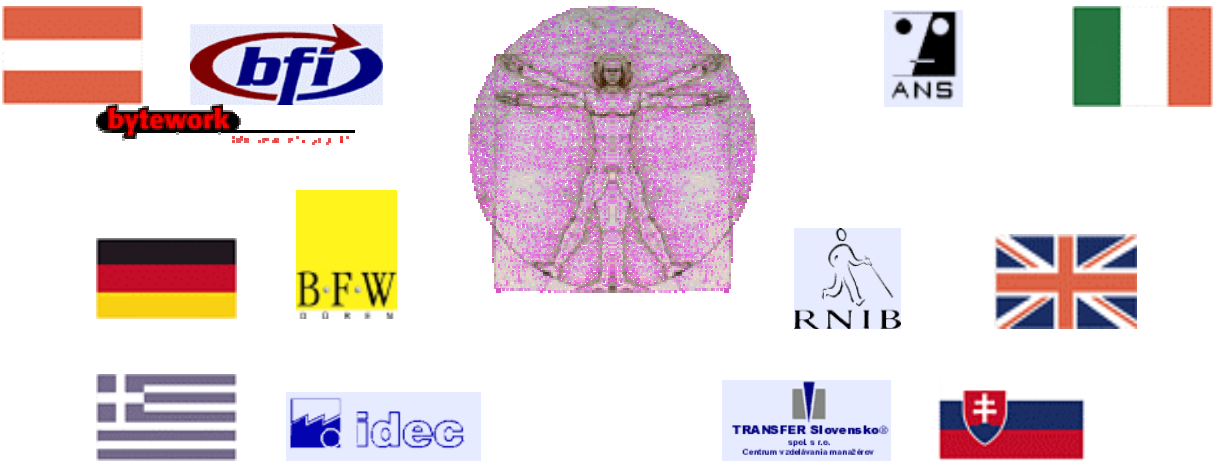
PROBIQ

Manuál
Borland C++ Builder 5.0



Education and Culture

Leonardo da Vinci

Tento projekt bol financovaný s podporou Európskej Únie.

Úvod k manuálu ProBiQ BCB5

Tento manuál a ucebná pomôcka je navrhnutý pre študentov a ucitelov, ktorí chcú programovat v prostredí Borland C++ Builder verzia 5. Použité materiály sú pestré a zaujímavé a poskytujú dostatok poznatkov k tomu, aby mohol jednotlivec získané zrucnosti využit na pracovisku.

Manuál sa zacína úvodom do základov programovania. Využívajúc rozmanité ucebných nástrojov, sprevádza študentov cez základné moduly a vrcholí v pokrocilých programovacích technikách.

Teoretický materiál, obsiahnutý v pociatocných kapitolách, môže pomáhat rozvíjat dobrú bázu poznatkov a bol zostavený tak, aby mohol byt používaný ako referencná prírucka alebo aj ako ucebná pomôcka.
Praktické úlohy a cvicenia v dalších castiach manuálu umožnia študentom precvicovat a overovat si získané poznatky.

Prajeme vám s týmto manuálom úspešný kurz!

Obsah

Kapitola 1: - Úvod do programovania

Ciel tejto kapitoly:
Naucit sa základy programovania, vrátane podstaty vyvíjania algoritmov.

1.1 Číselné sústavy

Ciel tejto podkapitoly:
Spoznat rozdielne číselné sústavy a byt schopný prevádzat čísla z jednej sústavy do druhej.

Táto kapitola poskytne prehlad číselných sústav a to predovšetkým tých, ktoré sa používajú v pocítacovej technológii: dvojková číselná sústava, šestnástková číselná sústava a osmicková číselná sústava.
Naucíme sa ako sú tieto sústavy prepojené i ako je možné konvertovat z jednej číselnej sústavy do druhej. Najprv zacneme s našou tradicnou desiatkovou číselnou sústavou.

1.1.1 Desiatková číselná sústava

Desiatková číselná sústava sa používa v každodennom pocítaní. Pozostáva z desiatich symbolov - arabských číslic. Každý symbol je interpretovaný podla svojej pozície v čísle a musí byt vynásobený mocninou desiatky. Z tohto dôvodu sa desiatková číselná sústava casto nazýva pozicná číselná sústava. Hodnota číslice v čísle sa nazýva jej pozicnou hodnotou. V závislosti od pozície je číslica vynásobená príslušnou mocninou základu číselnej sústavy. V desiatkovej sústave je základom číslo 10.

Príklad:
Zoberme číslo "240568". Ak zoberieme do úvahy pozicnú hodnotu číslic, musíme pocítat takto:

$2*10^5+4*10^4+0*10^3+5*10^2+6*10^1+8*10^0 = 240.568$
(pozn. prekladatela: bodka v čísle 240.568 oddeluje celú cast čísla od desatinnej casti ako slovenská desatinná ciarka)

V pocítacových systémoch sa využíva aj další typ číselnej sústavy - dvojková (tzv. binárna) číselná sústava. Pozrime sa, co sa deje v tejto sústave:

1.1.2 Binárna číselná sústava

Binárna číselná sústava je, tak ako desiatková číselná sústava, pozicná číselná sústava. Základom binárnej číselnej sústavy je číslo 2. To znamená, že pozicná hodnota číslice vzrastá z miesta na miesto po mocninách dvojky.

Z kolkých číslic pozostáva binárna číselná sústava?

Ak vydelíme nejaké celé číslo číslom 2, prenos bude 0 alebo 1. Takže binárna číselná sústava je zostavená z číslic 0 a 1.
Teraz uvidíme, preco je táto číselná sústava taká dôležitá pre pocítacové systémy. Pocítac pracuje s magnetizmom a elektrickým prúdom. Elektrický prúd môže, alebo nemusí prúdit, magnetické materiály môžu, alebo aj nemusia byt zmagnetizované. Toto všetko môže byt charakterizované dvomi číslami - 0 a 1.

Príklady:
Binárne číslo 11001011 má byt vyjadrené v desiatkovej číselnej sústave:
$(11001011)_2 = 1*2^7+1*2^6+0*2^5+0*2^4+1*2^3+0*2^2+1*2^1+1*2^0 = 203$
Naopak:
$(203)_{10}$ =

```
203/2 = 101 a zvyšok = 1;
                       najnižší bit (Least Significant Bit: LSB)
101/2 =  50 a zvyšok = 1;
 50/2 =  25 a zvyšok = 0;
 25/2 =  12 a zvyšok = 1;
 12/2 =   6 a zvyšok = 0;
  6/2 =   3 a zvyšok = 0;
  3/2 =   1 a zvyšok = 1;
  1/2 =   0 a zvyšok = 1.
                       najvyšší bit (Most Significant Bit: MSB)
```

Ak si precítame zvyšky zdola nahor dostaneme binárne číslo 11001011.

Na tomto príklade vidíme, že binárne číslo môžeme odvodit z čísla v desiatkovej sústave, ked číslo v desiatkovej sústave vydelíme základom 2 a zoberieme zvyšok.
Teraz prejdeme k dalšej, pre pocítace dôležitej číselnej sústave, k šestnástkovej číselnej sústave.

1.1.3 Šestnástková číselná sústava

Kvôli samotnému názvu tejto číselnej sústavy jednoducho pochopíme, že je to pozicná číselná sústava založená na císle 16. Ale ako sa dopracujeme až k šestnástim císliciam, ked doteraz ich poznáme iba desat? Matematici vyriešili tento problém nahradením císiel 10, 11, 12, 13, 14, a 15 velkými písmenami A, B, C, D, E a F.

Príklad:
$(0A6E)_{16} = 10*16^2+6*16^1+14*16^0 = 2,670$
(pozn. prekladatela: ciarka oddeluje tisícky)

Šestnástkový zápis císla vdesiatkovej sústave odvodíme jednoducho - císlo v desiatkovej sústave prevedieme do binárnej sústavy a potom binárne císlo rozdelíme po štvoriciach:

Príklad:
$(2670)_{10} = (0000\ 1010\ 0110\ 1110)_2 = (0A6E)_{16}$

Nakoniec sa pozrieme na dalšiu číselnú sústavu, ktorá sa v pocítacových systémoch neuplatnuje natolko ako binárna a šestnástková číselná sústava - na osmickovú číselnú sústavu.

1.1.4 Osmicková číselná sústava

Základom tejto pozicnej číselnej sústavy je číslo 8, ale používame len číslice od 0 po 7.

Príklad:
$(127)_8 = 1*8^2+2*8^1+7*8^0 = (87)_{10} = (0101\ 0111)_2 = (57)_{16}$
V tejto sústave môžeme zobrat binárne číslo a jeho cifry rozdelit na trojice tak, ako to vidíme v nasledujúcom príklade:
$(001\ 010\ 111)_2 = (127)_8$

1.1.5 Dalšie cvicenia

V týchto cviceniach by sme si mali osvojit pocítanie s binárnymi číslami ako i prevod medzi rôznymi číselnými sústavami.

a. Prevedte nasledujúce čísla v desiatkovej sústave na binárne čísla:
361
255
129

b. Prevedte nasledujúce císla v desiatkovej sústave na binárne císla:
1111100101
10000101
10101010

c. Prevedte nasledujúce císla v desiatkovej sústave na císla v šestnástkovej sústave:
894
777
1010

d. Prevedte nasledujúce císla v šestnástkovej sústave na binárne císla:
3AB
23F
ABC

e. Pokúste sa previest císla v cvicení D na binárne císla. V tomto bode pamätajte na metódu zostavovania štvoríc cifier.

1.2 Binárne císla

Ciel tejto podkapitoly:
Naucit sa pravidlá pocítania s císlami v binárnej sústave, najdôležitejšej z císelných sústav v pocítacovej technológii.

V tejto kapitole sa naucíme ako scitovat, odcitovat, delit, násobit dve binárne císla a že celý výpocet môže byt zredukovaný na jednoduché scítanie.

1.2.1 Teória pocítania s binárnymi císlami

Pri pocítaní s binárnymi císlami existujú nasledujúce pravidlá aritmetiky:

```
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 a 1 zvyšok pre nasledujúce miesto.
```

S týmito pravidlami je jednoduché vykonat scítanie 2 binárnych císel.
Príklad:

```
      1 1 1 0 1
+       1 0 1 0
```

```
zvyšok      1 1
    1 0 0 1 1 1
```

Pocítace nie sú schopné rozlišovat medzi rozdielnymi dlžkami binárnych císel. Ale toto ani nie je potrebné, lebo pocítac dáva každý údaj na špeciálne miesto v pamäti pocítaca. V závislosti od používaného pocítaca môže toto pamätové miesto (register) uchovávat 8, 16, alebo 32 bitov údajov.
Ak má pocítac 16 bitové registre, môže interpretovat predchádzajúci príklad takto:

```
     0000 0000 0001 1101
+    0000 0000 0000 1010
     0000 0000 0010 0111
```

Takzvaný najvyšší bit (na lavej strane) definuje algebraické znamienko, `0` oznacuje mínus a `1` oznacuje plus. Posledné císlo na pravej strane sa nazýva najnižší bit.
Ale ako by sme zapísali záporné císla? Zapíšeme ho použitím pravidla nazývaného jednotkový doplnok (Ones Complement):
0 vedie k 1, 1 vedie k 0.
Príklad:
`+23 je 0001 0111`
`-23 je 1110 1000`

Teraz dokážeme odcítat jedno binárne císlo od druhého:
Príklad:
Vypocítame `13+(-10)`:
`13 je 1101` v binárnej sústave
`10 je 1010` v binárnej sústave

```
          0000 1101
+         1111 0101    jednotkový doplnok
zvyšok 1  0000 0010
```

Teraz máme takýto problém: Co by sa malo spravit s posledným zvyškom? Pretože pamätové miesto môže prijat len 8 bitov, to jest jeden byte (pozn. cítaj bajt), zvyšok sa pridáva k najnižšiemu bitu. Takže `0000 0010` plus zvyšok vedie k 0000 0011.
Toto binárne císlo zodpovedá číslu 3 v desiatkovej sústave a predstavuje správny výsledok nášho výpoctu.
Poznámka: Posledný zvyšok sa casto pridáva k najnižšiemu bitu na

zaciatku. Toto sa nazýva dvojkový doplnok (Twos Complement):
Odcítanie binárnych císel sa uskutocnuje pripocítaním dvojkového doplnku.

Príklad:
Opät vypocítame `13 + (-10)`:

```
    1101 13 v desiatkovej sústave
    0101 jednotkový doplnok k 10
+   1    posledný zvyšok
 1  0011
```

Pretože sme už pridali posledný zvyšok, musíme odobrat vedúcu (prvú) 1.
Výsledok v desiatkovej sústave je 3.
Ako základné pravidlo si zapamätáme:
`Jednotkový doplnok + 1 k najnižšiemu bitu = dvojkový doplnok`
So zretelom na predchádzajúci text vidíme, že všetky výpocty môžu byt zredukované na jednoduché scítanie. Násobenie predstavuje len viacnásobné scítanie, delenie iba viacnásobné odcítanie a samotné odcítanie, ako je ukázané, je zredukované na obycajné scítanie.

Dalšie cvicenia:
a. Scítajte nasledujúce binárne císla:
`11101 a 10101`
`100101 a 111001`

b. Odcítajte nasledujúce binárne císla:
1011 od 111001
10010 od 10000101
1100 od 1011
1111 od 1101
Nezabudnite na zvyšok.

c. Vykonajte nasledujúce násobenia:
$(1011)_2 * 8_{10}$
$(1010)_2 * 4_{10}$
Našli ste nejaké zákonitosti?

Ako by podla vás mohol vyzerat výsledok nasledujúceho výpoctu:
`"1101 vynásobené císlom 16".`

1.3 Logika

Ciel tejto podkapitoly:

Naucit sa ako pocítac vykonáva všetky výpocty na základe troch logických funkcií.

V tejto kapitole sa pojednáva o logických prvkoch (hradlách) AND, NOT a OR. Napokon urobíme nejaké scítanie tak, ako to bude robit pocítac.

V predchádzajúcej kapitole sme naznacili, že všetky výpocty pocítaca môžu byt zredukované na scítanie. Ale ako pocítac uskutocnuje túto úlohu. Urcite pocítac nemôže pocítat ako ludia, ale vykonáva všetky výpocty používaním jednoduchých logických prvkov.
Tieto prvky sa tiež nazývajú logické hradlá.
Existujú 3 druhy hradiel:
Hradlo NOT (negácia)
Hradlo AND (konjunkcia)
Hradlo OR (disjunkcia)

Každé z hradiel budeme demonštrovat prostredníctvom tabulky hodnôt tiež nazývanej tabulka pravdivostných hodnôt.
Predstavuje všetky možné kombinácie dvoch vstupných hodnôt A a B a zobrazuje výstupnú hodnotu C.

1.3.1 Negácia (nie je pravda, že)

Prvok NOT je základným hradlom. Tento element len jednoducho otocí vstupnú hodnotu na jej doplnok. Kedže pocítac pozná len 0 a 1 nie je to problém.
Jednotkový doplnok získame takto:

```
A    C = NOT A
t    f
f    t
```

T zodpovedá pravde (true), f zodpovedá nepravde (false).
T a f sú casto zobrazené ako 1 alebo 0:

```
A    C = NOT A
1    0
0    1
```

1.3.2 Konjunkcia (a zároven)

Prvok AND má dve vstupné hodnoty. Ak má byt výstupná hodnota pravda, musia byt obe vstupné hodnoty pravda.

```
A    B    C = A AND B
```

```
1    1    1
1    0    0
0    1    0
0    0    0
```

1.3.3 Disjunkcia (alebo)

Prvok OR má tiež dve vstupné a jednu výstupnú hodnotu. Aby bola výstupná hodnota pravda, stací ak bude jedna vstupná hodnota pravda.

```
A    B    C = A OR B
1    1    1
1    0    1
0    1    1
0    0    0
```

1.3.4 Scítanie

S týmito logickými hradlami pocítac vykonáva všetky výpocty. Na príklade predvedieme ako funguje scítanie. Okrem výsledku C môže existovat aj zvyšok.

Polovicná scítacka (Half-Adder):
Poznámka pre inštruktora:
Táto schéma by sa mala postavit z lega alebo nakreslit na chrbte nevidiacich študentov.

A a B sú logické hodnoty, ktoré sa majú scítat, S je výsledok scítania, C prenos. Scítanie je založené na prvku XOR (vylucovací OR). Toto hradlo preveruje, že výstupná hodnota je pravda vtedy a len vtedy ked je práve jedna zo vstupných hodnôt pravda (vid tabulka pravdivostných hodnôt).
Tu je samotná tabulka pravdivostných hodnôt:

```
a    b    s    c
0    0    0    0
0    1    1    0
1    0    1    0
1    1    0    1
```

1.4 Pocítac a algoritmy

Ciel tejto podkapitoly:
Naucit sa aký je princíp výpoctu a co ovplyvnuje výkon pocítaca.

Obsah:
V tejto kapitole budeme hovorit o algoritmoch, procesoch, procesoroch, programoch a programovacích jazykoch.

1.4.1 Úvod

História viedla k dvom revolúciám. Prvá bola priemyselná revolúcia, druhá pocítacová revolúcia. V prvom prípade to viedlo k lepšiemu využitiu fyzickej sily, v druhom prípade k lepšiemu využitiu rozumových schopností.

Ale akože mal pocítac taký revolucný dopad? Za prvé pocítac je prístroj, ktorý robí jednoduché mechanické myšlienkové úlohy vykonávaním jednoduchých operácii velkou rýchlostou.
Jednoduchost operácií (typickými príkladmi sú scítavanie a porovnávanie dvoch císel) je v nepomere (disproportionate) k rýchlosti pocítania pocítaca (stovky miliónov až biliónov operácií za sekundu). Výsledkom je velké množstvo operácií, takže väcšina úloh môže byt vykonaná za krátky cas.

Ale toto vedie k problému. Usmernenie pocítaca tak, aby vykonával "prácu" znamená povedat, ktoré operácie by mali byt realizované. Musíme slovami popísat ako má byt "práca" vykonaná. Tento popis sa nazýva algoritmus. Algoritmus popisuje metódu na riešenie problému. Algoritmus pozostáva z postupnosti krokov (sequence), ktoré musia byt vykonané bez chýb, aby sme dospeli k požadovanému výsledku. Riadenie takéhoto algoritmu sa nazýva proces (process).

Náš každodenný život je plný algoritmov, napríklad:

Proces	Algoritmus	Typické kroky algoritmu
Pecenie koláca	Recept	odvážte múku a cukor, zoberte 3 vajícka
Hranie hudby	noty	interpretácia nôt
Štrikovanie pulóvra	vzor pre štrik.	prevrátené ocko, plné ocko

Na základe týchto príkladov je termín algoritmus jasný. Ale co je proces? Jednotka realizujúca procesy sa všeobecne nazýva procesor. Procesorom môže byt osoba alebo pocítac. Pocítac je velmi špecifickým procesorom, preto majú pocítace taký obrovský vplyv na náš každodenný život.

Univerzálny pocítac má tri hlavné casti:
Tie sú:
1. Procesor (Central Processing Unit - CPU)
2. Pamät, ktorá uchováva operácie algoritmu, informácie a údaje s ktorými sa operácia vykoná.

3. Vstupné a výstupné zariadenia, ktoré prenášajú algoritmus a údaje do hlavnej pamäte a cez ne pocítac zobrazí výsledok svojej práce.

1.4.2 Co robí pocítac vysoko výkonným?

Existujú tri hlavné charakteristiky pocítaca:

1.4.2.1 Rýchlost (speed)

CPU bežného pocítaca dokáže vykonat viac ako 2 bilióny operácií za sekundu (nazývané MIPS). Velmi komplexné úlohy môžu byt tak vypocítané vykonaním tisícok jednoduchých operácií.
Avšak si musíme uvedomit, že existujú problémy, ktoré nie sú adekvátne riešitelné prostredníctvom pocítaca, ako napr. neštruktúrovaná práca ako sú riadiace rozhodnutia.

1.4.2.2 Spolahlivost (reliability)

Napriek všeobecnému presvedceniu sú pocítacové chyby nezvycajné. Vyhlásenia o vysokých telefónnych úctoch sú pripisované väcšinou nesprávnym vstupným hodnotám (input) alebo chybám v algoritmoch. Toto je silná i slabá stránka pocítacov. Pocítac vykonáva predvolený algoritmus nezávisle na tom, ci je tento algoritmus správny alebo nie.

1.4.2.3 Pamät (memory)

Jedna z hlavných charakteristík pocítaca je schopnost uchovávat obrovské množstvo informácií, ktoré môžu byt velmi rýchlo sprístupnené. Kapacita pamäte a prístupová rýchlost záležia najmä na pamätovom médiu.

1.4.2.4 Programy a programovacie jazyky

Už pred tým bolo spomenuté, že procesor potrebuje algoritmus na vykonanie procesu. Algoritmus musí byt vyjadrený spôsobom, ktorému pocítac rozumie a je schopný ho aj vykonat. Procesor musí byt schopný interpretovat algoritmus, to znamená, že
a. musí vediet, co znamená každý jeden krok
b. aby bol schopný vykonat operáciu.
Ak je procesorom pocítac, algoritmus musí byt vyjadrený vo forme programu. Program sa zapisuje v programovacom jazyku. Samotná cinnost, ktorú musíme vykonat pri zapísaní algoritmu do programu, sa nazýva programovanie. Každý krok v algoritme je vyjadrený ako inštrukcia

alebo príkaz v programe. Teda program pozostáva z rady inštrukcií, pricom každá definuje operáciu, ktorá musí byt pocítacom vykonaná.

Typ inštrukcie v programe závisí na používanom programovacom jazyku. Existuje množstvo programovacích jazykov. Každý má svoju vlastnú sadu inštrukcií. Základné jazyky sa nazývajú tzv. strojové jazyky (machine language). Sú zostavené takým spôsobom, že každá inštrukcia môže byt priamo pochopená pocítacom. To znamená, že CPU je schopný rozumiet každej jednej inštrukcii a je schopný ju okamžite vykonat. Pretože tieto inštrukcie sú tak jednoduché (napr. scítanie dvoch císel), každá môže vyjadrovat len malú cast algoritmu. Preto je na vyjadrenie väcšiny algoritmov potrebné velké množstvo inštrukcií. Programovanie v strojovom jazyku je únavné a casovo nárocné.
Programovanie sa zjednodušilo vyvinutím tzv. vyšších programovacích jazykov napr. Cobol, Fortran, C alebo BASIC.
Tieto sa dajú ovela jednoduchšie naucit a sú tiež pre používatelov jednoduchšie ako strojové jazyky. Jednotlivá inštrukcia zahrna väcšiu cast algoritmu. Samozrejme, že aj tu musí pocítac nájst význam každej jednej inštrukcie. Spojenie medzi vyššími programovacími jazykmi a strojovými jazykmi, je stále velmi úzke. Takmer všetky programy napísané vo vyšších programovacích jazykoch musia byt, pred tým než môžu byt spustené, preložené do strojového kódu. Preklad vyjadruje každú inštrukciu vo vyššom programovacom jazyku radom ekvivalentmých inštrukcií v strojovom kóde.

Tento proces je predvedený v nasledujúcom príklade:
Algoritmus

Programovanie
vedie k
programu vo vyššom programovacom jazyku

Preklad
vedie k
programu v strojovom jazyku
vedie k
interpretácii prostredníctvom CPU

(Preferovaný proces sa vykoná)

1.5 Návrh algoritmu

Ciel tejto podkapitoly:
Naucit sa vyhýbat chybám v štruktúre algoritmov, ako sa zredukovaním problému na jednotlivé kroky, ktoré musia byt realizované na vyriešenie problému, vyvíja samotný program. Navyše sa naucit o niektorých nástrojoch, ktoré pomáhajú riešit problémy.

Hned na zaciatku tejto kapitoly budeme rozoberat 3 chyby, ktoré môžeme urobit pri programovaní: syntaktické, sémantické a v logické. Potom budeme hovorit o princípe zhora nadol ("top-down") na riešenie problémov. Nakoniec sa naucíme ako používat nástroje, ktoré pomáhajú riešit problémy: postupnost krokov, vetvenie a opakovanie.

1.5.1 Syntax, sémantika, logika: tri druhy chýb

Po objasnení idei programovania v predchádzajúcej kapitole sa teraz bližšie pozrieme na návrh algoritmov. Ako bolo už predtým uvedené, pocítac musí byt schopný interpretovat algoritmus, aby mohol vykonat popísaný proces. To znamená, že procesor musí byt schopný:

a. pochopit vyjadrenie algoritmu
b. vykonat špecifické operácie

Zamerajme sa na bod a):

Pochopenie algoritmu si vyžaduje dva kroky. Najprv musí byt procesor schopný identifikovat symboly, v ktorých je algoritmus prezentovaný, a pripísat im význam. K splneniu týchto úloh musí mat procesor poznatky o slovnej zásobe a gramatike jazyka, použitého na vyjadrenie algoritmu.

Kvantum gramatických pravidiel urcujúcich ako sa správne používajú symboly jazyka sa nazýva syntax jazyka. Program, ktorý je napísaný so správnou syntaxou sa nazýva syntakticky správny, co znamená, že každý použitý symbol v programe je použitý správnym spôsobom. Chyba v syntaxi sa nazýva syntaktická chyba. Na interpretáciu pocítacového programu sa obycajne vyžaduje syntaktická presnost.
Výnimky z tohto pravidla sú povolené iba ak je procesor dost bystrý a aj po vykonaní syntaktickej chyby odhadne zamýšlanú správnu syntax. V praxi je to velmi zriedkavé.

Druhým krokom kpochopeniu algoritmu je pripísanie významu každému jednému kroku valgoritme. Toto sa deje vo forme operácií, ktoré musí procesor vykonat. Napr. rovnica výnos = cena * množstvo znamená, že dve čísla nazývané cena a množstvo musia byt vynásobené a tretie číslo nazývané výnos predstavuje výsledok. Význam špeciálnych foriem výrazov v jazyku sa nazýva sémantika. Programovacie jazyky sú, co sa týka ich syntaxe a sémantiky, zostavené velmi jednoducho. Takže program sa dá syntakticky jednoducho analyzovat, odhliadnuc od jeho sémantiky. Inak povedané, skoro vo väčšine programovacích jazykov je velmi jednoduché objavit syntaktické chyby, ale je velmi tažké objavit sémantické chyby.

Na objavenie sémantických rozdielov musíme poznat špecifikované objekty. Obzvlášt musíme vediet o vlastnostiach týchto objektov a prepojeniach medzi nimi. To znamená, že procesor môže objavit sémantické rozdiely, iba ak má dostatok informácií o objektoch s ktorými algoritmus pracuje. Jednoduchý príklad to vysvetlí ovela presnejšie.
Predpokladajme, že procesor odhalí inštrukciu "Napíš názov trinásteho mesiaca v roku". Ak procesor vie, že je len dvanást mesiacov v roku, objaví túto sémantickú chybu pred tým, než sa bude pokúšat túto inštrukciu vykonávat. Ak iný procesor nevie nic omesiacoch a rokoch, bude túto inštrukciu akceptovat ako platnú a posnaží sa ju vykonat napr. neúspešne pohladá mesiac v kalendári.
Pokial inštrukcia nie je vykonávaná, procesor si nevšimne, že neexistuje trinásty mesiac a až potom bude táto nezrovnalost jasná.

Popri syntaktických a sémantických chybách existuje ešte tretí druh chýb, ktoré stoja za zmienku. Sú to logické chyby. Program môže byt syntakticky správny a bez sémantických nesprávností, ale nemusí popísat želanú procedúru správnym spôsobom. Pozrime sa na nasledujúci algoritmus na vypocítanie obvodu kruhu:
"Vypocítaj obvod tak, že vynásobíš polomer a pí.
Tento algoritmus je správny sémanticky i syntakticky. Ale vedie k nesprávnemu výsledku kvôli chýbajúcemu koeficientu 2, co vedie k logickej chybe.
V tomto bode by sme mali jasne konštatovat, že pocítac nie je schopný odhalit logickú chybu. Na nájdenie logických chýb musí clovek strávit cas pozorovaním postupnosti krokov algoritmu a používaním testovacích údajov.

1.5.2 Stratégia zhora nadol (top-down) pri riešení problémov

Štruktúra algoritmu je obycajne prekvapujúco zložitá, najmä ked je

vykonávaný proces komplikovaný. Nárocnost zostavenia algoritmu sa zvyšuje ak je procesorom pocítac. Pocítacu chýba intuícia a cit (pre nieco) a nie je schopný zistit, ci algoritmus popisuje vytúženú procedúru. Bežnou chybou je, že postup je takmer, ale nie úplne dotiahnutý.

Preto pri zostavovaní algoritmu musíme zabezpecit, aby algoritmus presne popisoval proces a všetky okolnosti boli známe. Ak je proces velmi komplexný, návrh je velmi nárocný. V skutocnosti úspech vývojára závisí na presnom metodickom prístupe k zostavovaniu algoritmov. Tento prístup sa nazýva návrh zhora nadol (top-down-design).
"Návrh zhora nadol" je najnovšou verziou starého divide et conquera" (lat. rozdeluj a panuj). Princípom je odskúšanie výkonného procesu vo viacerých malých krokoch, pricom každý jeden je popísaný menej rozsiahlym algoritmom, jednoduchším ako celý pôvodný proces. Pretože každý z týchto ciastkových algoritmov je jednoduchší ako celý algoritmus, vývojár lahšie porozumie ako ho treba zostavit. Z toho dôvodu je schopný navrhovat podrobnejšie, ako ked sa pokúša zaobchádzat s celým algoritmom.

Samotné ciastkové algoritmy môžu byt tiež rozdelené na menšie casti, lebo cím sú jednoduchšie, tým detailnejšie a presnejšie môžu byt vyjadrené. Zdokonalenie v zostavovaní algoritmu týmto spôsobom môže postupovat až kým sú kroky dostatocne detailné a presné na ich vykonanie procesorom.

Jednoduchý príklad tento úkon vysvetlí. Predpokladajme, že máme malého robota pracujúceho ako lokaj. Tento robot má algoritmus na prípravu šálky kakaa. Prvá verzia tohto algoritmu môže vyzerat takto:
1. Daj zovriet mlieko,
2. daj do šálky kakao,
3. napln šálku mliekom.
Pravdepodobne tieto kroky nie sú dostatocne podrobné, takže robot im nebude rozumiet. Preto musíme doplnit každý jeden krok radou jednoduchších krokov, pricom budeme každý krok špecifikovat podrobnejšie ako pôvodný:
Napríklad krok
1. Daj zovriet mlieko.
môže byt rozšírený ako:
1.1. Napln nádobu mliekom,
1.2. zapni varic,
1.3. pockaj kým mlieko zovrie,
1.4. vypni varic.

Rovnakým spôsobom
2. Daj do šálky kakao.
môže byt rozšírené ako:
2.1. Otvor nádobu s kakaom,
2.2. naber za plnú lyžicku kakaa,
2.3. vysyp kakao do šálky,
2.4. zatvor nádobu s kakaom.
A tretí krok:
3. Napln šálku mliekom.
môže byt rozšírený ako:
3.1. Lej z nádoby mlieko do šálky až kým (until) šálka nie je plná.
Tým sme ukoncili prvé rozšírenie nášho pôvodného algoritmu.
Ak je náš robot schopný interpretovat všetky ztýchto krokov správnym spôsobom, tak je algoritmus rozšírený a návrh je skompletizovaný.
Je však tiež možné, že vylepšenie môže íst o jeden krok dalej. Krok
1.1. Napln nádobu mliekom.
môže byt rozšírený ako:
1.1.2. Prines mlieko z chladnicky,
1.1.3. nalej mlieko do nádoby,
1.1.4. daj zvyšok mlieka späť do chladnicky.

Po dalšom rozširovaní už môže byt každý jeden krok interpretovaný robotom. V tomto bode je návrh algoritmu ukoncený.

1.5.3 Nástroje na riešenie problémov

Pre vývoj algoritmov sú prístupné aj nástroje. V tomto prípade využívame postupnost krokov (sequence), vetvenie (selection) a opakovanie (iteration).

1.5.4 Postupnost krokov (sequence)

V predchádzajúcej casti je algoritmus zrejmý. Zahrna jednoduché kroky, ktoré musia byt vykonané jeden za druhým. Takýto algoritmus predstavuje postupnost krokov, co znamená, že:

- Krok sa vykonáva v urcitom momente.
- Každý krok sa vykonáva len raz, žiadny sa neopakuje, žiadny sa nevynecháva.
- Dôležité je poradie. Každý krok sa robí jeden za druhým.
- Algoritmus je ukoncený vykonaním posledného kroku.

Algoritmus pozostávajúci len z postupnosti krokov je neprispôsobivý, pretože proces jeho vykonávania je velmi prísny (rigid) a nemôže byt za

žiadnych okolností pozmenený. Musíme vziat do úvahy, že kombinácia krokov tvoriacich postupnost je velmi primitívny algoritmus. V nasledujúcich castiach sa pozrieme na ovela zložitejšie schémy.

1.5.5 Vetvenie (selection)

Vetvenie (selection) za urcitých podmienok používame na pozmenenie postupnosti krokov. Spomenme si na robota. Náš robot by nebol schopný pripravit šálku kakaa keby bola nádoba s kakaom prázdna. V takomto prípade by sme mali dostat inštrukciu ako táto:
- Zober z policky novú, plnú nádobu kakaa.
Rozdiel medzi oboma prípadmi je rozšírený novou formuláciou:
Zober z policky nádobu kakaa.
(IF) Ak je nádoba prázdna
(THEN) tak zober z police novú.
Odstrán vrchnák z nádoby
Všeobecná formulácia tejto inštrukcie je:

IF podmienka
    THEN krok

Kde podmienka oznacuje okolnosti, kedy by sa mal krok vykonat. Ked sa splní podmienka, potom sa vykoná krok, inak nie.
V príklade uvedenom vyššie je vetvenie použité na upresnenie, ci urcitý krok má byt vykonaný alebo nie. Rozšírenie je realizované ponúknutím dvoch alternatívnych spôsobov. Všeobecná formulácia takejto možnosti je:

IF podmienka
    THEN krok 1
    ELSE krok 2

Kde podmienka jasne upresnuje, ci krok 1 alebo krok 2 majú byt vykonané. Vezmime do úvahy, že je možné použit aj dalšiu podmienku IF na mieste kroku 2. Toto sa nazýva viacnásobné vetvenie.
Prednostou vetvenia je, že umožnuje procesu prejst algoritmom rôznymi spôsobmi v závislosti od okolností. Vnorené vetvenie poskytuje niekolko spôsobov. Bez vetvenia by nebolo možné napísat algoritmus pre praktické použitie.

1.5.6 Opakovanie (iteration)

Prostredníctvom opakovania (iteration) sa pokúšame nájst možnosti

zopakovania urcitých castí algoritmu. Napríklad pátrame po urcitom mene v nekonecnom zozname mien. Môžeme byt schopní vytvorit algoritmus, ktorý pracuje výhradne s vnoreným vetvením, ale toto riešenie by nebolo za žiadnych okolností uspokojujúce.
Tu sa objavuje opakovanie, ktoré je všeobecne formulované ako:

```
REPEAT
        algoritmus
UNTIL podmienka
```

Toto znamená, že cast algoritmu medzi rezervovanými slovami REPEAT (opakuj) a UNTIL (až kým) musí byt opakovaná až kým nebude platit podmienka umiestnená za UNTIL.
Tomuto opakovaniu hovoríme cyklus (loop) a cast, ktorá má byt zopakovaná sa nazýva telo cyklu (loop body). Podmienka nasledujúca za UNTIL sa nazýva podmienka ukoncenia cyklu (termination condition). Další názov pre tento druh cyklu je cyklus s podmienkou na konci.

Prednostou opakovania je možnost vyjadrit proces s nedefinovanou dlžkou pomocou algoritmu s pevnou dlžkou. Ale jednou z najvážnejších chýb viacerých vývojárov je zanedbanie presného definovania podmienky ukoncenia. Tento prípad sa nazýva uzavretý cyklus (closed loop), co znamená, že tento cyklus sa nikdy neukoncí a algoritmus nikdy neprestane pracovat.
Cyklus repeat ... until ... nie je jediným možným opakovaním. Tento cyklus má na konci svoju podmienku ukoncenia, co znamená, že telo cyklu sa vykoná aspon raz.
V niektorých prípadoch nie je nutné vykonat telo cyklu, lebo podmienka na zaciatku cyklu nie je splnená. Pre tento prípad existuje cyklus while ... repeat .... (kým platí ... opakuj ...)
Všeobecný zápis je:

```
WHILE podmienka REPEAT
        telo cyklu
```

To znamená, že telo cyklu bude vykonané len pokial je podmienka splnená. Ale v tomto prípade telo cyklu nemusí byt vykonané ani raz. Preto sa tento druh cyklu nazýva cyklus s podmienkou na zaciatku.

Posledným z opakovaní je velmi jednoduché opakovanie. V tomto prípade je pocet opakovaní známy ešte pred tým ako sa cyklus zacne vykonávat.

Všeobecný zápis je:

REPEAT n-krát
telo cyklu

Kde N predstavuje akékolvek celé číslo. Tento druh cyklu (loop) sa nazýva cyklus s konecným poctom opakovaní.
Ako dlho tento cyklus trvá, je známe od zaciatku a koniec je tiež jasný. Tento druh cyklu predstavuje bezpecný spôsob návrhu algoritmov, ale nie je taký výkonný ako oba predchádzajúce cykly.

1.5.7 Dalšie cvicenia

Tu sú uvedené nejaké krátke algoritmy pre rôzne typy cyklov. Pokúste sa týmto príkladom porozumiet.
1. Prvý príklad pre vetvenie je algoritmus, ktorý používate, ked prídete k semaforu.

(IF) Ak je svetlo cervené alebo oranžové
(THEN) tak zastan
(ELSE) inak chod dalej

2. Ak sa semafor pokazí, mohli by sme rozšírit algoritmus:

(IF) Ak nie je na križovatke žiadne svetlo,
(THEN) tak šoféruj s mimoriadnou opatrnostou
(ELSE) (IF) inak Ak je svetlo na križovatke cervené alebo oranžové
(THEN) tak zastan
(ELSE) inak pokracuj v jazde
(Toto je príklad viacnásobného vetvenia.)

3. Hladáte najväcšie z troch císel. Môžete však porovnávat len dve císla naraz. Ak sú tieto císla oznacené X, Y a Z , algoritmus je:

(IF) Ak x > y
(THEN) (IF) tak Ak x > z
(THEN) tak vyber x
(ELSE) inak vyber z
(ELSE) (IF) inak Ak y > z
(THEN) tak vyber y
(ELSE) inak vyber z

4. V zozname mien vyhladávate urcité meno a chcete získat i adresu. Bol by tento algoritmus riešením?

Precítaj prvé meno v zozname.
(REPEAT) opakuj
    (IF) ak je to hladané meno
        (THEN) tak zapamätaj si priradenú adresu.
        (ELSE) inak cítaj nasledujúce meno v zozname
(UNTIL) až kým sa meno nenájde alebo je koniec zoznamu

Co by sa stalo, keby v casti UNTIL chýbal text "alebo je koniec zoznamu".

5. Chcete získat súcin prvých n císiel. V matematike tomu hovoríme n-faktoriál alebo n! Algoritmus môže byt:

Zober hodnotu n
nastav súcin na 1
(REPEAT) opakuj pre každé celé císlo od 1 po n
    vynásob súcin celým císlom
zapíš súcin

Ku ktorému druhu cyklu patrí tento príklad?

Viac príkladov na návrh algoritmov
Navrhnite algoritmus, ktorý nájde všetky prvocísla. Ako by sme mohli pocítat ak je císlo prvocíslom. Ako by mohol vyzerat algoritmus, ktorý nájde všetky prvocísla najbližšie k hocijakému císlu?
Známym príkladom v matematike je urcenie najväcšieho spolocného delitela dvoch celých císel. Ak nepoužívate cyklus s podmienkou na konci, tak to zdôvodnite.

1.6 Prekladové programy

Ciel tejto podkapitoly:
Naucit sa, ktoré z nástrojov slúžia na preloženie kódu napísaného v danom programovacom jazyku do strojového kódu.

V tejto kapitole sa budeme ucit o kompilátore, linkeri (spojovací program), assembleri a interpretri a tiež ako sa tieto nástroje používajú.

Ako bolo spomenuté v predchádzajúcich kapitolách väcšina programov,

ktoré nie sú napísané v strojovom jazyku, ale vo vyššom programovacom jazyku, ako sú C a Pascal, sa musia preložit do strojového kódu. Na toto slúžia nástroje ako kompilátor, linker, assembler a interpreter. Teraz budeme tieto nástroje identifikovat. Na zaciatku zopakujeme definíciu prekladacieho programu.
Prekladací program je nástroj, ktorý cíta inštrukcie vo vyššom programovacom jazyku alebo jazyku assemblera, analyzuje a prevádza ich do strojových inštrukcií rovnakého významu.
Okrem vlastného prekladania (= konverzie výrazov na iné) sa pôvodný program syntakticky testuje a v prípade potreby sa zobrazujú chybové hlásenia.

Assembler je prekladový program, ktorý konvertuje pôvodné inštrukcie písané v pôvodnom jazyku assemblera do výsledných inštrukcií primeraného strojového jazyka. Termín jazyk assemblera tu používame po prvýkrát a preto ho aj hned vysvetlíme. Jazyk assemblera je nástroj reprezentujúci strojové kódy ako zapamätatelnejšie a zrozumitelnejšie. Inštrukcie sú tak zobrazené mnemotechnickými skratkami, s ktorými sa pracuje ovela jednoduchšie.
Príkaz na spocítanie dvoch císel môže vyzerat ako:

```
ADD 3, 4 (scítaj)
```

Kompilátor je prekladací program, ktorý konvertuje pôvodný kód písaný vyšším programovacím jazykom na výsledné inštrukcie v strojovo orientovanom jazyku.
Výsledným jazykom kompilovania je spravidla samotný strojový jazyk (výsledný kód), ale preklad sa niekedy robí aj do jazyka assemblera špecifického pre konkrétny pocítac. Pocas prekladu je pôvodná inštrukcia obycajne rozdelená na vela inštrukcií v strojovom kóde (tzv. preklad 1 ku n).

Assembler a kompilátor prekladajú celé pôvodné programy do výsledných programov pred tým, než sú spustené. Pocas tohto prekladu nie je možné do programu zasahovat.

Výsledné programy vytvorené assemblerom a kompilátorom nie sú spúštatelné, lebo sa ešte musia pridat niektoré casti programu, ktoré pôvodný program potreboval, ako napr. vstupno/výstupné procedúry.
Toto vykonáva linker, ktorý zároven prepocítava príslušné (externé) adresy vzhladom na zaciatok programu. Teraz je program nacítatelný a spustitelný.

Interpreter je program, ktorý ihned prekladá a vykonáva existujúci zdrojový kód. To znamená, že sa nevytvára žiadny výsledný kód. Nevýhodou je, že touto metódou vytvoríme pomaly spustitelný program. Spomínaný postup spájania (linking) sa nepoužíva. Program beží dynamicky, bez pôsobenia používatela. Prednostou interpretera je interaktívne programovanie.

Príkladmi interpretera sú programovacie jazyky ako BASIC, APL, VBA, Perl. Tieto sú väčšinou interpretované. Iné vyššie programovacie jazyky ako Pascal, Cobol, Fortran, C, Java alebo Visual Basic využívajú rôzne kompilátory a spojovacie programy (linker).

1.7 Príloha: Riadiace štruktúry v praxi

Ciel:
Naucit sa nieco o realizácii rôznych riadiacich štruktúr v rôznych jazykoch.

V tejto kapitole budeme rozoberat operácie cyklus (loop) s podmienkou na konci, cyklus s podmienkou na zaciatku, cyklus s konecným poctom opakovaní a vetvenie (selection) v jazykoch Basic, C/C++, a Pascal.
Dalej ukážeme jednotlivé riadiace štruktúry v bežných programovacích jazykoch C/C++, Pascal a BASIC.

1.7.1 Cyklus s podmienkou na konci

BASIC:
```
DO
        telo cyklu
LOOP UNTIL podmienka
```

C/C++:
```
do
        {
                telo cyklu
        }
while (podmienka)
```

Pascal:
```
repeat
        telo cyklu
```

until podmienka

V jazykoch Basic a Pascal nájdete slovo until (až kým) pri podmienke ukoncenia cyklu. V oboch týchto prípadoch môže niekto pochopit cyklus s podmienkou na konci v zmysle, že cyklus je pravdivý, ked sa dosiahne podmienka ukoncenia. Na rozdiel od C/C++, kde zistíte, že podmienka ukoncenia je while (kým). Tu sa cyklus opakuje tak dlho, pokial podmienka platí (t. j. je pravdivá).

1.7.2 Cyklus s podmienkou na zaciatku

BASIC:
DO WHILE podmienka
    telo cyklu
LOOP

C/C++:
while (podmienka)
    {
        telo cyklu
    }

Pascal:
while podmienka
    begin
        telo cyklu
    end

1.7.3 Cyklus s pevným poctom opakovaní

BASIC:
FOR pocitadlo=1 TO 20
    Loop telo cyklu
NEXT pocitadlo

C/C++:
FOR (pocitadlo =0; pocitadlo <20; pocitadlo ++)
    {
        telo cyklu
    }

Pascal:

```
FOR pocitadlo:=1 to 20 do
        begin
                telo cyklu
        end
```

Na zlepšenie citatelnosti sme v tejto casti použili odsadenie blokov kódu. V programovacích jazykoch ako C a Pascal sú takéto bloky zapísané pomocou symbolov {} (zložené zátvorky) v C a príkazov „begin“ (zaciatok) a „end“ (koniec).

1.7.4 Vetvenie

```
BASIC:
IF podmienka
        THEN vetva 1
        ELSE vetva 2
END IF

C/C++:
if (podmienka) vetva 1;
        else vetva 2

Pascal:
if podmienka
        then vetva 1
        else vetva 2
```

Kapitola 2: Základy C++

Ciel tejto kapitoly:
Získat prehlad o histórii C, C++, prostredia Borland C++ Builder a pozriet sa aj na iné nástroje pre C++.

Študent by mal tiež získat strucný prehlad o C/C++ a o jeho prepojení na iné programovacie jazyky.

2.1 História C++, Borland C++ a definícia tohto programovacieho jazyka

V tejto casti budeme hovorit o rozvoji C v sedemdesiatych rokoch a dalšom rozvoji C++ vrokoch osemdesiatych. Potom si ukážeme, ako Windows prispel k rozvoju grafických vývojárskych prostredí (graphical development environments).

C bol vyvinutý vrokoch 1971/72 vA&T Bell Laboratories pre operacný systém UNIX. Jeho tvorcami boli Denis Ritchie a Brian Kernighan. Predchodcom C bol programovací jazyk nazývaný B. V roku 1978 C dostal svoju prvú presnú definíciu a v roku 1983 the American National Standard Institute (ANSI) štandardizoval C na ANSI C.

V tom case bol C ešte stále procedurálnym programovacím jazykom. To znamená, že raz naprogramovaný kód mohol byt použitý len pre jeden projekt a nemohol byt dalej rozvíjaný ci (jednoducho) použitý vdalšom projekte.
Z tohto dôvodu bol C dalej rozvinutý na C++, objektovo orientovaný programovací jazyk (OOP), ovela výkonnejší ako C. Objektovo orientovaný kód sa môže použit viac ako raz, lebo každý objektovo orientovaný program je samostatný celok (tzv. "zapúzdrený") a nový OO-projekt môže byt odvodený z existujúceho OO-projektu (tzv. pretaženie - overloading a dedicnost - inheritance). Takže, používaním OOP nemusíte znova "objavovat koleso" pre každý nový programovací projekt. Podstatou C++ je stále C, takže casto sú oba spomenuté spolocne ako C/C++.

Ked sa zacal používat Microsoft Windows, vývojári museli naprogramovat okná, ponuky, kurzory pre myš, tlacidlá, vstupné polícka, rozbalovacie zoznamy atd. Toto bolo velmi zdlhavé a nudné. Tak niektorí výrobcovia

(ako spolocnosti Microsoft alebo Borland) priniesli na trh nástroje, ktoré obsahovali všetky tieto komponenty a tieto nástroje používali programátori. Nazývajú sa Integrované vývojárske prostredia (Integrated Development Environments - IDE). Najznámejšie sú Visual Basic a Visual C++ od spolocnosti Microsoft, a Delphi a C++ Builder od spolocnosti Borland.

Borland bol už slávny svojimi rýchlymi kompilátormi pre C/C++ (kompilátory sú nástroje, ktoré vytvoria aplikáciu z programovacieho kódu). Takže Borland C++ Builder je programovací nástroj založený na IDE, ktorý využíva C a C++ na programovanie aplikácií.

2.2 C++ - jeho kategorizácia a prednosti

C++ je, ako bolo pred tým spomenuté, objektovo orientovaný programovací (OOP) jazyk. Nie je však jediný. Existujú aj iné ako napr. SmallTalk, Java, Delphi. Niektorí si myslia, že to nie je najlepší OOP jazyk, ale je široko používaný, vlastne samotný Microsoft Windows je naprogramovaný prevažne v C++ a tzv. WinAPI-funkciách, funkcie dostupné pre programátorov priamo od spolocnosti Microsoft pre priamu interakciu s Windows, sú napísané v C++.

Vela iných programovacích jazykov je odvodených z C/C++ napr. Java, Perl, PHP. Takže ak ovládate programovanie v C/C++, lahko môžete prejst k týmto programovacím jazykom. C++ používa rovnakú syntax (spôsob akým zapisujete programovací kód) ako C. A co viac, môžete napísat C programy s C++, pretože C je vlastne C++ bez OOP.
Z toho dôvodu sa nazýva C/C++. Koncepcia OOP je v tomto manuáli predstavená a doplnená príkladmi o nieco neskôr.

C++ sa lahko rozšíril o niektoré funkcie. Jednou z hlavných koncepcií C++ je použitie funkcií pre programy, ktoré sú uchované v súboroch nazývaných "knižnice" (libraries). Takže ak chcete sprístupnit urcité, vami naprogramované funkcie, môžete ich zverejnit spolu s knižnicou a každý iný programátor ich bude schopný použit. Vela iných programovacích jazykov naväzovalo na túto koncepciu, ale toto je ešte stále, jedna z koncepcií ktorá robí C/C++ takým výkonným.

Dalšou prednostou C/C++ je, že môžete programovat na nízkej úrovni, velmi blízko kvami používanému hardvéru. Väcšina tzv. "volaní funkcií" hardvéru sú C volania?. Takže ak chcete vyvinút programy, ktoré využívajú osobité funkcie zariadení, používajte C/C++.

Borland-Builder prichádza v troch verziách, rozdielnych vo vybavení a v cene:
The Borland C++ Builder Standard: najmenšie vybavenie, bez nástrojov na pripojenie databázy;
The Borland C++ Builder Professional: väcšie vybavenie a nástroje na pripojenie databázy;
The Borland C++ Builder Enterprise: väcšina vybavenia pre velké programovacie skupiny a pre rozvoj sietových aplikácií.

Kapitola 3: Úvod do IDE BCB 5

Ciel tejto kapitoly*:*
Prejst úvodom do Integrovaného vývojového prostredia (IDE) Borland C++ Builder 5.

3.1 IDE BCB5

IDE (Integrated Development Environment) je výkonný editor, ktorý nám pomôže vytvárat programy pre Windows a pre DOS. Dalej spomínané veci sú menej dôležité.

IDE nám pomôže

a) vytvárat programy vzhladom podobné Windows, s oknami, s ponukami, s tlacidlami, s rozbalovacími zoznamami, kurzormi myši a pod. bez nutnosti zaoberat sa detailami.
b) jednoducho písat funkcie pre naše tlacidlá, ponuky, rozbalovacie zoznamy, a pod.
c) jednoducho spravovat aplikácie pozostávajúce z mnohých castí
d) jednoducho hladat chyby (nazývané “bugy”) v našom kóde
e) jednoducho písat výkonné aplikácie s možnostou prepojenia s databázou
f) jednoducho písat internetové aplikácie
g) velmi jednoducho robit mnoho iných vecí, ktoré programátor robí.

Preco programátori, špeciálne nevidiaci programátori, potrebujú návod k BCB? IDE programu Borland Builder nie je obsiahnuté v jednom aplikacnom okne, ale pozostáva z mnohých okien plávajúcich na ploche, za ktorými je pracovná plocha MS Windows (alebo iné programy, ktoré ste nechali otvorené na pozadí). Teda, ak cítame obrazovku riadok po riadku s našim cítacom obrazovky, nájdeme informácie, ktoré patria kBuilderu ako aj informácie, ktoré patria k iným programom.

Ak spustíme Builder po prvý krát, objavia sa tieto 4 okná:

1) “Builder C++ - Project1” - okno v hornej casti obrazovky obsahujúce aplikacnú ponuku a panely nástrojov.
2) “Object Inspector” - okno na lavej strane obrazovky.
3) Okno “Form1”, prislúchajúce k nášmu prvému programu, nad prvým a napravo od druhého okna.
4) “Unit1.cpp” - okno, ktoré pod oknom “Form1” nie je viditelné, obsahujúce kód nášho programu.

Môžeme zatvorit každé okno okrem prvého okna “Builder C++”, pretože tým zatvoríte Borland Builder C++ 5.

Okno “Builder C++” aktivujeme klávesom ALT. Okno “Object Inspector” aktivujeme funkcným klávesom F11. Medzi oknami “Form1” a “Unit1.cpp” sa prepíname funkcným klávesom F12. Ak máme problém s prepínaním, stlacme najprv ESC.

Ako sme už možno uhádli, názvy okien “Form1” a “Unit1.cpp” sú len docasné názvy pre náš prvý projekt s docasným názvom “Project1”. Ak našu prácu prvý krát uložíme, dostanú okná názvy, ktoré sme im dali.
Tak isto môžeme pristupovat ku všetkým oknám cez ponuku “View”, ALT+V. Je ich mnoho, ale teraz sa nebudeme zmienovat o všetkých, pretože pre nás nebudú mat význam skôr, než zacneme programovat. Ak sme omylom jedno z nich otvorili, zatvoríme ho klávesovou skratkou ALT+F4. Tento príkaz nezatvorí Borland Builder, ale zatvorí aktívne okno.
Našu prácu v Borland Builder-i môžeme skoncit aktivovaním ponuky “File” a “Exit, alebo skrátene ALT+F, X.
Hlavné okno “Builder C++” bude opísané vdalšej kapitole, ale niektoré okná opíšeme teraz.

3.1.1 Okno “Form”

Okno “Form1” je plocha našej prvej aplikácie. Je to okno na ktoré budeme umiestnovat tlacidlá, písat text, umiestnovat rozbalovacie zoznamy a pod. Tieto sa nazývajú "komponenty" (components).

Ak sme funkcným klávesom F12 aktivovali okno “Form”, budeme sa prepínat medzi všetkými komponentmi, ktoré sme umiestnili do formulára.
Ak sme aktivovali jeden z komponentov vo formulári (alebo formulár samotný), stlacme F11 a aktivujme okno “Object Inspector”.

3.1.2 Okno “Unit”

Okno “Unit” obsahuje všetok kód programu - ten, ktorý je generovaný automaticky (velmi dobre!) a ten, ktorý píšeme sami (ešte lepšie!).
Okno unit je “editor” Borland Builder-u C++ 5.

Pretože Builder generuje kód automaticky, okno nie je prázdne ani ak ho otvoríme prvý krát.

Okno unit obsahuje všetky súbory unitu, do ktorých sme uložili kód, a tiež aj nejaké dalšie textové súbory. Tieto súbory môžeme aktivovat stlacením CTRL-TAB. Ak súbor unitu nemôžeme dosiahnut stlacením CTRL-TAB, musíte ho najprv otvorit.

Súbor unitu, alebo pomocný súbor, môžeme zatvorit stlacením CTRL+F4.

3.1.3 Inšpektor objektov ("Object Inspector")

Okná “Object Inspector-a” obsahujú informácie o komponentoch vo formulári, ktorý sme pred tým aktivovali.

Je tu:

- zoznam momentálne používaných objektov na tomto formulári,
- zoznam vlastností (properties) zvoleného objektu,
- a na dalšej záložke zoznam udalostí (events) zvoleného objektu.

Podrobnejšie:

a) Meno aktuálneho komponentu, nasledované dvojbodkou a druhom komponentu, ku ktorému prislúcha aktuálny komponent, napr. “Form1:TForm1", co znamená komponent s názvom “Form1” typu “TForm1” (ak ste skúsený programátor môžete povedat “objekt

Form1 triedy TForm1"). Táto informácia sa nachádza na vrchu inšpektora objektov pod titulkom okna. Môžeme k nej pristupovat stlacením klávesovej skratky CTRL+šipka dolu. Týmto otvoríme zoznam všetkých komponentov vo formulári a môžeme jeden z nich nastavit ako aktívny.

b) Zoznam s "vlastnostami" (properties) aktuálneho komponentu ako názov, dlžka, výška a mnoho iných. Tieto informácie sú zobrazované v tabulke s dvoma stlpcami. Lavý stlpec obsahuje názov vlastnosti, pravý hodnotu vlastnosti, napr. v lavom “Name”, v pravom “Form1”. Každý riadok obsahuje informácie o jednej vlastnosti. Medzi vlastnostami sa môžeme prepínat šípkou hore alebo šípkou dolu. Medzi pravým a lavým stlpcom sa môžeme prepínat klávesom TAB.
Názvy vlastností v lavom stlpci sú utriedené podla abecedy. Rovnako môžeme pristupovat ku konkrétnej vlastnosti ak poznáme jej meno. Ak budeme písat názov vlastnosti v lavom stlpci, budeme sa približovat k zvolenej vlastnosti tým viac, cím viac písmen napíšeme. Napríklad ak stlacíme “n” presunieme sa na riadok s prvou vlastnostou, ktorá zacína písmenom “n”, v našom prípade k vlastnosti “Name”. Stlacme kláves TAB a prepneme sa do pravého stlpca, ktorý obsahuje aktuálnu hodnotu vlastnosti “Name”, co je názov aktuálneho formulára - “Form1”. Napíšme nový názov a premenujeme náš formulár (možno náš formulár pomenujeme “frm_MyFirstForm”, kde zaciatok “frm” hovorí, že pomenovaný komponent je typu “Form” a cast za znakom podciarknutie je nejaký nami zvolený názov).
S “Properties” môžeme urcit mnoho vlastností aktuálneho komponentu. Napr. ho môžeme bez použitia myši umiestnit do

formulára s vlastnostami “Top” a “Left”, kde “Top” je vzdialenost medzi horným okrajom formulára a horným okrajom aktuálneho komponentu a “Left” je vzdialenost medzi lavým okrajom formulára a lavým okrajom aktuálneho komponentu.
Ak je aktuálny komponent formulár, vzdialenosti sú medzi okrajmi obrazovky a formulára.
Neskôr uvidíme, že vlastnosti komponentu môžeme menit v kóde programu.
Ak je kurzor na riadku vlastnosti a stlacíme F1, vyvoláme pomocníka k tejto vlastnosti.
Niektoré vlastnosti musia mat urcité preddefinované hodnoty. Potom pravý stlpec s hodnotou je realizovaný ako rozbalovací zoznam, ktorý môžeme otvorit stlacením klávesovej skratky ALT+šípka dolu. Napríklad vlastnost “Enabled” (odblokovaný) môže byt “true” (pravda) alebo “false” (nepravda). Tieto hodnoty sú obsiahnuté v rozbalovacom zozname.
Niektoré vlastnosti môžu byt komplikovanejšie a prislúcha im dialógové okno. Existencia dialógového okna urcitej vlastnosti je indikovaná tlacidlom s troma bodkami, umiestneným na pravom okraji stlpca s hodnotou. Dialógové okno môžeme otvorit stlacením klávesovej skratky CTRL+ENTER. Napríklad vlastnost “Font” má dialógové okno pre nastavenie písma.
A nakoniec niektoré vlastnosti obsahujú skupinu ciastkových vlastností, napr. vlastnost “Font”. Existujú dve informácie, ktoré indikujú skupinu vlastností: prvou je znamienko plus (+) na lavej strane názvu vlastnosti a druhou je uzavretie hodnoty vlastnosti do hranatých zátvoriek. Stlacme kláves plus (+) v lavom stlpci s názvom vlastnosti pre otvorenie skupiny a stlacme šípku dolu pre

prehliadku skupiny. Stlacme kláves mínus (-) pre zatvorenie skupiny (aby sa zatvorila, kurzor musí byt na názve skupiny).

c) Okrem “Properties” tu je aj dalšia záložka “Events” (udalosti). Medzi “Properties” a “Events” sa môžeme prepínat stlácaním klávesovej skratky CTRL+TAB.

Udalosti spúšta používatel programu, napr. kliknutím, dvojitým kliknutím, stlacením klávesu a mnohými dalšími cinnostami alebo cinnostami okien ako otváranie a zatváranie formulára. Udalosti sú vhodne pomenované “OnClick”, “OnDblClick”, “OnKeyPress”, “OnShow”, “OnClose”. K udalosti môžeme pristupovat rovnako ako k vlastnosti. Na zaciatku sú však hodnoty všetkých udalostí prázdne. Hodnotou udalosti je názov funkcie, urcitej casti kódu programu ktorá urcí napr. tlacidlu co robit, ak nastane udalost “OnClick”.

Ak sa kurzorom nastavíme do stlpca s hodnotou urcitej udalosti a stlacíme CTRL+ENTER prepneme sa do okna “Unit1.cpp” (ak sme ho doteraz neuložili a nepomenovali). Ak bola predtým hodnota prázdna, teraz obsahuje názov funkcie a funkcia s týmto názvom je automaticky vytvorená v okne unit (aj so všetkým co potrebujeme pri definovaní funkcie). Teraz už len napíšeme náš kód programu. Jednoduché, pravda?

Ak už bola k udalosti priradená funkcia, okno unitu je automaticky aktivované a kurzor je nastavený na tejto funkcii.

3.1.4 Zoznam okien (“Window-List”)

Ak máme otvorených viacero okien (niekedy sa to stane), môžeme otvorit dalšie okno - zoznam okien (“Window-List”) v ponuke “View”, alebo stlacením ALT+0. V tomto zozname nájdeme všetky otvorené okná. Na aktiváciu jedného z nich použijeme šípky a ENTER.

3.1.5 Zoznam komponentov (“Component-List”)

Ak chceme do formulára umiestnit jeden alebo viac komponentov otvoríme zoznam komponentov (component-list) v ponuke “View”, skrátene ALT+V, C.

Zoznam obsahuje všetky dostupné typy komponentov.

Na presúvanie sa medzi komponentmi môžeme použit šípku hore a šípku dolu. Ale pozor: v závislosti na tom akú verziu Builder-a používame môže zoznam obsahovat viac ako 100 komponentov. Teda je jednoduchšie, ak vieme aspon približný názov komponentu. Väcšina názvov komponentov zacína písmenom “T”. Teda napíšeme T plus meno komponentu a tým ho aktivujeme, napr. “TBu” pre nájdenie “TButton”. Potom stlacíme ENTER a tým ho vložíme do formulára. Vždy ked stlacíme ENTER, vložíme do formulára další komponent zvoleného typu (ak vkladáte komponenty typu tbutton, sú štandardne pomenované “Button1”, “Button2”, “Button3”, atd.). Ak sme skoncili, stlacíme ESC, cím zatvoríme zoznam komponentov.

3.1.6 Okno správ (“Messages-Window")

Ak sa objaví okno správ, znamená to, že nieco s kódom programu nie je v poriadku, že obsahuje chybu. Okno správ sa zobrazí automaticky ak spustíme našu aplikáciu príkazom “Run” z ponuky “Run” alebo stlacením F9. Ak Builder nájde chybu v kóde nášho programu, pozastaví spustenie aplikácie a ukáže riadok s chybou a okno správ, v ktorom chybu opíše. Takto ju môžeme jednoducho opravit.

Pri štandardných nastaveniach je ale okno správ pre nevidiacich používatelov tažko dostupné, alebo úplne nedostupné pre tých, co nepoužívajú pocítacovú myš. Okno správ je štandardne umiestnené v spodnej casti okna unitu. Ale môžeme do neho kliknút, stlacením shift+F10, otvorit kontextovú ponuku a deaktivovat položku “Dockable” (pozn. prekladatela: ci bude okno správ súcastou okna unitu alebo bude samostatné). Potom môžeme k oknu správ pristupovat cez zoznam okien (ALT+0) alebo cez kontextovú ponuku okna unitu a položku “Message-

Window", skrátene SHIFT+F10, M.

Okno správ bude samostatné (undocked) až kým nezavrieme Builder.

V okne správ nájdeme všetky chybové správy. Nastavme sa na správu, precítajme si ju, stlacme CTRL+ENTER alebo CTRL+S a presunieme sa do okna unitu na miesto, kde sa chyba vyskytla. Po tom ako sme opravili chybu v kóde programu, presunme sa späť do okna správ a urobme to isté s dalšou chybou.

Po opravení všetkých chýb znovu spustime našu aplikáciu stlacením F9.

3.1.7 Dalšie okná

Dalšie okná budú opísané na mieste, kde ich budeme používat po prvý krát.

3.2 Ponuky, panely nástrojov a panely komponentov prostredia BCB5

Ciel tejto podkapitoly:
Získat prehlad o ponukách a paneloch nástrojov prostredia Builder. Takisto sa naucit nieco o špeciálnom paneli, o paneli komponentov ("component-bar").

3.2.1 Ponuky

Ponuky sú tu opísané len všeobecne, pretože nie je potrebné poznat každý príkaz v každej ponuke ak ich nemôžeme zlúcit s nejakou funkciou pri programovaní.

V hlavnej ponuke sa nachádza deväť ponúk.

Ponuka "File" (súbor), skrátene ALT+F:
Tak ako v iných programoch, aj tu sú príkazy New (nový), Open (otvorit) a Save (uložit). Pretože Builder pozná viac ako jeden typ súboru, je tu viac ako jeden príkaz New, Open a Save.

Ponuka "Edit" (úpravy), skrátene ALT+E:
Ponuka Edit obsahuje príkazy Cut (vystrihnút), Copy (kopírovat), Paste

(vložit), Delete (vymazat) a nejaké dalšie podobné príkazy, ale tak isto aj príkazy na zmenu velkosti a umiestnenia komponentov vo formulári.
Ponuka "Search" (hladat), skrátene ALT+S:
Tu sa nachádzajú príkazy na vyhladávanie a nahrádzanie textu a príkazy typu chod na (go to).
Ponuka "View" (zobrazit), skrátene ALT+V:
Tu môžete nájst všetky okná Builder-a.
Ponuka "Project" (projekt), skrátene ALT+P:
Táto ponuka obsahuje množstvo príkazov na správu nášho aktuálneho projektu.
Ponuka "Run" (spustit), skrátene ALT+R:
Tu sú príkazy na spúštanie a testovanie našej aplikácie a príkazy na ladenie aplikácie (tzv. debugging = hladanie chýb).
Ponuka "Component" (komponent), skrátene ALT+C:
Túto ponuku potrebujeme, ak chceme používat komponenty, ktoré Builder štandardne neobsahuje, ale ich vytvoril iný programátor, aby zjednodušil život kamarátom programátorom.
Ponuka "Tools" (nástroje), skrátene ALT+T:
Tu môžeme vo všeobecnosti menit niektoré predvolené nastavenia editora, IDE vo všeobecnosti.
Ponuka "Help" (pomocník), skrátene ALT+H:
Slovo pomocník hovorí samo za seba.

3.2.2 Panely nástrojov

Štandardne máme pät panelov nástrojov umiestnených na pravej strane hlavnej ponuky a v dvoch riadkoch pod hlavnou ponukou. Zobrazovanie alebo nezobrazovanie každej z nich je možné nastavit v kontextovej ponuke, skrátene SHIFT+F10.

Panel nástrojov "Standard" (štandardné) je prvý pod hlavnou ponukou a obsahuje tlacidlá

New (nový)

Open (otvorit - so zoznamom naposledy otváraných súborov)
Save (uložit)

Save All (uložit všetko)

Open Project (otvorit projekt)

Add File to Project (pridat súbor k projektu)

Remove from Project (odstránit z projektu)

Panel nástrojov "View" (zobrazit) je pod panelom standard a obsahuje tlacidlá

View Form (zobrazit formulár)

View Unit (zobrazit unit)

Toggle Form/Unit (prepínat medzi formulár/unit)

New Form (nový formulár)

Panel nástrojov "Debug" (ladit) je na lavo od panelu view a pozostáva z tlacidiel

Run (spustit - so zoznamom naposledy spúštaných programov)

Pause (pozastavit)

Trace Into (pozn. prekladatela: sledovat najbližšiu inštrukciu)

Step Over (pozn. prekladatela: sledovat najbližšiu inštrukciu funkcie)

Panel nástrojov "Custom" (s volitelným obsahom) je nalavo od panelu standard a obsahuje len jedno tlacidlo Help Contents (obsah pomocníka).

Panel nástrojov "Desktop" (plocha) je nalavo od hlavnej ponuky a obsahuje zoznam plôch. Predvolená hodnota je "None" (žiadne).

Tlacidlo "Save Desktop" (ulož plochu)

Tlacidlo "Set Debug Desktop"

3.2.3 Panel komponentov

Panel komponentov, alebo paleta komponentov, je šiesty panel nástrojov. Obsahuje všetky komponenty, ktoré môžeme použit v projekte alebo vo formulári. Pretože je komponentov velmi vela, je tento panel zložený z registrov alebo záložiek. Prvá záložka má názov "Standard" (štandardné). Každá záložka obsahuje urcité komponenty.

Tento panel je umiestnený pod hlavnou ponukou, napravo od všetkých ostatných panelov a jeho výška je rovná výške dvoch štandardných

panelov nástrojov.

Tento panel nástrojov nemôže byt aktivovaný klávesovou skratkou, musíme na neho kliknút.

Ak sme ho aktivovali, môžeme použit kontextovú ponuku, SHIFT+F10 na výber požadovanej záložky.

Ked sme ale zvolili požadovanú záložku musíme použit kurzor myši, aby sme mohli vybrat komponent.

Na vkladanie komponentov do formulára je vhodnejšie používat zoznam komponentov než panel komponentov.

Kapitola 4: Vytváranie jednoduchých aplikácií pre Windows

Ciel tejto kapitoly:
Naucit sa ako vytvorit jednoduchý program v C++ v prostredí Borland Builder, zoznámit sa a mat prehlad o dôležitých a nevyhnutných komponentoch prostredia Borland Builder.

4.1 Vytváranie jednoduchých aplikácií pre Windows

4.1.1 Navrhovanie formulárov pre Windows (Windows Forms)
4.1.2 Inšpektor objektov (Object Inspector): vlastnosti (properties) a udalosti (events)
4.1.3 Vlastnosti a udalosti objektu (Object)
4.1.4 Inšpektor objektov OI
4.1.5 Názvy vlastností a udalostí
4.1.6 Presun do Inšpektora objektov
4.1.7 Predchádzanie chýb pocas pohybovania sa v OI
4.1.8 Aktualizácia grafických vlastností objektu (graphic characteristic)
4.1.9 Definovanie obsluhy udalosti (event handler)

4.2 Navrhovanie formulára (Form): vlastnosti a udalosti

4.2.1 Základné vlastnosti (properties) formulára
4.2.2 Základné udalosti (events) formulára

4.3 Pridávanie komponentov do formulára

4.3.1 Základné grafické komponenty. Cvicenie
4.3.2 Pridávanie komponentu do formulára. Cvicenie
4.3.3 Umiestnovanie komponentov
4.3.4 Tabulátorové poradie (Tab order)

4.4. Základné komponenty formulára: pridávanie vlastností a udalostí

4.4.1 Edit (Editacné políčko)
4.4.2 Memo (Viacriadkové editacné políčko)
4.4.3 Button (Tlacidlo)
4.4.4 Label (Menovka)
4.4.5 ListBox (Zoznam)
4.4.6 ComboBox (Kombinovaný rámik)
4.4.7 CheckBox (Zaciarkávacie políčko)

4.1 Vytváranie jednoduchých aplikácií pre Windows

Ciel tejto podkapitoly: Naucit sa vytvorit jednoduchú aplikáciu pre Windows.

V tejto casti si predstavíme aplikáciu (application), jej formulár (Form), jej komponenty (Components) spolu s ich vlastnostami (properties) a udalostami (events).

Všeobecne: Windows aplikácia je program spustitelný v niektorom operacnom systéme od spolocnosti Microsoft: Windows 95/98, Windows ME, Windows 2000.
Windows aplikácia je vo všeobecnosti vytvorená skupinou grafických komponentov (okien, editacných polícok, tlacidiel, panelov ponúk, atd.), ktoré umožnujú používatelovi vykonávat zadanú úlohu (napr. na editovanie jednoduchého textu použijeme aplikáciu Poznámkový blok (Notepad)).

Skratky:

IO = okno Object Inspector.
BCB = Borland C++ 5 Builder.
Up = šípka hore.
Down = šípka dolu.
Right = šípka vpravo.
Left = šípka vlavo.

Navrhovaná dohoda o pomenovaní BCB komponentov:

btn_	Button (tlacidlo)
chk_	Check box (zaciarkávacie políčko)
cmb_	Combo box (kombinovaný rámik)
edi_	Edit box (editacné políčko)
frm_	form (formulár)
lbl_	Label (menovka)
lst_	List box (zoznamový rámik)
mem_	Memo (viacriadkové editacné pole)
rad_	Radio button (prepínac)
rdg_	Radio group (skupina prepínacov)

4.1.1 Navrhovanie formulárov pre Windows (Windows Forms)

Ciel tejto podkapitoly: Naucit sa vytvorit formulár

V tejto casti si ukážeme ako navrhnút grafickú štruktúru formulára.

Prvým krokom na to, aby sme vytvorili windows aplikáciu, je vytvorenie jej hlavného formulára. Formuláre sú ekvivalentom operacného systému Windows.

Windows aplikácia pozostáva zaspon jedného formulára. Ked otvoríme BCB5, predvoleným aktívnym prvkom je prázdny formulár. Prázdny formulár môžeme prirovnat k základom budovy, ktorú chceme postavit. Najskôr si musíme stanovit úcel aplikácie, ktorú chceme vytvorit; potom navrhneme grafickú štruktúru formulára, tak aby súvisela so zvoleným úcelom. Úlohy, ktoré musíme splnit, sú:
a) Nastavenie vlastností formulára (velkost, farba, atd.)
b) Umiestnenie všetkých potrebných komponentov do formulára

c) Nastavenie vlastností komponentov tak, aby boli zosúladené a aby fungovali.

4.1.2 Inšpektor objektov (Object Inspector): vlastnosti (properties) a udalosti (events)

Ciel tejto podkapitoly: Ozrejmit si pojem vlastnost (Property) a udalost (Event) a spôsob používania Inšpektora objektov (Object Inspector)

V tejto casti si ukážeme ako používat Object Inspector na menenie vlastností a udalostí

Formulár, rovnako ako komponenty, má vlastnosti a udalosti. Na to, aby sme si mohli definovat vlastnosti a udalosti objektu, musíme sa zoznámit s modulom nazývaným Object Inspector (Inšpektor objektov).

4.1.3 Vlastnosti a udalosti objektu

Vlastnosti (properties) sú hodnoty objektu: napríklad meno, šírka, výška, farba a pod.
Udalosti (events) sú používatelove akcie, ktoré vykonáva na urcitom objekte, ako napr. Stlacenie klávesu, alebo kliknutie myši. Windows aplikácia je ovládaná udalostami, ktoré sú vyvolané používatelovými akciami.
Práve písanie funkcií, ktoré spracujú udalosti je hlavnou úlohou programátora vo Windows.
Táto funkcia sa stáva aktívnou vtedy, ked nastane udalost (napríklad funkcia, ktorá spracúva udalost **OnClick** sa stane aktívnou vtedy, ked nastane udalost **Click**.).

4.1.4 Inšpektor objektov (Object Inspector)

Všeobecne: Inšpektor objektov je okno IDE, ktoré nám umožní nastavit vlastnosti a umožní nám prístup k správcovi udalostí formulára a jeho komponentov.

Okno Inšpektora objektov pozostáva z troch castí:
a) Kombinovaného rámika, ktorý nám umožnuje vybrat si komponent,

ktorý chceme zmenit; kombinovaný rámik sa nachádza hned pod titulným riadkom **Object Inspector**.

b) Zoznamu **Properties**, ktorý sa týka oznaceného komponentu.
c) Zoznamu **Events**, týkajúceho sa oznaceného komponentu.

Názvy záložiek pre zoznamy **Properties a Events** sú jedinou informáciou použitelnou pri práci s hmatovým displejom.
Oba zoznamy sú zoradené do dvoch stlpcov; nalavo môžeme nájst abecedne usporiadané vlastnosti alebo udalosti, ktoré sa vztahujú na vyznacený komponent, napravo si môžeme vyznacit alebo zadat hodnotu vlastnosti alebo meno udalosti.

4.1.5 Názvy vlastností a udalostí

Názvy vlastností a udalostí v lavom stlpci sú slová bežne používané v programovacom jazyku; sú bez medzier alebo iných oddelovacov, prvé písmeno každého slova je velké, napríklad **TabOrder** alebo **OnKeyPress**. Názvy udalostí sú vyznacené predponou "**On**".

4.1.6 Presun do okna Object Inspector

Na pohyb v OI sa používajú nasledujúce príkazy:
(F11) na vyvolanie okna Object Inspector.
(Ctrl+Down): (**posun smerom hore na hmatovom displeji**) na dosiahnutie kombinovaného rámiku s komponentmi.
(Up alebo Down) na listovanie v zozname komponentov vo formulári, vrátane formulára.
(Enter) na vyznacenie komponentu.
(Ctrl+Tab): na prepínanie medzi zoznamom komponentov a zoznamom udalostí.
(Up) alebo (Down): na listovanie vo vlastnostiach a udalostiach, niektoré vlastnosti majú zoznamy ciastkových vlastností, ktoré môžeme vyvolat pomocou klávesu **Tab** na odznacenie textu vlastnosti v pravom stlpci; (Left) na zatvorenie zoznamu, (Right) na jeho otvorenie.
(Alt+Down): na otvorenie zoznamu parametrov vlastností (pozn. prekladatela: nefungovalo).
(Enter) na potvrdenie alebo (Ctrl+Enter) na presun k parametru, ktorý si prajeme oznacit.

4.1.7 Predchádzanie chýb pocas pohybovania sa v OI

Zvycajne, ked máme oznacené vlastnosti alebo udalosti, je zobrazený obsah pravého stlpca (ktorý obsahuje hodnotu vlastnosti alebo názov udalosti); ak vtomto momente stlacíme akýkolvek kláves na klávesnici, zmeníme zobrazenú informáciu.
Aby sme predišli tomuto problému budeme postupovat takto:
A) Stlacíme **Tab**, aby sme predišli vybratiu hodnôt z pravého stlpca.
B) Listujeme v zozname vlastností a udalostí až kým nedosiahneme vlastnost alebo udalost, ktorú hladáme.
C) Znova stlacíme Tab, aby sme videli pravý stlpec.
D) Zmeníme hodnotu.

4.1.8 Aktualizácia grafických vlastností objektu (graphic characteristic)

Hodnoty vlastností, najmä grafických, sa aktualizujú pocas úprav alebo stlacením klávesu Enter a pocas prechodu k inej vlastnosti. Napríklad, názov, ktorý je zobrazený v titulnom riadku formulára sa bude aktualizovat pocas jeho zadávania. Ak sa raz hodnota vlastnosti width (šírka) formulára zmení, a ak potom stlacíme Enter, alebo ak sa posunieme na inú vlastnost, šírka sa ihned prestaví.

4.1.9 Definovanie obsluhy udalosti (event handler)

V pravom stlpci zoznamu udalostí bude možné prispôsobit si názov oznacenej udalosti.
(Ctrl+Enter) slúži na otvorenie textového okna a na prácu s vyznacenou udalostou. Pôvodný názov udalosti sa objaví v pravom stlpci, v poli vedla vyznacenej udalosti; napravo, nad formulárom, sa objaví okno na zadávanie textu; vo vnútri tohoto okna môžeme vpísat kód týkajúci sa spracovania oznacenej udalosti.
(F11) na prepínanie medzi formulárom a Inšpektorom objektov a/alebo oknom na zadávanie textu.

4.2 Navrhovanie formulára (Form): vlastnosti a udalosti

Ciel tejto podkapitoly: Naucit sa navrhovat formulár (Form).

V tejto casti si ukážeme, ako pridat nový formulár do aplikácie, na ktorej

práve pracujeme.

Windows aplikácia môže pozostávať z jedného alebo z viacerých formulárov; v BCB5 nájdeme prázdny, predvolený formulár.
Na to, aby sme mohli pridat nový formulár do aplikácie, na ktorej práve pracujeme, musíme otvorit ponuku **File** a vybrat položku **New Form**.

4.2.1 Základné vlastnosti formulára

Vlastnost Name: umožnuje nastavit meno formulára vrámci aplikácie. Toto meno pomáha identifikovat komponent vo vnútri programu.
Vlastnost Caption: umožnuje nastavit nadpis formulára tak, ako sa má zobrazit v titulnom riadku.
Vlastnost Width: umožnuje nastavit šírku formulára v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku formulára v obrazových bodoch (pixel).
Vlastnost Colour: umožnuje nastavit farbu podkladu formulára.
Samozrejme, že formulár obsahuje aj množstvo iných vlastností; na získanie ich opisu vyznacte vlastnost a stlacte F1.

4.2.2 Základné udalosti formulára

OnShow: Urcuje co sa stane, pocas otvárania aplikácie a zobrazovania formulára.
OnActivate: Urcuje co sa stane, ked formulár znovu aktivujeme, napríklad stlacením (Alt+Tab).
OnClick: Urcuje co sa stane, ked klikneme myšou na formulár.
OnKeyPress: Urcuje co sa stane, ked stlacíme kláves na klávesnici.
OnClose: Urcuje co sa stane, ked zatvoríme formulár.

4.3 Pridávanie komponentov do formulára

Ciel tejto podkapitoly: Naucit sa ako pridat komponenty do formulára

V tejto casti si predvedieme základné grafické komponenty formulára.

Okrem formulárov je aplikácia vytvorená aj z iných grafických

komponentov, ktoré nám umožnujú s aplikáciou pracovat .

4.3.1 Základné komponenty formulára (pridávanie vlastností a udalostí)

TEdit: jednoriadkové editacné políčko, ktoré sa používa na pridávanie alebo menenie textu.
TMemo: viacriadkové editacné políčko, ktoré sa používa na pridávanie alebo menenie textu.
TButton: s tlacidlo vyvoláva spustenie jednej alebo viacerých akcií.
TLabel: textová menovka, ktorá je urcená len na cítanie.
TListBox: zoznamový rámik umožnuje výber položky zo zoznamu viacerých položiek
TCombBox: kombinovaný rámik je kombinácia zoznamu zobrazených položiek a editacného políčka. Položku je v zozname možné vyznacit stlacením jej zaciatocného písmena. Ak vyznacenú položku zoznamu potvrdíme stlacením (Enter) umiestni sa do editacného políčka.
TCheckBox: zaciarkávacie políčko môžeme použit na vyznacenie jednej alebo viacerých možností z ponúkaných alternatív.
TRadioButton: prepínace používame na výber medzi viacerými možnostami. Každá možnost automaticky ruší predchádzajúcu.

Teraz, po tomto dlhom teoretickom úvode si vyskúšame praktické cvicenie.

Cvicenie: Navrhnite formulár na zozbieranie osobných údajov študentov vašej školy.

Na zostavenie takéhoto formulára potrebujete:
editacné políčko (Edit) na krstné meno,
editacné políčko (Edit) na priezvisko,
dva prepínace (RadioButton) na urcenie pohlavia,
kombinovaný rámik (ComboBox) na zadanie veku,
zaciarkávacie políčko (checkBox) na zadanie postihnutia,
editacné políčko (Edit) na zadanie adresy,
zoznam (listBox) na miesto trvalého bydliska,
viacriadkové políčko (Memo) na poznámky,
tlacidlo (Button) na potvrdenie zadaných údajov.

Pred každým zo štyroch editacných políčok (tri jedno- a jedno viacriadkové) bude menovka, ktorá bude popisovat ich význam.

4.3.2 Pridávanie komponentu do formulára

Na to aby sme videli zoznam všetkých predvolených základných komponentov musíme ísť do ponuky **View** a vybrat **Component list** (zoznam komponentov); pre tento zoznam neexistuje skratka.
Zobrazí sa nám okno **Components**; toto okno pozostáva zeditacného políčka a zoznamu (sú síce od seba oddelené, ale pracujú ako kombinovaný rámik) a z tlacidla **Add to form**; medzi všetkými komponentmi sa dá pohybovat pomocou klávesu **Tab**.

Po otvorení okna sa objaví komponent, ktorý bol pred zatvorením vyznacený ako posledný; jeho názov je zobrazený a zvýraznený v editacnom políčku.
Pri všetkých komponentoch zacínajúcich písmenom "T" stací zadat "T" a za ním prvé písmeno názvu komponentu; na obrazovke sa oznací prvý možný komponent vabecednom poradí a celý jeho názov sa zobrazí a zvýrazní veditacnom políčku; (jediný komponent, ktorý nezacína na "T" je "Frame", v tomto prípade zadáme "F").
Stlacíme Up alebo Down na pohyb v zozname komponentov.
Stlacíme Enter na pridanie vyznaceného komponentu do formulára.
Stlacíme Esc na opustenie okna.

Aby sme pridali komponenty do nášho formulára (na zozbieranie osobných údajov), musíme:
1. Presunút sa do panelu ponúk (Alt).
2. Presunút sa do ponuky **View** (Right).
3. Presunút sa na položku **Component List** (Down), Enter.
4. Ked sa okno Component List otvorí, presunút sa na **TButton** (Down), Enter.

Pozor: Po stlacení Enter stále uvidíme okno Component; na tejto úrovni však nie je dôležité overovat, ci sa komponenty pridali do formulára správne.

6. Presunút sa na **TcheckBox** (Down), Enter.
7. Presunút sa na **TcomboBox** (Down), Enter.
8. Presunút sa na **Tedit** (Down), trikrát Enter.
9. Presunút sa na **Tlabel** (Down), štyrikrát Enter.
10. Presunút sa na **TlistBox** (Down), Enter.
11. Presunút sa na **Tmemo** (Down), Enter.
12. Presunút sa na **TradioButton** (Down), dvakrát Enter.
13. Esc na zatvorenie okna **Component List**.

Teraz by sme v zozname komponentov OI mali mat 15 položiek.
1. Presunme sa do OI (F11).
2. (Ctrl+Down) na otvorenie kombinovaného rámika komponentov.
3. (Up) alebo (Down) na listovanie v zozname.

4.3.3 Umiestnovanie komponentov

Po pridaní komponentu sa tento automaticky umiestni doprostred formulára; ak pridávame naraz viac komponentov, budú sa prekrývat. Preto bude nevyhnutné nastavit vlastnosti každého komponentu, a tak ich správne umiestnit.

4.3.4 Tabulátorové poradie (Tab order)

Formulár môže obsahovat viac rôznych komponentov; na optimalizovanie pohybu medzi komponentami vdialógovom okne je nevyhnutné nastavit vlastnost **Tab Order**. Táto vlastnost priraduje komponentom poradie, v akom sa po nich budeme pohybovat pomocou tabulátora; napríklad, v okne na zbieranie adries by bolo nelogické, keby sme sa po stlacení **Tab** presunuli z Meno na Mesto, bez toho aby sme prešli cez Adresa.

4.4 Základné komponenty formulára: pridávanie vlastností a udalostí

Ciel tejto kapitoly: Naucit sa ako usporiadat komponenty vo formulári

V tejto casti sa dozvieme o vlastnostiach komponentov a o najbežnejších udalostiach formulára.

4.4.1 Edit (editacné políčko)

Na to, aby sme do formulára pridali komponent **Edit**, musíme oznacit **TEdit** v okne **Components**, stlacit **Enter**, a **Esc** na zatvorenie okna .
Potom umiestnime doprostred formulára jednoriadkové editacné políčko nazvané **Edit#**; názov komponentu sa vpoli objaví automaticky ako text. Komponent vyzerá ako dlhý, úzky obdlžnik, pozadie a text majú farbu

predvolenú vo Windows; kontúra komponentu vyzerá ako "vyrytá".

Základné vlastnosti objektu Edit

Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie.
Vlastnost Text: umožnuje nastavit obsah textových komponentov.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).
Vlastnost TabOrder: umožnuje nastavit tabulátorové poradie.

Základné udalosti objektu Edit

Udalost OnChange: urcuje správanie sa aplikácie ked sa text v editacnom polícku zmení.
Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnDblClick: urcuje správanie sa aplikácie, ked myšou vykonáme dvojklik na komponente.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je stlacený kláves.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked uvolníme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane

aktívnym (receive the focus).
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.2 Memo (Viacriadkové editacné políčko)

Na pridanie objektu **Memo** do formulára musíme vokne **Components** vybrat **Tmemo**, stlacit **Enter** a napokon stlacit **Esc** na zatvorenie okna. Viacriadkové textové pole, ktoré sa nazýva **Memo#**, sa zobrazí v strede formulára; názov komponentu sa v poli ukáže ako text. Komponent vyzerá ako obdlžnik, farba pozadia a písma je použitá taká, aká je pôvodne vo Windows; komponent má trojrozmerný vzhlad.

Základné vlastnosti objektu Memo

Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie.
Vlastnost Text: používa sa priradenia textového obsahu komponentu.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).
Vlastnost TabOrder: umožnuje nastavit tabulátorové poradie.
ScrollBars: umožnuje nastavit viditelnost vodorovného a zvislého rolovacieho pásu.

Základné udalosti objektu Memo

Udalost OnChange: urcuje správanie sa aplikácie, ked sa zmení obsah textu.
Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.

Udalost OnDblClick: urcuje správanie sa aplikácie, ked myšou vykonáme dvojklik na komponent.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym (receive the focus).
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.3 Button (Tlacidlo)

Na to, aby sme do formulára pridali komponent **Button**, musíme oznacit **TButton** v okne **Components**, stlacit **Enter**, a **Esc** na zatvorenie okna.
Uprostred formulára sa objaví tlacidlo nazvané **Button#**; názov komponentu sa zobrazí ako menovka (label). Komponent vyzerá ako dlhý, úzky obdlžnik, pozadie a text majú farbu predvolenú vo Windows; kontúra komponentu vyzerá ako vyrytá.

Základné vlastnosti objektu Button (Button object)

Vlastnost Caption: umožnuje nastavit text, ktorý opisuje konkrétny ovládací prvok.
Vlastnost Default: ak je táto vlastnost nastavená na **True** (pravda) tlacidlo je prepojené na kláves **Enter**. Stlacením Enter z ktorejkolvek polohy vo formulári bude možné vyvolat kód **obsluhy udalosti** daného komponentu (pozn. prekladatela: okrem tých komponentov, ktoré sú samé aktivované klávesom Enter).
Ak je táto vlastnost nastavená na **True**, okraj tlacidla bude cierny.
Vlastnost Cancel: Ak túto vlastnost nastavíme na **True**, tlacidlo bude prepojené s tlacidlom **Esc**. Stlacením Enter z ktorejkolvek polohy vo formulári bude možné vyvolat kód **obsluhy udalosti** daného komponentu.
Vlastnost Name: reprezentuje identifikacné meno komponentu vrámci

aplikácie.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).

Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).
Vlastnost TabOrder: umožnuje nastavit tabulátorové poradie.

Základné udalosti objektu Button (Button object)

Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym (receive the focus).
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.4 Objekt Label (Label object)

Na pridanie objektu **Label** do formulára musíme vokne **Components** vybrat **Tlabel**, stlacit **Enter** a napokon stlacit **Esc** na zatvorenie okna.
Menovka s názvom **Label#** sa zobrazí uprostred formulára; názov

komponentu sa objaví na menovke. Komponent vyzerá ako textová menovka, pozadie je priehladné a farba textu je taká, ako je predvolená vo Windows; okraj komponentu je viditelný len ak je oznacený pocas navrhovania.

Základné vlastnosti a udalosti objektu Label (Label object)

Vlastnost Caption: nadpis umožnuje nastavit text, ktorý opisuje konkrétny ovládací prvok.
Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).

Základné udalosti objektu Label (Label object)

Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.

4.4.5 Objekt ListBox (ListBox object)

Na to, aby sme do formulára pridali komponent **ListBox**, musíme oznacit **TListBox** v okne **Components**, stlacit **Enter**, a potom **Esc** na zatvorenie okna.

Zoznam pomenovaný **ListBox#** sa umiestni doprostred formulára. Komponent vyzerá ako obdlžnik, farba pozadia a písma je použitá taká, aká je predvolená vo Windows; obrys komponentu má trojrozmerný vzhlad.

Základné vlastnosti objektu ListBox (ListBox object)

Vlastnost Items: umožnuje nám do zoznamu zadat položky. Stlacením **Ctr+Enter** sa otvorí **StringList Editor**, v nom je možné zadat položky, každú na samostatný riadok; na to aby boli správne zobrazené, doporucujeme použit co najkratší názov položky zoznamu.
Vlastnost IntegralHeight: umožnuje nastavit výšku v obrazových bodoch (pixel) tak, aby sme dostali požadovaný pocet riadkov.
Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie.
Vlastnost Text: umožnuje nastavit obsah textových komponentov.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).
Vlastnost TabOrder: umožnuje nastavit tabulátorové poradie.

Základné udalosti objektu ListBox (ListBox object)

Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnDblClick: urcuje správanie sa aplikácie, ked myšou vykonáme dvojklik na komponent.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked tlacidlo myši je stlacené.

Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym.
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.6 Objekt ComboBox (ComboBox object)

Na pridanie komponentu **ComboBox** do formulára musíme v okne **Components** vybrat **TComboBox**, stlacit **Enter** a napokon stlacit **Esc** na zatvorenie okna.
Zoznam/editacné polícko nazývané ComboBox sa zobrazí v strede formulára; názov komponentu sa objaví veditacnom riadku okna. Je to zložený komponent, vyzerá ako jednoriadkové editacné polícko (viditelný je len editacný riadok, nie zoznam); na pravom okraji polícka je malé trojrozmerné tlacidlo, na ktorom je malá šípka smerujúca nadol.

Základné vlastnosti objektu ComboBox (ComboBox object)

Vlastnost Items: umožnuje nám do zoznamu zadat položky. Stlacením **Ctr+Enter** sa otvorí **StringList Editor**, v nom je možné zadat položky, každú na samostatný riadok; na to aby boli správne zobrazené, doporucujeme použit co najkratší názov položky zoznamu.
Vlastnost Name: reprezentuje identifikacné meno komponentu vrámci aplikácie.
Vlastnost Text: umožnuje nastavit obsah textových komponentov.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na

komponent presunieme kurzor myši (tzv. bublinkový popis).
Vlastnost TabOrder: umožnuje nastavit tabulátorové poradie.

Základné udalosti objektu ComboBox (ComboBox object)

Udalost OnChange: urcuje správanie sa aplikácie, ked sa zmení obsah textu.
Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnDblClick: urcuje správanie sa aplikácie, ked myšou vykonáme dvojklik na komponent.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym.
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.7 Objekt CheckBox: pridávanie vlastností a udalostí

Na to, aby sme do formulára pridali komponent **CheckBox**, musíme oznacit **TCheckBox** v okne **Components**, stlacit **Enter**, a potom **Esc** na zatvorenie okna.
Zaciarkávacie políčko nazvané **CheckBox#** bude umiestnené do stredu formulára; názov komponentu sa objaví na menovke pola. Je to zložený komponent, vyzerá ako štvorcek s grafickou charakteristikou jednoriadkového textového pola; môže byt zaciarknutý alebo nezaciarknutý; hned napravo je vždy menovka.

Základné vlastnosti objektu CheckBox (CheckBox object)

Vlastnost Checked: urcuje stav pola: zaciarknuté alebo nezaciarknuté.

Vlastnost Caption: urcuje text nadpisu pola.
Vlastnost Alignment: urcuje polohu textu napravo alebo nalavo od zaciarkávacieho políčka.
Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie (pozn. prekladatela: oproti originálu zrušená vlastnost text, pretože pre tento komponent neexistuje).
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).

Základné udalosti objektu CheckBox (CheckBox object)

Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.
Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym.
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

4.4.8 Objekt RadioButton (RadioButton object)

Na to, aby sme do formulára pridali komponent **RadioButton**, musíme v okne **Components** oznacit **TRadioButton**, stlacit **Enter**, a **Esc** na zatvorenie okna. Tlacidlo volby nazvané **RadioButton#** sa zobrazí doprostred formulára, názov komponentu sa zobrazí ako menovka tlacidla. Je to zložený komponent, vyzerá ako krúžok; na pravej strane je vždy menovka.

Základné vlastnosti objektu RadioButton (RadioButton object)

Vlastnost Caption: urcuje text nadpisu krúžku prepínaca.
Vlastnost Checked: Pravda/Nepravda (True/False) ci je vyznacený alebo nie.
Vlastnost Alignment: urcuje polohu textu vedla krúžku (napravo alebo nalavo).
Vlastnost Name: reprezentuje identifikacné meno komponentu v rámci aplikácie.
Vlastnost Width: umožnuje nastavit šírku komponentu v obrazových bodoch (pixel).
Vlastnost Height: umožnuje nastavit výšku komponentu v obrazových bodoch (pixel).
Vlastnost Top: umožnuje nastavit vzdialenost od horného okraja formulára v obrazových bodoch (pixel).
Vlastnost Left: umožnuje nastavit vzdialenost od lavého okraja formulára v obrazových bodoch (pixel).
Vlastnost Color: umožnuje nastavit farbu pozadia komponentu.
Vlastnost Font: (s podponukou): umožnuje nastavit vlastnosti použitého písma.
Vlastnost Hint: umožnuje vložit textový riadok, ktorý sa zobrazí, ak na komponent presunieme kurzor myši (tzv. bublinkový popis).

Základné udalosti objektu RadioButton (RadioButton object)

Udalost OnClick: urcuje správanie sa aplikácie, ked myšou klikneme na komponent.
Udalost OnDblClick: urcuje správanie sa aplikácie, ked myšou vykonáme dvojklik na komponent.

Udalost OnKeyDown: urcuje správanie sa aplikácie, ked je kláves stlacený.
Udalost OnKeyUp: urcuje správanie sa aplikácie, ked uvolníme kláves.

Udalost OnKeyPress: urcuje správanie sa aplikácie, ked na klávesnici stlacíme nejaký znak.
Udalost OnMouseDown: urcuje správanie sa aplikácie, ked je tlacidlo myši stlacené.
Udalost OnMouseUp: urcuje správanie sa aplikácie, ked pustíme tlacidlo myši.
Udalost OnMouseMove: urcuje správanie sa aplikácie, ked je kurzor myši umiestnený na komponente.
Udalost OnEnter: urcuje správanie sa aplikácie, ked sa komponent stane aktívnym (receive the focus).
Udalost OnExit: urcuje správanie sa aplikácie, ked komponent prestane byt aktívnym.

Cvicenie.
Dalším krokom v našom cvicení je usporiadat pridané komponenty vo formulári tak, aby boli usporiadané správne a logicky. Nastavíme vlastnosti každého komponentu vrátane vlastností formulára.

A. Zacneme od formulára (**Form**).
1. Presunieme sa do okna **Object Inspector** (F11).
2. Presunieme sa do výberového rámika s komponentmi (Ctrl+Down).
3. Presunieme sa na **Form 1** (Up) alebo (Down).
4. Enter.
5. Presunieme sa na **Caption** (Up) alebo (Down) a zadáme "Údaje o študentoch".
6. Presunieme sa na **Height** (Down) a zadáme "600".
7. Presunieme sa na **Name** (Down) a zadáme "údaje".
8. Presunieme sa na **ShowHint** (Down), otvoríme výberové pole (Alt+Down) a vyberieme **True** (Up) alebo (Down) alebo stlácame (Ctrl+Down) kým ho nedosiahneme.
9. Presunieme sa na Width (Down) a zadáme "800".
10. Presunieme sa na **WindowsState** (Down) otvoríme výberové pole (Alt+Down) a vyberieme **wsMaximized** (Up) alebo (Down) alebo stlácame (Ctrl+Down) kým ho nedosiahneme.

Teraz musíme nastavit vlastnosti prvého komponentu, menovky (LabelName), ktorá bude pred editacným políckom "meno študenta"; vo formulári sa zobrazí vlavo hore. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v OI, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme

pracovat (Ctrl+Down).
3. Presunieme sa na **Label1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na Font (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Meno".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "10".
12. Presunieme sa na **Name** (Down) a zadáme "LabelName".
13. Presunieme sa na **TabOrder** (Down) a zadáme "".
14. Presunieme sa na **Top** (Down) a zadáme "10".
15. Presunieme sa na **Width** (Down) a skontrolujeme, ci je nastavená na hodnotu 60.

B. Dalšou úlohou je nastavit vlastnosti dalšieho komponentu, editacného políčka "meno študenta" (LabelName); bude sa nachádzat napravo od prvého komponentu (menovky).

1. Skontrolujeme, ci sa ešte stále nachádzame v OI, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Edit1** (Up) alebo (Down).
4. (Enter)
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter)
7. Vyberieme písmo Arial 14.
8. (Enter)
9. Presunieme sa na **Hint** (Down) a zadáme "".
10. Presunieme sa na **Left** (Down) a zadáme "70".
11. Presunieme sa na **Name** (Down) a zadáme "EditName".
12. Presunieme sa na **TabOrder** (Down) a zadáme "0".
13. Presunieme sa na **Text** (Down) a stlacíme **Del**, aby sme vymazali text, ktorý tu je predvolený.
14. Presunieme sa na **Top** (Down) a zadáme "10".
15. Presunieme sa na **Width** (Down) a zadáme "250".

Teraz musíme nastavit vlastnosti tretiemu komponentu, menovke LabelSurname, ktorá bude pred editacným políckom "priezvisko študenta"; zobrazí sa vpravo a trocha dalej od predchádzajúceho editacného políčka,

ako keby tvorila novú skupinu komponentov. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v OI, ak nie, presunieme sa tam funkcným klávesom (F11).
2. Vo výberovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Label2** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Priezvisko".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "410".
12. Presunieme sa na **Name** (Down), a zadáme "LabelSurname".
13. Presunieme sa na **Top** (Down) a zadáme "10".
14. Presunieme sa na **Width** (Down) a skontrolujeme, ci je nastavená na hodnotu 76.

Teraz musíme nastavit vlastnosti štvrtého komponentu, editacného políčka "priezvisko študenta" (EditSurname); bude sa nachádzat napravo od svojej menovky. Postupujeme takto:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Edit2** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Hint** (Down) a zadáme "".
10. Presunieme sa na **Left** (Down) a zadáme "496".
11. Presunieme sa na **Name** (Down) a zadáme "EditSurname".
12. Presunieme sa na **TabOrder** (Down) a zadáme "1".
13. Presunieme sa na **Text** (Down) a stlacíme Del, aby sme vymazali text, ktorý tu je predvolený.
14. Presunieme sa na **Top** (Down) a zadáme "10".

15. Presunieme sa na **Width** (Down) a zadáme "250".

Teraz si nastavíme vlastnosti piateho komponentu, prvého z dvoch prepínacov (RadioButtonMale), ktoré použijeme na zaznacenie pohlavia, zobrazí sa pod menovkou mena študenta, trochu nižšie, tak akoby mal formovat novú skupinu komponentov.
Budeme postupovat nasledovne:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **RadioButton1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na Font (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Muž".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "10".
12. Presunieme sa na **Name** (Down), a zadáme "RadioButtonMale".
13. Presunieme sa na **TabOrder** (Down) a zadáme "2".
14. Presunieme sa na **Top** (Down) a zadáme "70".
15. Presunieme sa na **Width** (Down) a zadáme "65".

Teraz si nastavíme vlastnosti šiesteho komponentu, druhého z prepínacov (RadioButtonFemale), ktorým oznacíme pohlavie, zobrazí sa napravo od predchádzajúceho. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **RadioButton2** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Žena".
10. Presunieme sa na **Hint** (Down) a zadáme "".

11. Presunieme sa na **Left** (Down) a zadáme "85".
12. Presunieme sa na **Name** (Down), a zadáme "RadioButtonFemale".
13. Presunieme sa na **TabOrder** (Down) a zadáme "3".
14. Presunieme sa na **Top** (Down) a zadáme "70".
15. Presunieme sa na **Width** (Down) a zadáme "90".

Teraz si nastavíme vlastnosti siedmeho komponentu, kombinovaného rámika pre vek (ComboBoxAge); zobrazí sa napravo od predchádzajúceho prepínaca, trocha dalej, nepatrí ani do jednej skupiny komponentov a od ostatných je graficky izolovaný. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **ComboBox1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Vek".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Items** (Up) or (Down).
12. (Ctrl+Enter).
13. Ked sa otvorí okno **StringListEditor** zadáme "0 - 10".
14. (Enter).
15. Zadáme "20 - 30".
16. (Enter).
17. Tabulátorom **Tab** sa presunieme na **OK** a potvrdíme klávesom Enter, tým sa zároven zavrie toto okno. Zatial sme vytvorili tri položky, ktoré budeme môct vybrat neskôr.
18. Presunieme sa na **Left** (Down) a zadáme "205".
19. Presunieme sa na **Name** (Down), a zadáme "ComboBoxAge".
20. Presunieme sa na **Text** (Down), a zadáme "Oznacte vek...".
21. Presunieme sa na **TabOrder** (Down) a zadáme "4".
22. Presunieme sa na **Top** (Down) a zadáme "70".
23. Presunieme sa na **Width** (Down) a zadáme "150".

Teraz si nastavíme vlastnosti ôsmeho komponentu, zaciarkávacieho políčka (CheckBoxHandicap) na oznacenie postihnutia (ak nejaké pripadá do úvahy); zobrazí sa napravo od predchádzajúceho kombinovaného

rámika, trocha dalej; nepatrí ani do jednej skupiny komponentov a od ostatných je graficky izolovaný. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **CheckBox1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Postihnutie".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "385".
12. Presunieme sa na **Name** (Down), a zadáme "CheckBoxHandicap".
13. Presunieme sa na **TabOrder** (Down) a zadáme "5".
14. Presunieme sa na **Top** (Down) a zadáme "70".
15. Presunieme sa na **Width** (Down) a zadáme "105".

Teraz musíme nastavit vlastnosti deviatemu komponentu, menovke (LabelAddress), ktorá bude pred editacným políckom adresy; zobrazí sa pod prvým prepínacom, trochu nižšie, ako keby tvorila novú skupinu komponentov. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektoreobjektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. Vo výberovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Label3** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Adresa".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "10".
12. Presunieme sa na **Name** (Down), a zadáme "LabelAddress".
13. Presunieme sa na **Top** (Down) a zadáme "130".
14. Presunieme sa na **Width** (Down) a skontrolujeme, ci je nastavená na

hodnotu 72.

Dalším krokom je nastavenie vlastností desiateho komponentu, editacného políčka s adresou (EditAddress); zobrazí sa napravo od svojej menovky. Postupujeme takto:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Edit3** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Hint** (Down) a zadáme "".
10. Presunieme sa na **Left** (Down) a zadáme "92".
11. Presunieme sa na **Name** (Down) a zadáme "EditAddress".
12. Presunieme sa na **TabOrder** (Down) a zadáme "6".
13. Presunieme sa na **Text** (Down) a stlacíme Del, aby sme vymazali text, ktorý tu je predvolený.
14. Presunieme sa na **Top** (Down) a zadáme "130".
15. Presunieme sa na **Width** (Down) a zadáme "500".

Dalším krokom je nastavenie vlastností jedenásteho komponentu, zoznamu (ListBoxCity), v ktorom sa bude vyznacovat miesto bydliska; zobrazí sa napravo od editacného políčka adresy, ako súcast tejto skupiny údajov. Postup je takýto:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. V ponukovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **ListBox1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Height** (Up) a zadáme "30".
10. Presunieme sa na **Hint** (Down) a zadáme "".

11. Presunieme sa na **Items** (Down).
12. (Ctrl+Enter).
13. Ked sa otvorí okno **StringListEditor** zadáme "Vyznacte mesto...":
14. (Enter),
15. zadáme "Bratislava",
16. (Enter),
17. zadáme "Banská Bystrica",
18. (Enter),
19. zadáme "Košice",
20. (Enter).
21. Tabulátorom **Tab** sa presunieme na **OK** a potvrdíme klávesom Enter, tým sa zároven zavrie toto okno. Zatial sme vytvorili tri položky, z ktorých si neskôr budeme môct vyberat.
22. Presunieme sa na **Left** (Down) a zadáme "602".
23. Presunieme sa na **Name** (Down), a zadáme "ListBoxCity".
24. Presunieme sa na **TabOrder** (Down) a zadáme "7".
25. Presunieme sa na **Top** (Down) a zadáme "130".
26. Presunieme sa na **Width** (Down) a zadáme "160".

Teraz musíme nastavit vlastnosti dvanástemu komponentu, menovke (LabelNote), ktorá bude pred viacriadkovým políckom (memo) pre poznámky; zobrazí sa pod menovkou pre adresu, trochu nižšie, ako keby tvorila novú skupinu komponentov. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. Vo výberovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Label4** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na Font (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Poznámky".
10. Presunieme sa na **Hint** (Down) a zadáme "".
11. Presunieme sa na **Left** (Down) a zadáme "10".
12. Presunieme sa na **Name** (Down), a zadáme "LabelNote".
13. Presunieme sa na **Top** (Down) a zadáme "190".
14. Presunieme sa na **Width** (Down) a skontrolujeme, ci je nastavená na hodnotu "40".

Teraz musíme nastavit vlastnosti trinástemu komponentu, objektu "Memo" (MemoNote); zobrazí sa napravo od svojej menovky. Môžeme to urobit nasledujúcim spôsobom:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. Vo výberovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Memo1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).
7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Hint** (Down) a zadáme "".
10. Presunieme sa na **Left** (Down) a zadáme "60".
11. Presunieme sa na **Lines** (Down).
12. (Ctrl+Enter).
13. Ked sa otvorí okno **StringListEditor**, stlacíme **Back**, aby sme vymazali predvolený text "Memo1".
14. Tabulátorom **Tab** sa presunieme na **OK** a stlacením (Enter) potvrdíme a zároven zatvoríme okno.
15. Presunieme sa na **Name** (Down), a zadáme "MemoNote".
16. Presunieme sa na **TabOrder** (Down) a zadáme "8".
17. Presunieme sa na **Text** (Down) a stlacíme Del, aby sme vymazali predvolený text.
18. Presunieme sa na **Top** (Down) a zadáme "190".
19. Presunieme sa na **Width** (Down) a zadáme "700".

Teraz nastavíme vlastnosti štrnásteho a posledného komponentu, tlacidla (ButtonAdd), ktorým potvrdíme a uložíme údaje, ktoré sme doteraz zadali; v okne sa zobrazí vpravo dole. Postup je nasledovný:

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov, ak nie, presunieme sa tam funkcným klávesom (F11).
2. Vo výberovom poli sa presunieme sa na ten komponent, s ktorým ideme pracovat (Ctrl+Down).
3. Presunieme sa na **Button1** (Up) alebo (Down).
4. (Enter).
5. Presunieme sa na **Font** (Up) alebo (Down).
6. (Ctrl+Enter).

7. Vyberieme písmo Arial 14.
8. (Enter).
9. Presunieme sa na **Caption** (Up) a zadáme "Pridaj".
10. Presunieme sa na **Height** (Down) a zadáme "30".
11. Presunieme sa na **Hint** (Down) a zadáme "".
12. Presunieme sa na **Left** (Down) a zadáme "650".
13. Presunieme sa na **Name** (Down), a zadáme "ButtonAdd".
14. Presunieme sa na **TabOrder** (Down) a zadáme "9".
15. Presunieme sa na **Top** (Down) a zadáme "500".
16. Presunieme sa na **Width** (Down) a zadáme "100".

Na konci tohoto dlhého cvicenia už môžeme vidiet nejaké výsledky. Ak stlacíme funkcný kláves (F9) a pár sekúnd pockáme, na monitore uvidíme "spúštatelnú" verziu okna, ktoré sme práve navrhli; môžeme sa vnom pohybovat, zapisovat do editacných polícok, vyberat položky, zaciarkávat polícka pri položke pohlavie, avšak ak stlacíme tlacidlo Pridat, nic sa nestane: na to, aby nieco stalo, sa musíme naucit ešte viac.

4.5 Navrhovanie formulárov prístupných pre nevidiacich pou□ívatel ov

Ciel tejto podkapitoly: Naucit sa navrhovat formuláre prístupné pre nevidiacich používatelov

V tejto casti si povieme o niektorých jednoduchých pravidlách, ktoré nám umožnia vytvárat štandardné aplikácie vo Windows prístupné pre nevidiacich používatelov.

Aké doporucenia ku grafickému stvárneniu by sme mali dat zrakovo postihnutým programátorom, aby boli aplikácie, ktoré navrhujú, prístupné pre zrakovo potihnutých používatelov?
Cielom nie je vytváranie špeciálnych "vyclenených" aplikácií; ale používanie štandardných aplikácií pre Windows, v ktorých sa uplatnujú nasledujúce jednoduché pravidlá. Tieto pravidlá sú užitocné pre programátorov aj pre všetkých používatelov aplikácií.

Každý komponent má vlastnost Top a Left, tieto urcujú vzdialenost komponentu od horného a lavého okraja formulára v obrazových bodoch (pixel). Vlastnosti komponentu výška (Height) a šírka (Width) urcujú jeho velkost. Na navrhovanie velkostí komponentov a ich umiestnovanie si každý musí nájst svoj vlastný systém, avšak vo všeobecnosti sa používajú

násobky 10, a medzi komponentami sa vynecháva aspon 10 obrazových bodov (pixel). Možno najjednoduchšie bude pracovat so štvorcovými komponentami 50 bodov vysokých a 50 bodov širokých, s 50 bodovou medzerou medzi komponentami. Tak, ako pre jednoduchost navrhujeme aj v cvicení v dalšej kapitole. Avšak väcšina programátorov používa obdlžnikové komponenty so šírkou a výškou v pomere 2 ku 1. Ak budete pracovat na nejakom nárocnejšom formulári, skôr než ho zacnete programovat, navrhnite si ho, napríklad pomocou kociek Lega alebo na kolíkovej tabulke.

4.5.1 Pridržiavanie sa štandardov Windows

Vo všeobecnosti, grafická stránka aplikácie by mala byt podla možnosti co najpodobnejšia štandardom používaným vo Windows. Štandardné nastavenia vlastností komponentov, sú tie, ktoré nám ponúka pocítac: (Obrazovka, Vzhlad, Schéma).
Nastavenia používatela sú pre neho zjavne tie najvhodnejšie, preto by programátor nemal prepisovat tieto parametre, najmä:

1. Farbu komponentov a ich nadpisov, vrátane nadpisu formulára; Ak je z nejakého dôvodu nevyhnutné menit farbu pozadia alebo textu, doporucujeme zmenit aj farbu textu, inak sa môže stat, že sa nám zobrazí editacné políčko s ciernym textom na ciernom pozadí.
2. Okraje komponentov. Napríklad textové pole bez okrajov môže byt mylne pokladané za menovku a naopak.
3. Kurzor myši. Kurzory navrhnuté podla vlastného vkusu sa môžu na obrazovke zle zobrazovat alebo negatívne vplývat na prehladnost pracovného prostredia.

4.5.2 Typ a velkost písma

Písmo by nemalo byt menšie než 14 bodové, alebo by sa malo dat aspon nastavit podla priania používatela. Navrhujeme používat bežné typy písma (Arial, Times New Roman, Courier New); prípadne by sa typ tiež mohol dat nastavit. Predvolený typ písma je MS Sans Serif, 8 bodové.

4.5.3 Textový popis (bublinkový popis) komponentu.

Všetky komponenty by mali mat svoj textový popis, ktorý sa zobrazí, ak presunime kurzor myši na komponent; tento opis poskytne používatelovi

pomocou cítaca obrazovky informáciu o komponente. Text by mal byt obsiahnutý vo vlastnosti **"Hint"** daného komponentu, a potom aj vlastnost **"ShowHint"** musí byt nastavená na "True" (pravda). Ak sa chcete o vlastnosti "Hint" dozvediet viac, vyznacte ju a stlacte funkcný kláves "F1".

4.5.4 Správne umiestnovanie komponentov vo formulári

Zvláštny dôraz by sme mali klást na umiestnovanie komponentov vo formulári tak, aby aplikácia jednoducho fungovala.
Vieme, že vlastnosti "Width" (šírka) a "Height" (výška) urcujú velkost komponentu a "Top" a "Left" urcujú umiestnenie vo formulári. Preto je správne nastavenie vlastností "Top" a "Left" nevyhnutné aj ked chceme komponent umiestnit do pravej dolnej casti formulára.
Spravidla, ked už sme vyznacili rozmery formulára, môžeme umiestnovat komponenty; treba dbat najmä na:

1. Vynechávanie miesta medzi komponentami a okrajom najmenej 10 obrazových bodov (pixel).
2. Vynechávanie miesta medzi komponentami rovnakej skupiny, napr. menovka a editacné políčko, najmenej 10 obrazových bodov (pixel).
3. Vynechávanie miesta najmenej 10 obrazových bodov (pixel) medzi dvoma skupinami komponentov.
4. Skontrolovanie velkosti a dlžky textu, ktorý sa má zobrazit v komponente, ktorý sme práve umiestnili, predtým, než prejdeme na další komponent (napravo alebo pod)

POZOR: Nastavenie niektorých komponentov prebehne automaticky alebo poloautomaticky; najmä:

1. "Edit" (editacné políčko) a "ComboBox" (kombinovaný rámik). Výška komponentu sa nastaví automaticky na velkost textu nastavenej vlastnostou Font (písmo)
2. "Label" (menovka). Výška a dlžka komponentu sa nastaví automaticky na velkost textu nastavenej vlastnostou Font (písmo)
3. "Memo". (Viacriadkové textové pole), prispôsobovanie komponentu neprebieha automaticky.
4. "ListBox" (Zoznam). Nastavenie komponentu neprebehne automaticky. Nastavením vlastnosti "IntegralHeight" na "True" (pravda) sa výška komponentu prispôsobí tak, že bude obsahovat presne jeden alebo viac riadkov textu. Vyberte velkost textu, potom vo vlastnosti "Height" zadajte hodnotu rovnú alebo vyššiu výške riadkov, ktoré sa majú zobrazit, v obrazových bodoch, výška sa

potom automaticky nastaví na správnu hodnotu. Napríklad: text písmom "Arial", 14 bodové

Výška,	Riadok,	Autom. výška;
0 to 25	0	0
26 to 47	1	26
48 to 69	2	48
70 to 91	3	70

5. "Button" (tlacidlo), "CheckBox" (zaciarkávacie políčko) a "RadioButton" (prepínac). Velkosti komponentov sa menovke názvu neprispôsobujú automaticky. Na správne nastavenie vlastností komponentu doporucujeme vo vlastnosti Font použit písmo "Courier New" (neproporcionálne písmo) v štýle bold (tucné). V nasledujúcej tabulke sú uvedené navrhované výšky a šírky v obrazových bodoch pre velkosti rôznych názvov (v bodoch)

Velkost písma	Výška	Šírka (1 znak)
8	17	28+7 na znak
10	20	30+8 na znak
12	23	33+10 na znak
14	26	36+11 na znak
16	29	39+13 na znak
18	32	42+14 na znak

4.6 Prechod od formulára k aplikácii

Ciel tejto podkapitoly: Na príklade sa správnym spôsobom naucit vytvorit aplikáciu.

V tejto casti si krok za krokom ukážeme, ako vytvorit aplikáciu.

4.6.1 Príklad aplikácie "Hello"

Cvicenie.
Navrhneme jednoduchú aplikáciu, ktorá bude po opakovanom stlácaní klávesu zobrazovat slovo "Hello!" a za ním názov zakaždým inej európskej krajiny.

Na navrhnutie takejto aplikácie potrebujeme:

- pridat všetky potrebné komponenty do predvoleného formulára,
- nastavit vlastnosti komponentov,
- definovat obsluhy udalostí.

4.6.2 Pridávanie komponentov do "Hello"

Po pridaní komponentov **Tedit**, **Tlabel** a **TButton** (plus formulára, ktorý ako teraz vieme, sa v BCB pridá automaticky), nájdeme tieto štyri komponenty Vo výberovom poli komponentov inšpektora objektov: Button1, Edit1, Label1 and Form1

4.6.3 Nastavenie vlastností komponentov

Nastavenie vlastností pre **Button1**:
Caption = "" zrušíme a necháme prázdne
Font = Arial, Bold, 14 (vyberieme zo štandardného dialógového okna);
Hint = "Zmen..." (zadáme);
Left = 150 (zadáme);
ShowHint = True (vyberieme zo zoznamu);
TabOrder = 1 (zadáme);
Top = 70 (zadáme);
Width = 380 (zadáme).

Nastavenie vlastností pre **Edit1**:
Font = Arial, Bold, 14 (vyberieme zo štandardného dialógového okna);
Left = 150 (zadáme);
Top = 50 (zadáme);
Hint = "Zobrazit" (zadáme);
ShowHint = True (vyberieme zo zoznamu);
TabOrder = 0 (zadáme);
Text = "Stlac tlacidlo..." (zadáme);
Width = 380 (zadáme).

Nastavenie vlastností pre **Label1**:
Alignment = taRightJustify (vyberieme zo zoznamu);
Caption = "Description" (zadáme);
Font = Arial, 14 (vyberieme zo štandardného dialógového okna);
Hint = "Popis" (zadáme);

Left = 10 (zadáme);
Top = 10 (zadáme);
ShowHint = True (vyberieme zo zoznamu);
Width = 130 (zadáme).

Nastavenie vlastností pre **Form1**:
Name = "Hello!" (zadáme);
Caption = "Hello!" (zadáme).

4.6.4 Definovanie obslúh udalostí

Aby sme mohli napísat program pre funkciu, ktorá spracuje udalost, musíme sa stlacením funkcného klávesu (F11) premiestnit do okna **Unit1.cpp**.
Pod textom THello *Hello vynecháme riadok.

Napíšeme tento kód:

```
const int MAX_HELLO = 6;
String Edit[MAX_HELLO];
String Button[MAX_HELLO];
String Label[MAX_HELLO];
int counter_Edit=0;
int counter_Button=0;
int counter_Label=0;
```

Po návrate do OI oznacíme formulár **Hello**, potom sa presunieme do **Events**, oznacíme **OnCreate** a stlacíme (Ctrl+Enter)

Medzi zložené zátvorky vpíšeme tento kód:

```
Edit[0]="Hello Austria!";
Edit[1]="Hello England!";
Edit[2]="Hello Germany!";
Edit[3]="Hello Greece!";
Edit[4]="Hello Italy!";
Edit[5]="Hello Slovakia!";

Button[0]="Austria";
Button[1]="England";
Button[2]="Germany";
```

```
Button[3]="Greece";
Button[4]="Italy";
Button[5]="Slovakia";

Label[0]="Austria";
Label[1]="England";
Label[2]="Germany";
Label[3]="Greece";
Label[4]="Italy";
Label[5]="Slovakia";

Button1->Caption=Button[0];
counter_Button++;
```

Po návrate do OI vyznacíme komponent **Button1**, potom sa premiestnime do **Events**, oznacíme **OnClick** a stlacíme **Ctrl+Enter**.

Teraz medzi zložené zátvorky vpíšeme tento kód:

```
Edit1->Text=Edit[counter_Edit];
Button1->Caption=Button[counter_Button];
Label1->Caption=Label[counter_Label];

if (counter_Edit==MAX_HELLO-1)
        counter_Edit=0;
else
        counter_Edit++;

if (counter_Label==MAX_HELLO-1)
        counter_Label=0;
else
        counter_Label++;

if (counter_Button==MAX_HELLO-1)
        counter_Button=0;
else
        counter_Button++;
```

Teraz, po uložení programu, môžeme spustit program.

4.6.5 Projekt

Na vytvorenie aplikácie potrebujeme zacat projektom. Projekt obsahuje všetky zložky, ktoré sa skompilujú a vytvoria aplikáciu nazvanú: súbor "**.exe**".
Na zobrazenie komponentov navrhovanej aplikácie otvoríme ponuku **View/Project/Manager** (stlacíme Ctrl+Alt+F11) a pohybujeme sa po položkách ponuky so stromovou štruktúrou.
Každý komponent sa uloží s odlišnou príponou, podla typu komponentu. v našej aplikácii sa uložia nasledovné súbory:

Unit1.dfm obsahuje grafickú stránku formulára a komponentov donho pridaných.
Unit1.cpp obsahuje C++ kód obslúh udalostí alebo akékolvek dalšie funkcie.
Unit1.h obsahuje implementované funkcie C++ súborov.
Project1.bpr súbor projektu, ktorým sa dá projekt otvorit vo vývojovom prostredí C++.
Project1.cpp obsahuje funkciu "winmain" na zacatie vykonávania programu.
Project1.res (k tomu sa vrátime neskôr).

Projekt uložíme tak, že otvoríme ponuku (File/Save All) stlacením Ctrl+Shift+S.
Teraz otvoríme nový priecinok, aby sme mohli uložit formulár Unit 1, použijeme na to bežný spôsob ukladania súborov. Aj na uloženie projektu project 1 použijeme bežný spôsob ukladania. Pocítac sa nás opýta, ci chceme projekt uložit do rovnakého priecinka ako sme predtým uložili formulár; dalšou úlohou je zamenit "project1" za "hello".
Ak minimalizujeme aplikáciu C++ a otvoríme priecinok, ktorý obsahuje projekt, uvidíme výsledky. Teraz podme skompilovat a spustit program.

4.6.6 Kompilácia, spúštanie programu

Kompilovanie znamená transformáciu návrhu a implementácie projektu do strojového kódu; t.j. postupností binárnych inštrukcií, ktoré sú zrozumitelné pre operacný systém.

Teraz sa presunme do panelu ponúk (menu bar):
(Run/Run) alebo F9, po pár sekundách sa spustí Hello.exe
Prostredníctvom klávesu Tab sa presunte na tlacidlo "Hello", stlacením

(Enter) sa zakaždým správa v editacnom poli zmení.
(Alt+F4) slúži na ukoncenie programu.
V priecinku, do ktorého sme predtým uložili projekt teraz nájdeme štyri nové súbory.
Hello.exe = spúštatelný súbor
Hello.obj a **Unit1.obj** = binárne súbory, ktoré sme získali kompiláciou .cpp.
Hello.tds = s informáciami pre ladenie (debugging)

4.7 Ponuky (Menu)

Ciel tejto podkapitoly: Naucit sa ako pridat ponuky (menu) do aplikácie

Štandardné aplikácie vo Windows majú panel ponúk (Menu Bar). Panel ponúk je vodorovný riadok so základnými textovými linkami, ktorý sa nachádza hned pod titulným riadkom a nad panelom nástrojov (v prípade, že aplikácia má panel nástrojov). Ako vieme, kliknutím na položku sa objaví okno s viacerými podpoložkami.

4.7.1 Pridávanie komponentu TMainMenu

V BCB je panel ponúk komponent ako každý iný (rovnako ako Edit (editacné pole), Button (tlacidlo), Label (menovka) atd.), a preto ho musíme do formulára pridat.

- (F11) na presun do formulára **Hello!**,
- (Alt) na presun do panelu ponúk (**menu bar**),
- (Right) na presun do ponuky **View**,
- (Down) na presun na položku **Component List**,
- (Enter),
- (Down) posun na (**TMainMenu**),
- (Enter),
- (Esc) na zatvorenie okna **Component List**.

Komponent sa nám teraz zobrazil v strede formulára ako ikona; jeho pôvodný názov je **MainMenu1**.
(F11) na presun do okna **OI**.
Pri listovaní vo vlastnostiach **MainMenu1**, si môžeme všimnút, že tam nie sú žiadne vlastnosti, ktoré by sa vztahovali na umiestnenie, ako je napr.

vlastnost Top, Left atd. Toto znamená, že komponent **MainMenu** je grafický kontajner, ktorý si možno prispôsobit pomocou špecifických funkcií, podobných tým, ktoré sú v navigacnom paneli prehliadaca Internetu.

Dalšou úlohou je navrhnút grafickú štruktúru panelu ponúk (menu bar).

4.7.2 Navrhovanie panelu ponúk (Menu Bar)

Ak chceme navrhnút panel ponúk (menu bar), musíme splnit tieto úlohy:

- Object Inspector,
- (Ctr+Down) na presun do zoznamu komponentov, na ktorých chceme pracovat,
- (Up) alebo (Down) na presun do **MainMenu1**,
- (Enter),
- (Up) alebo (Down) na presun na **Name**,
- Zadáme "HelloMenuBar".

Každá položka ponuky "HelloMenuBar" je spravovaná ako sub-komponent, v okne OI má svoj zoznam vlastností a spúšta sa udalostou (OnClick). Toto si ukážeme neskôr.
Vlastnosti každej položky ponuky sú nasledovné:
Caption nastavuje nadpis položky v zozname (Súbor, Úpravy atd.)
Checked v zozname, nalavo od položky zobrazuje fajocku, fajocka má funkciu prepínaca
Enabled upozorní, že položka je aktívna.
Hint umožnuje nastavit bublinkový popis, ktorý sa zobrazí, ked umiestnime kurzor myši na komponent.
Name umožnuje nastavit identifikacné meno komponentu v rámci aplikácie, toto meno používa program na identifikáciu komponentu.
Shortcut z rozbalovacieho zoznamu umožní vybrat sadu klávesových skratiek, pomocou ktorých sa bude možné presunút na udalost **OnClick nejakej položky**.

Dalšou úlohou je zostavit náš panel ponúk (menu bar).
Základný zoznam ponuky **HelloMenuBar** bude zostavený zo Súbor, Zobrazit, Pomoc.
Ponuka **File** bude mat položku nazvanú **Close**, táto umožní opustit

aplikáciu.
Ponuka **View** bude mat položku nazvanú **Change**, ktorá bude spúštat udalost OnClick, a táto bude zakaždým menit jazyk v "Hello".
Ponuka **Help** bude mat položku nazvanú **Info**, bude slúžit na zobrazenie okna Help.

1. Skontrolujeme, ci sa ešte stále nachádzame v inšpektore objektov.
2. (Ctrl+Down) na presun do výberového pola komponentov, na ktorých chceme pracovat.
3. (Up) alebo (Down) na presun na **MainMenu1**.
4. (Enter).
5. (Up) alebo (Down) na presun na **Items**.
6. (Ctrl+Enter).
7. Zobrazí sa nám okno "Hello->HelloMenuBar"; nalavo hore je editacné políčko: toto je prvá položka hlavnej ponuky.
8. Zadáme "&File".
9. (Enter).
10. Teraz sa nám zobrazili dve nové editacné políčka, prvé, napravo od File, druhé (aktívne) je pod ním.
11. Zadáme "&Close".
12. (Enter).
13. Teraz sa nám pod **Close** zobrazí nové editacné políčko; stlacíme (Right), aby sme sa presunuli na prázdne editacné políčko napravo od **File**.
14. Zadáme "&View".
15. (Enter).
16. Teraz sa nám zobrazili dve nové editacné políčka, prvé, napravo od **View**, druhé (aktívne) je pod ním.
17. Zadáme "&Change".
18. (Enter).
19. Teraz sa nám pod **Change** zobrazilo nové editacné políčko; stlacením (Right) sa presunme napravo od View.
20. Zadáme "&Help".
21. (Enter).
22. Teraz sa nám zobrazili dve nové editacné políčka, jedno napravo od Help, druhé (aktívne) je pod ním.
23. Zadáme "&Info".
24. (Enter).
25. Pod Info sa zobrazí nové editacné políčko.

Co znamená symbol "&", ktorý sme zadali pred každú položku?

Je to automatická funkcia. Znak abecedy nasledujúci za týmto symbolom

bude podciarknutý; to znamená, že ak stlacíme (Alt+ podciarknuté písmeno) spustí sa udalost OnClick. Ak je už toto písmeno použité pri iných položkách, môžeme dat symbol "&" pred iné písmeno, ktoré chceme podciarknút.

Späť do OI.
(Ctrl+Down).
Uvidíme všetky položky implementovanej ponuky, plus položky, ktoré sme predtým pridali..
Vlastnost **Name** každej položky bude pozostávat z jej menovky plus "1", "File1,"View1","Help1" atd.

4.7.3 Pridávanie, odstranovanie a upravovanie panelov ponúk

Na to, aby sme pridali alebo vymazali položky v paneli ponúk **HelloMainMenu**, musíme:

1. Presunút sa do inšpektora objektov,
2. (Ctr+Down) presunút sa na komponent **HelloMenuBar**,
3. (Enter),
4. (Up) alebo (Down) na presun na položku **Items**,
5. (Ctrl+Enter)
6. Pomocou šípok sa presunút na položku, ktorú chceme zmazat.
7. (Del)
8. Pomocou šípok sa presunút na položku, ktorú chceme pridat.
9. (Ins)
10. Zadat menovku pre novú položku
11. (Enter)
12. Pomocou šípok sa presunút na položku, ktorú chceme pridat do podponuky.
13. (Ctrl+Right)
14. Zadáme menovku pre prvú položku podponuky.
15. Teraz uvidíme dve nové editacné políčka, prvé je napravo od položky, ktorú sme práve editovali, druhé (aktívne) je pod nou.
16. Dalej postupujeme bežným spôsobom.

Cvicenie: Zmente vlastnost položky v ponuke.
Presunieme sa do OI
(Ctrl+Down) vyznacíme komponent HelloMenuBar
(Enter)
(Up) alebo (Down) na presun na položku, ktorú chceme zmenit

A teraz musíme postupovat rovnako ako pri ostatných komponentoch.

4.7.4 Realizácia udalosti OnClick

Na implementáciu HelloMenuBar, musíme spúštat udalost **OnClick** pri **Close**, **Change** a **Info**.

Udalost OnClick pri Close.
Tento postup je pre nás pomerne známy.

V okne **Object Inspector**:
(Ctrl+Down) na vyznacenie komponentu **HelloMenuBar**,
(Enter)
(Up) alebo (Down) na presun na vlastnost **Items**
(Ctrl+Enter)
Šípkami sa presunieme na **Close**
Spät do OI
(Ctrl+Tab) na presun na **Events**
(Down) na presun na **OnClick**
(Ctrl+Enter)
V okne **Unit1.ccp**, medzi zložené zátvorky napíšeme tento kód:

```
Close()
```

Cvicenia:

Zrealizujte udalost **OnClick** pri **Change**; pridajte nasledujúci kód:

```
Button1Click(Sender)
```

Zrealizujte udalost **OnClick** pri **Info**; na jeden riadok pridajte nasledujúci kód:

```
Application->MessageBox ("Stlacte tlacidlo, ak
chcete zmenit správu", "Help",MB_OK);
```

Ukladanie aplikácie.

(File/Save All)
F9 pre kompiláciu
Spustite program

(Alt) na presun do panelu ponúk (menu bar)

Tak a teraz už môžeme vidiet výsledky našej práce.

Kapitola 5: Základy programovania v C++

Ciel tejto kapitoly:
Naucit sa vytvárat jednoduché programy na riešenie velmi jednoduchých problémov.

Riešenie jednoduchých problémov vrátane myšlienky udalostí (event), postupnosti krokov (sequence) a manipulácie s údajmi.
Použitie editacných políc ok (edit box), menoviek (label) a tlacidiel s príkazmi (command button).
Syntax C++:

premenné (variable) a konštanty (constant);
globálna and lokálna viditelnost (scope);
znakové (character) a císelné (numeric) typy údajov;
operátory pre císelné a znakové typy údajov;
konverzia medzi rôznymi typmi údajov;
rezervované slová (keyword) a koncové znaky (terminator);
komentáre (comment);

5.1 Umiestnenie komponentov

Každý z komponentov má vlastnosti `Top` (hore) a `Left` (vlavo). Urcujú vdzialenost koponentu od horného a od lavého okraja formulára v obrazových bodoch (pixel), pricom vlastnosti `height` a `width` urcujú jeho velkost. Budeme potrebovat zaviest vlastné pravidlá pre umiestnovanie komponentov a pre ich velkosti. Vo všeobecnosti by sme mali pracovat s násobkami desiatich a medzi komponentmi nechat medzeru aspon 10 bodov. Najjednoduchšie bude pracovat s komponentmi vysokými 50 bodov a širokými 50 bodov a s medzerou 50 bodov medzi komponentmi. Pre jednoduchost to doporucujeme v cviceniach nasledujúcej kapitoly. Napriek tomu väcšina programátorov používa obdlžníkové komponenty so šírkou, ktorá je aspon dvojnásobkom ich výšky. Ak pracujeme s komplikovaným formulárom, musíme ho najprv navrhnút - pokúsme sa použit pri rozmiestnovaní komponentov stavebnicu alebo kolíkovú tabulku (peg board) - a až potom zacnime s programovaním.

Problém 1 - Scitovanie dvoch císel

V prvom cvicení budú na vstupe (input) 2 císla, spocítame ich a výstupom (output) bude výsledok scítania.

Vstup sa casto realizuje pomocou editacných políc ok, zatial co výstup pomocou menoviek. Používatel môže menit vstup, ale nemal by mat možnost menit výstup, pretože je to výsledok výpoctu.

K tomuto cviceniu budeme potrebovat 2 editacné políc ka pre 2 scitované císla a 1 menovku pre výsledok. Každé z editacných políc ok by malo mat nablízku menovku s vhodným nadpisom (caption), ktorý urcuje jeho význam. Menovka pre výsledok bude obsahovat aj nadpis.
Všimnime si, že používame menovky na 2 úcely. Menovku používame na popísanie susedného pola s vstupnými alebo výstupnými údajmi, taktiež donho umiestnujeme výstupné údaje.
Na záver potrebujeme komponent, pomocou ktorého budeme spúštat udalost scitovanie císel. Existuje vela spôsobov, my použijeme tlacidlo s príkazom na ktoré klikneme pred samotným zadávaním 2 císel.

Zacnime nový projekt a do formulára pridajme:
2 editacné políc ka - edit box (`Tedit`)
3 menovky - label (`Tlabel`)
1 tlacidlo s príkazom - command button (`Tbutton`).

Pripomienka:
Pridáme ich cez okno s komponentmi (Components box) (Alt-V,C). Napíšeme jedno alebo dve zaciatocné písmená zmena komponentu a stlacíme (Enter). Ak už máme všetky potrebné komponenty, tak okno s komponentmi zatvoríme stlacením (Escape).

Všetky potrebné komponenty sú teraz nahromadené na kope v strede formulára. Preto je dalšou úlohou rozmiestnenie komponentov vo formulári a priradenie zmysluplných mien a nadpisov k nim. Mená ako “Edit1” alebo “Label2” nám síce povedia o aký druh komponentu ide, no nehovoria nic o ich úcele. Ak v programe použijeme niekolko formulárov, taktiež by mali mat zmysluplné mená. Nakolko naše prvé programy budú mat len po jednom formulári, môžeme ponechat pôvodné meno (Name) “Form1”.

V tomto programe umiestnime všetky komponenty do vodorovného pásu pozdlž horného okraja formulára. Takto ich precítame takmer ako bežný výsledok scitovania. Každý komponent bude mat šírku (Width) 50 obrazových bodov (pixels), medzera medzi dvoma komponentmi bude 50 bodov. Takže každý komponent bude mat vlastnost `Top` (hore) nastavenú

na 0, a vlastnost Width nastavenú na 50. Prvý komponent bude mat vlastnost Left (vlavo) nastavenú na 0, další na 100, potom 200 atd. v tomto formulári bude 6 komponentov, musíme preto skontrolovat, ci je formulár dostatocne široký. Vlastnost Width (šírka) mu nastavíme na minimálne 600. Budeme sa snažit používat štvorcové komponenty, preto aj ich vlastnost Height (výška) bude nastavená na 50. Zistíme, že tomu nebude vždy tak, najmä pri menovkách, no to nás teraz nemusí trápit.

Zacneme s komponentom "Label1". Zmeníme jeho vlastnosti Name (meno) a Caption (nadpis) a potom ho presunieme.
(Pripomienka: (F11) pre okno Object Inspector, potom (Ctrl+Down) k zoznamu komponentov.)
Zmeníme vlastnost Name na lbl_firstnumber.
Zmeníme vlastnost Caption na "Prvé císlo".
Zmeníme vlastnost WordWrap na true. Týmto zabezpecíme, aby bol celý nadpis v menovke viditelný.
Presunieme ho do lavého horného rohu formulára tak, že mu nastavíme obe vlastnosti Top aj Left na 0.
Vlastnosti Width a Height nastavíme na 50.
Ak zmeníme výšku menovky a potom nejakú inú vlastnost, môže sa stat, že sa výška zmení na pôvodnú hodnotu. Teraz to naozaj nie je dôležité, no je dobré byt informovaný, že sa nieco takéto môže stat.

Dalej umiestnime komponent "Edit1". Bude obsahovat prvé císlo z nášho súctu. Opät budeme menit vlastnosti Name a Caption a potom ho presunieme.
Zmeníme vlastnost Name na edi_firstnumber.
Zmeníme vlastnost Text na medzeru (blank space), pretože po spustení programu tu budeme zapisovat císlo.
Zmeníme vlastnost Position na Top = 0, Left = 100, Height = 50, Width = 50.
Toto bude prvý komponent, ktorý v programe použijeme, preto sa uistíme, že jeho vlastnost TabOrder (poradie pri stlácaní (Tab)) má hodnotu 0.
Všimnime si, že prvý komponent má TabOrder rovný 0. Coskoro si zvykneme na spôsob pocítania v C++, kde zväcša zacíname pocítat od 0 a nie od 1.

Dalej potrebujeme umiestnit tlacidlo. Vlastnost Name zmeníme na btn_add a Caption zmeníme na Scítaj!. Jeho pozíciu zmeníme pomocou vlastností Top = 0, Left = 200, Height = 50, Width

= 50.

Teraz sa pokúsime správne umiestnit komponenty “Label2” and “Edit2” v našom formulári. Použijeme ich pre druhé číslo. Urobíme tak ešte skôr, než si pozrieme nižšie uvedené riešenie.

Riešenie:
Label2 má mat vlastnosti Name = lbl_secondnumber, Caption = Druhé číslo, Top = 0, Left = 300, Height = 50, Width = 50.
Edit2 má mat vlastnosti Name = edi_secondnumber, Text je prázdny, Top = 0, Left = 400, Height = 50, Width = 50.

Label3 bude na zaciatku obsahovat nadpis (Caption) “Výsledok”. Neskôr bude obsahovat aj samotný výsledok. Umiestnime ho vo formulári a potom si pozrieme nižšie uvedené riešenie.

Riešenie:
Label3 má mat vlastnosti Name = lbl_answer, Caption = Výsledok, Top = 0, Left = 500, Height = 50, Width = 50.

Všimnime si, že po umiestnení komponentov vo formulári sa môžeme medzi nimi pohybovat pomocou klávesu (Tab). Medzi komponentmi, ktoré sú približne na jednej vodorovnej ciare (v riadku) sa môžeme pohybovat aj pomocou šípok vlavo a vpravo. Medzi komponentmi, ktoré sú približne na jednej zvislej ciare (v stlpci) sa môžeme pohybovat aj pomocou šípok hore a dolu. Vyskúšajme si použitie šípok.

Navrhli sme formulár. Ak sa teraz pokúsime program spustit, bude skompilovaný a spustený, ale v podstate nic neurobí. Vyskúšajme to! Ked zadáme dve císla a klikneme na tlacidlo Scítaj!, nepozorujeme žiadnu zmenu. Preco je to tak? Vieme co by sa malo stat po stlacení tlacidla Scítaj!. Je to ocividné, mali by sa scítat císla. Avšak pocítac vie urobit len to, co mu prikážeme aby urobil, a my sme vlastne nepovedali, že chceme po stlacení tlacidla scítat císla.

Urobíme dvojklik na tlacidle Scítaj!, alebo stlacíme (F11) pre otvorenie okna Object Inspector. Potom stlacíme (Ctrl+Down) pre zoznam komponentov, vyberieme btn_add a pomocou (Ctrl+Tab) sa presunieme na záložku Events. Nájdeme udalost OnClick, pomocou (Tab) sa

presunieme do pravého stlpca, ktorý bude prázdny a stlacíme (Ctrl+Enter). Tým sa presunieme do okna skódom a nájdeme kus kódu patriaceho udalosti nazvanej `btn_addClick`, ktorý pre nás vygeneroval BCB5. Keby sme sa vrátili do okna Object Inspector pre `btn_add`, zistili by sme, že tu bola pre nás pomenovaná udalost `OnClick`.

Otvorí sa okno s kódom pre Unit1 a kurzor by mal byt umiestnený vo vnútri funkcie `btn_addClick`. Toto je udalost, ktorá sa spúšta kliknutím na tlacidlo Scítaj!. Kód by mal vyzerat takto:

```
void __fastcall TForm1::btn_addClick(TObject
*Sender)
{

}
```

Zložené zátvorky tu majú velký význam. Sú akýmsi kontajnerom pre všetok kód, ktorý patrí funkcii. Pozorne tento kód vpíšme medzi zátvorky:

```
int num1, num2, answer;
num1 = edi_firstnumber->Text.ToInt();
num2 = edi_secondnumber->Text.ToInt();
answer = num1 + num2;
lbl_answer->Caption = "Výsledok = " +
String(answer);
```

Teraz sa opät pokúsime spustit program. Ked teraz klikneme na tlacidlo Scítaj!, mali by sme získat výsledok rovný súctu zadaných císel.

Akú carovnú formulu sme to práve napísali, že náš program pracuje presne tak, ako sme chceli? Program v C++ obsahuje množstvo funkcií. Prvé funkcie, s ktorými sa stretávame, sú špeciálne funkcie nazývané event handler (funkcia, ktorá spracuje udalost). Takáto funkcia obsahuje kód, ktorý spracúva spustenú udalost. V našom prípade bola udalost spustená kliknutím na tlacidlo Scítaj!. Toto je udalost "`OnClick`", C++ ju pre nás pomenuje - priradí jej meno, ktoré sa skladá z mena tlacidla a slova "Click". Kód funkcie je vždy uzavretý v zložených zátvorkách `{ }`. Funkcie môžu mat aj parametre, ktoré sa uvádzajú bezprostredne za menom funkcie, a sú uzavreté v okrúhlych zátvorkách `( )`. Ešte sa k tomu vrátime, avšak zatial ponecháme C++, aby pre nás automaticky generoval parametre, zatial co sa zameriame na kód v zložených zátvorkách.

Náš kód pre túto funkciu (funkciu, ktorá spracuje udalost) má päť riadkov, ktoré potrebujeme pozorne analyzovat.

```
int num1, num2, answer;
```

Pri väčšine funkcií budeme potrebovat deklarovat nejaké premenné. Tento príkaz hovorí kompilátoru, že budeme mat tri celocíselné premenné (integers). Všimnime si, že premenné v zozname sú oddelené ciarkami. C++ skracuje `integer` na `int`. Sú to lokálne premenné, ktoré sú platné len v rámci funkcie a zanikajú pri dosiahnutí jej koncovej zloženej zátvorky `}`.

Premenné môžeme deklarovat kdekolvek v rámci kódu, no je dobrým zvykom deklarovat všetky na jednom mieste, a to na zaciatku funkcie. Meno premennej musíme vždy písat znak po znaku presne tak, ako sme ho uviedli pri deklarácii premennej. Ked sa teda objaví chyba pri kompilácii "`Undefined symbol`" (symbol nie je definovaný), jednoducho porovnáme znak po znaku premennú v príkaze s jej deklaráciou.
V kóde sa taktiež objaví pomerne vela bodkociarok.

Interpunkcia je dôležitou súcastou kódu. Pri bežnom písaní uspejeme aj ked zabudneme nejaké ciarky alebo ich použijeme tam, kde by bolo vhodnejšie použit bodkociarky. Ale v programovaní je interpunkcia rovnako dôležitá ako pravopis. V C++ sa bodkociarka používa ako koncový znak (terminator) príkazu (ako napr. deklarácie alebo príkazu na uskutocnenie niecoho). V niektorých jazykoch píšeme každý príkaz na nový riadok. C++ je jazyk s volným formátovaním, co znamená, že programátori môžu kód stlácat alebo roztahovat podla lubovôle. Vo všeobecnosti je jednoduchšie cítat roztiahnutý kód. Je to výhodné pre cloveka, nie však pre pocítac. Kompilátor nepotrebuje mat kód rozdelený na jednotlivých riadkoch, potrebuje však príkazy oddelené bodkociarkami a bloky kódu, akými sú napr. funkcie, uzavreté v zložených zátvorkách.

```
num1 = edi_firstnumber->Text.ToInt();
```

Toto je priradovací príkaz. Identifikátoru nalavo od znaku rovná sa je priradená hodnota výrazu napravo. V tomto prípade je premennej num1 priradená hodnota, ktorú sme vpísali do editacného polícka. C++ používa symbol `->` na spojenie objektu s konkrétnou vlastnostou. V tomto prípade je to vlastnost `Text` objektu `edi_firstnumber`. Ale nechceme `Text` ako

retazec znakov, chceme Text ako celé číslo. Funkcia ToInt() konvertuje retazec znakov na celé číslo.

```
num2 = edi_secondnumber->Text.ToInt();
```

Toto je další priradovací príkaz, velmi podobný tomu predchádzajúcemu. Nastavuje num2 na hodnotu rovnú číslu , ktoré sme vpísali do druhého editacného políčka.

```
answer = num1 + num2;
```

Tento priradovací príkaz len scíta dve čísla a výsledok vloží do premennej "answer". Pamätajme, že priradovací príkaz musí mat identifikátor pre výsledok nalavo. Ak napíšeme

```
num1 + num2 = answer;
```

objaví sa chyba pri kompilácii.

```
lbl_answer->Caption = "Výsledok = " +
String(answer);
```

Toto je další priradovací príkaz. Nalavo je vlastnost Caption komponentu lbl_answer. Kedže bola premenná answer deklarovaná ako celé číslo (int), musíme ho najprv skonvertovat späť na text a až potom môže byt zobrazené vkomponente label. Použijeme na to funkciu String(). Aj znak + tu má nové použitie, spojenie dvoch retazcov. Výsledkom je, že druhý retazec je pripojený na koniec prvého, resp. zretazený (concatenated). Tento silný nástroj, ktorý majú jazyky podobné C++ nazývame "pretaženie" operátorov (overloading). Znak + "vie" z kontextu ci má spocítat dve čísla alebo zretazit dva retazce. Úvodzovky okolo "Výsledok = " hovoria pocítacu, že má dat na výstup presne to (znak po znaku), co je v nich uvedené. Casto to nazývame "string literal".

Všimnime si, že sme tu použili funkcie dvoma rôznymi spôsobmi:

```
edi_secondnumber->Text.ToInt()
String(answer)
```

V prvom prípade je syntax

```
objectname.functionname()
```

a v druhom prípade máme syntax

```
functionname(parameter)
```

Ktorý zápis máme použit? Presne ten, ktorý potrebujeme

```
objectname.functionname(parameter)
```

C++ pre nás automaticky vygeneroval hlavicku funkcie pre udalost button click. Je takáto:

```
void __fastcall TForm1::btn_addClick(TObject
*Sender)
```

Objektom tu je `Tform1`, funkciou je `btn_addClick` a parametrom je `*Sender`. Pri volaní funkcie sa dvojitá dvojbodka `::` z deklarácie funkcie zmení na `.` alebo `->`.
Niekedy nepotrebujeme meno objektu, inokedy nepotrebujeme parameter. Dúfajme, že to bude jasnejšie ako bude kurz pokracovat. Zatial programujme podla uvedených príkladov a príliš sa tým nezatažujme.

5.2 Testovanie

Niekolkokrát spustíme náš program. Pracuje vždy správne? Co ak zadáme desatinné císla? Co sa stane ak namiesto císel zadáme znaky? testovanie je dôležitou fázou výroby programu. len ak dôkladne otestujeme program, môžeme si byt pomerne istí, že sme našli chyby. Kolko ste použili takých komercných programov, ktoré pracovali korektne a nikdy "nepadali"? Napriek velkému úsiliu vynakladanému na ich testovanie, sú velké programy len zriedkakedy bez chybiciek.

5.3 Komentáre

Dalšou dôležitou súcastou programovania, o ktorej sme ešte neuvažovali, je dokumentácia. Ak nacítame kód tohto programu po šiestich mesiacoch, budeme presne vediet co robí? Najlepším miestom na vysvetlenie co a ako program robí je samotný program. Takto urcite nestratíme dokumentáciu. Ak máme kód, máme aj vysvetlivky.

Existujú dva spôsoby, ktorými by sme mali dokladovat kód. Jedným z nich je "autodokumentácia", ked program hovorí sám o sebe, pretože sme použili zmysluplné mená premenných a napísali sme prehladný a lahko citatelný kód. Casto máme na výber niekolko spôsobov ako zapísat kus programu. Zjednodušene povedané, mali by sme použit ten spôsob, ktorý je podla nás najzrozumitelnejší, a to aj vtedy, ak nie je jedným z najelegantnejších. Oceníme to neskôr, ked budeme musiet osobne, alebo niekto z našich kolegov, robit v programe nejaké zmeny.

Druhým spôsobom, prinajmenšom rovnako dôležitým, je použitie komentárov. Komentáre by nemali byt niecím navyše, co do programu vložíme napr. predtým, než ho odovzdáme ucitelovi na ohodnotenie. V každom programe, ktorý píšeme, by mali byt už od zaciatku. Zacnime teraz a komentáre nech sú neoddelitelnou súcastou nášho programátorského života.

C++ ponúka dva druhy komentárov. Ak kompilátor nájde kdekolvek v kóde znaky `//` považuje zvyšok riadku za komentár. Iným spôsobom je uzatvorenie komentárov medzi symboly `/*` a `*/`. Takýto komentár môže byt v rámci jedného riadku, alebo môže presahovat cez niekolko riadkov.

Co potrebujeme na komentovanie? Každý program by mal zacínat komentárov, v ktorom je uvedené kto a kedy program napísal, prípadne aj císlo verzie a dátum poslednej zmeny. Nasledovat by malo strucné vysvetlenie toho, co program robí. Môžeme si tým ušetrit hodiny brodenia sa v kódoch, pri hladaní konkrétneho programu. Náš program by mohol zacínat napríklad takto:

```
/*
  Zadaj a spocítaj dve celé císla, zobraz
výsledok.
  MPW 01/01/02
  Verzia 1.0 naposledy opravované 01/01/02
*/
```

Taktiež by sme mali na zaciatok každej funkcie pridat komentár, ktorý by vysvetloval co robí:

```
void __fastcall TForm1::btn_AddClick(TObject
*Sender)
// spocítaj celé císla z editacných políc ok,
```

```
zobraz výsledok
```

Obcas budeme potrebovat komentáre v kóde. V jednoduchom programe nám samotné meno premennej povie všetko potrebné. Ak by tomu tak nebolo, pridáme krátky komentár objasnujúci ich význam. Podobne aj dômyselný kus kódu potrebuje poznámku na jeho objasnenie. Budeme si pamätat aj po rokoch, preco pocítadlo zacína na císle 3? Ak nie, pridajme komentár. Spolahnime sa na vlastný úsudok pri urcovaní rozumnej miery komentovania. nasledujúci komentár je nepotrebný a len ruší program - program je tažšie a vôbec nie lahšie citatelný.

```
answer = num1 + num2;   /* spocíta num1 a num2 a
výsledok uloží do answer */
```

Doplnme do nášho kódu vhodné komentáre a potom uložme kód ako `unit_addnums.cpp` a projekt ako `proj_addnums.bpr`.

5.4 Viac výpoctov

Teraz do nášho programu pridáme tlacidlá pre odcítanie, násobenie a delenie. Pozícia tlacidla Scítaj! je (`Top = 0, Left = 200`). Tlacidlo pre odcítanie umiestnime pod neho na pozíciu (`Top = 100, Left = 200`). Umiestnime ho teraz do formulára. Priradíme mu nadpis (`Caption`) “Odcítaj!” a meno “`btn_Subtract`”.

Kód pre tlacidlo Odcítaj! bude velmi podobný kódu pre tlacidlo Scítaj!. Skopírujeme a vložíme kód z udalosti `btn_addClick` do udalosti `btn_subtractClick`. Vo výpocte zmeníme znamienko + na –. Spustime program a otestujme, ci pracuje správne s novým tlacidlom `Odcítaj!` a nadalej aj s tlacidlom `Scítaj!`.

Nezabudli sme pridat komentár pre novú udalost a zmenit komentár pre celý program?
Ak modifikujeme program, je dôležité otestovat, že aj "nezmenené" casti programu fungujú nadalej správne. Ak zapisujeme kód rešpektujúc modulárnu organizáciu, nemalo by dôjst k "vedlajším úcinkom" v iných castiach kódu, no radšej to vždy otestujme.

Teraz pridajme pod tlacidlo Odcítaj! aj tlacidlá pre násobenie a delenie, doplnme kód a komentáre a znovu otestujme program.

Ak sme písali kód v navrhovanom poradí, zistíme, že pos spustení programu sa pomocou (Tab) presúvame medzi komponentmi v poradí `edi_firstnumber`, `edi_secondnumber`, `btn_add`, `btn_subtract`, btn_multiply, btn_divide. Ak týmto komponentom skontrolujeme vlastnost `TabOrder`, zistíme, že rastie od 0 po 5 vo vyššie uvedenom poradí. `TabOrder` ktoréhokolvek z komponentov môžeme zmenit na číslo, ktoré ešte nie je použité. Ak sú v tejto postupnosti medzery, focus (kurzor) bude presunutý na komponent s najnižším z nasledujúcich čísel. Poexperimentujme a potom obnovme pôvodné poradie.

Pocas nášho testovanie by sme mali objavit aj problémy s delením. Ak delíme číslo 4 číslom 2, tak dostaneme správny výsledok 2. Ale co sa stane ak delíme číslo 5 číslom 2? Opät dostaneme výsledok 2. A je to správny výsledok! Povedali sme kompilátoru, že pracujeme s celými číslami. To znamená, že všetky desatinné casti čísel sú jednoducho zanedbané. Symbol delenia `/` je dalším z pretažených operátorov. Ak delíme dve celé čísla, výsledkom bude celé číslo; ak delíme dve reálne čísla s desatinnou castou, výsledkom bude reálne číslo. Co ak máme jedno celé a jedno reálne číslo? V tomto prípade bude celé číslo prevedené na reálne a potom budeme delit. Pri celocíselnom delení je výsledkom celocíselný podiel plus zvyšok. Teda 5 delené 2 dáva podiel 2 a zvyšok 1. Neskôr sa k tomu ešte vrátime.

Rozhodli sme sa, že ak bude nenulový zvyšok po delení, budeme chciet zobrazit aj desatinnú cast výsledku. 5 delené 2 by teda malo byt 2,5. C++ pozná niekolko typov údajov pre reálne čísla. Jeden z nich sa volá double. Zadeklarujme teda premennú answer v udalosti btn_divideClick ako `double`:

```
int num1, num2;
double answer;
```

Otestujme! Co sa stalo? preco sú výsledky aj nadalej nesprávne? Síce sme zmenili typ výsledku, ale nezmenili sme typ operandov (delenec a delitel). Stále delíme dve celé čísla, preto je aj výsledok celé číslo. 5 delené 2 je 2 a zvyšok je 1. Môžeme zmenit typ výsledku 2 na reálne číslo, no stále je to 2 alebo 2.00000, a nie z nicoho nic 2.5. Ak chceme previest "reálne" delenie, musíme mat reálne operandy. Zmenme typ `num1` and `num2` v DivideClick na `double`. Taktiež musíme zmenit konverznú funkciu pre text v editacných políckach. Namiesto funkcie `ToInt()`, budeme potrebovat funkciu `ToDouble()`. Skompilujme programa opät ho

otestujme. Teraz by už mal pracovat podla našich predstáv.

Pozrime sa späť na kód. V troch z udalostí kliknutia sme deklarovali num1 a num2 ako celé čísla, ale v `btn_divideClick` ako double (reálne čísla). Je to podstatné? Nie je, pretože sú navzájom nezávislé a viditelné na rôznych miestach. Celocíselnú premennú `num1` deklarovanú v `btn_addClick` kompilátor identifikuje ako `num1`, ktoré prislúcha do vnútra `btn_addClick`; je to nieco celkom iné než premenná v `btn_subtractClick`, ktorá je zhodou okolností tiež celocíselná a volá sa num1. Tieto dve celocíselné premenné s menom `num1` mimo svojich funkcií neexistujú, zaberajú rôzne casti pamäte a kompilátor si ich nikdy nepopletie. Obdobne, ak sa voláte John a patríte do rodiny Smithovcov, tak vás žiadna náhoda nepopletie tak, aby ste sa vybrali domov ako John z tej istej školy, ktorý patrí do rodiny Williamsovcov.

Uložme kód unitu a projekt pod vhodnými menami. Lachšie si zapamätáme co patrí ksebe ak budú mat rovnaké mená s predponami `unit_` alebo `proj_`. Budú sa líšit aj príponami `.cpp` pre kód a `.bpr` pre projekt.

5.5 Zamienanie penazí (zmenáren)

Úlohou je napísat program, ktorý bude zamienat rôzne meny.

Môžeme zamienat britské libry a euro, doláre a euro alebo iné meny podla vlastného výberu. V case písania tohto materiálu boli platné kurzy približne takéto:

£1 = €1,60
$1 = €0,93

ale môžeme použit aktuálne kurzy.

Skôr než zacneme písat kód pre náš formulár, zamyslime sa nad jeho návrhom.
Chceme zamienat libry na euro, euro na libry alebo oboje?
Ak budeme zamienat len jedným smerom, budeme potrebovat editacné políčko (edit box) pre vstup a menovku (label) pre výstup. Ale co s programom, ktorý zamiena oboma smermi? Budeme mat formulár s dvoma castami, a každá z nich bude mat editacné políčko a menovku?
Budeme mat formulár s dvomi editacnými políckami, z ktorých každé môže byt použité ako vstup alebo výstup, a dve tlacidlá?
Budem mat dalšie tlacidlo, ktoré pred novým zamienaním vymaže

predchádzajúce zamienanie?
Budeme mat len jednu sadu komponentov a pred zamienaním otestujeme, ci je jedno editacné políčko vyplnené a druhé je prázdne?

Možno si to momentálne nemyslíte, no písanie kódu je najjednoduchšou castou programovania. Tou nárocnou castou je návrh, pretože dobrý návrh na zaciatku programovania nám môže neskôr ušetrit vela sklamaní. Je velmi dôležité všetko si premysliet a navrhút ešte skôr, než zacneme písat kód. Vždy nás to láka sadnút si k pocítacu a zacat písat kód. Skúsme tomu odolat a najprv sa zamyslime nad návrhom. Z dlhodobého hladiska nám to naozaj ušetrí cas a námahu.

Z vyššie uvedených riešení je pravdepodobne najlepším to posledné, ale než ho zrealizujeme, potrebujeme si ešte osvojit niekolko programátorských techník. V riešení, ktoré teraz naprogramujeme použijeme jednu sadu editacných políčok pre údaje a tri tlacidlá pre zamienanie oboma smermi a vymazanie predchádzajúcich údajov.

Zacnime nový projekt a do formulára pridajme:
2 menovky (label)
2 editacné políčka (edit box)
3 tlacidlá (button)
Dve tlacidlá a dve editacné políčka umiestnime do horného riadku (s `Top = 0`). Vhodne zmenme ich velkost, pomenujme ich a pridajme nadpisy.
Do riadku pod nimi umiestnime tri tlacidlá.
Urobme to skôr, než si precítame nižšie uvedený návrh.

`Label1` môže mat
```
Name = lbl_pounds
Caption = Britské libry
Top = 0  Left = 0  Height = 50 Width = 50
```

`Label2` môže mat
```
Name = lbl_euros
Caption = Euro
Top = 0  Left = 200  Height = 50 Width = 50
```

`Edit1` môže mat
```
Name = edi_pounds
Text = (prázdny)
Top = 0  Left = 100  Height = 50 Width = 50
```

```
Edit2 môže mat
Name = edi_euros
Text = (prázdny)
Top = 0  Left = 300  Height = 50 Width = 50

Button1 môže mat
Name = btn_pound_to_euro
Caption = £ na €
Top = 100  Left = 0  Height = 50 Width = 50

Button2 môže mat
Name = btn_euro_to_pound
Caption = € na £
Top = 100  Left = 100  Height = 50 Width = 50

Button3 môže mat
Name = btn_clear
Caption = Vymaž!
Top = 100  Left =200  Height = 50 Width = 50
```

(Rada – pamätajme si, že je vhodné nastavit vlastnost `WordWrap` na `True`.)

Teraz potrebujeme kód pre zamienanie oboma smermi. Na tomto mieste zacneme rozlišovat medzi premennými a konštantami. Konštanta je velicina, ktorá sa nemení pocas behu programu, zatial co premenná je velicina, ktorá sa môže pocas behu programu menit. Výmenný kurz bude pocas behu nášho programu konštantou. Napíšeme to do nášho kódu. Suma, ktorú chceme zamenit sa môže pocas behu programu niekolkokrát menit. Tento údaj bude zadávat používatel. Aby sme sa uistili, že výmenný kurz nemôžeme niekde v programe omylom zmenit, zadeklarujeme ho pomocou rezervovaného slova `const` ako konštantu. Konštanty zvycajne používame vcelom programe, preto ich budeme deklarovat na zaciatku programu a nie vo vnútri funkcie. Hovoríme, že sú viditelné globálne. Cokolvek deklarované vo vnútri funkcie je viditelné lokálne - použitelné a viditelné len vo vnútri funkcie.

V okne s kódom (F11), deklarácie konštánt umiestnujeme za všetky príkazy zacínajúce znakom # a pred prvú funkciu. Nasledujúcu deklaráciu napíšeme za deklaráciu formulára (`TForm1 *Form1;`):

```
const double rate_ptoe = 1.6; /* výmenný kurz
libier na euro */
```

Taktiež potrebujeme výmenný koeficient pre prevod euro na libry. Bude našou druhou konštantou. Môžeme si ju vypocítat sami a portom zapísat do kódu. Má to však svoje nevýhody: môžeme urobit chybu; vyzerá to hlúpo, ked používame pocítac na iné výpocty a potom ideme sami pocítat; a napokon, ak sa kurz zmení, budeme musiet znovu pocítat. Dobrý program pri svojej aktualizácii vyžaduje len minimálny pocet zmien. Preto zadeklarujme druhú konštantu v závislosti od prvej:

```
const double rate_etop = 1/rate_ptoe; /* výmenný
kurz euro na libry */
```

Ked sa potom zmení výmenný kurz, stací nám zmenit náš kód len na jedinom mieste. Pridajme tento riadok s kódom po prvej deklarácii konštanty. Dalším zlatým pravidlom programovania je, že nic nemôžeme použit skôr než to zadeklarujeme. Ak by sme vložili deklaráciu `rate_etop` pred deklaráciu `rate_ptoe`, snažili by sme sa použit hodnotu `rate_ptoe` skôr než sme ju zadeklarovali a priradili sme jej hodnotu. Toto nám kompilátor nedovolí a obdržíme chybovú správu. Vyskúšajme to! je vhodné robit malé zámerné chyby, aby sme sa zoznámili s reakciami kompilátora. Ked potom neskôr dôjde k neúmyselej chybe pri kompilácii, budeme mat lepšiu predstavu, co tá chybová správa znamená. Co sa stane ak zabudneme dat na koniec riadku bodkociarku? Je to naozaj bežná chyba, no kompilátor nám len zriedkakedy otvorene povie, že problém je v chýbajúcej bodkociarke.

Ked sme už zapísali kód pre výmenný kurz, potrebujem zapísat kód pre udalosti prislúchajúce k jednotlivým tlacidlám. Budeme zadávat hodnotu do editacného políčka `edi_pounds`, klikneme na tlacidlo oznacené “£ to €” a precítame výsledok z editacného políčka `edi_euros`. Kód pre udalost má obsahovat:

1. Zoberme hodnotu z `edi_pounds` a prevedme ju na císlo.
2. Vynásobme pocet libier koeficientom pre prevod na euro.
3. Prevedme euro spät na text a dajme ho do `edi_euros`.

Na uloženie poctu libier a poctu euro budeme potrebovat v udalosti `btn_pound_to_euroClick` dve lokálne premenné. Obidve zadeklarujeme ako `double`:

```
double pounds, euros;
```

Pokúsme sa zapísat dalšie tri riadky z kódu udalosti. Skompilujme program a otestujme, ci pracuje správne. Dalšie dve tlacidlá zatial nic nerobia. Náš kód by mal vyzerat podobne ako tento:

```
double pounds, euros;
pounds = edi_pounds->Text.ToDouble();
euros = pounds * rate_ptoe;
edi_euros->Text = String(euros);
```

Teraz zapíšme kód pre udalost `btn_euro_to_poundClick`.

Na záver, zapíšme kód pre udalost `btn_clearClick`. Tu nám stací vyprázdnit dve editacné polícka. Vyskúšajme to skôr, než si pozrieme nižšie uvedené riešenie.

```
edi_euros->Text = "";
edi_pounds->Text = "";
```

Otestujme, ci program pracuje správne. Ešte môžeme spomenút niekolko vecí, ktoré by sme mohli neskôr vylepšit. Napr.: co sa stane ak klikneme na tlacidlo skôr než zadáme údaje do editacných polícok?

(Pripomienka – pridali sme do programu vhodné komentáre? Uložme unit a projekt.)

5.6 Dalšie cvicenia

1. Napíšte program na prevod teploty medzi Faradajovou a Celziovou stupnicou. Budete potrebovat tieto vzorce:

degF = degC * 9/5 + 32

degC = (degF – 32) * 5/9

Pocas testovania majte na pamäti, že voda zamrza pri teplote 0°C (32°F) a vrie pri teplote 100°C (212°F).

(Rada – budete pracovat s císlami typu `int` alebo `double`? Ak vaše výsledky nie sú správne, spomente si na predchádzajúci príklad o celocíselnom delení.)

2. Napíšte program na prevod dalších jednotiek, napr. gramov a libier, míl

a kilometrov, litrov a pínt.

3. Ceny lístkov do náucného parku sú takéto:

Deti do 3 rokov	ZADARMO
Deti od 3 – do 11	€ 10
Dospelí, 12 rocné a staršie deti	€ 15
Dôchodcovia	€ 12

Napíšte program, ktorý umožní zadat pocet osôb v každej kategórii a potom vypocítat celkovú sumu úctovanú na pokladnicnom bloku. Ceny lístkov nech sú vo vašom programe konštantami.

4. Program císlo 3 musíte prekompilovat vždy, ked sa zmenia ceny lístkov a pocas niektorých období v roku sa ceny menia takmer denne. Pozmente program tak, aby sa predvolené ceny lístkov po spustení programu objavili v editacných políckach a používatel ich mohol zmenit. (Zamyslite sa nad tým, ako to zrealizujete. Zmeníte editacnému polícku vlastnost `Text`, alebo naprogramujete výpocet po nacítaní formulára?)

5. Manažment chce vediet, kolko lístkov sa predalo z každého druhu a aj celkovú sumu penazí (tržbu). Pozmente váš program znovu tak, aby priebežne zobrazoval aktuálne celkové sumy po každom pokladnicnom bloku.

6. Znovu nacítajte táš program na jednoduché výpocty. Pridajte tlacidlá pre dalšie funkcie. Napríklad operátor % sa používa na výpocet zvyšku pri celocíselnom delení. Preto `5 / 2 = 2` a `5 % 2 = 1`. Pridajte dalšiu menovku, ktorá je viditelná len pri celocíselnom delení a je v nej zobrazený zvyšok.

7. Zrealizujte tie isté funkcie (scítanie, odcítanie a pod.) pre tri císla namiesto dvoch.

8. Zmente všetky císla na double namiesto integer.

9. Napíšte program, ktorý vyzve používatela, aby do editacného políca zapísal svoje meno. Po kliknutí na tlacidlo sa zobrazí individuálny pozdrav (s oslovením menom).

5.7 Zhrnutie

V tejto kapitole sme napísali naše prvé programy.

Zoznámili sme sa s komponentmi menovka (`label`), editacné políčko (`edit box`) a tlacidlo (`button`).

Vieme umiestnit komponenty vo formulári, zmenit ich mená (`name`) a nadpisy (`caption`).

Vieme napísat kód udalosti `OnClick`.

Vieme deklarovat globálne a lokálne premenné a konštanty.

Vieme používat koncové znaky (terminators).

Vieme používat priradovací príkaz.

Vieme prevádzat výpocty na celých číslach a na reálnych číslach.

Vieme prevádzat retazce z editacných políčok na čísla pre naše výpocty a spätne čísla na retazce pre zobrazovanie.

Vieme robit dokumentáciu k programu pomocou kometárov.

Vieme testovat naše programy.

Kapitola 6: Problémy a ich riešenie v BCB5

Ciel tejto kapitoly:
Naucit sa riešit zložitejšie problémy, používat programové konštrukcie pre vetvenie (selection) a opakovanie (iteration).

V tejto kapitole budú vysvetlené nasledujúce témy:
Riešenie jednoduchých problémov vrátane myšlienok vetvenia a opakovania;
Zápis príkazov `if, if … else` a `switch` v C++;
Zápis cyklov `while, for` a `do … while` v C++;
Retazce;
Použitie dialógových okien.

6.1 Podmienený príkaz *if*

V predchádzajúcej kapitole sme písali na prevod medzi librami a euro. Spomenme si, že sme navrhli niekolko spôsobov realizácie, vrátane nasledujúceho spôsobu:
“Budeme mat len jednu sadu komponentov a pred prevodom otestujeme ci je jedno editacné políčko vyplnené a druhé je prázdne? ...
Z vyššie uvedených riešení je pravdepodobne najlepším to posledné, ale než ho zrealizujeme, potrebujeme si ešte osvojit niekolko programátorských techník. V riešení, ktoré teraz naprogramujeme použijeme jednu sadu editacných políčok pre údaje a tri tlacidlá pre prevod oboma smermi a vymazanie predchádzajúcich údajov.”

Teraz prišiel cas, aby sme sa pozreli na riešenie. Bolo by vhodné vytvorit nový adresár pre príklady z tejto kapitoly. Prekopírujme don všetky súbory z programu na výmenu penazí. Nacítajme túto kópiu a pripomenme si, ako program pracuje. Použili sme tri tlacidlá na výmenu £ na €, € na £ a vymazanie vstupných políčok. Teraz by sme chceli, aby dve tlacidlá na zamienanie penazí splynuli do jedného tlacidla. Nazveme ho “Zamen!”.

Pri kliknutí na tlacidlo Zamen! sa dve editacné políčka môžu nachádzat v jednom zo štyroch stavov. Obe políčka môžu byt prázdne, alebo môžu obsahovat údaje. Ešte sa môže objavit jedna komplikácia - údaje nemusia byt císelné - no teraz sa tým nebudeme zatažovat.

Predpokladajme, že používatel korektne zadal údaje do jedného z dvoch

editacných polícok a potom klikol na `Zamen!`.
Udalost `OnClick` pre tlacidlo `Zamen!` by mala realizovat nasledujúce kroky:

1. zistit, ktoré editacné polícko obsahuje údaje,
2. konvertovat údaje v editacnom polícku z retazca na císlo,
3. uskutocnit zámenu penazí medzi jednotlivými menami,
4. zapísat výsledok do prázdneho editacného polícka.

Kroky 2, 3 a 4 by nemali byt problematické, nakolko sme ich už robili. Ako to bude s prvým krokom? V ktorom z editacných polícok je císlo a ktoré je prázdne? Lahšie je skontrolovat, ktoré z nich je prázdne. Najprv skontrolujeme, ci je komponent `edi_pounds` prázdny. Kód bude vyzerat takto:

```
if (edi_pounds -> Text == “”)
```

Všimnime si, že testovaná podmienka je vždy uzavretá v zátvorkách ( ). Podmienka obvykle obsahuje relacné operátory. V C++ méme k dispozícii šest relacných operátorov. Sú to:

rovná sa	==
nerovná sa	!=
väcšie ako	>
väcšie ako alebo rovné	>=
menšie ako	<
menšie ako alebo rovné	<=

Väcšina z nich je úplne samozrejmá, no budme obzvlášt pozorní pri používaní dvoch "znakov rovná sa" pre relácný operátor rovná sa. Ak na to pozabudneme a použijeme jeden znak rovná sa, nemusí nastat chyba pri kompilácii. Avšak objektu nalavo od znaku rovná sa nastavíme hodnotu napravo od znaku rovná sa a podmienka bude mat vždy hodnotu `true`. However, it will set the object on the left equal to the value on the right and Toto je sémantická (týkajúca sa významu) chyba a vôbec to nezodpovedá nášmu pôvodnému zámeru.

Kroky 2, 3 a 4 sa vykonajú ak je podmienka splnená (`true`), a tento kus kódu je uzavretý v zložených zátvorkách `{ … }`. Celé to bude vyzerat takto:

```
if (edi_pounds -> Text == “”)
{
```

```
   euros = edi_euros -> Text.ToDouble();
   pounds = euros * rate_etop;
   edi_pounds -> Text = String(pounds);
}
```

A co v prípade, ak je prázdny komponent edi_euros? Môžeme napísat další blok s podobným kódom:

```
if (edi_euros -> Text == “”)
{
pounds = edi_pounds -> Text.ToDouble();
euros = pounds * rate_ptoe;
edi_euros -> Text = String(euros);
}
```

Funguje to, ale mohli by sme to urobit efektívnejšie. Ak používatel podla inštrukcií pred kliknutím na tlacidlo zadal údaje do jedného z editacných polícok, tak je budo jedno, alebo druhé prázdne. Stací ak otestujeme jedno z editacných polícok, nepotrebujeme testovat obidve zvlášt. V C++ tomu syntakticky zodpovedá podmienený príkaz `if … else`. V tomto prípade dostávame:

```
if (edi_pounds -> Text == “”)
{
   euros = edi_euros -> Text.ToDouble();
   pounds = euros * rate_etop;
   edi_pounds -> Text = String(pounds);
}
else
{
   pounds = edi_pounds -> Text.ToDouble();
   euros = pounds * rate_ptoe;
   edi_euros -> Text = String(euros);
}
```

Teraz upravme náš program. Pridajme tlacidlo znadpisom `"Zamen!"` a udalost `OnClick` s vyššie uvedeným kódom. Ked sme si istí, že všetko funguje tak ako má, môžeme odstránit dve tlacidlá na zamienanie, ktoré už nepotrebujeme. Všimnime si, že potrebujeme z unitu (`.cpp`) a z príslušných súborov s hlavickou (`.h`) odstránit kód udalosti OnClick. K tomu budeme potrebovat otvorit súbor s hlavickou (header file).

Co sa stane, ak pri kliknutí na tlacidlo "Zamen!" obidve editacné políčka obsahujú údaje? Co sa stane, ak sú obidve prázdne? Ak nevieme odpovedat na tieto otázky, tak sme program dôkladne netestovali!
V prvom prípade, ak obidva komponenty obsahujú údaje, to co je v edi_pounds bude zamenené na euro a príslušná hodnota sa objaví v edi_euros. V druhom prípade program "spadne" (crash).

6.2 Dialógové okná

Pogram casto obsahuje kód, ktorý zabezpecí, aby pri chybe používatela program nespadol, ale aby sa objavila chybová správa. Pre chybovú správu sa zvykne používat dialógové okno. Používatel si precíta správu, klikne na tlacidlo OK a pokracuje. Dialógové okno v BCB5 je formulár.

Teraz vytvoríme dialógové okno, ktoré sa objaví v prípade, že používatel klikol na tlacidlo "Zamen!" ešte skôr, než zadal údaje pre zamienanie.

Pridajme do projektu nový formulár (File – New Form).
Zmenme jeho meno (`Name`) na `frm_NoData`, a jeho nadpis (`Caption`) na `"Chybné údaje"`.
Treba zmenit jeho velkost tak, aby sa objavil v strede hlavného okna. Môžu to byt tieto hodnoty:

```
Height = 150
Left = 300
Top = 200
Width = 300
```

Zmenme vlastnost `BorderStyle` na hodnotu `bsDialog`. (Týmto odstránime tlacidlá minimalizuj a obnov z pravého horného roku okna, aby používatel nemohol menit jeho velkost.)
Do formulára vložíme menovku (`label`) s nadpisom (`caption`), ktorý vysvetlí príslušnú chybu, napr. "Chyba - nezadali ste údaje pre zamienanie". Umiestnime menovku na vhodnú pozíciu.
Takýtéto dialógové okno má obvykle aj tlacidlo OK na jeho zatvorenie. Vhodným komponentom v BCB5 je `TbitBtn`. Pridajme jeden do formulára. Vhodná pozícia je pri dolnom okraji v strede. Zmeníme mu vlastnost `Kind` na `bkOK` (zapíšeme ju, alebo si ju vyberieme z rozbalovacieho zoznamu (dropdown list) pomocou Alt+šípka dolu). Toto preddefinované tlacidlo už obsahuje nadpis (caption) so zelenou "fajockou" (tick) a slovom “OK” a taktiež už obsahuje kód na zatvorenie formulára po kliknutí nan.
Uložme formulár ako `error.cpp` (Alt+F, S)

Teraz sa vrátme k pôvodnému formuláru, s ktorým sme pracovali. Existuje na to niekolko spôsobov, napr. (Shift+F12). Pravdepodobne sa ešte stále volá Form1. Teraz potrebujeme pridat kód na zobrazenie chybovej správy v prípade, ak sú obidve editacné políčka prázdne. Pomocou logických operátorov AND a OR môžeme kobinovat podmienky. V C++ ich zapisujeme ako && a | |.
Preto podmienku “ak je políčko libry prázdne a políčko euro je prázdne” môžeme zapísat takto:

```
if ((edi_pounds -> Text == “”) &&
    (edi_euros -> Text == “”))
```

Všimnime si dve skupiny zátvoriek. Vnútorné dvojice zátvoriek okolo každej z jednoduchých podmienok nie sú nevyhnutné, no vdaka nim je príkaz citatelnejší.

Ak táto podmienka platí, chceme aby sa objavil formulár pre chybu (dialógové okno):

```
frm_NoData->ShowModal();
```

Ak namiesto Show použijeme ShowModal, používatel bude musiet zatvorit dialógové okno predtým, než bude môct pokracovat s programom.

Môžeme vyskúšat pridat tento kód, no nerieši náš problém úplne. To co naozaj potrebujem sú dva oddelené testy, pricom druhý z nich závisí od výsledku prvého testu:

```
If je políčko pounds prázdne then
  if je políčko euro prázdne then
    daj chybovú správu
  else
    zamen euro na libry
else (políčko libry nebolo prázdne, tak)
  zamen libry na euro.
```

Hovoríme tomu vnorený podmienený príkaz (nested if statement). Vyskytuje sa pomerne casto. Kód vyzerá takto:

```
if (edi_pounds -> Text == “”)
{
```

```
  if (edi_euros -> Text == “”)
  {
      frm_NoData->ShowModal();
  }
  else
  {
   euros = edi_euros -> Text.ToDouble();
   pounds = euros * rate_etop;
   edi_pounds -> Text = String(pounds);
}
}
else
{
pounds = edi_pounds -> Text.ToDouble();
euros = pounds * rate_ptoe;
edi_euros -> Text = String(euros);
}
```

Vyskúšajme to skompilovat. Ak to urobíme, objaví sa chyba pri kompilácii. Preco? Spomenme si, co sme povedali predtým - všetko co chceme použit musí byt najprv zadeklarované. Udalost `OnClick` pre tlacidlo `"Zamen!"` vo `Form1` nic netuší o druhom formulári, ktorý sme vytvorili. Všetky informácie o druhom formulári sú v dvoch súboroch `error.cpp` a `error.h`. Volá sa súbor s hlavickou (header file). Vytvorilo ho IDE a obsahuje zoznam informácií, potrebných v prípade, že k tomuto formuláru bude pristupovat iný formulár. Potrebujeme povedat kompilátoru, aby zahrnul (`#include`) unit error do unitu currency. Prepnime sa do kódu pre formulár currency, a z hlavnej ponuky zvolme File, Include Unit Hdr (Alt+F,I). V dialógovom okne zo zoznamu vyberme súbor error.h. Potom skompilujme, spustime a ešte raz otestujme program.

6.2.1 Cvicenie

Ešte sme nevyriešili problém používatela, ktorý klikne na tlacidlo "Zamen!" a v obidvoch editacných políckach sú údaje. Pridáme dalšie chybové dialógové okno (alebo zmeníme už existujúce dialógové okno na viacúcelové), ktoré ošetrí túto situáciu?

Príklad - kontrola hesiel

Heslá potrebujeme velmi casto pri používaní pocítacov a podobných systémov ako sú napr. bankomaty. V tomto cvicení chceme, aby používatel zadal štvorcíslicový kód, a potom skontrolujeme, ci je správny. Pravidlo pre

správne kódy je: súcet prvých troch číslic vydelený siedmimi dáva zvyšok rovný štvrtej číslici.
Medzi správne kódy teda patria 1236, 9873 and 5551, medzi nesprávne kódy patria 5678, 2222 and 6543.
Algoritmus je celkom jednoduchý:

- Zadaj 4-ciferný kód;
- Klikni na tlacidlo pre kontrolu kódu;
- Skontroluj kód;
- Zobraz správu o (ne)správnosti kódu.

Potrebujeme vytvorit formulár v ktorom bude:

- Menovka (`label`) s inštrukciami;
- Komponent `edit`, v ktorom bude kód;
- Tlacidlo (`button`) na spustenie kontroly;
- Menovka alebo dialógové okno na zobrazenie výsledku kontroly. Vyskúšajme obidve!

Teraz vytvoríme formulár. V nasledujúcom texte predpokladáme, že komponent edit sa volá `edi_code` a výsledok kontroly bude zobrazovaný v menovke s menom `lbl_message`.

Na zaciatku predpokladáme, že používatel zadal číselný kód, teda že kód neobsahuje písmená ani iné znaky okrem číslic. Na kontrolu kódu potrebujeme nasledujúce kroky:

- Skonvertujme kód z komponentu edit na integer (celé čísla);
- Oddelme číslice kódu;
- Vypocítajme súcet prvých troch číslic a zvyšok pri delení súctu siedmimi;
- Skontrolujme, ci sa zvyšok rovná štvrtej číslici.

Uvažujme o štvorcifernom čísle 1234.
Prvú číslicu získame po delení číslom 1000. 1234 / 1000 = 1.
Štvrtú číslicu získame ako zvyšok po delení desiatimi. 1234 % 10 = 4.
Získanie dvoch stredných číslic je trošku obtiažnejšie.
1234 / 100 = 12 a 12 % 10 = 2. Teda (1234 / 100) % 10 = 2. Druhú číslicu získame tak, že číslo 1234 vydelíme číslom 100, a potom zoberieme zvyšok po delení výsledku číslom 10.
Tretiu číslicu získame tak, že číslo 1234 vydelíme číslom 10, a potom zoberieme zvyšok po delení výsledku opät číslom 10. 1234 / 10 = 123 a 123 % 10 = 3. Teda (1234 / 10) % 10 = 3.

Teraz môžeme zapísat kód pre udalost OnClick:

```
void __fastcall TForm1::btn_checkClick(TObject
*Sender)
{
int number, digit1, digit2, digit3, digit4,
remainder;
number = edi_code ->Text.ToInt();
digit1 = number /1000;
digit2 = (number / 100) % 10;
digit3 = (number / 10) % 10;
digit4 = number % 10;
remainder = (digit1 + digit2 + digit3) % 7;
if (digit4 == remainder)
   lbl_result -> Caption = "Kód je správny!";
else
   lbl_result -> Caption = " Kód je nesprávny!";
}
```

Dokoncme kód a spustime program. Dôkladne ho otestujme. Pozmenme kód tak, aby bol výsledok testovania v dialógovom okne a nie v menovke.

Co sa stane ak používatel zadá v kóde písmená alebo iné znaky? Program spadne. Predídeme tomu ak pred samotnými výpoctami otesujeme, ci je každý zadaný znak skutocne císlica. Použijeme na to typ String (retazec) z BCB5 a zacneme kopírovaním obsahu komponentu edit do retazca, ktorý môžeme otestovat znak po znaku. S každým pravidlom existujú aj výnimky a retazce patria k výnimkám z pravidla, že pocítanie v C++ zacína vždy od 0. Prvý znak vretazci je teda prvý a nie nultý. (Samozrejme, že je na to dobrý dôvod - kompatibilita s jazykom Pascal a prostredím Delphi, ktoré sa používajú na vytváranie mnohých odvodených tried (underlying classes) pre BCB). K znakom retazca pristupujeme pomocou hranatých zátvoriek […] a císla (resp. indexu) znaku, ktoré v nich uvádzame.

Zaciatok udalosti `OnClick` bude vyzerat takto:

```
// declarácie
String code;
int digit1, digit2, digit3, digit4, remainder;
// kopíruj kód z komponentu edit do retazca
code = edi_code -> Text;
// skontroluj ci je prvý znak císlica
```

```
if (code[1]>='0'&& code[1] <= '9')
   digit1 = code[1] - '0';
```

Posledné dva riadky vysvetlíme podrobnejšie. Potrebujeme overit, ci je prvý znak medzi 0 a 9. Overujeme však ci je medzi znakmi 0 a 9 a nie medzi číslami 0 a 9. Apostrofy (jednoduché úvodzovky vanglictine) sa v C++ používajú na oznacenie znakových konštánt alebo retazcov znakov (literal) a ich odlíšenie od císel. Prvú číslicu `digit1` teda získame tak, že zoberieme znak (je to predsa ASCII kód) a odcítame od neho ASCII kód nuly.
Tento príkaz `if` zopakujeme pre každú z troch dalších číslic v kóde.

Co budeme robit, ak ten znak nie je číslica? Potrebujeme nejaký systém oznacovania, aby sme sa neskôr mepokúšali prevádzat výpocty na neplatných údajoch a nespôsobili tak spadnutie programu. Najjednoduchším riešením bude deklarácia logickej premennej (true=pravda, false=nepravda). Spociatku môže byt nastavená na true a ak sa objaví problém, zmeníme ju na false. Celá udalost OnClick potom bude vyzerat takto:

```
void __fastcall TForm1::btn_checkClick(TObject
*Sender)
{
  String code;
  int digit1, digit2, digit3, digit4, remainder;
  bool valid = true;
  code = edi_code -> Text;
  if (code[1]>='0'&& code[1] <= '9')
      digit1 = code[1] - '0';
  else valid = false;
  if (code[2]>='0'&& code[2] <= '9')
      digit2 = code[2] - '0';
  else valid = false;
  if (code[3]>='0'&& code[3] <= '9')
      digit3 = code[3] - '0';
  else valid = false;
  if (code[4]>='0'&& code[4] <= '9')
      digit4 = code[4] - '0';
  else valid = false;
  if (valid)
  {
   remainder = (digit1 + digit2 + digit3) % 7;
```

```
   if (digit4 == remainder)
       lbl_result -> Caption = "Kód je správny!";
   else
       lbl_result -> Caption = "Kód je
nesprávny!";
  }
  else
      lbl_result -> Caption = "Kód je
nesprávny!";
}
```

Preco sme napísali kód v ktorom dva krát nastavujeme nadpis na nesprávny? Skúsme prepísat kód tak, aby sme to robili len raz. Môžeme to urobit, ale náš program bude ovela tažšie citatelný. Nadpis nastavujeme na neplatný za dvoch rôznych okolností – císelný kód, ktorý sa neriadi daným pravidlom, alebo kód obsahujúci aj iné znaky než len císlice.
Zadajme kód a otestujme ci program pracuje správne. Opät môžeme namiesto menovky použit dialógové okno.

6.2.2 Dalšie cvicenia na vetvenie

1. Napíšte program, ktorý bude prevádzat percentuálne hodnotenie na známky, napr.:

75% a viac je známka A,
66 – 74% je známka B,
pod 35% je známka F.

2. V cvicení 3 na konci predchádzajúcej kapitoly ste písali program na predaj lístkov do náucného parku. Upravte program tak, aby skupine ludí ponúkal rodinné lístky vtom prípade, ak sú lacnejšie. Rodinný lístok sa priznáva 2 dospelým a maximálne 4 detom za cenu lístka pre 2 dospelých a 2 deti. (Rada – niektoré komponenty by mali byt viditelné len ak ponúkame rodinné lístky.)

3. Opät pozmente program tak, aby ste dôchodcov mohli pri rodinných lístkoch zapocítat medzi deti.

4. Upravte program na kontrolu hesiel tak, aby boli správne tie kódy, ktoré sa skladajú zo štyroch císlic a prvá z nich je kontrolná – má sa rovnat súctu císlic 2, 3 a 4 to celé deleno 3. Správne sú teda kódy 3234 a 5674, ale kódy 4321 a 3330 sú nesprávne

6.3 Prepínac - príkaz *switch*

Príkaz `if` umožnuje vybrat jednu zdvoch alternatív. Ak máme niekolko alternatív, potrebujeme vnorené podmienené príkazy `if` a program môže vyzerat dost komplikovane. Prepínac - príkaz `switch` - ponúka iný spôsob realizácie viacnásobného vetvenia.

Napríklad ak máme dni vtýždni reprezentované ich poradovým číslom a chceme toto číslo previest na meno dna, môžeme použit viacero príkazov `if … else`, alebo citatelnejší príkaz `switch`:

```
int daynumber;
String dayname;
…
switch (daynumber)
{
  case 1: dayname = “Pondelok”;
          break;
  case 2: dayname = “Utorok”;
          break;
case 3: dayname = “Streda”;
          break;
case 4: dayname = “Štvrtok”;
          break;
case 5: dayname = “Piatok”;
          break;
case 6: dayname = “Sobota”;
          break;
case 7: dayname = “Nedela”;
          break;
    default: dayname = "neznámy";
}
```

Všimnime si, že medzi každou z možností (`case`) v príkaze `switch` máme `break;`. je to nevyhnutné preto, aby po vykonaní správneho príkazu prešlo riadenie na koniec príkazu `switch`. Bez prerušenia (`break`) by bola vykonaná nie len tá správna možnost, ale aj každá nasledujúca možnost až do konca.

Zoberme program so známkami, ktorý sme napísali vpredchádzajúcom cvicení 1. Pomocou príkazu `switch` pridajme ku každej známke vhodný komentár, napr.:

```
switch (grade)
{
  case ‘A’: lbl_grade -> Caption = “A –
vynikajúca práca!”;
  break;
…
```

Premená v príkaze switch môže byt typu integer, character, alebo akéhokolvek iného vymenovaného typu (enumerated type). Možnost `default` na konci nie je nevyhnutná, no zvykne sa používat, lebo môže pomôct zachytit nejaké chyby.

V dalšom cvicení budeme zistovat, ci je zadaný dátum platný. Používatel zadá císlo dna v mesiaci, císlo mesiaca v roku a rok. Program rozhodne, ci je dátum platný. Dátum 1 1 2001 bude teda platný, ale dátum 31 11 2002 nebude, nakolko november má len 30 dní. Dátum 13 13 2013 nebude platný, pretože rok má len 12 mesiacov, ale co napríklad dátum 29 02 2016? 29. februáru budeme venovat osobitnú pozornost, nakolko sa vyskytuje len v priestupnom roku. Pre zjednodušenie obmedzíme náš program len na dátumy od roku 1901 po rok 2099.

Pripravme formulár pre tento program a skontrolujme, ci súhlasíme s nasledujúcim návrhom.

Budeme potrebovat tri komponenty `edit` s menovkami (`label`) pre zadanie dna, mesiaca a roku. Pre výsledok budeme potrebovat menovku (`label`) alebo dialógové okno (`dialog box`) a tlacidlo (`button`) pre spustenie výpoctu.

Napíšme algoritmus - množinu krokov, ktorú má vykonat udalost `OnClick`. Mala by vykonat to, co je uvedené v nasledujúcom zozname, no nemusí to byt v tomto poradí:

1. Skontrolujme ci `edi_year` obsahuje císlice.
2. Skonvertujme císlo `vedi_year` na `integer` a skontrolujme, ci je medzi 1901 a 2001.
3. Skontrolujme ci `edi_month` obsahuje císlice.
4. Skonvertujme císlo `vedi_month` na `integer` a skontrolujme, ci je medzi 1 a 12.
5. Skontrolujme ci `edi_day` obsahuje císlice.
6. Skontrolujme, ci je císlo v `edi_day` typu `integer` a ci je medzi 1 a

31.
7. Ak je den 31, skontrolujme ci je mesiac 1,3,5,7,8,10 alebo 12.
8. Ak je den 30, skontrolujme ci mesiac nie je 2.
9. Ak je den 29 a mesiac je 2, skontrolujme ci je priestupný rok.

Teraz to už len naprogramujeme. Kroky 6, 7, 8 a 9 môžeme spolu zapísat pomocou príkazu `switch`, ktorým vymedzíme maximálny pocet dní v jednotlivých mesiacoch. V nasledujúcom kóde predpokladáme, že sme už otestovali rok a mesiac a obidve císla sú platné:

```
switch(month)
{
  case 1: case 3: case 5: case 7:
  case 8: case 10: case 12:
      maxdays = 31;
      break;
  case 2:
      if (year % 4 == 0)
          maxdays = 29;
      else
          maxdays = 28;
      break;
  default:
      maxdays = 30;
}
if ((day < 1) || (day > maxdays))
  valid = false;
```

Akonáhle sa logická hodnota `valid` zmení na `false`, tak nemá zmysel pokracovat v testovaní, pretože dátum je neplatný. Ako to zrealizujeme? Pravdepodobne použijeme niekolko vnorených príkazov `if`. Všimnime si, že ak má viac ako jedna možnost (`case`) v príkaze `switch` rovnaký kód, tak môžu byt tieto možnosti uvedené spolu, pricom nezáleží na ich poradí. Obycajne je predvolená možnost (`default`) uvedená ako posledná, ale nemusí to tak byt.
Dokoncme zápis nášho programu, uložme ho, skompilujme a otestujme.

6.4 Opakovanie

Opakovanie (iteration, repetition) je tretou zo základných riadiacich štruktúr v programovaní. Umožnuje nám raz napísat blok kódu a potom ho niekolko

krát opakovat. Existujú tri základné spôsoby zostrojenia cyklu:

- Cyklus s konecným poctom opakovaní (definite loop) sa používa vtedy, ak vopred vieme kolko krát potrebujeme kód opakovat. V mnohých jazykoch sa volá cyklus "for". Aj C++ má cyklus for, no nepracuje celkom tak, ako v iných jazykoch.
- Nekonecný cyklus (indefinite loop) sa používa vtedy, ked sa má kód opakovat pokial platí nejaká podmienka, alebo až kým nebude platit podmienka. Poznáme dva rôzne nekonecné cykly.
  - V cykle s podmienkou na zaciatku (pre-tested loop) sa podmienka "pokracovania" testuje pred vykonaním cyklu. Ak podmienka na zaciatku nie je splnená, cyklus sa vôbec nevykoná. V C++ sa tento cyklus volá cyklus “while”.
  - V cykle s podmienkou na konci (post-tested loop) sa podmienka "pokracovania" testuje až po vykonaní cyklu. Aj ked už na zaciatku podmienka nie je splnená, cyklus sa vždy vykoná aspon raz. V C++ sa tento cyklus volá cyklus “do … while”.

Ktorý z cyklov by sme mali použit? Casto na tom nezáleží, jeden z nich bude jednoducho vyzerat ako prirodzená možnost. Cyklus s podmienkou na zaciatku - cyklus while - je základným cyklom a môže byt použitý vždy. Uvedieme tri jednoduché príklady použitia troch cyklov.

Najprv chceme napísat program, ktorý bude pocítat priemer z dvoch císel. Vyskúšajme to a porovnajme naše riešenie s nižšie navrhnutým riešením.

Potrebujeme formulár s dvomi komponentmi edit pre 2 císla, dvomi menovkami (label) na popis editacných polícok, menovku pre výsledok a tlacidlo (button) pre spustenie výpoctu. Kód udalosti OnClick bude vyzerat približne takto:

```
double num1, num2, average;
num1 = edi_num1 -> Text.ToDouble();
num2 = edi_num2 -> Text.ToDouble();
average = (num1 + num2)/2.0;
lbl_answer -> Caption = String(average);
```

Predpokladajme teraz, že potrebujeme program na výpocet priemeru z desiatich císel. Môžeme pridat dalších 8 komponentov edi_num a v kóde použit 10 premenných num, no bude to dost zdlhavé. Co keby sme potrebovali priemer z 20, 200 alebo 2000 císel? V istom momente sa táto metóda stane nepoužitelnou.

Alternatívou je použitie jediného editacného políčka, vpisovanie čísel po jednom a následné klikanie na tlacidlo pre medzisúcet. Nadpis menovky môžeme pozmenit na "Zadaj n. číslo", pricom číslo n vždy aktualizujeme. Po zadaní všetkých čísel môžeme vypocítat priemer. Použijeme na to druhé tlacidlo. Týmto sme sa priblížili k myšlienke cyklu, no v skutocnosti ho nepoužívame, pretože opakovanie riadi používatel klikaním na tlacidlo pre medzisúcet a neriadi ho kód.
Tento príklad nás privedie k nasledujúcemu kódu:

```
TForm1 *Form1;
int total = 0;
int count = 0;
//---------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------
void __fastcall TForm1::btn_subtotalClick(TObject
*Sender)
{
int number;
number = edi_num -> Text.ToInt();
total += number;
count ++;
lbl_instructions -> Caption = "Zadaj" +
String(count+1) + ". číslo";
edi_num -> Text = "";
}
//---------------------------------------------
void __fastcall TForm1::btn_averageClick(TObject
*Sender)
{
float average;
average = float(total)/count;
lbl_answer -> Caption = "Priemer = " +
String(average);
}
//---------------------------------------------
void __fastcall TForm1::btn_clearClick(TObject
*Sender)
{
```

```
count = 0;
total = 0;
lbl_instructions -> Caption = "Zadaj 1. číslo";
lbl_answer -> Caption = "Priemer";
}
```

Premenné `total` (súcet) a `count` (pocet) budú používat všetky tri udalosti. Musia byt deklarované globálne, nie len v jednej z funkcií na spracovanie udalosti, aby k nim mali prístup všetky udalosti. Preto ich deklarujeme takmer na zaciatku kódu, hned za deklaráciou formulára.

Premenná `number` sa používa len na odovzdanie císla z editacného políčka a na jeho pripocítanie k súctu. Preto môže byt viditelná lokálne a budeme ju deklarovat vo funkcii na spracovanie udalosti `OnClick` pre `btn_subtotal`. V tejto funkcii uvádzame dva nové operátory, ktoré sú v C++ velmi casto pužívané.
Operátor += je skrátným zápisom a znamená: pripocítaj cokolvek z pravej strany k lavej strane a výsledok ulož späť do premennej na lavej strane.

```
total += number;
```

je len skrátený zápis pre

```
total = total + number;
```

K podobným skráteným operátorom patria `-= *= a /=`.
K bežným operátorom patrí `++` a znamená pripocítanie jednotky (increment).

```
count ++;
```

je len skrátený zápis pre

```
count = count + 1;
```

Velmi podobným operátorom je odpocítanie jednotky (decrement) `--`.
Mimochodom, jedno z vysvetlení názvu jazyka C++ hovorí, že vznikol len rozšírením jazyka C (incremental improvement on C).

Nakoniec sa pozrieme na výpocet priemeru vo fukcii na spracovanie udalosti `OnClick` pre `btn_average`.

```
float average;
average = float(total)/count;
```

Premenú average sme deklarovali ako float (môžeme použit aj double) pretože priemer z císel nemusí byt celé číslo. Z predchádzajúcej kapitoly však vieme, že pri delení celého čísla celým číslom získame opät celé číslo. Aby sme získali výsledok typu float (desatinné číslo), musíme sa uistit, že aspon jeden zcinitelov vsúcine je typu float. Toto sme urobili napísaním kódu float(total), ktorý je známy ako "pretypovanie“ (casting) a prevádza total z celého čísla na deatinné číslo.

Ak sme tak ešte neurobili, zapíšeme tento program a otestujeme ho.

Nevýhodou tohto riešenia oproti pôvodnému problému je, že pri zadávaní císel sme vždy stratili zadané číslo, aby sme mohli pokracovat v zadávaní dalšieho čísla do zoznamu. Mohli bys me ponechat všetky zadávané čísla na obrazovke a potom len kliknút na tlacidlo pre výpocet priemeru? Jedným z riešení je použitie iného komponentu na ukladanie císel – komponentu memo (viacriadkové editacné pole). Komponent memo je viacriadkový komponent edit. Čísla môžu byt zadávané do komponentu memo (jedno číslo na jeden riadok) a po kliknutí na tlacidlo sa vypocíta priemer.

Navrhnime vzhlad aplikácie pre tento nový program. Budeme potrebovat viacriadkové editacné pole (memo), menovku (label) pre inštrukcie s nadpisom (caption) "Vložte údaje, jeden na riadok", tlacidlo (button) a menovku (label) pre výsledok.

Kód udalosti OnClick je:

```
void __fastcall TForm1::btn_averageClick(TObject
*Sender)
{
int count;
int total=0;
float average;
count = mem_numbers->Lines->Count;
for (int i = 0; i < count; i++)
   total+= mem_numbers -> Lines ->
```

```
Strings[i].ToInt();
average = float(total)/count;
lbl_answer->Caption = "Priemer: " +
String(average);
}
```

Vlastnost `Lines->Count` komponentu `memo` udáva pocet riadkov s údajmi v tomto komponente. Obsah n-tého riadku je uložený v `Lines->Strings[n]`, pricom zacíname pocítat od 0. Vo funkcii sme použili cyklus `for`, pretože na zaciatku cyklu sme poznali pocet scítancov - je v premennej `count`. Samotný cyklus `for` pozostáva z dvoch riadkov:

```
for (int i = 0; i < count; i++)
   total+= mem_numbers -> Lines ->
Strings[i].ToInt();
```

Ak by malo byt v cykle for vykonaných viac než len jeden príkaz, kód by musel byt uzatvorený v zložených zátvorkách.
Riadok s for pozostáva z troch castí:

- inicializácia (initial condition), nastavenie `i = 0` (všimnime si, že je bežné deklarovat riadiacu premennú cyklu priamo v cykle; `i` je naozaj viditelné lokálne - len v dvoch riadkoch kódu);
- podmienka pokracovania (continuation condition), cyklus bude pokracovat pokial je `i` menšie ako `count`;
- prírastok (increment), `i` je zvýšené o 1 na konci každého prechodu cyklom.

V nasledujúcom riadku je retazec v i-tom riadku prevedený na celé císlo a pripocítaný k `total`.

Môžeme tu použit aj cyklus `while`. V tomto prípade bude kód vyzerat takto:

```
void __fastcall TForm1::btn_averageClick(TObject
*Sender)
{
int count;
int total=0;
int i = 0;
float average;
```

```
count = mem_numbers->Lines->Count;
while (i < count)
{
   total+= mem_numbers -> Lines ->
Strings[i].ToInt();
   i++;
}
average = float(total)/count;
lbl_answer->Caption = "Priemer:  " +
String(average);
}
```

Kód je velmi podobný, ale cyklus pozostáva zdvoch príkazov, pretože musíme výslovne zvýšit pocítadlo cyklu `i`.

Ak predpokladáme, že v zozname bude aspon jedno číslo, môžeme v tomto programe použit aj cyklus do … `while`:

```
void __fastcall TForm1::btn_averageClick(TObject
*Sender)
{
int count;
int total=0;
int i = 0;
float average;
count = mem_numbers->Lines->Count;
do
{
   total+= mem_numbers -> Lines ->
Strings[i].ToInt();
   i++;
}
while (i < count);
average = float(total)/count;
lbl_answer->Caption = "Priemer:  " +
String(average);
}
```

Skopírujme kód aspon jedného z cyklov, skompilujme a otestujme program.

Dalším príkladom využitia cyklov je prehladánie zoznamu. Pocet opakovaní

vopred nevime, pretože cyklus končí hned ako nájdeme danú položku. preto nie je vhodný cyklus s konecným poctom opakovaní (`for`) a obvykle sa používa cyklus `while`.

Pre tento program budeme potrebovat viacriadkové editacné pole (`memo`) pre zoznam mien, editacné políčko (`edit box`) pre vyhladávané meno, tlacidlo (`button`) a menovky (`label`) pre vysvetlenie významu viacriadkového editacného pola, editacných polícok a pre výsledok vyhladávania. Vyskúšajme nasledujúci kód pre udalost `OnClick`:

```
void __fastcall TForm1::btn_searchClick(TObject
*Sender)
{
String name;
int count, i=0;
bool found = false;
count = mem_numbers->Lines->Count;
name = edi_name->Text;
while (i<count && !found)
{
  if (name == mem_numbers->Lines->Strings[i])
      found = true;
  else
      i++;
}
if (found)
    lbl_answer -> Caption = "Meno sa našlo v
zozname";
else
    lbl_answer -> Caption = " Meno sa nenašlo v
zozname ";
}
```

Pozorne si precítajme kód a uistime sa, že mu rozumieme.

6.5 Cvicenia

Napíšte program, vktorom používatel bude môct zadat zoznam císel a potom urcit:

a) kolko krát sa dané císlo vyskytlo v zozname;
b) kolko císel v zozname je väcších ako dané císlo;

c) najväčšie a najmenšie číslo v zozname.

Kapitola 7: Dalšie riešenia

Ciel tejto kapitoly: Riešit dalšie problémy. Riešenia budú vychádzat z predtým používaných programovacích konštrukcií, predstavíme si dalšie užitocné a bežne používané komponenty.

V tejto kapitole si vysvetlíme nasledovné:
Riešenia problémov s použitím zoznamových rámikov (list box) a kombinovaných rámikov (combo box);
Riešenia problémov s použitím zaciarkávacích polícok (check box) a prepínacov (radio button).

7.1 Úvod s cvicením “Zoznam na nákup”

Našou úlohou je vytvorenie zoznamu na nákup, aby sme vedeli, co chceme najbližšie kúpit v supermarkete.
Najskôr sa musíme rozhodnút, co chceme vediet s našim zoznamom robit.
Zacnime si písat zoznam funkcií, ktoré by sme od takéhoto programu ocakávali.
Súhlasia vaše požiadavky s tými, ktoré v dalšom texte navrhujeme?

Pridat položky do zoznamu;
Zmazat položky zo zoznamu;
Zmenit položky v zozname;
Vyhodit zoznam a zacat nový.

Toto sú základné požiadavky aj pre mnohé ostatné aplikácie používajúce zoznam.
Pre zjednodušenie, vynecháme tretiu požiadavku, zmenenie položky v zozname. To isté totiž môžeme dosiahnut vymazaním nesprávnej položky a pridaním tej správnej.

„Luxusná“ verzia programu so zoznamom nám môže poskytnút aj dalšie možnosti, ako napr.:
abecedné zoradenie zoznamu;
spocítanie položiek zoznamu.

Zacneme so základnými požiadavkami, tie dalšie pridáme neskôr.

7.1.1 Navrhovanie formulára

Na vytvorenie základného formulára potrebujeme 5 komponentov:

3 tlacidlá (button), na pridanie a zmazanie položky a zmazanie zoznamu;
1 editacné políčko (edit box), do ktorého zapíšeme položku, ktorú chceme do zoznamu pridat;
1 zoznamový rámik (list box), ktorý bude obsahovat náš zoznam.

Položku do zoznamu pridáme tak, že do editacného políčka napíšeme jej názov a potom klikneme na tlacidlo "Pridaj!". Položku zo zoznamu zmažeme tak, že ju v zozname vyznacíme (vysvietime) a klikneme na tlacidlo "Zmaž!". Kliknutím na tlacidlo "Cistý zoznam" sa zoznam vyprázdni.

Pridajme do formulára komponenty s nasledujúcimi menami.

Komponent	**Meno (Name)**	**Nadpis/Text (Caption/Text)**
Form (Formulár)	frm_Shopping	Nakupovanie
Button (Tlacidlo)	btn_Add	Pridaj!
Button (Tlacidlo)	btn_Delete	Zmaž!
Button (Tlacidlo)	btn_Clear	Cistý zoznam
Edit box (Editacné políčko)	edi_item	
List box (Zoznamový rámik)	lst_shopping	

Pridáme menovky, aby sme formulár sprehladnili, napr. pridáme menovku, ktorou vysvetlíme, co treba zadat do editacného políčka.

Ak si dôkladne prezrieme vlastnosti zoznamového rámika, nájdeme aj vlastnost Items. Toto je vlastnost, ktorú budeme používat najcastejšie, pretože obsahuje zoznam položiek. Má svoje vlastné metódy ako sú Add (pridat), Delete (zmazat) a Count (spocítat), ktoré budeme v našom programe používat.

Najskôr napíšeme kód pre tlacidlo Pridaj! (Add).
Mal by sme zobrat text z editacného políčka a pridat ho do zoznamového rámika:

```
lst_shopping -> Items -> Add(edi_item -> Text);
```

Teraz napíšeme kód pre tlacidlo Cistý zoznam (Clear). Jednoducho použijeme metódu Clear vlastnosti Items:

```
lst_shopping -> Clear();
```

Otestujme program a presvedcíme sa, ci zatial funguje správne. Ked

klikneme na tlacidlo Zmaž! tak sa síce nic nedeje, ale dalšie dve tlacidlá by mali fungovat správne. Pripravme si zoznam vylepšení, ktoré ulahcia používanie programu. Na záver sa môžeme k tomuto zoznamu vrátit a pozriet sa, ci na nom ostali nejaké nezrealizované vylepšenia.

Naprogramujme tlacidlo Zmaž!. Ak má metóda Delete fungovat správne, musíme jej povedat, ktorá položka zo zoznamu má byt zmazaná. „Povieme“ jej to tak, že jej ako parameter odovzdávame císlo, alebo index položky.
Odovzdávanie parametrov je jeden zo spôsobov, ktorým rôzne funkcie a metódy zdielajú informácie. Parameter je uvedený v zátvorkách, za menom funkcie alebo metódy. Ak nepotrebujeme žiadne parametre, použijeme len prázdne zátvorky. Pozrime sa ešte raz na dve tlacidlá, ktoré sme už naprogramovali. Metóda Clear nemá žiadne parametre, pretože zmazat sa má všetko. Preto sme zapísali:

```
Clear();
```

Metóde Add sme museli povedat, co má pridat, takže sme tam dali parameter:

```
Add(edi_item -> Text);
```

Doteraz sme si všímali len vlastnosti, ktoré sú prístupné vo fáze vývoja a sú uvedené v okne Object Inspector. Komponenty však môžu mat aj iné vlastnosti, ktoré sú prístupné pocas behu programu, a tak nie sú uvedené v Inšpektore objektov. Jednou z nich je napr. vlastnost zoznamového rámika ItemIndex. Obsahuje index práve vyznacenej (vysvietenej) položky v zozname. Ak nie je vyznacená žiadna z položiek, obsahuje hodnotu -1. Preco? Takže, pripomenme si, v C++ sa zacína pocítat od 0, takže 0 je index prvej položky v zozname.
Kód tejto funkcie je:

```
int index;
index = lst_shopping -> ItemIndex;
lst_shopping -> Items -> Delete(index);
```

Skompilujme, spustime program a otestujeme, ci pracuje správne.

Vylepšenie riešenia

Co sa ešte nachádza na vašom zozname vylepšení?

Jedna jednoduchá, no pôsobivá crta, ktorú by sme mohli pridat, je zoradenie položiek v zozname vabecednom poradí po každom pridaní novej položky.
V Inšpektore objektov nájdeme vlastnost zoznamového rámika Sorted a nastavíme ju na true (pravda).
Znova skompilujeme a spustíme program. Co sa stane ak dvakrát pridáme tú istú položku? Prekáža to? Co by sme s tým mohli spravit?

Ked pridáme nejakú položku do zoznamu, co musíme urobit pred tým, ako pridáme dalšiu? Kde je kurzor? Porozmýšlajme o tom, kde by sme chceli mat kurzor po každej akcii. Metóda FocusControl nám umožní nastavit sa na akýkolvek komponent, ktorý si vyberieme. Dalšou akciou, po stlacení daktorého tlacidla v tomto programe, bude pravdepodobne pridávanie niecoho dalšieho do zoznamu. Takže na koniec kódu každej z troch udalostí Click by sme mohli pridat:

```
FocusControl(edi_item);
```

aby sme poslali kurzor na editacné políčko.

V našom pôvodnom zozname požiadaviek sa vyskytla aj požiadavka na spocítanie položiek v zozname. Toto umožnuje vlastnost Count vlastnosti Items.

```
numitems = lst_shopping -> Items -> Count;
```

Takisto budeme potrebovat aj menovku na zobrazenie poctu položiek. Dáme jej meno

```
lbl_count
```

a nadpis Pocet položiek: 0

K btn_ClearClick pridáme riadok na to, aby sa v menovke zobrazilo “Pocet položiek: 0”. K btn_AddClick a k btn_DeleteClick pridáme kód na spocítanie poctu položiek a následné zobrazenie v menovke:

```
lbl_count -> Caption = "Pocet položiek: " +
String(numitems);
```

(Rada – nezabudnime, že numitems budeme musiet deklarovat ako int.)

Znova skompilujme, spustime a otestujme náš program. Co sa stane, ak skúsime zadat položku bez toho, aby sme ju zapísali do editacného políčka? Alebo skúsme vymazat položku bez toho, aby sme ju vyznacili. Program bude mohutnejší, ak napíšeme kód, ktorý zachytí aj tieto dve udalosti.
V btn_AddClick použime príkaz if tak, aby sme položku pridávali len vtedy, ak editacné políčko nie je prázdne. Podobne použime príkaz if v btn_DeleteClick, aby sme skontrolovali, ci je položka pred zmazaním vyznacená. (Rada – akú hodnotu má index, ak žiadna položka nie je vyznacená?)

Teraz si môžeme skontrolovat konecný programu s nasledujúcim kódom:

```
void __fastcall
Tfrm_Shopping::btn_ClearClick(TObject *Sender)
{
   lst_shopping -> Clear();
   FocusControl(edi_item);
   lbl_count -> Caption = "Pocet položiek: 0";
}
//---------------------------------------------

void __fastcall
Tfrm_Shopping::btn_AddClick(TObject *Sender)
{
   int numitems;
   if (edi_item -> Text != "")
   {
      lst_shopping -> Items ->
 Add(edi_item -> Text);
      edi_item -> Text = "";
   }
   FocusControl(edi_item);
   numitems = lst_shopping -> Items -> Count;
   lbl_count -> Caption = "Pocet položiek: " +
String(numitems);
}
//---------------------------------------------

void __fastcall
Tfrm_Shopping::btn_DeleteClick(TObject *Sender)
{
```

```
   int index, numitems;
   index = lst_shopping -> ItemIndex;
   if (index != -1)
   lst_shopping -> Items -> Delete(index);
   FocusControl(edi_item);
   numitems = lst_shopping -> Items -> Count;
   lbl_count -> Caption = "Pocet položiek: " +
String(numitems);
}
```

7.2 Dalšie vysvetl ovanie s cvicením „Menu v reštaurácii“

V tejto úlohe chceme vytvorit menu v reštaurácii a pozvat používatela, aby si vybral jedlo.
Ako predjedlo bude vždy na výber medzi polievkou dna, ovocnou štavou alebo melónom.
Ako hlavné jedlo si bude možné vybrat medzi niekolkými jedlami, tieto sa budú denne menit.
Takisto si bude možné vybrat medzi syrom a sušienkami a/alebo kávou.
Môžeme ponúknut aj dezerty – ale to až neskôr!
Ako by sme toto mohli naprogramovat? Potrebujeme si vybrat co najvhodnejšie komponenty.

Kedže predjedlo sa dá vybrat z troch možností, dobre by nám poslúžili prepínace (radio button). Prepínace sú zvycajne navrhnuté tak, aby pracovali spolocne, takže ak je jeden zaciarknutý, ostatné sú automaticky nezaciarknuté. Avšak, ak použijeme tri oddelené prepínace, vždy budeme musiet skontrolovat všetky, aby sme zistili, ktorý z nich je zaciarknutý. Pre tento úcel je ale vhodnejšie použit komponent skupinu prepínacov (RadioGroup), ktorá, ako si môžeme všimnút aj v jej názve, zoskupuje prepínace. Potom stací, ak celú skupinu prepínacov skontrolujeme len raz a nájdeme index zaciarknutého prepínaca. Podobne ako zoznamový rámik v predchádzajúcom príklade, aj skupina prepínacov (RadioGroup) má vlastnost ItemIndex, a je hodnota je -1 ak nie je zaciarknutý ani jeden z prepínacov.

Hlavným jedlom bude jedno jedlo vybraté z viacerých možností, ktoré sa budú denne menit. Pre tento úcel by sme mohli použit zoznam, doplnat a menit ho vo fáze vývoja. Program by sme potom museli denne kompilovat, podla toho ako sa zmenilo menu, a zakaždým by sme museli vpísat nový zoznam. Ako alternatívu by sme mohli použit špecializovanejší

zoznamový rámik (list box) - kombinovaný rámik (combo box).
Syr a sušienky a káva sú nepovinné. Bude si ich možné vybrat nezávisle na iných jedlách. Tu sa výborne hodia zaciarkávacie políčka (check box).
Ked si používatel vyberie jedlo, stlací tlacidlo „Objednat!“ a jeho výber sa objaví v zoznamovom rámiku.

Zacnime teda nový projekt. Do formulára umiestnime komponenty podla nižšie uvedenej tabulky.

Komponent	**Meno (Name)**	**Nadpis/Text (Caption/Text)**
Form (Formulár)	frm_Restaurant	Reštaurácia
RadioGroup (Skupina prepínacov)	rgr_Starter	Predjedlo
ComboBox (Kombinovaný rámik)	cmb_Main	
CheckBox (Zaciarkávacie políčko)	chb_Cheese	Syr a sušienky
CheckBox (Zaciarkávacie políčko)	chb_Coffee	Káva
ListBox (Zoznamový rámik)	lst_Order	
Button (Tlacidlo)	btn_Order	Objednat!

Zacnime so skupinou prepínacov (RadioGroup). V Inšpektore objektov nájdeme k tomuto komponentu vlastnost Items, tabulátorom sa presunieme do pravého stlpca a stlacíme Ctrl+Enter. Prípadne, stací kliknút na malé políčko napravo. Otvorí sa okno String List Editor. Zadáme tri položky menu (polievka, džús, melón), každú na jeden riadok. Klikneme na tlacidlo OK, a zistíme, že vo formulári sa nám objavili tri pomenované prepínace.

Teraz urobme to isté s kombinovaným rámikom. Avšak teraz pridajme do Items položiek pre hlavné jedlo z menu, každú na osobitný riadok. Snažme sa nepoužívat dlhé opisy, lebo sa môže stat, že v poli formulára nebudú viditelné celé. Existuje viac typov kombinovaných rámikov. Ten najjednoduchší umožnuje používatelovi zadat jeho vlastný výber. Ale my chceme, aby si používatel mohol vybrat iba z nášho menu. Preto použijeme DropDownList (rozbalovací zoznam). Rozlicné typy kombinovaných rámikov ako napr. rozbalovací zoznam, môžeme nájst vo vlastnosti Style. Neskôr si s nimi môžeme poexperimentovat, teraz však vyberme rozbalovací zoznam.

So zaciarkávacími políckami (check box) a zoznamovým rámikom (list box) už netreba robit nic iné.

Kód bude vykonaný po kliknutí na tlacidlo Objednat!. Výber sa prevedie na

zoznam v zoznamovom rámiku. Aby používatel nechtiac nestlacil tlacidlo "Objednat!" dva krát, môžeme ho zmenit na nedostupné (disabled).

Prvou položkou, ktorá sa skopíruje do zoznamu je predjedlo. Skontrolujme, ci bola vybratá nejaká položka (ItemIndex sa nerovná -1), a potom tú položku pridajme do zoznamu. Spomenme si, že pôvodne sme do vlastnosti Items zadali zoznam retazcov. Teraz by sme teda mali znova dostat správny retazec z Items. Bude to teda String[index], kde index je císlo alebo Itemindex vybratej položky.

Kód teda vyzerá nasledovne:

```
int index;
index = rgr_Starter -> ItemIndex;
if (index >= 0)
{
   lst_Order -> Items -> Add(rgr_Starter -> Items
-> Strings[index]);
}
```

Dalej musíme skopírovat výber hlavného jedla. Kód bude vyzerat velmi podobne. Všimnime si, že už pre skupinu prepínacov nepoužívame premennú index, preto ju môžeme použit znova aj pre kombinovaný rámik. Nie je potrebné deklarovat dalšiu premennú pre index položky, vybratej z kombinovaného rámika.

```
index = cmb_Main -> ItemIndex;
if (index >= 0)
{
   lst_Order -> Items -> Add(cmb_Main -> Items ->
Strings[index]);
}
```

Ak si zákazník vybral syr a sušienky, je potrebné pridat do zoznamu aj tie:

```
if (chk_Cheese -> Checked)
{
   lst_Order -> Items -> Add("Biscuits and
Cheese");
```

Vlastnost zaciarkávacieho políčka Checked môže nadobúdat dve hodnoty – pravda (True) alebo nepravda (False). Ak políčko bolo zaciarknuté, jeho

hodnota bude true, a preto aj podmienka v okrúhlych zátvorkách bude true a položka sa pridá do zoznamu.

Pridajte podobný kód aj pre kávu.

Nakoniec chceme, aby bolo tlacidlo nedostupné. Aby sme to dosiahli, musíme nastavit hodnotu jeho vlastnosti Enabled na false:

```
btn_Order -> Enabled = false;
```

Teraz skompilujme, spustime a otestujeme program. Ak máte nejaké problémy pri ladení (debugging), celý kód pre udalost Click je uvedený nižšie:

```
void __fastcall
Tfrm_Restaurant::btn_OrderClick(TObject *Sender)
{
   int index;
   index = rgr_Starter -> ItemIndex;
   if (index >= 0)
   {
      lst_Order -> Items -> Add(rgr_Starter ->
Items -> Strings[index]);
   }
   index = cmb_Main -> ItemIndex;
   if (index >= 0)
   {
      lst_Order -> Items -> Add(cmb_Main -> Items
-> Strings[index]);
    }
    if (chk_Cheese -> Checked)
    {
      lst_Order -> Items -> Add("Biscuits and
Cheese");
    }
    if (chk_Coffee -> Checked)
    {
      lst_Order -> Items -> Add("Coffee");
    }
    btn_Order -> Enabled = false;
}
```

7.3 Cvicenia

A co takto nejaký dezert? Rozhodnite sa, co chcete ponúknut a navrhnite k tomu vhodný komponent. Pridajte výber dezertu do zoznamu.

Poexperimentujte si s rozlicnými typmi kombinovaných rámikov.

Ponúknite na výber aj nápoje po jedle, ci víno alebo nealkoholické nápoje k jedlu.

Alebo nejakú zeleninu? Alebo radšej šalát? Využite svoju predstavivost a experimentujte!

Kapitola 8: Distribúcia programu pomocou InstallShield

Ciel tejto kapitoly:
Naucit sa pripravit program na distribúciu pomocou funkcie prostredia Borland C++ Builder "Install Shield" a to spôsobom, ktorý je bežný pri akomkolvek komercnom programe.

8.1 Úvod

Dalším krokom po naplánovaní projektu, napísaní, ladení a kompilovaní BCB je distribúcia výsledného produktu.
Ako sme sa už mali možnost presvedcit, prostredníctvom BCB môžeme vytvárat spustitelné súbory **(.exe)**, ktoré bude možné spustit na ktoromkolvek pocítaci. Toto platí najmä pre jednoduché produkty.
Avšak programátor amatér sa casto stretne s komplexnými aplikáciami, ktoré vyžadujú štandardizované procedúry inštalátora. Inými slovami, potrebujeme nainštalovat konfiguracné súbory **(.INI)**, súbory registrov **(.reg)**, dynamické knižnice **(.dll)** atd; koncový používatel rozhoduje, do ktorého priecinka chce program nainštalovat, aký druh inštalácie zvolí (typickú, minimálnu alebo vlastnú), ci chce zástupcu programu umiestnit do štartovacej ponuky, ci akceptuje podmienky licencnej zmluvy atd.

Preto profesionálna verzia BCB obsahuje **InstallShield Express - Borland Limited Edition**. Installshield Express nie je automaticky nainštalovaný spolu s BCB, preto sa musíme presvedcit, ci nie je potrebné ho doinštalovat. Ide o jednoduchšiu verziu softvérového balíku urceného pre programátorov v BCB na distribúciu aplikácií. Pomocou tejto verzie **InstallShield** je možné pripravit hlavné casti projektu inštalátora. Opät sa stretávame s projektom: ak zvolíme **File\New**, budeme musiet zadat cestu a názov pre nový projekt.
Rada: odporúcame vytvorit samostatný priecinok pre každý projekt.

Stlacíme **Ok** a objaví sa okno podobné oknu **Tento pocítac** (My Computer): nalavo strom, ktorý obsahuje niekolko skupín krokov na vytvorenie inštalátora; napravo položky každej zo skupín; viac napravo parametre zvolenej položky a dalej okno pomocníka. Množstvo a umiestnenie rámcov, spôsob výberu hodnôt (editacné pole, výberové pole atd.) sa líši pri každej z položiek.
Pohyb medzi rámcami je podobný ako pohyb v okne **Tento pocítac**. S použitím spomínaných prostriedkov, postupovaním krok za krokom cez dostupné možnosti vytvoríme inštalátor s potrebnými vlastnostami na

distribúciu BCB aplikácie.
Niektoré rámce môžu byt pre cítace obrazovky ciastocne alebo úplne nedostupné.
Teraz sa budeme zaoberat jednotlivými krokmi smerujúcimi k vytvoreniu projektu inštalátora.

8.2 Usporiadanie inštalátora

Prvý krok pri vytváraní inštalátora je zosumarizovanie základných informácií o našej aplikácii a spolocnosti. Položky v tomto kroku umožnujú zadat základné informácie o projekte, definovat logické skupiny, ktoré budú eventuálne uchovávat všetky údaje našej aplikácie a urcit možnosti, dostupné koncovému používatelovi pri inštalácii našej aplikácie.
Základné delenie a vlastnosti, ktoré si definujeme v prvých krokoch bude prítomné až do konca procesu vytvárania inštalátora. Preto je dôležité jeho príprave venovat náležité množstvo casu, aby sme získali dobrý základ, na ktorom môžeme stavat.

8.2.1 Všeobecné informácie (General information view)

V zobrazení "General information view" nastavujeme všeobecné vlastnosti projektu, ako názov aplikácie a informácie o podpore. Zadanie väcšiny vlastností nie je povinné. Inštalátor bude dobre fungovat aj vtedy, ak akceptujeme predvolené hodnoty, resp. ich vôbec nezadáme. Zadanie nasledujúcich hodnôt je však povinné: Subject (predmet), Product Name (názov produktu), Publisher (výrobca produktu), Product Code (kód produktu), Upgrade Code (kód pre aktualizáciu), a Destination Folder (cielový priecinok).

8.2.2 Inštalované súcasti aplikácie (Features)

Features sú stavebnými kamenmi inštalátora. Pre koncového používatela predstavujú dôležitú cast aplikácie – ako napr. programové súbory, súbory pomocníka, clip art. Features obsahujú komponenty (components), ktoré zasa obsahujú súbory. V zobrazení "Features view" môžeme vytvorit features a subfeatures až do hlbky 15 úrovní.

8.2.3 Druhy inštalácie (Setup Types)

Druhy inštalácie ponúkajú pre koncového používatela rôzne zostavy inštalovaných aplikácií. Zostavy môžu byt užitocné, ak chceme s aplikáciou distribuovat dalšie prídavné funkcie, ktoré nie sú nevyhnutné pre

fungovanie samotnej aplikácie.

8.2.4 Spôsoby aktualizácie (Upgrade Paths)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.3 Urcenie údajov aplikácie

Potom, ako sme vytvorili základnú štruktúru aplikácie, vyplníme ju príslušným obsahom (údajmi). InstallShield obsahuje niekolko druhov zobrazenia slúžiacich tomuto úcelu. Súbory môžeme pridat priamo do preddefinovaných cielových priecinkov alebo podpriecinkov týchto priecinkov a následne vybrat feature, ku ktorej majú byt priradené. Ušetríme cas, ak do projektu pridáme merge modules – pripravené balícky casto používaných súborov aplikácií.

8.3.1 Súbory (Files)

V zobrazení "Files view" môžeme vybrat súbory aplikácie z ich umiestnenia v našom systéme a zadat umiestnenie v systéme koncového používatela, kde ich chceme nainštalovat. Rozhranie je rozdelené na dve casti: v hornej casti je zobrazený náš systém, v dolnej casti je niekolko preddefinovaných cielových priecinkov.
Toto rozhranie funguje presne ako Windows Explorer (Prieskumník), takže súbory môžeme z hornej casti do spodnej prenášat spôsobom tahaj a pusti (drag and drop). Súbory budú priradené tej feature, ktorá je práve vybraná v rozbalovacom zozname. Rozbalovací zoznam sa nachádza nad rozhraním. Priradenie môžeme kedykolvek zmenit výberom inej feature z rozbalovacieho zoznamu.

8.3.2 Súbory a inštalované súcasti aplikácie (Files And Features)

Zobrazenie "Files and Features view" nám umožnuje upravit priradenia súbor – feature. Napríklad predpokladajme, že sme presunuli všetky súbory aplikácie do cielových priecinkov bez toho, aby sme zmenili feature. V zobrazení "Files And Features view" môžeme presunút súbory z jednej feature do druhej (pomocou tahaj a pusti (drag and drop)) bez toho, aby sme ovplyvnili cielový priecinok. Ak potrebujeme nejaký súbor nainštalovat vo dvoch features, môžeme ho kopírovat z jednej feature a prilepit ho do druhej. Takýto súbor si zachová všetky svoje vlastnosti vnútri InstallShield. Dalšie informácie nájdeme v Associating Files with Features.

8.3.3 Objekty a pripájané moduly (Objects/Merge Modules)

V zobrazení "Objects/Merge Modules view" môžeme do inštalátora pridávat všeobecné prvky zabezpecujúce funkcnost – tzv. merge modules. Tieto prvky sú potrebné, ak aplikácia využíva všeobecné knižnice, ako napríklad MFC library. Takto nebudeme musiet tieto súbory zhromaždit a rucne pridávat vždy, ked budeme pripravovat na distribúciu program, ktorý takéto knižnice používa. Stací, ak vyberieme príslušný modul a potrebné súbory budú automaticky nainštalované do správneho priecinka.
V predchádzajúcich verziách InstallShield poskytovali objects podobnú funkciu ako terajšie merge modules. Vela spolocností, vrátane Microsoftu, vytvorilo mnoho všeobecných merge modules na zabezpecenie základných funkcií. Je to napríklad modul so súbormi Visual Basic run-time. Tieto moduly nám poskytnú takmer úplne bezpecnú metódu zapracovania technológie inej firmy do nášho inštalátora.

8.3.4 Závislosti (Dependencies)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.4 Konfigurácia ciel ového systému

S aplikáciami sú casto asociovaní zástupcovia, zápisy do databázy registry a INI súbory. V tomto kroku môžeme takúto rozcírené nastavenia aplikácie vykonat velmi rýchlo a jednoducho. V predchádzajúcich verziách InstallShield nebola takáto konfigurácia vždy priamo podporovaná. Teraz však môžeme vytvárat zástupcov aj vo viacerých rôznych umiestneniach cielového systému, editovat zápisy v databáze registry a v INI súboroch, to všetko v rámci IDE.

8.4.1 Zástupcovia a priecinky (Shortcuts/Folders View)

Zobrazenie "Shortcuts/Folders view" nám poskytuje nástroje na umiestnenie zástupcov súborov aplikácie vcielovom systéme. Niekolko základných a casto používaných predvolených priecinkov tu už nájdeme. Sú to Desktop (Pracovná plocha), ponuky Programs (Programy) a Send To (Odoslat kam). V týchto priecinkoch môžeme vytvárat zástupcov alebo dalšie podpriecinky.
V zobrazení "Shortcuts/Folders view" môžeme dalej urcit súbor s ikonami pre zástupcov, odovzdanie parametrov z príkazového riadku aplikácii a kombináciu horúcich klávesov, ktoré chceme používat. Dalšie informácie nájdeme v Shortcuts.

8.4.2 Registre (Registry)

Zobrazenie "Registry view" nám poskytuje možnost zostavit zoznam zmien, ktoré chceme vykonat v databáze registry v cielovom systéme. Ak máme na v systéme záznamy v databáze registry usporiadané tak, ako ich chceme mat usporiadané aj v databáze registry v cielovom systéme, môžeme ich presunút pomocou tahaj a pusti (drag and drop - podobne ako sme to robili so súbormi vzobrazení "Files view"). Záznamy môžu byt rovnako pripravené aj v REG súboroch, ktoré môžeme v tomto zobrazení importovat. Záznamy je samozrejme vždy možné editovat rucne a to aj pri použití vyššie spomínaných metód.

8.4.3 ODBC prostriedky (Resources)

V zobrazení "ODBC Resources view" môžeme do inštalátora zahrnút akýkolvek ODBC driver (ovládac) alebo translator (prekladac), ktorý sa nachádza v našom systéme. Ak chceme pridat ovládac alebo prekladac:
- v zozname nájdeme prostriedok, ktorý chceme pridat;
- vyberieme prostriedok a feature, s ktorou sa má tento ODBC prostriedok nainštalovat;
- pod zoznamom ODBC prostriedkov sa nachádzajú vlastnosti tohoto súboru, ktoré reflektujú jeho nastavenia v našom systéme. Ktorúkolvek hodnotu môžeme zmenit, rovnako môžeme pridat nové nastavenie.

Naviac v zobrazení "ODBC Resources view" môžeme pridat ODBC prostriedok do inštalátora
ODBC prostriedky môžeme pridávat okrem zobrazenia "ODBC resources view aj v zobrazeniach "Setup Design view" a "Components view". V týchto zobrazeniach sa s ODBC prostriedkami pracuje v režime rozšírených nastavení komponentu (advanced settings).

8.4.4 Prípony súborov (File Extension)

Vo File Extensions môžeme spojit istý typ súboru s našou aplikáciou. Ak dva krát klikneme na takýto súbor, spustí sa naša aplikácia a otvorí tento súbor. Ked otvárame textový súbor (.txt), tak vlastne posielame operacnému systému správu - hovoríme mu, aby otvorik Poznámkový blok (Notepad), aby sme si mohli pozriet obsah textového súboru. Ak by sme chceli dosiahnut podobnú funkcnost s našou aplikáciou a jej súbormi, môžeme vytvorit asociáciu s príponou súboru.

8.4.5 Premenné prostredia (Environment Variables)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.5 Prispôsobenie vzhl adu inštalátora

Akonáhle sme skoncili s usporadúvaním údajov aplikácie, môžeme sa zamerat na používatelské rozhranie nášho inštalátora. To, co zákazník vidí na obrazovke pocas inštalácie má velký vplyv na formovanie jeho vztahu k produktu a k spolocnosti. Inštalátor je casto prvou skúsenostou s produktom, ktorú používatel má. Ak inštalátor nevyzerá profesionálne, alebo je zložitý na ovládanie, prvá skúsenost zákazníka môže byt zlá.
Tento krok ponúka možnost prispôsobit všetky vizuálne stránky inštalátora tak, aby sa nám už pocas inštalácie podarilo zanechat dobrý dojem.

8.5.1 Dialógové okná (Dialogs)

V zobrazení "Dialogs view" môžeme urcit, ktoré zo štandardizovaných okien budú pocas inštalácie prezentované koncovému používatelovi. Niektoré z týchto dialógových okien sú pre inštalátor nevyhnutné a nie je ich možné vylúcit. V takom prípade bude zaškrtávacie políčko okna nedostupné. Pri ostatných oknách bude možné toto políčko zaciarknut alebo nechat nezaciarknuté.
Každému dialógovému oknu, ktoré v inštalátore použijeme, môžeme nastavit vlastnosti pomocou hárku s vlastnostami (property sheet) vpravo. Patrí k nim aj banner (transparent), ktorý je zobrazený nad každým dialógovým oknom a na ktorý môžeme umiestnit názov spolocnosti. Pod zoznamom dialógov sa nachádza ukážka toho, ako bude dialógové okno prezentované používatelovi. Pozor, táto vzorka však nebude reflektovat hodnoty zadané vo vlastnostiach jednotlivých dialógových okien.

8.5.2 Nástenky (Billboards)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.5.3 Text a správy (Text and Messages)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.6 Definícia po□iadaviek pre inštaláciu a prácu s aplikáciou

Naša aplikácia a jej inštalátor môžu mat urcité minimálne požiadavky. Tento krok nám umožnuje nastavit požiadavky, ktoré musí cielový systém

splnit a zvýšit funkcnost prostredníctvom dodatocných úkonov, pokial Windows Installer service nesplna požiadavky nášho inštalátora.

8.6.1 Požiadavky na cielový systém (Requirements)

Cielový systém musí byt schopný poskytnút potrebný výkon procesora, možnosti zobrazovacieho adaptéra a dostatocné množstvo pamäte na to, aby aplikácia mohla bez problémov fungovat. V zobrazení "Requirements view" môžeme teda vymedzit, aké požiadavky musí cielový systém splnit skôr, ako bude samotný produkt nainštalovaný. Ak cielový systém požiadavky, ktoré tu vymedzíme nesplní, používatelovi sa zobrazí chybové hlásenie a inštalátor skoncí.

8.6.2 Vlastné funkcie (Custom Actions)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.6.3 Podporné súbory (Support Files)

Táto funkcia je dostupná iba v plnej verzii InstallShield Express.

8.7 Príprava inštalátora (Prepare For Release)

Nakoniec musíme pripravit inštalátor na uvolnenie (release) softvéru. Budeme ho musiet skompilovat do funkcného inštalacného súboru, testovat tento súbor a presunút ho na cielové médium. Tento krok nám priblíži spomínané úlohy bez toho, aby sme museli opustit IDE.

8.7.1 Vytvorenie inštalátora (Build Your Release)

Vytvorením inštalátora završime všetku prácu, ktorú sme pri tvorbe tohto projektu vynaložili. To zahrna všetko programovanie a plánovanie, ktoré sme urobili ešte pred tým, ako sme vôbec rozmýšlali nad vytvorením inštalátora. Zobrazenie "Build Your Release view" nám umožní transformovat všetky informácie, ktoré sme do projektu inštalátora vložili, na funkcný inštalacný program pre Windows. Môžeme si tu zvolit jedno z množstva štandardných formátov médií, ako napríklad CD-ROM alebo rôzne typy DVD-ROM. Inštalátor môžeme transformovat aj priamo na disk image alebo na súbory so zvolenou velkostou pre naše disky.
Akonáhle sme si vybrali a nakonfigurovali typ média, môžeme vytvorit inštalátor. InstallShield nás neskôr bude informovat prostredníctvom protokolov (build logs) a správ (reports). Tieto sú jedinecné pre každé

vytvorenie inštalátora a preto nebudú prepísané neskoršími, ani ak pôjde o ten istý typ média.

8.7.2 Testovanie inštalátora (Test Your Release)

V zobrazení "Test Your Release view" môžeme náš novovytvorený inštalátor otestovat aby sme sa uistili, že všetko pôjde tak, ako má. Máme dve možnosti: run (spustit) alebo test (testovat) setup. Ak zvolíme run, inštalátor sa bude správat tak, ako by sme dva krát klikli na spustitelný súbor inštalátora. Budú nainštalované všetky súbory, zobrazené všetky dialógové okná a budú vykonané všetky dodatocné úkony. Ak zvolíme test, na cielovom systéme nebudú vykonané žiadne zmeny; bude zobrazené iba používatelské rozhranie a nebudú vykonané žiadne dodatocné úkony. Druhá spomínaná možnost je užitocná vtedy, ak chceme vidiet, ako inštalátor vyzerá bez toho, aby nám inštalácia zaberala miesto na disku.

8.7.3 Distribúcia aplikácie (Distribute Your Release)

Len co sme vytvorili inštalátor a presvedcili sa, že všetko funguje tak, ako sme chceli, môžeme v zobrazení "Distribute view" premiestnit našu aplikáciu na cielové médium. Môžeme ju kopírovat priamo na príslušné médium, alebo urcit priecinok, prípadne kopírovat ju priamo na FTP server aby bola dostupná z iných miest.

Kapitola 9: Riadenie projektu

Ciel tejto kapitoly:
Zoznámit sa s riadením väcších programovacích projektov.

V tejto kapitole si rozcleníme programovací projekt na rôzne úlohy. Sú to: formulácia problému, formulácia úloh, ich analýza, písanie kódu a testovanie. Okrem toho budeme hovorit aj o dokumentácii programovacieho projektu.

9.1 Úvod

Existuje niekolko možných spôsobov na riešenie urcitých problémov: osoba, ktorá problém odhalí, ho vyrieši samostatne. Toto sa podarí iba ak má osoba dostatocné zrucnosti, cas a najmä motiváciu. Ked problém dosiahne istý rozsah a komplexnost, osoba zistí, že na jeho riešenie potrebuje pomoc. V tomto prípade sú k dispozícii jeden, dvaja alebo aj viacerí zamestnanci a pokúšajú sa tak problém vyriešit spolocne. Mali by sme poznamenat, že nárocnost dosahovania spolocného riešenia vzrastá s velkostou skupiny. Nemali by sme napríklad zanedbat ani náklady na komunikáciu v rámci velkej skupiny. Bez organizovanej štruktúry môže dosahovanie spolocného riešenia lahko narazit na znacné tažkosti. Aby sme tomuto predišli, môžeme použit nástroje na riadenie projektu (project management) rozvíjané v odbore manažment.

Projektom tu myslíme znacne komplexný a jedinecný problém (ako napríklad výskumný projekt alebo projekt v krízovej situácii), ktorý sa má vyriešit v rámci urcitého casu a s danými prostriedkami.

9.2 Fázy vývoja projektu

Najdôležitejšími krokmi vo vývoji projektu sú v skutocnosti plánovanie a príprava, pretože úspešná realizácia projektu je možná len v prípade, ak týmto krokom venujeme dostatok casu a námahy.
V zásade rozlišujeme nasledujúce fázy:

Problém
Formulovanie problému
Metóda
    Hrubá analýza

Špecifikácia požiadaviek
    Podrobná analýza
Špecifikácia
    Programovanie
Program
    Testovanie
    Naplánovanie riadenia
Upravený program
    Naplánovanie riadenia
    Prevzatie údajov (assupmtion)
    Paralelná cinnost (action)
Pocítacová aplikácia

Prvou úlohou v celom projekte musí byt príprava kostry metód vrámci projektu. Sú to:

Opis projektu.
Špecifikácia práce: kvalitatívne a kvantitatívne vymedzenie želaného výsledku.
Zadefinovanie práce: prehladné opísanie prostriedkov, ktoré sa majú použit.
Popis postupu práce: druh, rozsah a výsledok potrebných krokov.
Termíny (najlepšie aj s informáciami o priebežných stretnutiach).
Rozpocet.
Priradenie projektov urcitým osobám.

9.3 Informácie a dokumentácia

Jednou z najdôležitejších pomôcok plánovania a kontroly projektu je informacný a dokumentacný systém. Kontrolou sa nemyslí pobádanie pracovníkov k práci. Kontrola sa považuje za prostriedok slúžiaci na odhalenie možných problémov ci casového oneskorenia tak, aby ešte bolo možné vykonat potrebné protiopatrenia. Na tento úcel je nevyhnutné sprístupnit pokyny, výsledky a ciastkové úlohy všetkým úcastníkom; najlepšie vytlacené. Toto sa deje aj pocas prípravy špecifikácie softvéru.

9.4 Špecifikácia softvéru

Špecifikácia softvéru zahrna fixné stanovenie bodov, ktoré by mal vytvorený program splnat.
Predstavuje zoznam všetkých spracúvaných údajov a predpokladov, na základe ktorých môžeme na konci projektu hodnotit jeho úspešnost.

Okrem toho sú v nej zahrnuté informácie o zodpovednosti, termíny, náklady a všetky ostávajúce vymedzenia. Každá nevyhnutná zmena špecifikácie softvéru musí byt dokumentovaná a schválená odberateIom. Napríklad špecifikácia dodania by mohla okrem iného obsahovat nasledujúce body:

Casti špecifikácie softvéru:

1. Všeobecný popis ciela
2. Samotný popis systému
   zostavenie a funkcia
   technické požiadavky (softvérové a hardvérové)
   Skúsenosti
3. Úloha následného systému
   Popis pracovného postupu
   Vstup (input) / výstup (output) údajov
   Spracovanie údajov
4. Súbory
   Typ súboru
   Vstupný (input) / výstupný (output) štandard
   Programovanie
   Editor
   Compilátor / Linker
   Pomôcka na krokovanie
   Predkladanie rozdielnych riešení
   Obmedzujúce podmienky
   Riešenie A.: výhody a nevýhody
   Riešenie B.: výhody a nevýhody
5. Funkcnost a popis
   Prostredie
   Hardvér a softvér
6. Rozsah objednávky a jej dorucenie
   Množstvo
   Termíny
7. Spracovanie projektu
   Dátum
   Miesto
8. Servisná kontrola
9. Rozsah služieb
   Záruka systému
   Rozsah dodávky a podmienky
   Dokumentácia
   Školenie

10. Hodnotenie
11. Odskúšanie a podmienky prevzatia

9.5 Dokumentácia softvéru

Zvláštnym problémom pocas prípravy softvéru je jeho dokumentácia. Niekto však môže tvrdit, že zdrojový text každého programu je zdokumentovaný a tak dalšie informácie nie sú potrebné. Každý, kto už niekedy odskúšal program napísaný niekým iným, vie aké to môže byt tažké.

Cím je projekt rozsiahlejší, tým bude pravdepodobnejšie, že sa použije komplexnejší softvér. Preto sa na príprave zúcastnuje niekolko programátorov.
Pri rozsiahlych programoch alebo softvérových balíkoch je dokumentácia nevyhnutná nielen na konci projektu, ale tiež aj pocas prípravy, aby sa mohol celý program bez problémov pospájat.
Aby každý programátor vedel, co presne musí robit, je nevyhnutné definovat všeobecné štandardy dokumentácie ešte pred rozdelením programátorskej práce.
Pri písaní softvérovej dokumentácie (pokial možno do špeciálnych formulárov) by sme mali mat na zreteli nasledujúce body:

Casti softvérovej dokumentácie
Názov projektu
Názov programu / císlo modulu
Úcel programu
Príprava / zmena (urcenie referencného programu)
Vyhotovil
Dátum
Nevyhnutný hardvér a systémový softvér
Štruktúry modulu
Rozhrania modulu (vstupy, výstupy)
Použité algoritmy a postupy
Obmedzenie využitia ako aj iné ukazovatele "špecifických vlastností".
Správy o priebehu testovania.

Sumár dokumentácií k jednotlivým modulov predstavuje po ukoncení vývoja softvéru celú dokumentáciu.
Ak dodržiavame túto procedúru, je možné zabezpecit, že všetky stránky projektu sú skompletizované nacas a podla požadovaných štandardov. Toto je dôležité najmä preto, lebo na konci projektu zvykne byt nedostatok

casu. Dobre naplánovaná dokumentácia taktiež ulahcuje údržbu a aktualizáciu.

Kapitola 10: Smerníky (pointers)

Ciel tejto kapitoly: Zoznámit sa so smerníkmi, ich definíciou a použitím.

V tejto kapitole sa naucíme priamo pristupovat do pamäte RAM cez smerník. Naucíme sa, ako pristupovat do urcitej casti pocítacovej pamäte, ako z nej cítat a zapisovat do nej údaje.

10.1 Definícia

Smerníky sú premenné, ktoré obsahujú adresu inej premennej ("ukazujú na inú premennú").
Príkazom

```
char *ptr;
```

deklarujeme smerník, ktorý ukazuje na premennú typu char.

So smerníkmi sú dovolené nasledujúce operácie:
a) priradenie adresy (assignment),
b) pristupovanie k hodnote premennej pomocou adresy (dereferencing),
c) porovnávanie (equality, disparity).

Hoci premenná typu smerník vždy obsahuje adresu, musí byt deklarovaný typ údajov, na ktoré ukazuje. Toto zarucuje, že napr. po inkrementácii smerník naozaj ukazuje na další element (pozn. prekladatela: posunie sa v pamäti o dlžku premennej).

Poznámka:
V aritmetike smerníkov sa nepocíta po bytoch, ale po blokoch. Teda `ptr+1` vždy ukazuje na další element, nie na nasledujúcu adresu.

Príklady:

```
// smerník typu integer
int *int_Zahl;
// smerník typu edit box
TEdit *edi_myEditBox;
```

10.2 Adresa premennej (referencing)

Adresa pamätovej bunky sa nazýva "reference". K tomu prislúchajúci operátor "&" vráti adresu premennej.
Príklad:

```
reference (&v);
```

Funkcii **reference** tento operátor odovzdá adresu pamätovej bunky s menom v.

10.3 Pristupovanie k hodnote premennej pomocou adresy (dereferencing)

Pristupovaniu k hodnote premennej pomocou adresy hovoríme "dereferencing". Obsah pamätovej bunky (buniek) získame, ak použijeme operáciu "dereferencing" pre premennú. Interpretácia tejto pamätovej bunky (buniek) závisí od typu údajov.

Operátor "dereferencing" "*" vráti obsah pamäte z adresy.

Príklad:

```
name = *ptr;
```

Premennej "name" odovzdá obsah pamätovej bunky (buniek), na ktorú ukazuje ptr.

10.4 Smerníky NULL a Void

NULL je smerník, ktorý prakticky neukazuje nikam. Momentálne neobsahuje adresu platných údajov.

Príklad:

```
float *fp = NULL;
```

Globálne premenné typu smerník sú štandardne inicializované na hodnotu NULL, lokálne premenné obsahujú nedefinovanú hodnotu.

Smerník typu **void** môže adresovat lubovolnú pamätovú bunku. Nie je

zviazaný so žiadnym typom. Smerníky typu void sú nástroje na urcovanie údajov, ktorých typ momentálne nie je známy.

Príklad:

```
void *unknown;
```

10.5 Vyu□itie smerníkov

10.5.1 Adresy premenných

V C/C ++ je možné pomocou smerníka ukazovat na lubovolné premenné.

Príklad 1: Ako pracujú smerníky

1. Vytvorte “Novú aplikáciu” vprostredí C++ Builder 5. Do formulára vložte dve editacné polícka (Edit), dve menovky (Label) (na sprístupnenie editacných políckok) a 4 tlacidlá (Button).
2. Nadpisy (Caption) menoviek môžu byt “&Temp-Variable:” (pomocná premenná) a “&Pointer:” (smerník). Nadpisy tlacidiel môžu byt “&Fill Temp-Variable” (zadaj pomocnú premennú), “Show T&emp Variable Value” (ukáž hodnotu pomocnej premennej), “Show P&ointer Value” (ukáž hodnotu smerníka) a “&Clear All” (zmaž všetko).

Zdrojový kód príkladu (v UNIT1):

3. V deklaracnej casti formulára definujte nasledujúce premenné (okrem Tfrm_Pointer *frm_Pointer, kde frm_Pointer je meno formulára – aj meno smerníka, ako jednoducho zistíte, v tomto príklade ju nepoužijeme):

```
// smerník
String *str_Ptr;
// premenná, na ktorú má smerník ukazovat
String str_Temp;
```

4. Do jedinej existujúcej funkcie vo formulári (tu: Tfrm_Pointer:Tfrm_Pointer) vložte nasledujúci kód:

```
__fastcall Tfrm_Pointer::Tfrm_Pointer(TComponent*
Owner)
: TForm(Owner)
{
```

```
// definujte, aby smerník ukazoval
// na adresu premennej
str_Ptr = &str_Temp;
// naplnte adresu, kam ukazuje smerník
// hodnotou ("dereferencing" smerníka)
*str_Ptr = "Hello World";
}
```

5. Do udalosti “OnClick”, tlacidla “Fill Temp-Variable”, vložte nasledujúci kód (v tomto prípade je názov tlacidla “btn_FillTempVariable” a meno editacného pola je “edi_TempVariable”):

```
void __fastcall
Tfrm_Pointer::btn_FillTempVariableClick(TObject
*Sender)
{
// do premennej priradte text napísaný
// v prvom editacnom poli
str_Temp = edi_TempVariable->Text;
}
```

6. Do udalosti “OnClick”, tlacidla “Show Temp Variable Value”, vložte nasledujúci kód (v tomto prípade je meno tlacidla “btn_ShowTempValue”):

```
void __fastcall
Tfrm_Pointer::btn_ShowTempValueClick(TObject
*Sender)
{
// vložte hodnotu premennej späť do
// prvého editacného pola
edi_TempVariable->Text = str_Temp;
}
```

7. Do udalosti “OnClick”, tlacidla “Show Pointer Value”, vložte nasledujúci kód (v tomto prípade je meno tlacidla “btn_ShowPointerValue”, “edi_Pointer” je meno druhého editacného pola):

```
void __fastcall
Tfrm_Pointer::btn_ShowPointerValueClick(TObject
*Sender)
{
```

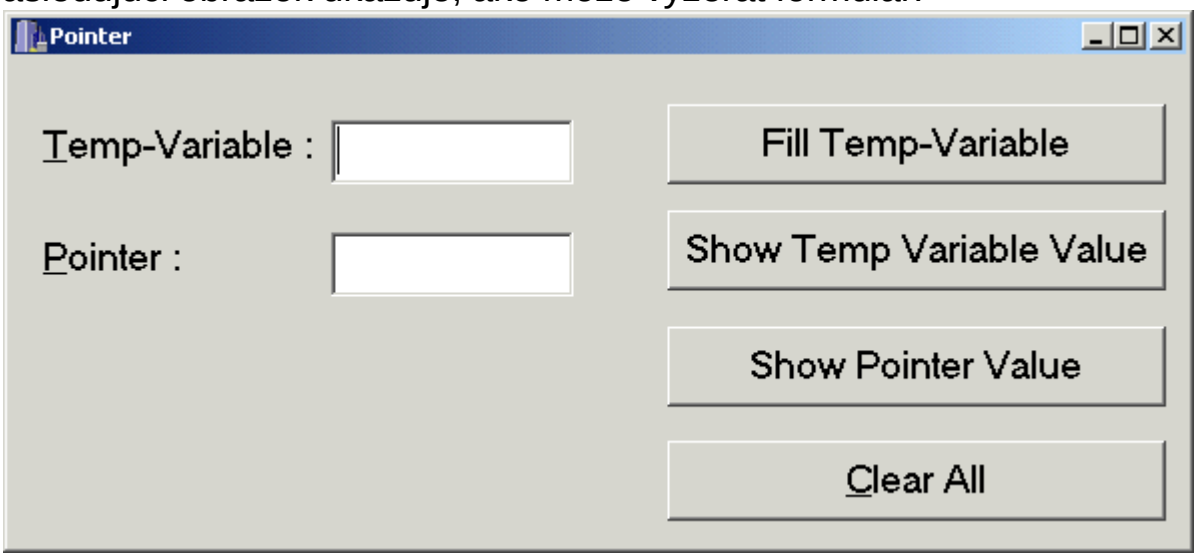
```
// do druhého editacného pola priradte hodnotu
// z adresy, na ktorú ukazuje smerník
//("dereferencing" smerníka)
edi_Pointer->Text = *str_Ptr;
}
```

8. Do udalosti “OnClick”, tlacidla “Clear All”, vložte nasledujúci kód (v tomto prípade je meno tlacidla “btn_Clear”):

```
void __fastcall
Tfrm_Pointer::btn_ClearClick(TObject *Sender)
{
// vyprázdnite obe editacné polia
edi_Pointer->Clear();
edi_TempVariable->Clear();
}
```

Vždy, ked kliknete na tlacidlá “Show Temp Variable Value” a “Show Pointer Value”, jedno po druhom obsahujú rovnaké hodnoty a zobrazujú ich v editacnom poli. Do prvého editacného pola môžete napísat novú hodnotu, stlacit tlacidlo “Fill Temp-Variable” a potom tlacidlo “Clear All”, aby sa vyprázdnili editacné polia. Obe predchádzajúce tlacidlá budú znovu zobrazovat rovnakú hodnotu.

Nasledujúci obrázok ukazuje, ako môže vyzerat formulár:



Príklad 2: Použitie dvoch smerníkov
Príklad môžeme rozšírit pridaním nasledujúceho kódu:

1. V tretom bode predchádzajúceho príkladu pridajme riadok:

```
// definuj druhý smerník
String *str_Temp2;
```

2. V štvrtom bode predchádzajúceho príkladu pridajme riadok:

```
// druhý smerník obdrží adresu premennej
// temp od prvého smerníka
str_Temp2 = &str_Temp;
```

3. V šiestom bode predchádzajúceho príkladu zmenme riadok s kódom na:

```
edi_TempVariable->Text = str_Temp2;
```

Tento príklad funguje tak ako prvý príklad.

Predchádzajúce príklady demonštrujú operáciu "referencing" a operáciu "dereferencing". Pomocou zretazovania alebo znovupriradenie môžeme vytvárat zaujímavé štruktúry údajov: spájané zoznamy, stromy, haldy, atd. Smerníky "rozumne" prepájajú jednotlivé premenné s pamätovými miestami.

Príklad 3: Využitie smerníka na prepínanie medzi komponentmi
Tieto príklady ukazujú silu smerníkov v reálnejšej situácii.

1. Vytvorte “Novú aplikáciu” v prostredí C++ Builder 5. Do formulára vložte dva zoznamové rámiky (ListBox), jedno editacné políčko (Edit), dve tlacidlá (Button) a jednu skupinu prepínacov (RadioGroup).
2. Prvky pomenujte takto:
   Listbox1: lst_Left
   Listbox2: lst_Right
   Edit1: edi_New
   Button1: btn_Add, Caption “Add” (pridaj)
   Button2: btn_Clear, Caption “Clear” (vymaž)
   RadioGroup1: rdg_SwitchBox, Caption “Switch Box” (prepínac)
3. Nastavte sa na vlastnost “Items” komponentu RadioGroup “rdg_SwitchBox” a stlacte Ctrl+Enter. Pridajte položku “& Left Listbox”, stlacením enter zariadkujte a pridajte položku “&Right Listbox”. Zavrite okno “String-List Editor”.
   Komponent RadioGroup bude teraz ukazovat dva prepínace (možnosti nastavenia).

Zdrojový kód príkladu (v súbore UNIT1):

4. V deklaracnej casti formulára definujte nasledujúce premenné:

```
// smerník typu listbox
TListBox *lst_Temp;
```

5. Do jedinej existujúcej funkcie pridajte nasledujúci kód:

```
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
// inicializuj (=nastav) smerník temp
// s adresou prvého zoznamového rámika
lst_Temp = lst_Left;
}
```

6. Do udalosti “OnClick”, tlacidla “Add”, vložte nasledujúci kód:

```
void __fastcall TForm1::btn_AddClick(TObject
*Sender)
{
// iba ak nie je edit-box prázdny
if (edi_New->Text != "")
{
// do zoznamového rámiku pridaj novú položku
// a použi hodnotu z editacného polícka
lst_Temp->Items->Add(edi_New->Text);
}
}
```

7. Do udalosti “OnClick”, tlacidla “Clear”, vložte nasledujúci kód:

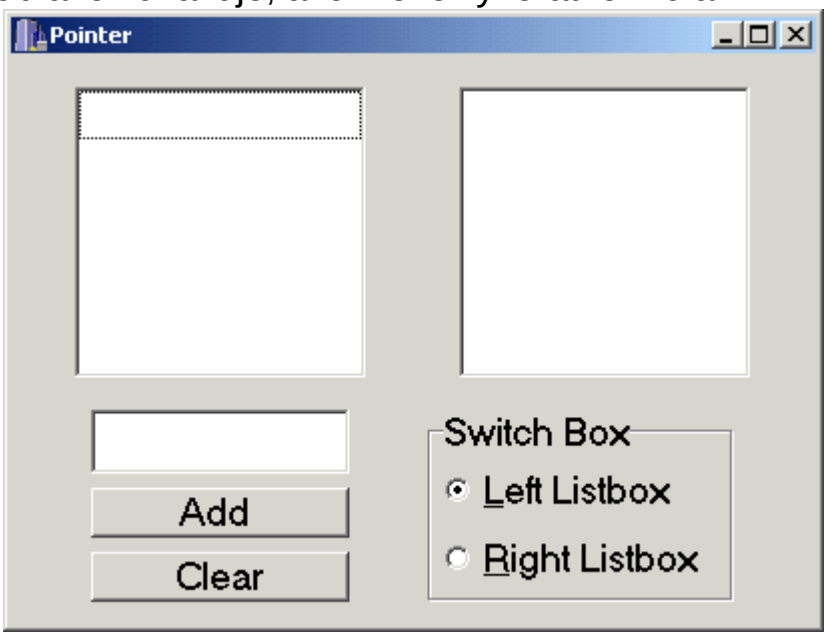
```
void __fastcall TForm1::btn_AddClick(TObject
*Sender)
{
// vyprázdni zoznamový rámik
lst_Temp->Clear();
}
```

8. Do udalosti “OnClick”, komponentu RadioGroup “rdg_SwitchBox”, vložte nasledujúci kód. Prvý prepínac “Left Listbox” (lavý zoznamový rámik) má hodnotu vlastnosti ItemIndex 0, druhý prepínac “Right Listbox” (pravý zoznamový rámik) má hodnotu ItemIndex 2:

```
void __fastcall
TForm1::rdb_SwitchBoxClick(TObject *Sender)
{
// ak je aktivovaný prepínac “Left Listbox”
if( rdb_SwitchBox->ItemIndex == 0 )
{
// smerník obdrží adresu lavého zoznamového
rámika
lst_Temp = lst_Left;
}
// ak je aktivovaný prepínac “Right Listbox”
else
{
// smerník obdrží adresu pravého zoznamového
rámika
lst_Temp = lst_Right;
}
}
```

V tomto príklade si môžeme pomocou prepínacov v komponente radio-group vybrat, s ktorým zoznamovým rámikom chceme pracovat. Smerník “lst_Temp2 obsahuje adresu pravého alebo lavého zoznamového rámika. Všetky príkazy ktoré pracujú so zoznamovým rámikom tu môžu byt vykonané na smerníku lst_Temp.

Nasledujúci obrázok ukazuje, ako môže vyzerat formulár:



10.5.2 Smerníky a vektory (polia)

Deklaráciou

```
int a[10];
```

Definujeme vektor pozostávajúci z 10 za sebou idúcich hodnôt typu integer, ku ktorým môžeme pristupovat inštrukciou

```
a[0] až a[9]
```

Pomocou smerníkov môžeme pristupovat aj k poliam. Teda pomocou

```
int *pa=&a[0]
```

smerník pa ukazuje na prvý element pola a. Pomocou jednoduchej smerníkovej aritmetiky môžeme pristupovat k celému polu:

```
a++;
*pa=120;
```

priradí do a[1] hodnotu 120.

Tiež môžeme napísat

```
*(pa+1)=120;
```

Poznámka:

Zistovanie adresy prvého elementu pola inštrukciou

```
int *pa = &a[0];
```

je síce lahko citatelné, ale v C/C++ nezvycajné. Ak napíšeme názov pola bez indexu, adresa pola bude vygenerovaná automaticky. Teda nasledujúca inštrukcia urobí to isté:

```
int *pa = a;
```

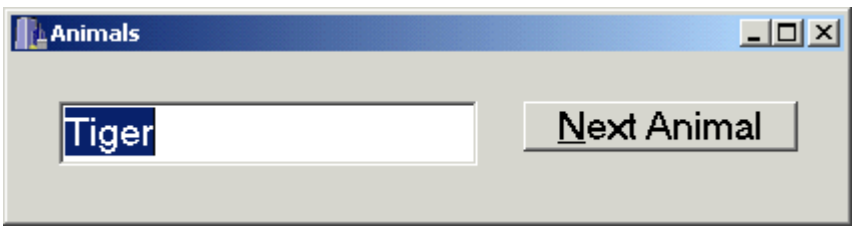
Príklad 4: Využitie smerníka ako pola

1. Vytvorte “novú aplikáciu” v prostredí C++ Builder 5. Do formulára vložte jedno editacné políčko (Edit) a jedno tlacidlo (Button).
2. Ovládacím prvkom dajte nasledujúce mená:
   Edit1: edi_Animal (zviera)
   Button1: btn_NextAnimal, Caption “Next Animal” (dalšie zviera)
   Caption Form1: “Animals” (zvieratá)
   Kód príkladu (v unit1) zacína smerníkom na formulár (form):

```
// kód v deklaracnej casti unitu
TForm1 *Form1;
// pole pre mená piatich zvierat
```

```
String astr_Animals[5];
// smerník priradený k prvému elementu pola
String *str_ptr = astr_Animals;
// riadiaca premenná
int i = 0;
//---------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner):
TForm(Owner)
{
// napln políčka pola menami zvierat,
// zacni políčkom s číslom 0
astr_Animals[0] = "Tiger";
astr_Animals[1] = "Lion";
astr_Animals[2] = "Elefant";
astr_Animals[3] = "Zebra";
astr_Animals[4] = "Hippo";
// zobraz prvé zviera v editacnom poli
edi_Animal->Text = *str_ptr;
}
//---------------------------------------------
void __fastcall
TForm1::btn_NextAnimalClick(TObject *Sender)
{
// zvýš riadiacu premennú o jedna
i++;
// ak riadiaca premenná dosiahne hodnotu 5
// potom ju nastav späť na 0;
if (i == 5) i = 0;
// nastav smerník tak, aby ukazoval na
// adresu zvierata na pozícii i v poli
// a zobraz ho v editacnom poli
edi_Animal->Text = *(str_ptr+i);
}
//---------------------------------------------
```

Nasledujúci obrázok ukazuje, ako môže vyzerat formulár:



10.5.3 Cvicenia

1. Cvicenie podobné príkladu 3: Vytvorte formulár, ktorý bude obsahovat 3 kombinované rámiky (combo-box), jedno editacné pole (edit-box) a skupinu prepínacov (radio-group). Vytvorte tlacidlá (button), na pridanie položky do kombinovaného rámika, na mazanie poslednej alebo prvej položky a na vyprázdnenie kombinovaného rámika. Na prepínanie medzi kombinovanými rámikmi použite skupinu prepínacov (radio-group).
2. Vytvorte aplikáciu pozostávajúcu z troch formulárov. Prvý formulár obsahuje tlacidlo (button) a skupinu prepínacov (radio-group). Druhý formulár obsahuje zoznamový rámik (list-box) s názvami miest. Tretí formulár obsahuje zoznamový rámik s PSC. Tlacidlo bude otvárat formulár. Skupina prepínacov (radio-group) rozhodne, ci bude otvárat formulár s názvami miest, alebo ten s PSC.
3. Vytvorte hazardnú hru (gamble). Použite 3 editacné polia (edit-box) a tlacidlo (button). Kliknutím na tlacidlo hra zacne a vygeneruje náhodné císlo v každom editacnom poli, napr. medzi 0 and 9. Na generovanie císel napíšte len jednu funkciu a pri naplnaní editacných polí použite smerník.
   Poznámka: Pri generovaní náhodných císel použite funkcie randomize() a random();

10.6 Vymenované typy

Ciel tejto podkapitoly: Naucit sa ako k celocíselným hodnotám priradovat identifikátory.

Najprv zadefinujeme vymenované typy. Potom preberieme nejaké príklady.

10.6.1 Definícia

Pomocou vymenovaných typov môžeme k číslam priradit identifikátory. Na definovanie vymenovaného typu použijeme klúcové slovo “enum”.
Napríklad:

```
enum TagDays {Mon, Tue, Wen, Thu, Fre, Sat, Sun}
Day;
```

Zadefinovali sme vymenovaný typ „Days”, ktorý priradí ku každému dnu císlo, pocínajúc Mon = 0 až po Sun = 6.
Vymenovaný typ môžeme definovat aj takto:

```
enum TagCoins{penny=1, twopence, nickel=penny+4,
dime=10, quarter=nickel*nickel} Coins;
```

Týmto priradíme penny =1, twopence = 2, nickel = 5, dime = 10, quarter = 25.
Identifikátory sa nazývajú enum-constants (konštanty vymenovaných typov).

10.6.2 Deklarácia

“Day” je premenná, ktorá je deklarovaná v rovnakom case ako vymenovaný typ, co nie je povinné.
Novú premennú “today” typu TagDays môžeme deklarovat takto

```
enum TagDays today;
```

10.6.3 Priradenie

Do premennej vymenovaného typu môžeme priradit iba samotné konštanty vymenovaného typu. Nemôžeme do nej priradit hodnotu typu integer.
Napr.:

```
Day=Thu;   // správne
Day=1; // nesprávne napriek tomu, že Thu má
hodnotu 1
```

Ku každej premennej typu int môžeme priradit konštantu vymenovaného typu, napr:

```
int today = Thu;
```

Príklad 5: Využitie vymenovaného typu na špecifikáciu elementov

1. Vytvorte “Novú aplikáciu”
2. Formulár definujte podobne ako v príklade 4:
3. Zdrojový kód v súbore unit1 zmente takto

```
// kód v deklaracnej casti unitu
TForm1 *Form1;
// pole pre mená piatich zvierat
String astr_Animals[5];
// definuj vymenovaný typ so zvieratami,
// v ktorom etiger = 0 a ehippo = 4
enum TagAnimals {etiger, elion, eelefant, ezebra,
```

```
ehippo} Animals;
// riadiaca premenná
int i = 0;
//---------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  // napln pole menami zvierat
  // s použitím vymenovaného typu Animals
  astr_Animals[etiger] = "Tiger";
  astr_Animals[elion] = "Lion";
  astr_Animals[eelefant] = "Elefant";
  astr_Animals[ezebra] = "Zebra";
  astr_Animals[ehippo] = "Hippo";
  // do editacného pola prirad meno
  // prvého zvierata s použitím konštanty
  // etiger vymenovaného typu
  edi_Animal->Text = astr_Animals[etiger];
}
//---------------------------------------------
void __fastcall
TForm1::btn_NextAnimalClick(TObject *Sender)
{
  // zvýš riadiacu premennú o jedna
  i++;
  // ak má riadiaca premenná hodnotu 5 tak
  // ju nastav späť na 0
  if (i == 5) i = 0;
  // zobraz meno zvierata na pozícii i
  edi_Animal->Text = astr_Animals[i];
}
//---------------------------------------------
```

10.7 Štruktúry

Ciel tejto podkapitoly: Naucíme sa co sú to štruktúry, ako sa definujú a na co sa používajú.

V tejto podkapitole zadefinujeme štruktúry. Potom si ukážeme ako ich používat a ako do nich nastavovat hodnoty.

10.7.1 Definícia

Štruktúry sa používajú na zaznamenávanie viacerých hodnôt rôzneho typu, ktoré nejako súvisia. Typický príklad štruktúry je záznam v databáze, napr. adresa. Pri definícii štruktúry používame klúcové slovo struct.
Štruktúru na zaznamenanie adresy definujeme takto:

```
struct TagAddress
{
  String Name;
  int Age;
  float Size;
} Address;
```

10.7.2 Deklarácia

V predchádzajúcej definícii štruktúry je deklarovaná aj premenná "Address", co je nepovinné.
Nové premenné typu TagAddress deklarujeme inštrukciou

```
struct TagAddress MyAddress;
```

Rovnako môžeme deklarovat aj smerníky a polia typu struct.
Napr.:

```
struct TagAddress *ptr_Address; // smerník typu
TagAddress
struct TagAddress FamilyAddresses[5]; // 5-
prvkové pole typu TagAddress
```

10.7.3 Deklarácia s použitím "typedef"

Pomocou klúcového slova "typedef" môžeme premenovat štruktúru.
Napr.:

```
typedef struct TagAdress OLDADRESS;
```

Môžeme ho ale použit aj na definovanie novej štruktúry:

```
// definícia typu
typedef struct {…..} NEWADRESS;
// deklaruj premenné
NEWADRESS Friends[10], GirlFriend, *BoyFriend;
```

Poznámka:
"Typedef" môžeme použit rovnako aj pre vymenované typy.

10.7.4 Priradovanie do elementov

Na priradenie alebo cítanie hodnôt z elementov štruktúry môžeme použit operátory “.”(bodka) alebo “->”(šípka).
Ak chceme uložit do premennej “MyAdress” typu TagAdress nejaké hodnoty s použitím operátora bodka, urobíme to takto:

```
// použitie operátora bodka
struct TagAddress MyAddress;
MyAddress.Name = “Jane Somebody”;
MyAddress.Age = 25;
MyAddress.Size = 1.7;
```

Ak chceme použit operátor šípka, definujeme najprv smerník. Potom do smerníka priradíme klúcovým slovom “new” štruktúru rovnakého typu. Po použití musíme smerník zrušit (destroy) klúcovým slovom “delete”.

```
// deklaruj smerník typu struct TagAdress, (iba
sa alokuje pamät)
struct TagAddress *MyAddress;
// prirad “reálnu” premennú typu struct TagAdress
MyAdress = new (struct TagAddress);
// prirad hodnoty do elementov
MyAddress->Name = “Jane Somebody”;
MyAddress->Age = 25;
MyAddress->Size = 1.7;
// zruš smerník (destroy)
delete MyAdress;
```

Príklad 6: Použitie štruktúry na ukladanie záznamov

1. Vytvorte “novú aplikáciu”.
2. Do formulára vložte kombinovaný rámik (combo-box), dve editacné polia (edit-box) a 3 menovky (label).
3. Umiestnite komponenty tak, že menovky (label) sú vlavo od kombinovaného rámiku (combo-box) a editacných polí (edit-box).
4. Komponenty pomenujte takto:
   ComboBox1: cmb_AnimalName (meno zvierata)
   Edit1: edi_AnimalClass (druh)
   Edit2: edi_AnimalSize (velkost)
   Labels nechajte tak ako sú.
5. Vlastnosti caption (nadpis) pre menovky (label) nastavte takto:
   Label1: Caption “&Name” (meno)
   Label2: Caption “&Kind” (druh)

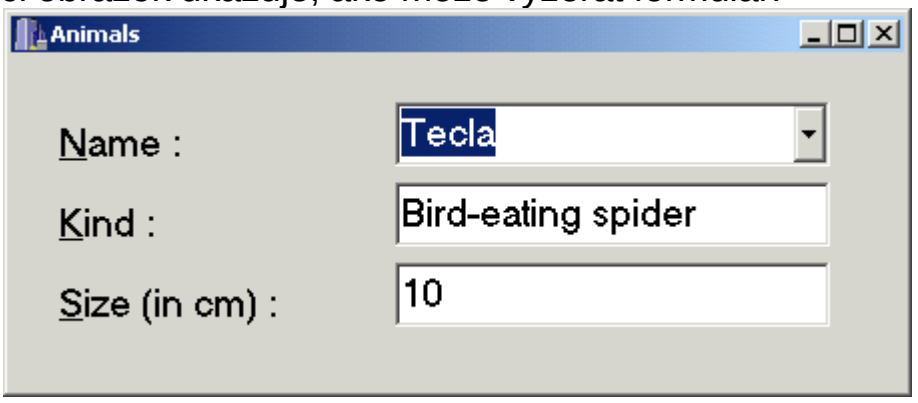
Label3: Caption “&Size (in cm)” (velkost v cm)

6. Na prepojenie menoviek (label) s korešpondujúcimi komponentmi použite vlastnost focuscontrol.
7. Do okna Unit1 napíšte nasledujúci kód:

```
TForm1 *Form1;
// definuj štruktúru Animals
struct Animals
{
String Name;
String Class;
int Size;  // v centimetroch

};
// deklaruj pole typu Animals
struct Animals MyAnimals[4];
//---------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
// 1. Zviera je pes
MyAnimals[0].Name = "Pluto";
MyAnimals[0].Class = "Dog";
MyAnimals[0].Size = 70;
// 2. Zviera je macka
MyAnimals[1].Name = "Carlo";
MyAnimals[1].Class = "Cat";
MyAnimals[1].Size = 30;
// 3. Zviera je kanárik
MyAnimals[2].Name = "Twiggy";
MyAnimals[2].Class = "Canary";
MyAnimals[2].Size = 7;
// 4. Zviera je pavúk-vtákožrút
MyAnimals[3].Name = "Tecla";
MyAnimals[3].Class = "Bird-eating spider";
MyAnimals[3].Size = 10;

// do kombinovaného rámiku vlož mená zvierat
for (int i = 0; i < 4; i++)
{
cmb_AnimalName->Items->Add(MyAnimals[i].Name);
}
```

```
}
//---------------------------------------------

void __fastcall
TForm1::cmb_AnimalNameChange(TObject *Sender)
{
// ak v kombinovanom rámiku vyberiete zviera,
// editacné polia budú ukazovat
// druh a velkost zvierata
edi_AnimalClass->Text =
MyAnimals[cmb_AnimalName->ItemIndex].Class;
edi_AnimalSize->Text =
MyAnimals[cmb_AnimalName->ItemIndex].Size;
}
//---------------------------------------------
```

Nasledujúci obrázok ukazuje, ako môže vyzerat formulár:



10.7.5 Cvicenia

1. Vytvorte formulár a vložte do neho editacné polia (edit-box) pre meno, adresu, atd. Na uloženie mien a adries viacerých osôb použite pole štruktúr. Na presúvanie sa na predchádzajúci a nasledujúci záznam a na zobrazovanie hodnôt aktuálneho záznamu použite tlacidlá (button). Aj na pridávanie a mazanie adries v štruktúre použite tlacidlá.
2. Použite cvicenie 1. Do formulára aj do štruktúry pridajte políčko “Gender” (pohlavie). Pridajte tlacidlo. Po kliknutí na tlacidlo sa môže otvorit druhý formulár, sú meno a adresa zapísané ako v liste. Sledujte formát adresy v súvislosti s oslovením podla pohlavia (pán, pani).
3. Použite štruktúru na ukladanie názvov hudobných titulov a mena súboru s titulom. Vytvorte formulár s kombinovaným rámikom

(combo-box) a tlacidlom (button). Kombinovaný rámik môže obsahovat skladby. Po kliknutí na tlacidlo bude skladba prehraná. Poznámka: Na prehrávanie hudobných súborov použite komponent TMediaPlayer.

10.8 Objektovo-orientované programovanie

Ciel tejto podkapitoly: Spoznat pojmy objektovo-orientovaného programovania. Deklarovat triedu, písat pre takúto triedu funkcie, definovat objekt a použit funkcie takéhoto objektu. Použit konštruktor (constructor) a deštruktor (deconstructor) objektu. Popísat a použit dedicnost (inheritance), polymorfizmus (polymorphism) a pretažovanie (overloading). Vediet, co reprezentujú pojmy "udalost" (event), "metóda" (method) a "vlastnost" (property) vo vizuálnom programovacom prostredí.

Najprv zacneme s definíciou objektovo-orientovaného programovania (object oriented programming) a preco by sme mali robit nejakú prácu navyše. Uvidíme, že všetko sa zacína triedou (classs) definovanou v hlavickovom súbore (header-file). Budeme písat funkcie pre takúto triedu a dozvieme sa o špeciálnych funkciách: konštruktor (constructor) a deštruktor (deconstructor).
V dalšom spoznáme pojmy dedicnost (inheritance), polymorfizmus (polymorphism) a pretažovanie (overloading) a naucíme sa ich používat.
Nakoniec sa pozrieme na vizuálne programovacie prostredia a na to, ako sú udalosti (events), metódy (methods) a vlastnosti (properties) v nich definované.

10.8.1 Definície

10.8.1.5 Abstraktný údajový typ (abstract datatype)

Až doteraz sme používali iba jednoduché údajové typy ako sú char, int, float, ale i pointer (smerník) a zložené údajové typy ako array (pole), struct, union (typ podobný struct, ale pristupovat možno v jednom casovom okamihu len k jednému prvku) a enum. Objektovo-orientované programovanie prináša nový údajový typ a to abstraktný údajový typ.

Definícia:
Abstraktný údajový typ je používatelom definovaný model struct, union alebo class (trieda), ktorý môže obsahovat údaje i funkcie ako prvky. Tieto

môžu byt deklarované ako private (súkromné) a public (verejné).

Výhody takejto údajovej štruktúry ležia v prísnom rešpekte a nemennosti modulárnych castí programu. Pri rozširovaní softvérových balíkov napísaných v procedurálnych programovacích jazykoch sa lahko môže stat, že napríklad vdaka prispôsobovaniu a rozširovaniu pôvodne velmi dobre definovaných rozhraní sa tieto "znejasnia" a stanú sa závislé na implementácii. Zdielané moduly a programové jednotky sú spletené a zamotané.

Poznámka:
Výrazy "trieda" (class) a "abstraktný údajový typ" (abstract data type) sú synonymá.

10.8.1.6 Zapúzdrenie (Encapsulation)

V tradicnom programovaní sa definujú údaje a rovnako i funkcie, ktoré údaje preberajú, spravujú a odovzdávajú. V objektovo-orientovanom programovaní sa spolu s údajmi definujú aj funkcie pracujúce s týmito údajmi. Toto spolocné definovanie (údajov a príslušných funkcií) sa nazýva "zapúzdrenie".

10.8.2 Deklarácia triedy

Deklarácia (declaration) sa podobá deklarácii “struct”. Uvádza ju rezervované slovo “class”, potom nasleduje meno triedy (teraz nazvaná “sphere” (gula)). Zložené zátvorky uzatvárajú prvky triedy a delia sa na dve casti “private:” (súkromné) a “public:” (verejné).

Príklad:

```
class sphere
{
   private:
   float f_radius;
   float f_volumn;
   float PI;
  public:
   void setRadius (float radius);
   float getRadius (void);
   float getVolumn (void);
   void calculateVolumn (void);
};
```

Deklarácia triedy musí byt ukoncená bodkociarkou (semicolon)!
V triede sú povolené premenné (variables) a funkcie (functions).
Deklarácia triedy je uložená v sprievodnom súbore unit-súbore (unit-file), v hlavickom súbore (header-file). Hlavickový súbor má rovnaké meno ako unit-súbor, no má inú príponu ".h", teda "sphere.h" je priradené ku "sphere.cpp".
Okno s hlavickovým súborom môžeme otvorit stlacením CTRL + F6 v okne unitu.

10.8.2.7 Private (súkromný) a public (verejný):

Prvky súkromnej sekcie (private:) môžu byt použité výlucne iba vo funkciách triedy. Sú tak chránené proti nepovolenému prezeraniu ci modifikácii z prostredia mimo triedy.
Každý príkaz, hoci aj z prostredia mimo triedy, sa môže dostat k prvkom vo verejnej sekcii (public:). Vdaka public: je možné takéto premenné menit i z prostredia mimo triedy.
Neexistuje pravidlo, ktoré by urcovalo, v ktorej sekcii sa ma ten-ktorý prvok nachádzat! Ako funkcie, tak aj premenné, môžu byt deklarované ako public: i private:. Z dôvodu garancie spolahlivej ochrany premenných pred vonkajším prístupom sa doporucuje umiestnit ich do súkromnej sekcie a vytvorit malé funkcie, ktoré umožnujú prístup k týmto premenným.
Triedy môžu byt deklarované tiež pomocou klúcového slova "struct" (trieda je štruktúra, ktorá obsahuje aj funkcie). Rozdiel je však v spôsobe, ako sa narába s prvkami, ktoré patria do nie oddelene definovanej sekcie (private: alebo public:). Kým vstruct sú štandardne (default) verejné, vtriede sa stanú štandardne súkromné.

Nasledujúci príklad vyjasnuje rozdiel medzi deklaráciami struct a class:

```
struct AClass
{
   int a;  // premenná verejne prístupná
   int getValue(); // žiadna inštrukcia public:
                   // nie je potrebná
};


class BClass
{
   int b;  // súkromná premenná
```

```
   int getValue(); // funkcia nie je zvonka
                   // dostupná
};
```

10.8.3 Objekt (Object)

Trieda je deklarácia nového typu. Aby sme mohli použit clenské funkcie a polia triedy, musíme mat definovaný objekt.

Príklad:
Funkcia “main” by mohla použit vyššie definovanú triedu nasledovne:

```
void main()
{
   sphere football;

   football.setRadius(15);
   football.calculateVolumn();
}
```

Definujeme objekt (pomenovaný "football") triedy "sphere“. Volanie funkcie sa uskutocnuje úplne rovnako ako sa pristupuje kprvkom triedy. Meno funkcie je oddelené od mena objektu bodkou.

Poznámka:
V literatúre sa "objekt" nazýva aj "inštancia" ("instance").

10.8.4 Definícia clenských funkcií triedy (element functions)

Deklarácie funkcií vtriede sú len prototypy (prototypes). Tieto musia byt neskôr definované.

Príklad:
Funkcia “setRadius” (nastav polomer), ktorá nacíta novú hodnotu polomeru gule, by mohla vyzerat nasledovne:

```
void sphere::setRadius (float radius)
{
   f_radius=radius;
}
```

Táto funkcia priradí súkromnej triednej premennej “f_radius” hodnotu radius

(polomer), ktorá jej bola odovzdaná.

Definícia vyzerá ako bežná definícia funkcie až na meno triedy a dve dvojbodky (sphere::) nachádzajúce sa pred menom funkcie. Toto jednoznacne hovorí kompilátoru, že ide o clenskú funkciu triedy “sphere”. Je totiž možné, že i iné triedy obsahujú funkciu s menom “setRadius”, ktoré však nemajú nic spolocné s funkciou sphere::setRadius.

10.8.5 Konštruktor (Constructor)

Každá premenná by mala byt priradená ku správnej hodnote skôr, ako je použitá. To isté platí pre prvky triedy. Pri takomto ponímaní situácie by mala existovat aspon jedna funkcia, ktorá inicializuje vnútro triedy. Kedže toto je velmi dôležité, a je to nezávislé od ostatných zložiek triedy, funkcie, ktoré inicializujú údajové prvky objektu, nazývame osobitným menom: konštruktor.
“Konštruktor” musíme takto charakterizovat, hoci C++ mu dáva aj dalšie úlohy. Jeho identifikácia je založená na jeho volaní: jeho meno je rovnaké, ako je meno triedy objektu, ktorý má inicializovat. Pocet a typ argumentov udávajú kompilátoru dodatocnú informáciu, ktorý príslušný konštruktor sa má použit. Konštruktor s minimálnym poctom argumentov sa napríklad nazýva "štandardný konštruktor" ("default constructor"). Je volaný vždy, ked program príde vaplikácii kobjektu a tento nemá urcený inicializátor (initializer). To znamená, že takéto volanie sa objaví pred prvou inštrukciou programu, skôr, ako aplikácia dosiahne tento objekt.
Konštruktory sú volané automaticky a na rozdiel od iných funkcií, nemôžu byt volané vlubovolnom bode programu. Avšak konštruktory môžu volat dalšie funkcie. Konštruktory nemôžu vracat hodnotu, ani len void!

Príklad:
Konštruktor triedy tlacidla by mohol vyzerat nasledovne:

```
class sphere
 {
   private:
    float f_radius;
    float f_volumn;
    float PI;
   public:
    sphere();  // konštruktor
    void setRadius (float radius);
    float getRadius (void);
```

```
    float getVolumn (void);
    void calculateVolumn (void);
};
```

10.8.6 Deštruktor (deconstructor)

Kedže život každého objektu je obmedzený, existuje tiež automatický systém jeho odstránenia, ktorému zodpovedajú isté clenské funkcie. Tieto funkcie nazývame deštruktory (destructors). Ich úlohou je uvolnit všetky prvky patriace k objektu – t.j. dynamicky rezervované bloky pamäte. Volania deštruktora vytvára kompilátor automaticky.
Život objektu nekoncí skôr, ako bol zavolaný jeho deštruktor. Ak objekt opustí priestor, deštruktor je zavolaný automaticky (pri statických objektoch!).
Deštruktor nesie meno triedy uvedené tildou (~). Deštruktor nemá návratový typ (ani len void) a má prázdny zoznam argumentov. V každej triede je práve jeden deštruktor. Ak deštruktor nebol definovaný a trieda potrebuje deštruktor, tak kompilátor jej automaticky jeden vytvorí.
Štandardné deštruktory (default destructors) sú vždy verejné, teda v deklarácii triedy prináležia do sekcie public:. Každý deštruktor implicitne volá deštruktory svojich údajov po tom, co ukoncil vlastné inštrukcie, a to v opacnom poradí, ako boli deklarované.

Poznámka:
Písanie volaní deštruktorov v deštruktore môže mat kritické dôsledky.
Ak z nejakých dôvodov dôjde k dvojnásobnému volaniu deštruktorov pri súcasnom viacnásobnom uvolnovaní oblasti pamäte, môže dôjst ku kolapsu systému (crash).

Tu sú niektoré crty (features) deštruktorov:

1. V triede je iba jeden deštruktor.
2. Deštruktory nemajú paramatere.
3. Deštruktory nemajú návratovú hodnotu, a to ani void.
4. Deštruktory sú volitelné. Ak trieda nemusí po svojej cinnosti upratovat (clean), nepotrebuje deštruktor.
5. Deštruktory sú volané automaticky. Programy nikdy nevolajú deštruktory.

Príklad:

```
class sphere
 {
   private:
```

```
    float f_radius;
    float f_volumn;
    float PI;
   public:
    sphere();  //konštruktor
    void setRadius (float radius);
    float getRadius (void);
    float getVolumn (void);
    void calculateVolumn (void);
    ~sphere()  //deštruktor;
};
```

Príklad 8: Trieda “sphere” (gula)

1. Vytvoríme “New Application” (novú aplikáciu).
2. Na formulár umiestnime dve Labels (menovky) , dve Edits (editacné políčka) a päť Buttons (tlacidiel).
3. Usporiadame komponenty tak, že menovky sú vlavo od editacných políčok. Tlacidlá môžme umiestnit kamkolvek.
4. Komponenty pomenujeme (name) nasledovne:
   Edit1: edi_radius
   Edit2: edi_volumn
   Button1: btn_getRadius
   Button2: btn_setRadius
   Button3: btn_getVolumn
   Button4: btn_calculateVolumn
   Button5: btn_ClearAll
   Necháme Labels tak ako sú.
5. Nastavíme Caption (nadpis) nasledovne:
   Label1: &Radius (polomer)
   Label2: &Volumn (objemn)
   btn_getRadius: Get R&adius (vrát polomer)
   btn_setRadius: Set Ra&dius (nastav polomer)
   btn_getVolumn: Get V&lumn (vrát objem)
   btn_calculateVolumn: Calculate Vo&lumn (vypocítaj objem)
   btn_ClearAll: &Clear All (vymaž všetko)
6. cez ponuku “Files”-“New…” ("Súbor"-"Nový…") pridáme nový unit do okna unitov (unit-window).
7. Uložíme unit pod menom “sphere”.
8. Stlacíme CTRL + F6 vo vnútri “sphere.cpp”. Otvorí sa súbor “sphere.h”.
9. Do hlavickového súboru (header file) pridáme nasledujúci kód:

```
class sphere
 {
   private:
    float f_radius;
    float f_volumn;
    float PI;
   public:
    sphere();  // konštruktor
    void setRadius (float radius);
    float getRadius (void);
    float getVolumn (void);
    void calculateVolumn (void);
    ~sphere()  // deštruktor;
 };
```

10. Uložíme hlavickový súbor.
11. Vrátime sa do okna “sphere.cpp”.
12. Do sekcie deklarácií (declaration section) cpp-súboru pridáme nasledujúci kód:

```
// vkladá knižnicu matematických funkcií
// z nej použijeme funkciu Power() (mocnina)
#include <math.hpp>
```

13. Do cpp-súboru pridáme tento kód:

```
// koštruktor gule (sphere), ktorý sa spustí
// pri generovaní objektu
sphere::sphere()
{
 // inializujeme všetky súkromné premenné
 f_radius = 0;
 f_volumn = 0;
 PI = 3.14;
}
//-----------------------------------------
// deštruktor gule, ktorý sa spustí pri
// zrušení objektu (object is destroyed)
sphere::~sphere()
{
  // v tomto prípade nie je co robit
}
```

```
//---------------------------------------------
// nastavuje (set) polomer (radius) gule
void sphere::setRadius (float radius)
{
 f_radius = radius;
}
//---------------------------------------------
// vracia (return) súcasnú hodnotu polomeru
float sphere::getRadius (void)
{
  return f_radius;
}
//---------------------------------------------
// vracia aktuálnu hodnotu objemu (volume) gule
float sphere::getVolumn (void)
{
  return f_volumn;
}
//---------------------------------------------
// vypocítava (calculate) objem gule
void sphere::calculateVolumn (void)
{
  f_volumn = 4 * Power(f_radius, 3) * PI / 3;
}
//---------------------------------------------
```

14. Uložíme súbor sphere.cpp.
15. Aktivujeme okno unitu formulára (pomenované štandardne “unit1”).
16. Otvoríme ponuku “File”–“Include Unit Hdr” ("Súbor"–"Vložit Unit Hlav.").
17. Zvolíme sphere.h a ukoncíme cez OK. Teraz je sphere.h pridané ako

```
#include “sphere.h”
```

do unitu formulára a teraz môžeme použit našu triedu “sphere” (gula).

18. Do sekcie deklarácií v onke unitu (za TForm1 *Form1;) napíšeme nasledujúci kód

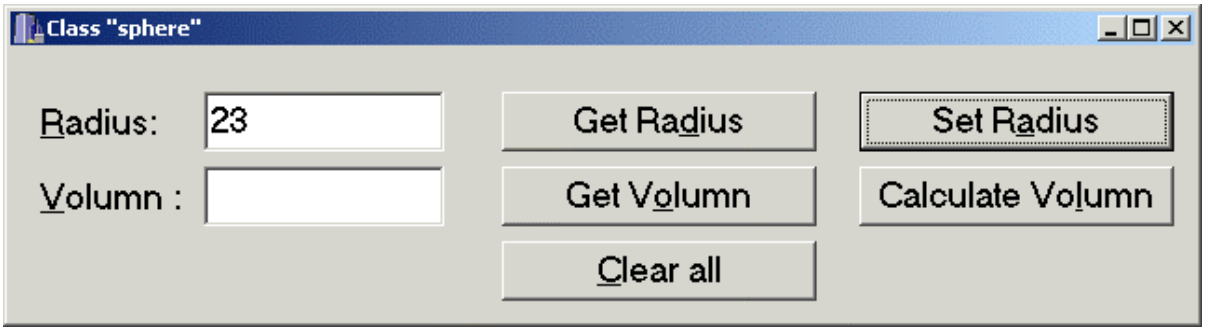
```
// deklarujeme objekt "football" triedy "sphere"
sphere football;
```

19. Pridáme nasledujúci kód na koniec okna unitu:

```
void __fastcall
TForm1::btn_setRadiusClick(TObject *Sender)
{
  football.setRadius ( edi_radius-
>Text.ToDouble());
}
//---------------------------------------------
void __fastcall
TForm1::btn_getRadiusClick(TObject *Sender)
{
  edi_radius->Text = football.getRadius();
}
//---------------------------------------------
void __fastcall
TForm1::btn_getVolumnClick(TObject *Sender)
{
  edi_volumn->Text = football.getVolumn();
}
//---------------------------------------------
void __fastcall
TForm1::btn_setVolumnClick(TObject *Sender)
{
  football.calculateVolumn();
}
//---------------------------------------------
void __fastcall TForm1::btnClearAllClick(TObject
*Sender)
{
  edi_radius->Clear();
  edi_volumn->Clear();
}
//---------------------------------------------
```

20. Teraz môžeme otestovat triedu "sphere" (gula) tak, že skompilujeme formulár a odskúšame niekolko hodnôt pre “radius” (polomer). Zistíme, že ak je editacné políčko pre polomer prázdne, alebo obsahuje retazec (string) a stlacíme "Set Radius" (Nastav polomer), aplikácia vygeneruje výnimku (exception). Tento problém budeme riešit v nasledujúcej kapitole.

Nasledujúci obrázok ukazuje, ako by mohol formulár vyzerat:



10.8.7 Cvicenie

1. Navrhnite triedu “kružnica”, ktorá vypocítava všetky "hodnoty kružnice"
   Vstupné hodnoty: polomer, priemer, Pi
   Výstupné hodnoty: polomer, priemenr, Pi, obsah, obvod.
   Vytvorte jednoduchý formulár na zobrazovanie triedy.
2. Použite cvicenie 1 z kapitoly 3.5 a riešte problém pomocou triedy.
3. Použite cvicenie 3 z kapitoly 3.5 a riešte problém pomocou triedy.
4. Navrhnite triedu “Vtáky”. Trieda by mala obsahovat vela vlastností a vecí, ktoré môžu robit, a to také, ktoré platia pre všetky vtáky, napr. ci majú perie, akej sú farby, niektoré z nich vedia a niektoré nevedia lietat, ich velkost, rozpätie krídiel, co jedia, atd..
   Vytvorte formulár na navrhovanie vtákov, napr. kura, holub, a aj vtákov, ktoré vôbec neexistujú.

10.9 Preta□ovanie (Overloading)

Do definície funkcie môžeme pridat funkciu s rovnakým menom, ako má už existujúca funkcia, ktorá sa líši iba v type alebo v pocte parametrov. Toto sa nazýva pretažovanie funkcie.
Teraz možno použit oba typy funkcií. Trieda sa sama rozhodne, ktorý druh funkcie má byt použitý.

Príklad:
Do verejnej sekcie definície triedy "sphere" (gula) v hlavickovom súbore pridáme

```
void setRadius (String radius);
```

Toto pretaží funkciu setRadius. Teraz môžeme zavolat funkciu setRadius aj vtedy, ked je parameter typu String.

Príklad 8: Pretaženie funkcie triedy sphere (gula)

1. Otvoríme súbor sphere.h.
2. Do verejnej sekcie pridáme nasledujúcu definíciu funkcie:

```
void setRadius (String radius);
```

3. Do súkromnej sekcie pridáme nasledujúcu definíciu funkcie:

```
int checkRadius(String radius)
```

   Táto funkcia skontroluje, ci prevzatý retazec môže byt pretypovaný na float.
4. Uložíme súbor sphere.h a aktivujeme okno sphere.cpp.
5. Do súboru sphere.cpp pridáme nasledujúci kód:

```
// pretažená funkcia pre parameter typu string
void sphere::setRadius (String radius)
{
  // Ak je prevzatý polomer číslo, pretypuj ho
  // na float a prirad do radius.
  // Inak nastav radius na 0
  if (checkRadius(radius))
  {
   f_radius = radius.ToDouble();
  }
  else
 {
   f_radius = 0;
 }
}
//---------------------------------------------
// skontroluj, ci prijatý retazec možno
// pretypovat na float
int sphere::checkRadius(String rstr_radius)
{
  // správny retazec môže obsahovat iba
  // tieto znaky
    String str_Numbers = "1234567890,";
  // ak je retazec prázdny, je nesprávny;
  // vrát false
  if (rstr_radius == "") return 0;
  // skontroluj každý znak v prijatom retazci,
  // ci je správny. Ak nie, vrát false
```

```
  for (int i = 1; i <= rstr_radius.Length();i++)
 {
   // Vezmi znak na pozícii i (funkcia
   // Substring(i,1) vystrihne 1 znak
   // na pozícii i) v prijatom retazci
   // "rstr_radius" a skontroluj, ci je
   // v retazci "str_numbers".
   // Ak nie, funkcia Pos() vráti 0.
  if(str_Numbers.Pos(rstr_radius.SubString(i, 1))
  == 0)
  {
   return 0;
  }
 }
 // ak sme prešli úspešne všetky testy,
 // vrát true
 return 1;
}
```

6. Aktuvujeme okno unitu formulára a zmeníme kód udalosti (event) btn_setRadius-Onclick na:

```
void __fastcall
TForm1::btn_setRadiusClick(TObject *Sender)
{
  //football.setRadius ( edi_radius-
  >Text.ToDouble());
  football.setRadius ( edi_radius->Text);
}
```

7. Opät spustíme aplikáciu. Teraz môžeme napísat do editacného políčka pre polomer lubovolnú hodnotu a stlacit “Set Radius”. Takto už nedostaneme výnimku.

10.10 Dedicnost (Inheritance)

Obcas chceme pridat dalšiu funkcnost do triedy, no triedu nemožno menit, kedže sa v súcasnej podobe využíva. Môžeme definovat novú triedu, ktorá zdedí všetky funkcie existujúcej triedy a získa tiež nové funkcie alebo predefinuje už existujúce funkcie starej triedy. Pôvodná trieda sa nazýva "základná trieda" (base class), nasledovníka nazývame "odvodená trieda" (derived class).
Dedenie urcíme v definícii triedy tým, že pridáme dvojbodku za meno

triedy, nasledované špecifikátorom public a menom základnej triedy.

Príklad:

```
class B: public A
 {
  // ...
 };
```

Trieda A je základnou triedou triedy B.

10.10.1 Viacnásobná dedicnost (Multiple-inheritance)

Ak jedna trieda má dve alebo viac základných tried, tak hovoríme o viacnásobnej dedicnosti.

Príklad:

```
class C: public A, public B
{
  // ...
};
```

Triedy A a B sú základné triedy triedy C.

10.10.2 Chránené prvky (Protected-elements)

Okrem súkromnej (private) a verejnej (public) sekcie môže existovat i dalšia sekcia: chránená (protected). Táto má zmysel iba ak využívame dedicnost. Chránená sekcia základnej triedy je viditelná v odvodenej triede. Z prostredia mimo triedy prvky z chránenej sekcie nie sú prístupné.

Príklad:

```
class sphere
{
  private:
   float f_radius;
   float f_volumn;
  protected:
   float PI;
   int checkRadius(String radius);
  public:
   sphere();
   void setRadius (float radius);
   float getRadius (void);
   float getVolumn (void);
```

```
   void calculateVolumn (void);
   void setRadius (String radius);
   ~sphere();
};
```

10.10.3 Prekrývanie (Overriding)

V odvodenej triede môžeme zadefinovat funkciu, ktorá sa úplne líši od tej, ktorá je v triede základnej. Funkcia má definíciu zhodnú s tou v základnej triede, no kód je iný. Toto nazývame prekrývanie (overriding). Prekrývanie je podobné ako pretažovanie.

Príklad:
Trieda newsphere (nová gula), odvodená z triedy sphere (gula), definuje funkciu checkRadius (over polomer), ktorá je už definovaná v základnej triede sphere.

```
class newsphere : public sphere
{
  protected:
   int checkRadius(String radius);
};
```

Príklad 9: Dedicnost tried a prekrývanie funkcií

1. Pridáme nové okno unitu.
2. Uložíme ho a pomenujeme “newsphere”.
3. Otvoríme hlavickový súbor “newsphere.h”
4. Ako odkaz na základnú triedu (cross-refer) pridáme riadok:

```
#include "sphere.h"
```

5. Definujeme odvodenú triedu “newsphere” nasledovne:

```
class newsphere : public sphere
{
 private:
 protected:
  int checkRadius(String radius);

 public:
  newsphere();
  void setPi(String Pi);
```

```
  float getPi(void);
  void setRadius (String radius);
  ~newsphere();};
```

6. Uložíme hlavickový súbor
7. Otvoríme súbor sphere.h:
8. Do definície triedy sphere pridáme novú sekciu “protected” (chránené) a do nej presunieme premennú Pi a funkciu checkRadius.

```
protected:
  float PI;
  int checkRadius(String radius);
```

9. Aktivujeme okno “newsphere.ccp”.
10. Pridáme nasledujúci kód:

```
// konštruktor triedy newsphere
newsphere::newsphere()
{
// nie je co robit
}
//---------------------------------------------
// deštruktor triedy newsphere
newsphere::~newsphere()
{
// nie je co robit
}
//---------------------------------------------
// nová funkcia, ktorá umožní používatelovi
// nastavit hodnotu Pi
void newsphere::setPi(String Pi)
{
// zavolaj funkciu checkRadius triedy newsphere
if (checkRadius(Pi))
{
PI = Pi.ToDouble();
}
}
//---------------------------------------------
// nová funkcia, ktorá vráti hodnotu Pi
float newsphere::getPi(void)
{
return PI;
```

```
}
//---------------------------------------------
// prekrývajúca funkcia nastavujúca polomer
// (radius) gule (sphere)
void newsphere::setRadius (String radius)
{
// zavolaj funkciu checkRadius triedy
// newsphere
if (checkRadius(radius))
{
//zavolaj funkciu triedy sphere
sphere::setRadius(radius);
}
else
{
// zavolaj funkciu triedy sphere
sphere::setRadius(0);
}
}
//---------------------------------------------
// prekrývajúca funkcia preverujúca polomer
int newsphere::checkRadius(String rstr_radius)
{
// premenná na pocítanie výskytov ciarky
int colonFound = 0;

// najprv skontroluj hodnotu cez funkciu
// základnej triedy
if (!sphere::checkRadius(rstr_radius))
{
return 0;
}
else
{
// skontroluj každý znak retazca
for (int i = 1; i <= rstr_radius.Length();i++)
{
// ak je to ciarka, zvýš pocítadlo
if(rstr_radius.SubString(i, 1) == ",")
{
colonFound++;
}
```

```
}
}
// ak nájdeš viac ako jednu ciarku, vrát false
if (colonFound > 1) return 0;
// ak sme prešli všetky testy úspešne,
// vrát true
return 1;
}
```

11. Aktivujeme formulár.
12. Do formulára pridáme Edit (editacné políčko) a dve Buttons (tlacidlá).
13. Pomenujeme ich nasledovne:
    Edit1: edi_Pi
    Button1: btn_getPi
    Button2: btn_setPi
14. Aktivujeme okno unitu formulára.
15. Zmeníme príkaz include z sphere.h na newsphere.h:

```
#include "newsphere.h"
```

16. Zmeníme kód unitu nasledovne:

```
// deklaruj ukazovatel na objekt "football"
// triedy "sphere"
newsphere *football;
//sphere football;
//---------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
 // prirad smerník ku skutocnému
 // objektu newsphere
 football = new newsphere;
}
//---------------------------------------------
void __fastcall
TForm1::btn_setRadiusClick(TObject *Sender)
{
 //football.setRadius ( edi_radius-
 >Text.ToDouble());
 football->setRadius ( edi_radius->Text);
```

```
}
//---------------------------------------------
void __fastcall
TForm1::btn_getRadiusClick(TObject *Sender)
{
 edi_radius->Text = football->getRadius();
}
//---------------------------------------------
void __fastcall
TForm1::btn_getVolumnClick(TObject *Sender)
{
 edi_volumn->Text = football->getVolumn();
}
//---------------------------------------------
void __fastcall
TForm1::btn_setVolumnClick(TObject *Sender)
{
 football->calculateVolumn();
}
//---------------------------------------------
void __fastcall TForm1::btnClearAllClick(TObject
*Sender)
{
 edi_radius->Clear();
 edi_volumn->Clear();
 edi_Pi->Clear();
}
//---------------------------------------------

void __fastcall TForm1::btn_setPiClick(TObject
*Sender)
{
 football->setPi(edi_Pi->Text);
}
//---------------------------------------------

void __fastcall TForm1::btn_getPiClick(TObject
*Sender)
{
 edi_Pi->Text = football->getPi();
}
//---------------------------------------------
```

10.11 Polymorfizmus (Polymorphism)

Niekedy nie je jasné, ktorá funkcia sa má v programe použit: ci tá zo základnej alebo tá zodvodenej triedy. Toto môže nastat, ak používame smerníky na deklaráciu obsluhy (handle) triedy tak, ako sa to deje v príklade 9.
Inak povedané, kompilátor v case kompilácie nevie, objekt ktorej triedy je v nom priradený.
Problém je vyriešený, ak definujeme funkciu ako "virtuánu" ("virtual").

Príklad:
Použime naše triedy sphere (gula) a newsphere (nová gula).
Do smerníka priradíme základnú triedu

```
sphere *ball = new sphere;
```

Ak zavoláme funkciu

```
ball->setRadius(4);
```

všetko je v poriadku a zavolá sa správna funkcia.
Ak však priradíme smerník takto:

```
sphere *ball = new newsphere;
```

a potom zavoláme funkciu

```
ball->setRadius(4);
```

zavolá sa funkcia triedy sphere, hoci existuje rovnaká funkcia v triede newsphere a táto by mala byt zavolaná.
Problém môžeme riešit priradením smerníka tak, že napíšeme

```
newsphere *ball = new newsphere;
```

no v mnohých prípadoch toto v case návrhu (design-time) nevieme povedat.

Iné, lepšie riešenie tohoto problému je definícia setRadius ako virtuálnej (virtual) funkcie aspon v základnej triede sphere, v lepšom prípade aj vo všetkých odvodených triedach:

```
virtual void setRadius(float radius);
```

Ak to urobíme takto, zavolá sa správna funkcia, hoci kompilátor nevie, ktorý objekt bude použitý pocas behu programu (runtime).

10.11.1 Cvicenia

1. Použite triedu “birds” (vtáky) z cvicenia 4 kapitoly 4.7 a vytvorte

odvodenú triedu “predátory” (predátor).

2. Použite cvicenie 2 z kapitoly 4.7 a vytvorte odvodenú triedu, ktorá vie vytvorit poštovú adresu tak, ako bolo popísané v cvicení 2 kapitoly 3.5.

10.12 Zoznam retazcov (StringList)

Ciel tejto podkapitoly: Naucit sa co je to StringList a ako sa používa pri práci s textom. Taktiež sa naucit ako otvorit textový súbor a ako uložit text do súboru.

V tejto kapitole sa pozrieme na to, ako používat zoznamy retazcov (stringlist). Budeme cítat retazce zo zoznamu retazcov, nacítavat súbor do zoznamu retazcov a ukladat zoznam retazcov ako súbor.

10.12.1 Definícia

Stringlist je komponent, do ktorého ukladáme text. O retazcoch (Strings) môžeme uvažovat ako o riadkoch textu alebo ako o prvkoch pola. Zoznam retazcov (Stringlist) je teda text s aspon jedným riadkom. K jednotlivým riadkom pristupujeme ako k prvkom pola alebo k položkám, pomocou indexu. Retazce môžeme pridávat, odstranovat alebo môžeme vymazat celý zoznam retazcov (stringlist).
Stringlist je skrytý (hidden) komponent, co znamená, že ho do formulára nemôžeme pridat pomocou zoznamu komponentov.
Stringlist definujeme tak, že do funkcie, ktorá vyhradí (alokuje) pamät pre stringlist, pridáme nasledujúci kód:

```
TStringList *MyText;
```

Na priradenie vyhradenej pamäte k reálnemu zoznamu retazcov (stringlist) musíme urobit nasledujúce:

```
MyText = new TStringList;
```

10.12.2 Metódy (methods) a vlastnosti (properties)

Teraz môžeme použit zoznam retazcov (stringlist):

```
MyText->Add(“text novej položky”);
```

pridá retazec“ text novej položky ” do zoznamu retazcov (stringlist).

```
MyText->Delete(index);
```

odstáni retazec na pozícii “index”.

```
MyText->Clear();
```
vymaže celý zoznam retazcov.
```
MyText->Count;
```
spocíta kolko retazcov obsahuje zoznam retazcov (stringlist).
```
MyText->String[i];
```
precíta retazec na pozícii i, pocítat zacne od 0. Túto vlastnost môžeme použit aj na nastavenie hodnoty retazca na pozícii i.
Pomocou nasledujúceho kódu môžeme otvorit textový súbor a jeho riadky uložit do zoznamu retazcov (stringlist):
```
MyText->LoadFromFile(filename);
```
alebo uložit obsah zoznamu retazcov (stringlist) do súboru:
```
MyText->SaveToFile(filename);
```

10.12.3 Podobné komponenty

Existujú 2 komponenty, ktoré fungujú podobne ako zoznam retazcov (stringlist), pretože používajú tú istú základnú triedu - TStrings.
Prvým komponentom je TRichEdit, ktorý vjednotlivých riadkoch (Lines) ukladá text. Používatel môže pracovat stextom, ktorý je vkomponente RichEdit zobrazený.
Druhým komponentom je TStringGrid, ktorý používa bunky (Cells), rozdelené na stlpce a riadky. Aj tu môže používatel modifikovat text v bunkách.
V obidvoch prípadoch môžeme otvorit súbor, zobrazit jeho obsah v komponente a taktiež uložit zmeny, ktoré používatel urobil.

Príklad 10: Použitie zoznamu retazcov (Stinglist) na otvorenie súboru.

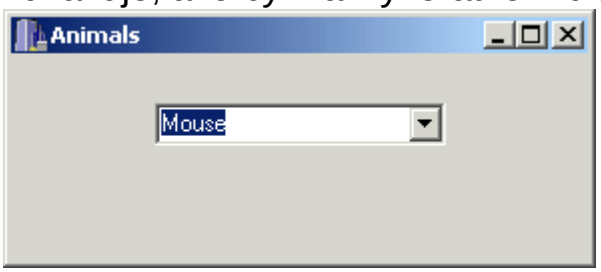
1. Vytvorte novú aplikáciu.
2. Zmente nadpis (caption) formulára na “Zvieratá” (animals).
3. Pridajte do formulára kombinovaný rámik (combo-box).
4. Pomenujte ho “cmb_Animals”, vymažte vlastnost Text.
5. V jednoduchom textovom editore vytvorte textový súbor a zapíšte don mená zvierat.
   Uložte ho do priecinku (folder) s projektom a dajte mu meno “animals.txt”.
6. Priradte udalost OnCreate k formuláru a k funkcii FormCreate pridajte nasledujúci kód:

```
//definuje smerník ako StringList
TStringList *stl_Animals;

//priradí smerník k reálnemu zoz. retazcov
```

```
stl_Animals = new TStringList;

//cíta textový súbor a riadky ukladá do
stringlist
//textový súbor obsahuje pod sebou mená zvierat
stl_Animals->LoadFromFile("animals.txt");
//každé zo zvierat z textového súboru pridá do
kombinovaného rámika
for (int i = 0;i<stl_Animals->Count;i++)
{
  cmb_Animals->Items->Add(stl_Animals-
  >Strings[i]);
}
// zruší priradenie smerníka
delete stl_Animals;#include "sphere.h"
```

Ak spustíme aplikáciu, otvorí sa súbor, precítajú sa mená zvierat a vyplní sa kombinovaný rámik.

Nasledujúci obrázok ukazuje, ako by mal vyzerat formulár:



10.12.4 Dialógové okno File

V tomto prípade môže dôjst k výnimke (exception), ak nie je nájdený súbor s menami zvierat.
Môžeme cez to prejst ak použijeme dialógové okno "Otvorit súbor" (open-file-dialog), ktoré je jednoducho komponentom TOpenDialog.

Príklad 11: Použitie dialógovího okna File na otvorenie súboru.

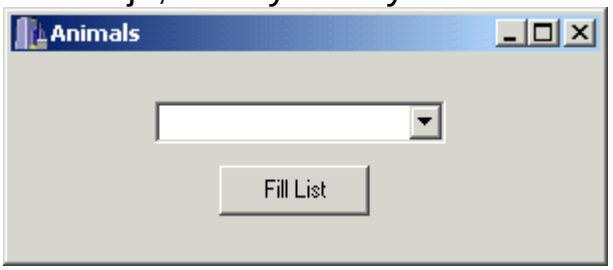
1. Pridajte tlacidlo do formulára z príkladu 10.
2. Pridajte komponent TOpenDialog do formulára. Môžete ho umiestnit kdekolvek, lebo pocas behu programu nie je viditelný.
3. Zmente meno komponentu opendialog na “dlg_OpenFile”.
4. Pridajte kombinovaný rámik do formulára.
5. Pomenujte ho “cmb_Animals”, vymažte vlastnost Text.

6. Pridajte tlacidlo do formulára.
7. Pomenujte ho “bnt_FillList”, nastavte jeho nadpis (caption) na “Vypln zoznam!”
8. Presunte celý kód z funkcie FormCreate do obsluhy udalosti OnClick tlacidla a pridajte nejaké nové riadky podla nasledujúceho:

```
void __fastcall TForm1::bnt_FillListClick(TObject *Sender)
{
    // definuje smerník ako StringList
    TStringList *stl_Animals;

    // nastavuje filter pre súbory v dialógovom okne Otvorit tak
    // aby boli zobrazené len textové súbory
    dlg_OpenFile->Filter = "Text files (*.txt)|*.TXT";
    // otvorí dialógové okno File
    if(dlg_OpenFile->Execute())
    {
        // priradí smerník k reálnemu zoz. retazcov
        stl_Animals = new TStringList;

        // cíta textový súbor otvorený v dialógu Open
        // riadky ukladá do zoz. retazcov
        // textový súbor obsahuje pod sebou mená zvierat
        stl_Animals->LoadFromFile(dlg_OpenFile->FileName);
        // pridá každé zo zvierat v textovom súbore do kombinovaného rámika
        for (int i = 0;i<stl_Animals->Count;i++)
        {
            cmb_Animals->Items->Add(stl_Animals->Strings[i]);
        }
        // zruší priradenie smerníka
        delete stl_Animals;
    }
}
```

teraz môžeme urcit, kde sa má ukladat súbor, ktorý chceme otvorit pre mená zvierat.

Nasledujúci obrázok ukazuje, ako by mal vyzerat formulár:



10.12.5 Cvicenia

1. Vytvorte aplikáciu s komponentom TMemo. Na spracovanie súboru použite jeho vlastnost Lines. Vytvorte ponuky “File Open” a “File Save” na otváranie a ukladanie súborov.
2. Vytvorte aplikáciu s komponentom TImage. Na spracovanie súboru použite jeho vlastnost Picture. Vytvorte ponuku “File Open” na otváranie BMP-súborov.

10.13 Filestreams (súborové prúdy údajov)

Ciel tejto podkapitoly: Zoznámit sa s myšlienkou filestreams a ich použití pri vstupných a výstupných súboroch.

V tejto kapitole získame prehlad o filestreams (súborových prúdoch údajov). Budeme ich používat na cítanie (input) a zapisovanie (output) do súborov.

10.13.1 Súborové operácie s prúdmi

Ak je súbor otvorený, musíme rozlišovat medzi textovým a binárnym režimom (mode).

10.13.1.1 Textový režim

Textový režim znamená, že pocas cítania údajov sa sekvencia CR/LF transformuje na znak '\n'. A naopak, pocas zapisovania sa znak '\n' transformuje na sekvenciu CR/LF.

10.13.1.2 Binárny režim

V binárnom režime nedochádza k automatickej transformácii. Znaky sú cítané a zapisované bez zmien.

10.13.1.3 Triedy „ifstream“ a „ofstream“

Aby sme mohli používat file streams (súborové prúdy údajov), musíme pripojit (bound) hlavickový súbor <fstream.h>.
Pomocou tried ifstream a ofstream sa uskutocnuje (established) priame spojenie so súborom. Ifstream otvára súbor na cítanie (input), ofstream otvára súbor na zápis (writing) - obidve štandardne v textovom režime!

Príklad: Kopírovanie súboru v textovom režime

```
#include <fstream.h>


int main (void)
{
 char ch;

 ifstream inf ("FILE.IN");
 ofstream outf ("FILE.OUT");

 if (!inf) cerr << "Chyba vo vstupnom súbore";
 if (!outf) cerr << "Chyba vo výstupnom súbore";

 while (inf.get (ch))
 outf.put (ch);
}
```

Vo vyššie uvedenom príklade sú znaky cítané zo súboru FILE.IN a zapisované do súboru FILE.OUT, až kým už nie je možné cítat žiaden znak zo súboru FILE.IN.

File streams (súborové prúdy údajov) sú vstupno/výstupné prúdy ako cin a cout. Preto sa tu používajú operátory pre prúdy ako "<<" a ">>". Riadky 14 a 15 z predchádzajúceho príkladu by sme mohli upravit takto:

```
while (inf >> ch)
  outf << ch;
```

Táto metóda má však nevýhodu, pretože sa ignorujú konce riadkov (LF - line feed). Ak má byt súbor otvorený v binárnom režime, tak konštruktory pre vstupný a výstupný súbor umožnujú deklarovat file stream (súborový prúd údajov) a potom ho spojit so súborom. Dalšie režimy práce so

súborom (mode bits) môžeme uviest pri otváraní súboru.

Príklad: Otvorenie súboru v binárnom režime

```
// generuj input-stream
ifstream inf;

...

// uskutocni spojenie
inf.open ("FILE.IN", ios::binary);

...

inf.close();
```

Vo vyššie uvedenom príklade otvárame prúd (stream) pre súbor FILE.IN v binárno režime. Bit pre režim práce so súborom (file mode bit) nie je povinný. Môžeme použit režim (mode bit) z nasledujúcej tabulky.

MODE-BIT	ACTION / AKCIA
`ios::app`	Pridávanie údajov na koniec súboru
`ios::ate`	Po otvorení presun na koniec súboru
`ios::in`	Otvorenie na cítanie (implicitne pre ifstream)
`ios::out`	Otvorenie na zápis (implicitne pre ofstream)
`ios::binary`	Otvorenie súboru v binárnom režime
`ios::trunc`	Ak existuje, obsah súboru sa prepíše. (implicitne ak je uvedený ios::out is, a nie je uvedený ios::app ani ios::ate)
`ios::nocreate`	Chyba pri otváraní, súbor neexistuje
`ios::noreplace`	Chyba pri otváraní, súbor existuje a nie je uvedený ios::app ani ios::ate

10.14 Obsluha chýb (Error-handling) a výnimky (Exceptions)

Ciel tejto podkapitoly: Naucit sa narábat s výnimkami (exceptions).

V tejto kapitole budeme používat rezervované slová try a catch na zachytenie výnimiek, ktoré sa objavia.

10.14.1 Definícia

Niekedy sa stane, že funkcia použije nevhodné hodnoty. Možno sú príliš velké alebo príliš malé, možno nie sú ocakávaného typu alebo dokonca chýbajú. Vtedy môžeme použit príkaz "try", aby aplikácia vyskúšala spustit kód. Ak to zlyhá, zachytí (catch) generovanú výnimku a urobí nieco iné, napr. zobrazí upozornenie
Vo všeobecnosti sa používa takýto príkaz:

```
try
{
    //jeden alebo viac riadkov kódu
    //ktoré treba vyskúšat
    //ak sú ok, tak ich spusti
}
catch(…)
{
    //kód co robit, ak sa vyskytne chyba
}
```

Príklad 12: Obsluha chýb pri otváraní súboru

1. Použite príklad 10.
2. Zmente kód funkcie FormCreate nasledovne:

```
void    __fastcall    TForm1::FormCreate(TObject
*Sender)
{
    TStringList *stl_Animals;

    //tu zacína príkaz “try”
    try
    {
        stl_Animals = new TStringList;
        stl_Animals->LoadFromFile("animal.txt");
    }
    //ak sa vyskytne výnimka, v dialógovom okne
    //sa zobrazí upozornenie na chybu
    //a zvyšok funkcie sa nevykoná
    catch(...)
    {
        Application->MessageBox("Súbor
        'animal.txt'   sa   nenašiel",   "Chyba",
        MB_OK);
```

```
        return;
    }
    for (int i = 0;i<stl_Animals->Count;i++)
    {
        cmb_Animals->Items->Add(stl_Animals-
        >Strings[i]);
    }
    delete stl_Animals;

}
```

3. Ak aplikáciu spustíme, nastane výnimka, pretože súbor “animal.txt” neexistuje. Síce nastane výnimka, ale výsledok nie je ocakávaný. Existujú dva spôsoby ako testovat aplikáciu:
   a) spustit “make” alebo “build” a vykonat EXE-súbor
   b) íst do ponuky Tools – Debugger Options, zapísat (register) “Language Exceptions”, a deaktivovat “Stop on Delphi Exceptions”. Opät spustit aplikáciu.

10.14.2 Cvicenia

1. Skontrolujte, ci môžete k cviceniam z predchádzajúcej kapitoly pridat obsluhu chýb, napr. pri otváraní súboru, ukladaní do súboru, odovzdávaní (exchange) parametrov nesprávneho typu a pod.

Kapitola 11: Nadstavbový modul 1 Knižnica vizuálnych komponentov (VCL)

Ciel tejto kapitoly: Naucit sa co je to VCL a ako ju použit.

V tejto kapitole budú vysvetlené tieto témy:

1. Borland Visual Component Library (Knižnica vizuálnych komponentov, VCL).
2. Skrátený prehlad VCL.
3. Štruktúra VCL.
4. Triedy aplikácie a formulára.
5. Trieda retazca.
6. Trieda množiny.
7. Komponentové triedy

11.1 Úvod

Prostredie Borland C++ Builder ponúka programátorovi knižnicu tried (framework) nazývanú **VCL** (Visual Component Library, Knižnica vizuálnych komponentov).
Na pochopenie pojmu a užitocnosti frameworku môžeme tento pojem porovnat so stavbou domu. Jestvujú dve možnosti:
Stavat dom od základu, tehlicka ku tehlicke, škridla ku škridle, atd.
Stavat ho z prefabrikovaných dielcov.

Niet pochýb, ktorú možnost si zvolíme pri úlohe z IT.
Framework je sada prefabrikovaných modulov, ktoré možno okamžite použit a upravit podla potrieb programátora.
Každé moderné prostredie má svoj vlastný framework: napríklad Visual C++ používa "MFC" (Microsoft Foundation Class Library, Knižnica základných tried Microsoftu).
VCL vzniklo ako súcast **Delphi**, základ všetkých súcasných prostredí Borlandu. Delphi používajú **Object Pascal**, ktorý je odvodený z jazyka **Pascal**; dôsledkom toho je **VCL** napísaná v jazyku **Object Pascal** a upravená pre **C++ Builder**. **VCL** je jadro prostredia Borland C++ Builder.
Ako môžeme nájst **VCL v** prostredí Borland C++ Builder?
Celkom lahko: každá operácia v BCB je spätá s VCL; vždy, ked vkladáme do alebo manipulujeme s formulárom, tlacidlom, editacným políckom, atd., vytvárame novú inštanciu komponentu a **VCL** ju uskutocnuje. Ked vložíme tlacidlo do formulára, vytvoríme inštanciu triedy **TButton**, ktorá je

štandardne pomenovaná **Button1**. **Button1** zdedil charakteristiku **VCL** komponentu **TButton** a vdaka práci programátora získava vlastnú osobnost a správanie.

11.2 Strucný prehl ad VCL

VCL má hierarchickú štruktúru ako pyramída. Napriek svojmu pomenovaniu komponenty Knižnice Vizuálnych Komponentov nie sú výlucne vizuálne; napríklad **TTimer** (komponent generujúci pravidelne casovanú udalost (event)) nie je vizuálny.
VCL komponenty (s menitelnými vlastnostami (properties)) môžu byt použité ako vo fáze vývoja (design time), tak i pocas behu programu (runtime).

Ked vložíme komponent do formulára, využívajúc tak možností vizuálneho programovania, ktoré nám prostredie ponúka, BCB automaticky zapíše príslušný kód v C++. Ak ale chceme vytvorit a aktivovat komponent pocas behu programu, potom je nutné napísat kód v C++ vlastnorucne tak, ako v nasledujúcom príklade:

```
TOpenDialog* dlg = new TOpenDialog(this);
dlg->Title = "Otvor nový súbor";
dlg->Execute();
```

Všetky **VCL** objekty sa alokujú dynamicky. Ich vytvorenie sa vykoná cez operátor **new**.

Jedinou výnimkou sú štruktúry operacného systému Windows, ktoré VCL iba reprodukuje, ako **TPaint**:

```
TPaint Paint(10, 10, 200, 200);
```

V tomto prípade sa alokácia uskutocní staticky a parametre udávajú pozíciu komponentu.

Triedy **VCL** nemajú štandardné parametre funkcií.
Ak napríklad chceme zobrazit správu, musíme definovat: text správy, názov okna a tlacidlá, ktoré sa majú zobrazit (príznaky (flags)):

```
Application->MessageBox("Toto je správa",
"Správa", MN_OK);
```

V iných prostrediach:

```
MessageBox("Toto je správa")
```

11.3 Štruktúra VCL

Nie je nutné poznat naspamät všetky detaily štruktúry **VCL**; ale je dôležité poznat štruktúru ako takú.
V nasledujúcom si ukážeme stromovú ponuku **VCL**.

```
TObject;
-TPersistent;
-  -  TComponent;
-  -  -  [nevizuálne komponenty];
-  -  -  -  TTimer, atd.;
-  -  -  [vizuálne komponenty];
-  -  -  -  TControl;
-  -  -  -  -  TGraphicControl;
-  -  -  -  -  -  TShape, atd.;
-  -  -  -  -  -  -  TWinControl;
-  -  -  -  -  -  -  -  TButton, etc.
```

Kedže sme sa s ním už tolkokrát stretli v **TButton**, **TEdit**, **TLabel**... atd., urcite sme sa s prefixom "**T**" spriatelili.
Na vrchole pyramídy sa nachádza **TObject**, predok (ancestor) všetkých tried; toto je zjavne ocakávané, kedže sa stretávame s objektovo–orientovaným programovaním.
TPersistent zabezpecuje pre ostatné komponenty ukladanie do súborov a pamäti. **TComponent** je základná trieda všetkých komponentov; nevizuálne sa odvodzujú priamo z **TComponent**, vizuálne komponenty však z nasledujúcej triedy **TControl**.
Základná trieda pre grafické komponenty a pre triedy ovládacích prvkov (controls) Windows je **TControl.**

Ked vložíme formulár, jeho komponenty a jeho ovládacie prvky do projektu, v BCB kóde sa vytvoria príslušné triedy a ukazovatele (pointers); meno premennej ukazovatela sa odvodí z vlastnosti **Name** samotného komponentu.

11.4 Triedy aplikácie a formulára

Vo **VCL** sa trieda aplikácie (**TApplication**) a formulára (**TForm**) odvodzujú

priamo z **TComponent**.

TApplication zapúzdruje (encapsulate) základné operácie programu pre OS Windows a obsluhuje cinnosti ako:
správa ikony aplikácie;
zobrazenie on-line pomocníka;
zobrazenie správ.
Niektoré vlastnosti **TApplication** (**Icon** (ikona), **HelpFile** (súbor pomocníka) a **Title** (názov okna)), možno nastavit na stránke **Application** (aplikácia) okna **Project Options** (možnosti projektu).

TForm zapuzdruje základné operácie akéhokolvek okna a tak, ako sme už videli v predchádzajúcich kapitolách, obsahuje velký pocet vlastností, metód, udalostí.

11.5 Trieda retazca

VCL obsahuje triedu **AnsiString**, jednoducho premenovanú triedu **String** (retazec), ktorá emuluje typ **Pascal long string** (ako sme si už spomínali, **VCL** je napísaná v jazyku **Object Pascal**). Táto trieda sa používa vo väcšine **VCL** komponentov a umožnuje všetky druhy operácií s retazcami.

Niektoré komponenty, ktoré boli pôvodne napísané v **Delphi**, vyžadujú triedu **SmallString**.

11.6 Trieda mno□iny

Niektoré typy údajov (data types) existujú v jazyku **Pascal** a neexistujú v **C++** (a naopak); jedným z nich je funkcia **set** (množina). Množiny sa hojne využívajú vo **VCL**, takže je dôležité rozumiet tomu, ako fungujú.
Na pochopenie toho, co to **množina** je, môžeme použit vlastnost **Style** (štýl) triedy **TForm**. Táto vlastnost môže obsahovat jednu alebo viacero z nasledujúcich hodnôt: **fsBold**, **fsItalic**, **fsUnderline** a **fsStrikeout**.
Ako už bolo spomenuté, **C++** nezahrna **množiny**, riešením teda bola **šablóna** (template) triedy, jednoducho nazvaná **set,** emulujúca tento typ.
Obycajne sa síce atribúty (attributes) textu nastavujú pocas fázy vývoja, ale ich nastavovanie môže byt potrebné aj pocas behu programu (runtime).
Uvažujme prípad, že chceme pridat atribúty bold (hrubé písmo) a italic (kurzíva) do štýlu textu objektu (**TEdit**).

```
TFontSyles Styles;
Styles << fsBold << fsItalic;
```

```
Edit1->Font->Style = Styles;
```

Operátor **<<** sa využíva na pridávanie prvkov do množiny; je tiež potrebné vykonat priradenie množiny do vlastnosti **Font->Style** komponentu na uplatnenie požadovaného štýlu.

Ak teraz chceme odstránit **fsItalic** z množiny, a teda aj z editacného polícka, musíme napísat nasledujúci kód:

```
Styles >> fsItalic;
Edit1->Font->Style = Styles;
```

Operátor **>>** sa používa na odstranovanie prvkov z množiny.

Na zistenie, ci je element prvkom množiny, povedzme napríklad **fsBold**, použijeme metódu **Contains()**:

```
Bold = Edit1->Font->Style.Contains(fsBold);
if (Bold) do something();
```

Niekedy sa môže stat, že potrebujeme úplne vyprázdnit množinu. Vtedy použijeme metódu **Clear**:

```
Edit1->Font->Style.Clear();
```

11.7 Komponentové triedy

Teraz si ukážeme zoznam tried odvodených z **TComponent**.

11.7.1 Triedy štandardných ovládacích prvkov

Zo štandardných ovládacích prvkov Windows sme sa už stretli s **TEdit**, **TBbutton**, atd.. Do tejto skupiny však patria aj tieto:
TPanel, kontajner (container) pre panely nástrojov (tool bars), stavové riadky (status bars) a iné;
TBitBtn a **TSpeedButton**, obrázkové tlacidlá;
TImage, na zobrazovanie obrázkov vo formulári;
TBevel, na vytvorenie štvorcov a ciar vo formulári;
TStringGrid a **TDrawGrid**, na prezentáciu informácií v podobe tabuliek.

11.7.2 Triedy volitelných (custom) ovládacích prvkov

VCL obsahuje skupinu tried, ktorá zahrna hlavné ovládacie prvky pre **32 bitové** aplikácie systému **Windows**, napríklad **TRichEdit**; niektoré z týchto tried sú pomerne zložité a budeme sa s nimi zaoberat neskôr.

11.7.3 Triedy štandardných dialógových okien (Dialog box)

Skupina tried, ktoré sa vztahujú na dialógové okná (dialog windows), obsahuje hlavné dialógové okná systému Windows. Do tejto skupiny sa radia nasledujúce: **TOpenDialog**, **TSaveDialog**, **TOpenPictureDialog**, **TSavePictureDialog**, **TFontDialog**, **TColorDialog**, **TPrintDialog**, **TPrinterSetupDialog**, atd..
Všetky komponenty tejto skupiny sú nevizuálne. Dôsledkom toho ich nemožno zobrazit pocas programovania.

11.7.4 Systémové triedy

Skupina systémových komponentov zahrna tak vizuálne, ako aj nevizuálne triedy. K tejto skupine prislúchajú:- **TTimer**, systémový casovac (timer) prostredia Windows; jediná udalost ním tvorená je **OnTimer**, ktorá je štandardne generovaná (triggered) vždy, ked sa casovac spustí. Interval medzi udalostami **OnTimer** sa nastavuje pomocou vlastnosti komponentu **Interval**. **TTimer** je nevizuálny komponent;
- **TMediaPlayer**, na reprodukciu multimediálnych súborov;
- **TPaintBox**, na kreslenie.

11.7.5 GDI triedy

GDI (Graphics Device Interface, Rozhranie Grafického Zariadenia) triedy slúžia na zobrazovanie obrázkov a textov v oknách systému Windows; nie sú viazané na žiadny konkrétny komponent, no mnohé komponenty využívajú inštancie týchto tried ako vlastnosti. Napríklad vlastnost **Font**, ktorú má mnoho komponentov, je inštanciou triedy **TFont**.
Medzi **GDI** triedami nájdeme:
TCanvas, na kreslenie a zobrazovanie obrázkov a textu.
TBrush, reprezentujúci oblast, ktorá môže byt farebne pokreslená alebo môže priamo zobrazit obrázok.
TBitmap, spravujúci rastrové obrázky (bitmap),
TFont sa stará o vlastnosti písma (font).

11.7.6 Triedy utilít (utility)

V skupine tried utilít nájdeme:
TIniFile na cítanie a zápis konfiguracných súborov systému Windows (**.INI**);
TRegistry a **TRegKeyInfo** na správu konfiguracných registrov (registry) systému Windows;
TStringList, na cítanie a ukladanie retazcov.
TList, ktorý vytvára polia objektov lubovolného typu. Pole sa automaticky upravuje pri pridávaní a odstranovaní prvkov;
TStream, TFileStream, TmemoryStream a TResourceStream, na cítanie a zápis do prúdov (streams).

11.7.7 Databázové triedy

Ak potrebujeme komunikovat s databázou, máme k dispozícii osobitnú skupinu databázových tried. Tieto budú predmetom nasledujúcej kapitoly.

11.8 Práca s údajovým modulom VCL

Ciel tejto podkapitoly: Naucit sa pracovat s údajovým modulom VCL (VCL data module)

V tejto podkapitole si ozrejmíme nasledujúce témy:

1. Vytváranie VCL databázovej aplikácie
2. Prehlad architektúry databázovej aplikácie
3. Vytvorenie novej VCL aplikácie
4. Nastavovanie komponentov prístupu k údajom (data)
5. Nastavovanie spojenia s databázou
6. Nastavovanie jednosmerného súboru údajov (unidirectional dataset)
7. Nastavenie správcovského (provider), klientského (client) súboru údajov a zdroja údajov
8. Navrhovanie používatelského rozhrania
9. Vytváranie mriežky (grid) a navigacného panelu
10. Pridávanie tlacidla
11. Písanie obsluhy udalosti (event handler)
12. Písanie obsluhy udalosti príkazu Aktualizuj! (Update Now!)
13. Písanie obsluhy udalosti FormClose (zatvorenie formulára)

11.8.1 Vytváranie VCL databázovej aplikácie

Toto cvicenie nás bude viest vytvorením VCL C++ aplikácie, ktorá nám

umožní zobrazovat a aktualizovat ukážkovú databázu zamestnancov. Je písané pre tie edície BCB, ktoré obsahujú databázové komponenty (Professional Edition, Profesionálna edícia). Tieto komponenty vytvárajú prístup k databáze. Využívajú však také vlastnosti a crty komponentov, ktoré Personal edition (osobná edícia) neobsahuje. Naviac, na úspešné dokoncenie cvicenia, musíme mat nainštalovaný InterBase.
Ak využijeme štandardnú (default) inštaláciu BCB, tak bude interBase jej súcastou.

11.8.2 Prehlad architektúry databázovej aplikácie

Architektúra databázovej aplikácie sa môže zdat na prvý pohlad velmi zložitá. No využitie viacerých komponentov zjednodušuje vývoj a údržbu vlastnej databázovej aplikácie.
Databázová aplikácia sa skladá z troch hlavných castí: používatelské rozhranie, skupina komponentov prístupu k údajom a databáza samotná. v tomto cvicení vytvoríme dbExpress databázovú aplikáciu. Iné databázové aplikácie majú rovnakú architektúru.
Používatelské rozhranie obsahuje ovládacie prvky so znalostou údajov (data-aware controls) ako je mriežka, takže používatel môže editovat (edit) údaje a ukladat (post) ich do databázy. Komponenty prístupu k údajom zahrnajú zdroj údajov, klientský súbor údajov (dataset), správcu (provider) údajov, jednosmerný súbor údajov a komponent pripojenia. Zdroj údajov má úlohu kanálu medzi používatelským rozhraním a klientským súborom údajov. Klientský súbor údajov je srdce aplikácie, kedže obsahuje množinu záznamov zo základnej databázy, ktoré sú uložené (buffered) vpamäti. Správca prenáša údaje medzi klientským súborom údajov a jednosmerným súborom údajov, ktorý vyberá (fetch) údaje priamo z databázy. Nakoniec komponent spojenia vytvára spojenie s databázou. Každý typ jednosmerného súboru údajov používa iný typ komponentu spojenia.

11.8.3 Vytváranie novej VCL aplikácie

Skôr, ako zacneme cvicenie, vytvorme priecinok na ukladanie zdrojových súborov. Potom vytvorme a uložme nový projekt.

1. Vytvorme priecinok s názvom **Probiq** na ukladanie súborov projektu, ktoré vytvoríme pocas práce na tomto cvicení.
2. Zacneme s novým projektom. Na jeho vytvorenie vyberieme z ponuky File|New|Application (Súbor|Nový|Aplikácia).
3. Zvolíme File|Save All (Súbor|Uložit všetko) a uložíme tak všetky súbory na disk. Ked sa objaví dialógové okno ukladania, presunieme

sa do nášho priecinku **Probiq** a uložíme každý súbor pod jeho štandardným menom.

V dalšom môžeme hocikedy uložit prácu volbou File|Save All. Ak sa rozhodneme nedokoncit toto cvicenie na jediné posedenie, tak si ho môžeme otvorit z uloženej verzie cez File|Reopen (Súbor|Znovu otvor) a vybrat si naše cvicenie zo zoznamu.

11.8.4 Nastavovanie komponentov prístupu k údajom

Komponenty prístupu k údajom reprezentujú tak údaje (súbory údajov), ako aj komponenty, ktoré spájajú tieto súbory údajov s inými castami našej aplikácie. Každý z týchto komponentov prístupu k údajom ukazuje na nasledujúci nižší komponent. Napríklad, zdroj údajov ukazuje na klientský súbor údajov, klientský súbor údajov ukazuje na správcu, atd.. Preto, ked nastavujeme naše komponenty prístupu k údajom, pridávame komponenty v správnom poradí zacínajúc od vrstvy spojenia (connection layer).
V dalších témach budeme pridávat databázové komponenty na vytvorenie spojenia s databázou, jednosmerného súboru údajov, správcu, klientského súboru údajov a zdroja údajov. Potom vytvoríme používatelské rozhranie aplikácie. Tieto komponenty sa nachádzajú v okne zoznamu komponentov (component list).

Tip: Dobrá myšlienka je oddelit používatelské rozhranie na vlastnom formulári a umiestnit komponenty prístupu k údajom do údajového modulu. My však v našom príklade, aby sme veci urobili co najjednoduchšie, umiestnime používatelské rozhranie a všetky ostatné komponenty na ten istý formulár.

11.8.5 Nastavovanie spojenia s databázou

Stránka dbExpress obsahuje sadu komponentov, ktoré poskytujú rýchly prístup k SQL databázovým serverom.
Potrebujeme pridat komponent spojenia, aby sme sa mohli pripojit na databázu. Typ komponentu spojenia závisí na tom, aký typ komponentu súboru údajov používame. V tomto cvicení použijeme komponenty TSQLConnection a TSQLDataSet.
Ako pridat komponent spojenia dbExpress:

1. Zvolme komponent TSQLConnection zo zoznamu komponentov (vid kapitola 4.4 - ako sa dostaneme ku zoznamu komponentov). Komponent dostane štandardné meno SQLConnection1.

2. V inšpektore objektov (object inspector) nastavíme jeho vlastnost ConnectionName na IBLocal (ide o rozbalovací (drop-down) zoznam).
3. Nastavíme vlastnost LoginPrompt na false (nepravda). (Nastavením tejto vlastnosti na false zabránime výzve na prihlásenie pri každom prístupe do databázy.)

   TSQLConnection komponent má editor spojenia (connection editor). Na zobrazenie editora spojenia asociovaného s komponentom si zvolíme zobrazenie stromu objektov (object tree view) z ponuky okien (windows menu); vyberieme si komponent spojenia (SQLConnection1); otvoríme kontextovú ponuku a zvolíme si editovanie vlastnosti spojenia (Edit Connection Property).

   Editor spojenia používame na volbu konfigurácie komponentu TSQLConnection alebo na editovanie spojení uložených vsúbore dbxconnections.ini. Všetky zmeny vykonané vdialógovom okne sú zapísané práve do tohto súboru pri kliknutí na OK. Nadôvažok, pri kliknutí na OK sa zvolenému spojeniu priradí hodnota vlastnosti ConnectionName komponentu SQL spojenia.

4. V editore spojenia špecifikujeme cestu (pathname) k databázovému súboru nazvanému employee.gdb, podla umiestnenia v našom systéme. V tomto cvicení sa pripojíme na ukážkovú databázu InterBase, employee.gdb, ktorú nám poskytuje C++Builder. Štandardná inštalácia InterBase umiestnuje employee.gdb do C:\Program Files\Common Files\Borland Shared\Data. Túto cestu zadajme do pola hodnoty asociovaného s klúcom s názvom Database.

5. Skontrolujme, ci polícka User_Name a Password obsahujú prípustné hodnoty. Ak sme nezmenili štandardné hodnoty, tak nepotrebujeme menit polícka. Ak je prístup k databáze spravovaný niekým iným ako nami, tak možno budeme musiet dostat na prístup do databázy používatelské meno (username) a prístupové heslo.

6. Ked dokoncíme kontrolu a editáciu políck, klikneme na OK, cím zatvoríme editor spojenia a uložíme vykonané zmeny.

   Tieto zmeny sa zapíšu do súboru dbxconnections.ini a zvolenému spojeniu sa priradí ako hodnota vlastnost ConnectionName komponentu SQL Connection.

7. Z ponuky vyberieme File|Save All a uložíme celý náš projekt.

11.8.6 Nastavovanie jednosmerného súboru údajov

Aj najjednoduchšia databázová aplikácia používa súbor údajov na prístup k informáciám v databáze. v dbExpress aplikáciách používame jednosmerný súbor údajov (unidirectional dataset). Jednosmerný súbor údajov číta údaje z databázy, ale údaje neaktualizuje.

Ako pridáme jednosmerný súbor údajov:

1. Zo zoznamu komponentov si zvolíme TSQLDataSet.
2. Cez inšpektora objektov nastavíme jeho vlastnost SQLConnection na SQLConnection1 (databázové spojenie, ktoré sme v predchádzajúcom vytvorili).
3. Nastavíme vlastnost CommandText na “select * from SALES”, cím špecifikujeme príkaz, ktorý má súbor údajov vykonat.
4. Nastavíme Active na true (pravda), cím otvoríme údajový súbor.
5. Zvolíme File|Save All a uložíme projekt.

11.8.7 Nastavenie správcovského a klientského súboru údajov a zdroja údajov

Správcovské komponenty umožnujú klientským súborom údajov získavat údaje z iných súborov údajov. Správca dostáva požiadavky na údaje (data requests) z klientského súboru údajov, vyberá údaje, balí ich (package) a údaje vracia klientskému súboru údajov. Správca získava aktualizácie z klientských súborov údajov a tie zavádza na databázový server.

Ako pridat správcu:

1. Zo zoznamu komponentov si zvolíme komponent TDataSetProvider.
2. Pomocou inšpektora objektov nastavíme vlastnost správcu DataSet na SQLDataSet1.

Klientský súbor údajov ukladá (buffers) svoje údaje do pamäti. Takisto si ukladá aktualizácie, ktoré sú zaslané do databázy, do vyrovnávacej pamäti (cache). Taktiež môžeme využit klientské súbory údajov na poskytovanie údajov komponentom ovládacích prvkov so znalostou údajov v používatelskom rozhraní, využívajúc komponent zdrojových údajov.

Ako pridat klientský súbor údajov:

1. Zo zoznamu komponentov vyberieme komponent TClientDataSe.

2. Vlastnost ProviderName nastavíme na DataSetProvider1.
3. Nastavíme vlastnost Active na true (pravda), cím umožníme prenos údajov do našej aplikácie.

Zdroj údajov sa prepája s klientským súborom údajov pomocou ovládacích prvkov so znalostou údajov. Každý ovládací prvok so znalostou údajov musí byt asociovaný s komponentom zdroja údajov, co mu umožnuje manipuláciu a zobrazovanie údajov. Rovnako, všetky súbory údajov musia byt asociované s komponentom zdroja údajov, aby bolo možné zobrazovat ich údaje ci manipulovat s nimi cez ovládacie prvky so znalostou údajov vo formulári.

Ako pridat zdroj údajov:

1. Zo zoznamu komponentov vyberieme komponent TDataSource.
2. Vlastnost zdroja údajov DataSet nastavíme na ClientDataSet1.
3. Zvolíme File|Save All, cím uložíme celý projekt.

Až doposial sme pridávali do našej aplikácie nevizuálnu infraštruktúru databázy. Teraz potrebujeme navrhnút a vytvorit používatelské rozhranie.

11.8.8 Vytváranie používatelského rozhrania

Teraz potrebujeme do aplikácie pridat vizuálne ovládacie prvky tak, aby jej používatelia mohli zobrazovat, editovat a ukladat údaje na disk. V zozname komponentov sa nachádza sada ovládacích prvkov so znalostou údajov, ktoré pracujú s údajmi v databáze a vytvárajú používatelské rozhranie. My zobrazíme databázu pomocou mriežky, pridáme niekolko príkazov a navigacný panel.

11.8.9 Vytváranie mriežky a navigacného panelu

Ako vytvoríme používatelské rozhranie aplikácie:

1. Zacneme pridaním mriežky do formulára. Zo zoznamu komponentov si zvolíme komponent TDBGrid a umiestnime ho do formulára.

2. Nastavíme vlastnosti komponentu DBGrid tak, aby sme mriežku ukotvili (anchor).

3. V **inšpektore objektov** nájdeme vlastnost **Anchors**, otvoríme kontextovú ponuku, zvolíme **Expand** (rozvinút) a nastavíme hodnoty ciastkových vlastností (subproperties) **akLeft, akTop, akRight** a

akBottom na hodnotu **True.**

4. Zarovnáme (align) mriežku s dolným okrajom (bottom) formulára nastavením vlastnosti Align na alBottom. Mriežku tiež môžeme zväčšit bud tahaním (drag) alebo nastavením vlastnosti Height na 400.

5. Vlastnost mriežky DataSource nastavíme na hodnotu DataSource1. Ked to vykonáme, mriežka sa zaplní údajmi z databázy zamestnancov. Ak sa tak nestane, teda údaje sa nezobrazia, uistime sa, že sme správne nastavili vlastnosti všetkých objektov formulára tak, ako sme si vysvetlili v predchádzajúcich inštrukciách.

6. Ovládací prvok DBGrid zobrazuje údaje aj vo fáze vývoja (design time), ak pracujeme v IDE. Toto nám umožnuje overit si, že sme sa správne pripojili k databáze. Bohužial, vo fáze vývoja nemôžeme editovat údaje; aby sme mohli editovat údaje v tabulke, musíme spustit aplikáciu.

7. Zo zoznamu komponentov vyberieme ovládací prvok TDBNavigator a umiestnime ho do formulára. Databázový navigátor (database navigator) je nástroj, ktorý umožnuje pohyb po záznamoch súboru údajov (napríklad pomocou šípok dalej (next) a späť (previous)) a vykonáva operácie s údajmi.

8. Navigacnému panelu nastavíme hodnotu vlastnosti DataSource na DataSource1. Navigátor teda bude hladat údaje v klientskom súbore údajov.

9. Nastavíme vlastnost navigacného panelu ShowHint na true. (Nastavenie ShowHint na true umožnuje zobrazovanie bublinkového popisu (help hints) pri nastavení kurzora nad jednotlivé položky navigacného panelu pocas behu programu.)

10. Vyberieme File|Save All a uložíme projekt.

11. Stlacíme F9, cím skompilujeme (compile) a spustíme (run) projekt. To isté môžeme urobit aj tak, že si vyberieme položku Run (spustit) z ponuky Run.

Ked spustíme projekt, program sa otvorí v okne, ktoré vyzerá tak, ako sme si ho navrhli vo formulári. Teraz môžeme testovat navigacný panel

s databázou zamestnancov. Napríklad sa môžeme pomocou príkazových šípok (arrow commands) presúvat od záznamu (record) k záznamu, príkazom + pridávat záznamy a pomocou príkazu - záznamy odstranovat.

Tip: Ak by sme zistili chybu pocas testovania casných verzií našej aplikácie, jednoducho zvolíme Run|Program Reset (Spustit|Zastavit program), cím sa vrátime k fáze vývoja (design-time view).

11.8.10 Pridávanie tlacidla

Tento odstavec opisuje, ako pridat do aplikácie tlacidlo Aktualizuj! (Update Now!). Toto tlacidlo sa používa na uplatnenie všetkých zmien, ktoré používatel urobil v databáze, ako sú editovanie záznamov, pridávanie nových záznamov alebo vymazávanie záznamov.
Na pridanie tlacidla:

1. Zo **zoznamu komponentov** si vyberieme **TButton** a umiestnime ho do formulára;
2. tlacidlo umiestnime do formulára vpravo hore;
3. zo **zoznamu komponentov** si vyberieme **TActionList** a umiestnime ho do formulára;
4. z **ponuky okien** zvolíme **Object TreeView** (Zobrazenie stromu objektov);
5. v okne **Object TreeView** vyberieme **ActionList1**;
6. otvoríme kontextovú ponuku a vyberieme **Action List Editor** (Editor zoznamu akcií);
7. opät otvoríme kontextovú ponuku a vyberieme **New Action** (Nová akcia);
8. vrátime sa spät k vlastnostiam tlacidla a pre vlastnost **Action** nastavíme **Action1**.

Nastavíme vlastnost Caption (nadpis) na Aktualizuj!. Ak chceme spustit aplikáciu, tlacidlo sa objaví v šedej farbe. To sa zmení až potom, co pridáme obsluhu udalosti, ktorá tlacidlo oživí.

11.8.11 Písanie obsluhy udalosti

Väcšina komponentov na palete komponentov (Component palette) obsahuje udalosti (events) a väcšina z nich má aj štandardnú udalost (default event). Bežnou štandardnou udalostou je OnClick, ktorá sa vyvolá vždy, ked sa na komponent, ako je TButton, klikne. Ak si zvolíme vo formulári komponent a klikneme na záložku Events (udalosti) v inšpektore

objektov, uvidíme súpis všetkých udalostí komponentu.
Dalšie informácie o udalostiach a obsluhe udalostí možno nájst v kapitole 4.6.4

11.8.12 Písanie obsluhy udalosti príkazu Aktualizuj! (Update Now!)

Najprv napíšeme obsluhu udalosti pre tlacidlo Aktualizuj!:

Do modulu Unit1.cpp napíšeme nasledujúcu funkciu Action1Execute:

```
void __fastcall TForm1::Action1Execute(TObject
*Sender)

  {

if(ClientDataSet1->State == dsEdit ||
ClientDataSet1->State == dsInsert)

    ClientDataSet1->Post();
    ClientDataSet1->ApplyUpdates(-1);
  }
```

Táto obsluha udalosti najprv skontroluje stav, v ktorom sa nachádza databáza. Ak sa posunieme z práve zmeneného záznamu, tento je automaticky odoslaný do databázy. Ak však neopustíme zmenený záznam, databáza ostane vrežime editovania alebo vkladania. Príkaz if odosiela všetky údaje, ktoré mohli byt zmenené, no neboli odovzdané do klientského súboru údajov. Nasledujúci príkaz potom zavádza aktualizácie, ktoré sa nachádzajú v klientskom súbore údajov, do databázy.

Poznámka: Ak používame dbExpress, zmeny údajov nie sú automaticky prenesené do databázy. Na to, aby sme zapísali všetky aktualizované, vložené a vymazané záznamy z klientského súboru údajov, musíme zavolat metódu ApplyUpdates.

11.8.13 Písanie obsluhy udalosti FormClose (zatvorenie formulára)

Nakoniec napíšeme obsluhu udalosti, ktorá sa vyvolá pri zatváraní aplikácie. Aplikáciu môžeme zavriet cez ponuku File|Exit (Súbor|Skoncit) alebo kliknutím na X v pravom hornom rohu okna (pozn. prekladatela: autor má na mysli ikonu "Zavriet", poslednú v skupine ikon v pravom hornom rohu okna, ktorá je symbolicky oznacená krížikom tvaru X). Pri

oboch spôsoboch program skontroluje, ci neexistujú nedokoncené (pending) databázové aktualizácie a podla potreby zobrazí okno správ a požiada používatela o rozhodnutie, co robit s nedokoncenými zmenami. Takýto kód môžeme umiestnit do obsluhy udalosti Exit, no akékolvek nedokoncené zmeny databázy budú stratené, ak používatel ukoncí aplikáciu pomocou X.

1. Klikneme na hlavný formulár tak, aby bol zvolený (teda nie žiadny konkrétny objekt vo formulári, ale formulár samotný);
2. V inšpektore objektov prejdeme na záložku Events, aby sme mohli vidiet všetky udalosti formulára;
3. Zvolíme udalost **OnClose**;
4. Stlacíme **Ctrl+Enter** a prejdeme do okna s kódom, kde sú vložené nasledujúce riadky:

```
void      __fastcall      TForm1::FormClose(TObject
*Sender, TCloseAction &Action)

{
```

Priamo tam, kde sa nachádza kurzor (medzi zložené zátvorky), napíšeme:

```
Action = caFree;
   if(ClientDataSet1->State    ==    dsEdit    ||
ClientDataSet1->State == dsInsert)
       ClientDataSet1->Post();
   if(ClientDataSet1->ChangeCount > 0) {
      int   Ret   =   Application->MessageBox("Máte
nedokoncené databázové aktualizácie. Chcete ich
zapísat    do    databázy?    ",    "Nedokoncené
aktualizácie", MB_YESNOCANCEL);


      if(Ret == IDYES)
        ClientDataSet1->ApplyUpdates(-1);

     else
        if(Ret == IDCANCEL)
           Action = caNone;
}
```

Táto obsluha udalosti skontroluje stav databázy. Ak jestvujú nedokoncené zmeny, tieto sú postúpené do klientského súboru údajov, kde sa zvýši pocítadlo zmien. Potom, skôr ako sa aplikácia uzavrie, zobrazí sa okno so správou, s otázkou co sa má vykonat so zmenami. Možné odpovede sú Yes (áno), No (nie), alebo Cancel (zrušit). Odpoved Yes zavedie aktualizácie do databázy; No uzavrie aplikáciu bez uloženia zmien do databázy; no a Cancel zruší ukoncenie aplikácie, teda ani nezapíše zmeny do databázy, ani neukoncí aplikáciu - tá ostane bežat.

Dokoncíme volbou File|Save All, cím uložíme projekt. Potom stlacením F9 spustíme aplikáciu.

A máme to hotové! Môžeme vyskúšat aplikáciu a pozriet si, ako pracuje.

11.9 Vytváranie SDI aplikácie

Ciel tejto podkapitoly: Naucit sa vytvárat aplikáciu s jednodokumentovým rozhraním (SDI, Single Document Interface). Vysvetlit si to na príklade, v ktorom bude vytvorený jednoduchý textový editor.

11.9.1 Vytváranie textového editora ako SDI aplikácie

V tejto kapitole bude vysvetlené nasledujúce:

1. Vytvorenie textového editora - prírucka
2. Zacatie novej aplikácie
3. Nastavenie hodnôt vlastností
4. Pridanie komponentov do formulára
5. Pridanie podpory pre ponuku a panel nástrojov (toolbar)
6. Pridanie obrázkov do manažéra akcií (Action Manager)
7. Pridanie akcií do manažéra akcií
8. Pridanie štandardných akcií do manažéra akcií
9. Pridanie panelu nástrojov
10. Vymazanie textovej oblasti (text area)
11. Písanie obsluhy udalostí
12. Vytvorenie obsluhy udalosti príkazu New (Nový)
13. Vytvorenie obsluhy udalosti príkazu Open (Otvorit)
14. Vytvorenie obsluhy udalosti príkazu Save (Uložit)
15. Vytvorenie obsluhy udalosti príkazu Save As (Uložit ako)
16. Vytvorenie súboru Help (Pomocník)
17. Vytvorenie obsluhy udalosti príkazu Help Contents (Obsah

pomocníka)
18. Vytvorenie obsluhy udalosti príkazu Help Index (Register pomocníka)
19. Vytvorenie okna About (O aplikácii)
20. Dokoncenie našej aplikácie

Poznámka:
Táto prírucka je urcená pre profesionálnu edíciu (Professional edition) prostredia C++ Builder.

11.9.1.1 Zacatie novej aplikácie

Skôr, ako zacneme s novou aplikáciou, vytvoríme si adresár, kde budeme udržiavat zdrojové súbory:

1. Na našom pocítaci vytvoríme adresár s názvom **TextEditor**.
2. Zacneme nový projekt tak, že zvolíme **File|New|Application** (Súbor|Nový|Aplikácia) alebo použijeme štandardný projekt, ktorý je už štandardne otvorený po štarte prostredia C++ Builder.

Každá aplikácia je reprezentovaná projektom. Ak spustíme C++ Builder, štandardne sa vytvorí prázdny projekt a automaticky sa vytvoria nasledujúce súbory:

- **Project1.cpp:** súbor zdrojového kódu (source-code file) asociovaný s projektom. Nazývame ho súbor projektu (project file).
- **Unit1.cpp:** zdrojový súbor asociovaný s hlavným formulárom (main form) projektu. Oznacujeme ho ako súbor unitu (unit file).
- **Unit1.h:** hlavickový súbor (header file) je asociovaný s hlavným formulárom projektu. Nazývame ho hlavickový súbor unitu (unit header file).
- **Unit1.dfm**: súbor zdrojov (resource), ktorý nesie informácie o hlavnom formulári projektu. Hovoríme o súbore formulára (form file).

Každý formulár má svoj vlastný súbor unitu (**Unit1.cpp**), hlavickový (**Unit1.h**) súbor a súbor formulára (**Unit1.dfm**). Ak vytvoríme druhý formulár, vytvorí sa automaticky druhý súbor unitu (**Unit2.cpp**), hlavickový súbor (**Unit2.h**) a súbor formulára (**Unit2.dfm**).

3. Z ponuky zvolíme **File|Save All** a uložíme tak všetky súbory na disk. Ked sa objaví dialógové okno Uložit ako (Save As):

   Presunieme sa do nášho priecinka **TextEditor**.

Uložíme **Unit1** pod jeho štandardným menom **Unit1.cpp**.
Uložíme projekt pod menom **TextEditor.bpr**. (Spustitelný súbor (executable) bude mat rovnaké meno ako súbor projektu, ale s príponou (extension) .exe .)

V dalšom opätovne ukladáme našu prácu volbou **File|Save All**.
Ked náš projekt uložíme, C++ Builder vytvorí v našom priecinku s projektom rôzne doplnkové súbory. Tieto súbory však nevymazávame.

11.9.1.2 Nastavovanie hodnôt vlastností

Ked otvoríme nový projekt, C++ Builder štandardne zobrazuje hlavný formulár projektu, pomenovaný **Form1**. Používatelské rozhranie, ako aj iné casti našej aplikácie, vytvoríme umiestnovaním komponentov na tento formulár.
Vedla formulára uvidíme **Inšpektor objektov** (Object Inspector), ktorý môžeme použit na nastavovanie hodnôt vlastností formulára a komponentov na nom umiestnených. Pri nastavovaní vlastností sa C++ Builder sám udržuje náš zdrojový kód. Hodnoty, ktoré nastavíme v inšpektore objektov, sa nazývajú nastavenia vo fáze vývoja (design-time).

1. V **inšpektore objektov** vyhladáme vlastnost formulára Caption (nadpis) a napíšeme '**Vytvorenie textového editora - príručka**' (Text Editor Tutorial) ako náhradu za štandardný nadpis '**Form1**'. Všimnime si, že nadpis formulára sa mení tak, ako ho prepisujeme.
2. Spustíme formulár stlacením **F9**, hoci na nom nie sú žiadne komponenty.
3. Do zobrazenia vo fáze vývoja formulára Form1 sa vrátime, ak stlacíme **Alt+F4**.

11.9.1.3 Pridávanie komponentov do formulára

Skôr, ako zacneme pridávat komponenty do formulára, si potrebujeme premysliet, ako má vyzerat co najvhodnejšie používatelské rozhranie (UI, user interface) našej aplikácie. UI je to, co umožnuje používatelovi interagovat s aplikáciou a mal by byt navrhnutý s ohladom na co najjednoduchšie použitie.
C++ Builder obsahuje mnoho komponentov, ktoré predstavujú jednotlivé casti aplikácie: ponuky (menus), panely nástrojov (toolbars), dialógové okná (dialog boxes) a mnohé dalšie vizuálne a nevizuálne programové prvky.

Aplikácia textového editora potrebuje editacnú oblast (editing area), stavový riadok (status bar) na zobrazovanie informácií ako meno práve editovaného súboru, ponuky a panely nástrojov s tlacidlami (buttons) pre lahší prístup k príkazom. Krása vytvárania používatelského rozhrania pomocou prostredia C++ Builder je v tom, že môžeme skúšat rôzne komponenty a sledovat, ci sa správajú tak, ako si želáme. Takýmto spôsobom môžeme velmi rýchlo vytvorit prototyp rozhranie aplikácie.
Na zaciatku vytvárania textového editora vložíme do formulára textovú oblast a stavový riadok.

1. Na vytvorenie textovej oblasti najprv pridáme komponent **RichEdit**. Vyberieme si z ponuky **View|Component List** (Zobraz|Zoznam komponentov), zvolíme **TRichEdit** a stlacíme tlacidlo **Add to form** (Pridaj do formulára); potom stlacíme **Esc,** cím zatvoríme **Zoznam komponentov** (Component List).
2. Každý komponent v C++ Builder je triedou (class); umiestnením komponentu do formulára vytvoríme inštanciu (instance) tejto triedy. Akonáhle sa teda komponent objaví vo formulári, C++ Builder generuje kód potrebný na vytvorenie inštancie tohoto objektu v našej aplikácii pocas behu programu.
3. So zvoleným komponentom **RichEdit**, si vyhladáme v **inšpektore objektov** vlastnost **Align** (zarovnanie) a nastavíme ju na **alClient** (vyberáme si z položiek).
4. Komponent **RichEdit** teraz vyplna celý formulár, takže sme získali velkú editacnú plochu. Volbou hodnoty **alClient** vlastnosti **Align**, sa velkost ovládacieho prvku **RichEdit** zmení pri každej zmene velkosti okna, aj pri jej obnovení (resize), tak, aby tento ovládací prvok vyplnil celé okno.
5. Vyberieme si **View|Component List**, zvolíme **TStatusBar** a stlacíme tlacidlo **Add to form**; potom stlacíme **Esc**, cím zatvoríme **zoznam komponentov**.
6. Zvolíme komponent **StatusBar**, v **inšpektore objektov** vyberieme vlastnost **Panels** a stlacíme **Ctrl+Return**, cím otvoríme dialógové okno **Editing StatusBar1->Panels** (Editácia StatusBar1->Panels).
7. Urobíme **pravý klik** a zvolíme **Add** (pridat), aby sme mohli pridat panel pre stavový riadok. Tento panel bude zobrazovat cestu a meno súboru, ktorý bude otvorený a editovaný v našom textovom editore.
8. V **inšpektore objektov** nastavíme vlastnost **Text** na "**untitled.txt**" ("nepomenovaný.txt"). Pri používaní editora bude editovaný, no doposial neuložený súbor pomenovaný ako "**untitled.txt**".
9. Vyberieme **Window|Editing StatusBar1->Panels** (Okná|Editovat

StatusBar1->Panely). Potom ho zatvoríme.

10. Tak a teraz už máme hlavnú editacnú oblast používatelského rozhrania pripravenú.

11.9.1.4 Pridávanie podpory pre ponuku a panel nástrojov (toolbar)

Aby aplikácia mohla ponúkat svoju funkcnost, mala by mat ponuku (menu), príkazy (commands) a pre pohodlie používatelov aj panel nástrojov. Hoci príkazy môžeme písat oddelene, C++ Builder ponúka zoznam akcií (action list), ktorý centralizuje akcie (actions), obrázky (images) a programové kódy príkazov ponuky a tlacidiel panelu nástrojov.
Je zvykom, že akcie, ktoré sú spojené s príkazmi ponuky, majú meno prvoúrovnovej ponuky (top-level menu) a meno príkazu. Napríklad akcia **FileExit** prislúcha k príkazu **Exit** (Koniec) z ponuky **File** (Súbor). Nasledujúca tabulka je súpisom rôznych príkazov ponuky, potrebných v našom textovom editore, a uvádza ci tá-ktorá akcia má asociované tlacidlo v paneli nástrojov:

Ponuka	**Príkaz**	**Na paneli nástrojov?**	**Opis**
File (Súbor)	New (Nový)	Áno	Vytvorí nový súbor.
File (Súbor)	Open (Otvorit)	Áno	Otvorí na editovanie už existujúci súbor.
File (Súbor)	Save (Uložit)	Áno	Ukladá aktuálny súbor na disk.
File (Súbor)	Save As (Uložit ako)	Nie	Ukladá súbor pod novým menom (taktiež umožnuje uložit nový nepomenovaný súbor pod zadaným menom).
File (Súbor)	Exit (Koniec)	Áno	Ukoncí program editora.
Edit (Úpravy)	Cut (Vystrihnút)	Áno	Vymaže text a uloží ho do schránky (clipboard).
Edit (Úpravy)	Copy (Kopírovat)	Áno	Text skopíruje a uloží do schránky (clipboard).
Edit (Úpravy)	Paste (Prilepit)	Áno	Vloží text zo schránky (clipboard).
Help (Pomocník)	Contents (Obsah)	Nie	Zobrazí obsah pomocníka, z ktorého môžeme pristupovat k témam (topics) pomocníka.

Help (Pomocník)	Index (Register)	Nie	Zobrazí register pomocníka.
Help (Pomocník)	About (O aplikácii)	Nie	Zobrazí okno s informáciou o aplikácii.

11.9.1.5 Rozdiely medzi editorom manažéra akcií (Action Manager editor) a editorom zoznamu akcií (Action List editor)

V závislosti na edícii nášho prostredia C++ Builder sú dve možnosti, ako spravovat akcie a obrázky ponuky a panela nástrojov. Všetky edície prostredia C++ Builder obsahujú editor zoznamu akcií, ktorý poskytuje priestor centralizujúci odpovede a odozvy používatelských príkazov. Editor zoznamu akcií je súcastou Borland Component Library for Cross Platform (CLX, Knižnica komponentov Borlandu pre križovanie platforiem) a mal by byt použitý namiesto editora manažéra akcií, ak je predpoklad budúceho prechodu na iné platformy (ako napríklad Linux).

Editor manažéra akcií poskytuje istú osobitú funkcnost, ktorá je však prístupná len ako súcast Knižnice vizuálnych komponentov (VCL). Tá je však špecifická len pre platformu Windows. V editore manažéra akcií sa nachádza dialógové okno Customize (prispôsobit), ktorý vie poskytnút také akcie ponuky, ktoré sú prispôsobitelné koncovým používatelom. Tieto majú niektoré z vlastností Microsoft Office (ako je skrývanie zriedka používaných položiek ponuky). Naviac manažér akcií umožnuje daleko rýchlejší proces vývoja aplikácie, kedže akcie možno jednoducho pretiahnut z dialógového okna manažéra akcií Customize do komponentu ponuky (menu) vo formulári.

Upozornenie:
Táto príručka používa dialógové okno manažéra akcií Customize podla edícií Enterprise a Professional prostredia C++ Builder. Ak máme Enterprise alebo Professional edíciu, uskutocníme "Pridávanie ponuky a obrázky panela nástrojov (Enterpsise a Profesional)" ("Adding menu and toolbar images (Enterprise and Professional)").

11.9.1.6 Pridávanie obrázkov do manažéra akcií (Action Manager)

V tejto sekcii budeme pridávat obrázky, ktoré sa budú používat so zväzkami akcií (Action Bands).
V mnohých prípadoch by sme do nášho formulára pridávali komponent **ImageList1** a importovali by sme naše vlastné obrázky. V tejto príručke však budeme šetrit cas tým, že budeme importovat zoznam obrázkov,

ktorý bol použitý pri tvorbe C++ Builder IDE. Pokial nepridáme našu vlastnú grafiku, **ImageList1** na **zozname komponentov** (Component List) bude používat predvolené obrázky pre štandardné akcie.
Ako pridáme už existujúci zoznam obrázkov:

1. Ak máme nainštalovaný C++ Builder v štandardnom adresári, zvolíme **File|Open** (Súbor|Otvorit) a nastavíme **C:\Program Files\Borland\CBuilder6\Source\vcl\actnres.pas**. Aby sa hladaný súbor zobrazil, nastavíme v dialógovom okne Open (otvorit) filter "Files of type:" (typy súborov) na **"Any file.*.*"** (všetky súbory).

2. Vyberieme si komponent **ImageList1,** prekopírujeme ho a prilepíme ho do nášho formulára (copy and paste). Kedže je to nevizuálny komponent, nezáleží na tom, kam ho prilepíme.

Poznámka:
Aby sme skopírovali **ImageList1**, **klikneme pravým tlacidlom myši** na komponente a klikneme na **Edit|Copy** (Úpravy|Kopírovat). Potom na našom formulári opät vykonáme **pravý klik** a vyberieme **Edit|Paste** (Úpravy|Prilepit).

3. Zatvoríme **okno StandardActions** (štandardné akcie).

4. Zvolíme **Window|Object TreeView** (Okno|Zobrazenie stromu objektov), vyberieme **komponent ImageList1**, potom **pravý klik** a vyberieme si **ImageList Editor** (editor zoznamu obrázkov), aby sa nám zobrazili všetky obrázky, ktoré môžeme použit.

Vlastnosti indexu obrázkov pre tak štandardné akcie, ako aj **ImageList1**, ktorý sme pridali:

Command (Príkaz)	**Vlastnost ImageIndex (index obrázku)**
Edit|Cut (Úpravy|Vystrihnút)	0
Edit|Copy (Úpravy|Kopírovat)	1
Edit|Paste (Úpravy|Prilepit)	2
File|New (Súbor|Nový)	6
File|Open	7

(Súbor|Otvorit)	
File|Save (Súbor|Uložit)	8
File|SaveAs (Súbor|Uložit ako)	30
File|Exit (Súbor|Koniec)	43
Help|Contents (Pomocník|Obsah)	40

11.9.1.7 Pridávanie akcií do manažéra akcií

Najprv pridáme manažér akcií a potom pridáme akcie.

1. Vyberieme si **View|Component List** (Zobrazit|Zoznam komponentov), zvolíme **TActionManager** a stlacíme tlacidlo **Add to form** (Pridaj do formulára); potom stlacíme **Esc**, cím zatvoríme **Zoznam komponentov** (Component List). Kedže je to nevizuálny komponent, môžeme ho umiestnit na lubovolnom mieste formulára.

   Tip:
   Na zobrazenie nadpisov nevizuálnych komponentov sa prepneme do formulára, zvolíme **Tools|Environment Options** (Nástroje|Možnosti prostredia), vyberieme si stránku **Preferences** (Preferencie), zaškrtneme **Show component captions** (Zobrazit nadpisy komponentov), a klikneme na **OK**. Takisto podržanie kurzora myši nad komponentom (hovering over) zobrazí jeho meno.

2. Vo formulári si **vyberieme ActionManager1** a pomocou **Inšpektora objektov** nastavíme vlastnost **Images** (obrázky) na **ImageList1**.
3. Dalej pridáme akcie do **Manažéra akcií** a nastavíme ich vlastnosti. Pridáme neštandardné akcie, ktorým budeme musiet nastavit všetky vlastnosti, ako aj akcie štandardné, ktoré už majú svoje hodnoty automaticky nastavené.
4. Zvolíme Window|Object TreeView (Okno|Zobrazenie stromu objektov), vyberieme si ActionManager1, potom pravý klik a zvolíme Customize (Prispôsobit). Zobrazí sa nám dialógové okno Editácia Form1->ActionManager1 (Editing Form1->ActionManager1), alebo sa objaví manažér akcií.
5. Urobíme pravý klik na editor manažéra akcií a zvolíme New Action (Nová akcia).
6. Uistíme sa, že je zvolené No Category (žiadna kategória), a že

v zozname akcií a v Action1 je nastavené under Actions (pri akciách). V inšpektore objektov nastavíme nasledujúce vlastnosti:

- Za **Caption** (nadpis) napíšeme "**&New**" (nový). Uvedomme si, že ampersand (&) pred písmenom spôsobí, že toto písmeno sa stane skratkou (shortcut) pre prístup k príkazu.
- Za **Category** (kategória) napíšeme "**File**" (súbor) (toto zoskupuje príkazy File na jednom mieste).
- Za **Hint** (bublinkový popis) napíšeme "**Create file**" (vytvor súbor) (toto bude tip pomocníka k nástroju (Help tooltip)).
- Uistíme sa, že **ImageIndex** je nastavený na **6** (Toto by sa malo zhodovat so zoznamom obrázkov, ktorý sme importovali. Môžeme tiež kliknút na šípku dole (pozn. prekladatela: otvorit rozbalovací zoznam) a vybrat správny obrázok).
- Za **Name** (meno) napíšeme **"FileNew"** (pre príkaz File|New (Súbor|Nový)) a stlacíme Enter, cím uložíme zmenu.

7. Uistíme sa, že v okne editácie **Form1->ActionManager1** je zvolený File (súbor). Vykonáme **pravý klik** na **editor manažéra akcií** a zvolíme **New Action** (nová akcia).
8. V **inšpektore objektov** nastavíme nasledujúce vlastnosti:

- Do Caption napíšeme "&Save" (uložit).
- Uistíme sa, že Category je nastavená na "File".
- Do Hint zadáme "Save file" (uložit súbor).
- Do ImageIndex nastavíme 8.
- Do Name vložíme "FileSave" (za príkaz File|Save (Súbor|Uložit)).

9. Vykonáme **pravý klik** na **editor manažéra akcií** a zvolíme **New Action**.
10. V **inšpektore objektov** nastavíme nasledujúce vlastnosti:

- Za Caption napíšeme "&Index" (register).
- Za Category napíšeme "Help" (pomocník).
- ImageIndex nie je potrebný. Necháme teda štandardnú hodnotu.
- Za Name vložíme "HelpIndex" (pre príkaz Help|Index (Pomocník|Register)).
- Pravý klik na editor manažéra akcií a zvolíme New Action.
- V inšpektore objektov nastavíme nasledujúce vlastnosti:
- Za Caption napíšeme "&About" (o aplikácii).
- Uistíme sa, že v Category máme "Help".
- Opät, ImageIndex nie je potrebný. Ponecháme teda štandardnú

hodnotu.
- Za Name vložíme "HelpAbout" (príkaz Help|About (Pomocník|O aplikácii)).

11. Na obrazovke si ponecháme dialógové okno manažéra akcií Prispôsobit.
12. Uložíme si našu prácu volbou File|Save All (Súbor|Uložit všetko).

11.9.1.8 Pridávanie štandardných akcií do manažéra akcií

V dalšom pridáme štandardné akcie (open (otvorit), save as (uložit ako), exit (koniec), cut (vystrihnút), copy (kopírovat), paste (prilepit), a help contents (obsah pomocníka)) do manažéra akcií.

1. **Editor manažéra akcií** by mal byt stále zobrazený. Ak nie, tak zvolíme **Window|Object TreeView** (Okno|Zobrazenie stromu objektov), zvolíme **ActionManager1**, vykonáme **pravý klik** a zvolíme **Customize** (Prispôsobit).
2. Urobíme **pravý klik** na **editor manažéra akcií** a zvolíme **New Standard Action** (Nová štandardná akcia).
3. Objaví sa dialógové okno **Standard Action Classes** (triedy štandardných akcií).
4. Posunieme sa na kategóriu **Edit** (úpravy) a pomocou klávesu **Ctrl** vyberieme **TEditCut**, **TEditCopy**, a **TEditPaste**. Klikneme na **OK**, cím tieto akcie pridáme do novej kategórie **Edit** v **Categories list** (v zozname kategórií) dialógového okna editácie Editing **Form1->ActionManager1**.
5. Vykonáme **pravý klik** na **editor manažéra akcií** a zvolíme **New Standard Action** (nová štandardná akcia).
6. Posunieme sa na kategóriu **File** (súbor) a zvolíme **TFileOpen**, **TFileSaveAs** a **TFileExit**. Klikneme na **OK**, cím tieto akcie pridáme do kategórie **File**.
7. Urobíme **pravý klik** na **editor manažéra akcií** a zvolíme **New Standard Action**.

8. Posunime sa (scroll) ku kategórii **Help** (pomocník) a zvolíme **THelpContents**. Klikneme na **OK**, cím túto akciu pridáme do kategórie **Help**.

   Poznámka:
   Náš vlastný príkaz **Help|Contents** (Pomocník|Obsah) zobrazuje súbor pomocníka (Help file) vždy zobrazením záložky (tab) Help

Contents (obsah pomocníka). Štandardný príkaz **Help|Contents** otvára poslednú zobrazenú stranu záložky, a to ci už Contents (obsah), Index (register) alebo Find (hladat).

Teraz máme pridané všetky štandardné akcie, ktoré v našej aplikácii potrebujeme. Štandardné akcie majú vlastnosti automaticky nastavené vrátane indexu obrázku. Index obrázku však možno zmenit, cím sa zobrazí iný obrázok.

9. Ak chceme radšej použit iné ako štandardné obrázky pre akcie, tak môžeme. Napríklad zvolme akciu **File|Open** v dialógovom okne editácie Edit **Form1->ActionManager1**. Teraz zmenme štandardný obrázok na obrázok 7 zo zoznamu.
10. Použijeme tlacidlo Close (zatvorit), aby sme ukoncili editor manažéra akcií.
11. Zvolíme **File|Save All**, cím uložíme všetky naše zmeny.

11.9.1.9 Pridávanie panela nástrojov

Akonáhle už máme nastavené akcie v dialógovom okne **manažéra akcií** Prispôsobit, môžeme pridat niektoré zrovnakých akcií, aké boli použité v ponukách skupín akcií (action band) panelu nástrojov, co bude pripomínat panel nástrojov zMicrosoft Office 2000, ked s tým budeme hotoví.

1. Zvolíme **View|Component List** (Zobraz|Zoznam komponentov), vyberieme si **TActionToolBar** a stlacíme tlacidlo **Add to form** (pridat do formulára); potom stlacíme **Esc**, aby sme zatvorili **zoznam komponentov** (Component List). Pod panelom ponuky sa objaví prázdny panel nástrojov skupiny akcií.
2. Ak **editor manažéra akcií** nie je zobrazený, otvoríme ho (zvolíme **Window|Object TreeView** (Okno|Zobrazenie stromu objektov), vyberieme **ActionManager1**, urobíme **pravý klik** a zvolíme **Customize** (prispôsobit)), a zvolíme **File** v **zozname kategórií**. v **zozname akcií** vyberieme **New**, **Open**, **Save**, a **Exit** a pretiahneme (drag) tieto položky na **panel nástrojov**. Tieto sa automaticky objavia ako tlacidlá, každé s príslušným priradeným obrázkom.
3. V dialógovom okne **manažéra akcií Prispôsobit**, pretiahneme kategóriu **Edit** na **panel nástrojov**. Všetky príkazy kategórie **Edit** by sa mali objavit na paneli nástrojov.

Poznámka:

Ak náhodou pretiahneme nesprávny príkaz na panel nástrojov, môžeme ho jednoducho odstránit (delete). Iba označíme položku v **zobrazení stromu objektov** (Object TreeView) a stlacíme **Del**.

4. Zvolíme File|Save All, cím uložíme naše zmeny.
5. Stlacíme F9, aby sme skompilovali a spustili projekt.

   Tip:
   Ked spustíme náš projekt, C++ Builder otvorí program v okne behu programu (runtime window) tak, ako sme ho navrhli vo formulári.

   Náš textový editor je už pomerne funkcný. Ak si vyznacíme text v textovej oblasti, tlacidlá Cut (vystrihnút), Copy (kopírovat), a Paste (prilepit) by mali pracovat. Ponuky a panel nástrojov by mali fungovat, hoci niektoré príkazy sú neaktívne, sivej farby. Na aktiváciu niektorých príkazov ešte budeme musiet napísat obsluhy udalostí.

6. Na návrat do módu návrhu stlacíme **Alt+F4**.

11.9.1.10 Vymazanie textovej oblasti (text area)

Ked sme spustili program, v textovej oblasti sa objavilo meno **RichEdit1**. Tento text však môžeme odstránit pomocou **editora zoznamu retazcov** (String List editor). Ak neodstránime text teraz, mali by sme ho odstránit pocas inicializácie hlavného formulára ako posledný krok.
Ako vymazat textovú oblast:

1. V hlavnom formulári vyberieme komponent **RichEdit1**.
2. V **inšpektore objektov** zvolíme vlastnost **Lines** (riadky) a stlacíme **Ctrl+Enter** , cím sa zobrazí **editor zoznamu retazcov**.
3. V **editore zoznamu retazcov** vyznacíme a vymažeme text "**RichEdit1**" a klikneme na **OK**.
4. Uložíme zmeny a opät spustíme program.

Editovacia textová oblast je teraz pri zobrazení hlavného formulára prázdna.

11.9.1.11 Písanie obsluhy udalostí

Doteraz sme vytvárali aplikáciu bez písania sebamenšieho kódu. Využívajúc **inšpektor objektov** na nastavovanie vlastností objektov vo fáze vývoja sme prišli na velkú výhodu programovania v RAD (pozn. prekladatela: Rapid Application Development, Rýchly vývoj aplikácie)

prostredí C++ Builder‘s RAD. V tejto kapitole napíšeme funkcie, ktoré sa nazývajú obsluha udalostí (event handler). Tieto odpovedajú a reagujú na používatelské vstupy pocas behu aplikácie. Obsluhy udalostí poprepájame s položkami v ponuke a na paneli nástrojov, takže ak je niektorá položka zvolená pocas behu našej aplikácie, vykoná sa kód v príslušnej obsluhe.

Pre neštandardné akcie musíme vytvorit obsluhu udalostí. Pre štandardné akcie, ako sú **File|Exit** (Súbor|Koniec) a **Edit|Paste** (Úpravy|Prilepit), sú už udalosti obsiahnuté v ich kóde. Akokolvek, pre niektoré štandardné akcie, ako napríklad príkaz **File|Save As** (Súbor|Uložit ako), môžeme napísat našu vlastnú obsluhu udalostí, aby sme mohli príkaz prispôsobit našim potrebám.
Kedže všetky položky ponuky a panelu nástrojov sú spojené v **manažéri akcií** (Action Manager), tak môžeme písat obsluhy udalostí práve tam.

11.9.1.12 Vytvorenie obsluhy udalosti príkazu New (Nový)

Na vytvorenie obsluhy udalosti príkazu New:

1. Zvolíme **View**|**Units** (Zobraz|Unity) a vyberieme **Unit1** na zobrazenie kódu asociovaného s **Form1**.
2. Najprv potrebujeme deklarovat meno súboru, ktorý bude používat obsluha udalosti tak, že pridáme vlastnú vlastnost pre meno súboru, aby bola globálne dostupná z ostatných metód. Otvoríme súbor **Unit1.h** tak, že **pravým klikom** v editore kódu (code editor) v súbore **Unit1.cpp** zvolíme **Open Source/Header File** (Otvorený zdroj/Hlavickový súbor). V hlavickovom súbore vyhladáme a nastavíme sa do sekcie verejných deklarácií (public declarations section) triedy **TForm1**, a do riadku pod:

```
public: // User declarations
```

napíšeme:

```
AnsiString FileName;
```

3. Stlacíme F12, cím sa vrátime spät do hlavného formulára.
4. Prejdeme do inšpektora objektov a zvolíme FileNew z nášho komponentu.
5. Vyberieme záložku **Events** (udalosti), potom zvolíme **OnExecute** a stlacíme **Ctrl+Return**.
6. Práve tam, kde je umiestnený kurzor, v editore kódu (medzi { a }),

napíšeme nasledujúce riadky:

```
RichEdit1->Clear();
FileName = "untitled.txt";
StatusBar1->Panels->Items[0]->Text = FileName;
```

7. Vyberieme **File|Save All**.

11.9.1.13 Vytvorenie obsluhy udalosti príkazu Open (Otvorit)

Ak máme otvorit súbor v textovom editore, chceli by sme, aby sa objavilo štandardné dialógové okno (dialog box) systému Windows. Už sme do **manažéra akcií** pridali štandardný príkaz **File**|**Open** (Súbor|Otvorit), ktorý automaticky obsahuje dialógové okno. Bohužial, stále potrebujeme prispôsobit obsluhu udalosti príkazu.

1. Prejdeme do **inšpektora objektov** a vyberieme spomedzi komponentov **FileOpen1** (pozn. prekladatela: vyberieme napr. pomocou rozbalovacieho zoznamu komponentov).
2. Zvolíme vlastnost **Dialog**, stlacíme **Tab** a potom **Vpravo**, aby sme expandovali jeho vlastnosti. **Dialog** je referencovaný komponent, ktorý vytvára dialógové okno Open (otvorit). C++ Builder pomenúva tento dialóg štandardne **FileOpen1->OpenDialog**. Ked sa zavolá metóda Execute (vykonat) komponentu **OpenDialog1**, vyvolá sa štandardné dialógové okno na otváranie súborov.
3. Nájdeme a nastavíme vlastnost **DefaultExt** na "**txt**".
4. Vyberieme **Filter** (filter) a stlacíme **Ctrl+Return**, cím sa zobrazí **editor filtra** (filter editor).
   - V prvom riadku v stlpci Filter Name (názov filtra) napíšeme "Text files (*.txt)" (textové súbory). Do stlpca Filter napíšeme "*.txt".
   - Do druhého riadku v stlpci Filter Name zadáme "All files (*.*)" (Všetky súbory (*.*)) a do stlpca Filter zadáme "*.*".
   - Klikneme na OK.
5. Nastavíme Title (titulok) na "Open file" (otvorit súbor). Tieto slová sa ukážu v hornej casti dialógového okna Open (otvorit).
6. Prejdeme na záložku Events (udalosti). Zvolíme OnAccept a stlacíme Ctrl+Return, cím otvoríme editor kódu.
7. Práve tam, kde sa nachádza kurzor (medzi { a }) napíšeme nasledujúce riadky:

```
RichEdit1->Lines->LoadFromFile (FileOpen1->Dialog
```

```
->FileName);
FileName = FileOpen1->Dialog->FileName;
StatusBar1->Panels->Items[0]->Text = FileName;
```

Tip:
Aby sme náš kód písali efektívne a rýchlejšie, môžeme použit nástroje nahliadania do kódu (Code Insight tools), ktoré nám to umožnia. Napríklad, ked napíšeme šípku (->) po RichEdit1, objaví sa dialógové okno doplnenia (completion dialog box). Napíšeme “l” a objaví sa Lines : TStrings; v hornej casti dialógového okna. Stlacíme Enter alebo na nom urobíme dvojklik, cím tento text pridáme do nášho kódu.
A to je to, co potrebujeme pre príkaz File|Open a dialógové okno Otvorit.

11.9.1.14 Vytvorenie obsluhy udalosti príkazu Save (Uložit)

Obsluhu udalosti príkazu **Save** vytvoríme nasledovne:

1. Prejdeme do **inšpektora objektov** a vyberieme si komponent **FileSave**.
2. Prejdeme do záložky **Events**, zvolíme **OnExecute** a stlacíme **Ctrl+Return**, aby sme mohli vytvorit obsluhu udalosti.
3. Práve tam, kde sa nachádza kurzor (medzi { a }) napíšeme nasledujúce riadky:

```
if (FileName == "untitled.txt")
   FileSaveAs1->Execute();
else
   RichEdit1->Lines->SaveToFile(FileName);
```

Tento kód hovorí textovému editoru, aby zobrazil dialógové okno **Uložit ako** (save As), ak súbor ešte nebol pomenovaný a používatel by ho teda mal pomenovat. Inak jednoducho uloží súbor pod jeho súcasným menom. Dialógové okno **Uložit ako** je definované v obsluhe udalosti príkazu **Save As**. **FileSaveAs1 BeforeExecute** je meno automaticky generované pre príkaz **Save As**.
To je všetko k príkazu **File|Save**.

11.9.1.15 Vytvorenie obsluhy udalosti príkazu Save As (Uložit ako)

Ked je volaná metóda Execute (vykonat) objektu **SaveDialog**, vyvolá sa

štandardné dialógové okno systému Windows **Uložit ako** (save As) na ukladanie súborov. Obsluhu udalosti príkazu **Save As** vytvoríme takto:

1. Prejdeme do **inšpektora objektov** a zvolíme spomedzi komponentov **FileSaveAs1**.
2. Zvolíme vlastnost **Dialog**, stlacíme **Tab** a potom **Vpravo**, aby sme expandovali jeho vlastnosti. **Dialog** referencuje komponent dialogóvého okna **Uložit ako** a zobrazí vlastnosti dialógového okna **Uložit ako**.
3. Zvolíme a nastavíme **DefaultExt** na "**txt**".
4. Vyberieme **Filter** a stlacíme **Ctrl+Return**, cím sa zobrazí **editor filtra** (Filter editor). V **editore filtrov** špecifikujeme filtre typov súborov rovnako ako pre dialógové okno **Otvorit** (Open).

   - V prvom riadku v stlpci Filter Name (Názov filtra) napíšeme "Text files (*.txt)" (Textové súbory (*.txt)). Do stlpca Filter napíšeme "*.txt".
   - Do druhého riadku v stlpci Filter Name zadáme "All files (*.*)" (Všetky súbory (*.*)) a do stlpca Filter zadáme "*.*".
   - Klikneme na OK.

5. Nastavíme nadpis (Title) na "Save As" (Uložit ako).
6. Nastavíme sa na záložku Events, potom na BeforeExecute a stlacíme Ctrl+Return, aby sme vytvorili obsluhu udalosti.
7. Práve tam, kde sa náchadza kurzor, napíšeme nasledujúce riadky:

```
FileSaveAs1->Dialog->InitialDir = ExtractFilePath
(Filename);
```

8. Záložka Events by mala byt stále zobrazená. Vyberieme OnAccept a stlacíme Ctrl+Return, aby sme napísali obsluhu udalosti.
9. Na miesto, kde sa nachádza kurzor, napíšeme nasledujúce riadky:

```
FileName = FileSaveAs1->Dialog->FileName;
RichEdit1->Lines->SaveToFile(FileName);
StatusBar1->Panels->Items[0]->Text = FileName;
```

10. Zvolíme File|Save All, cím uložíme všetky zmeny.
11. Aby sme videli, ako aplikácia teraz vyzerá, stlacíme F9.
12. Spät do fázy vývoja sa vrátime stlacením Alt+F4.

11.9.1.16 Vytvorenie súboru Help (Pomocník)

Velmi dobrá myšlienka je vytvorit súbor Help (súbor pomocníka), ktorý vysvetluje, ako používat našu aplikáciu. C++ Builder ponúka Microsoft Help Workshop v adresári C:\Project Files\Borland\CBuilder6\Help\Tools, ktorý obsahuje informácie o navrhovaní (designing) a kompilovaní (compiling) súboru pomocníka systému Windows (Windows Help file). V našej ukážkovej aplikácii textového editora si používatel môže zvolit Help|Contents (Pomocník|Obsah) alebo Help|Index (Pomocník|Register) ako prístup ksúboru pomocníka (Help file), a to ci už kobsahu, tak ku registru.
Už skôr sme vytvorili akcie HelpContents a HelpIndex vmanažéri akcií (Action Manager) alebo v editore zoznamu akcií (Action List editor), ktoré zobrazujú záložku Contents (obsah) alebo Index (register) skompilovaného súboru pomocníka. Potrebujem priradit konštantné hodnoty k parametrom pomocníka (help parameters) a vytvorit obsluhy udalostí, ktoré zobrazujú to, co je chceme.

Aby sme mohli používat príkazy skupiny príkazov Pomocník (Help), musíme vytvorit a skompilovat súbor pomocníka systému Windows. Vytváranie súborov pomocníka je mimo rozsahu tejto príručky. Akokolvek, môžeme si stiahnut ukážkový rtf súbor "**TextEditor.rtf**", súbor pomocníka "**TextEditor.hlp**" a súbor obsahu "**TextEditor.cnt**":

1. V Prieskumníkovi (Explorer) systému Windows, v našom adresári **C:\Program Files\Borland\CBuilder6\Help, otvoríme B6X1.zip**
2. Rozbalíme (extract) a uložíme súbory "**.hlp**" a "**.cnt**" do adresára nášho **textového editora**. **Poznámka:**
3. Takisto môžeme v našom projekte použit lubovolný súbor "**.hlp**" alebo "**.cnt**" (ako napríklad niektorý zo súborov pomocníka prostredia C++ Builder a s ním asociovaný súbor "**.cnt**"). Budeme ich musiet skopírovat do adresára projektu a premenovat ich na "**TextEditor.hlp**" a "**TextEditor.cnt**", aby ich aplikácia vedela nájst.

11.9.1.17 Vytvorenie obsluhy udalosti príkazu Help Contents (Obsah pomocníka)

Na vytvorenie obsluhy udalosti príkazu **Help Contents**:

1. Ideme do **inšpektora objektov** a zvolíme komponent

HelpContents1.
2. Vyberieme záložku **Events**, zvolíme **OnExecute** a stlacíme **Ctrl+Return,** aby sme vytvorili obsluhu udalosti.
3. Práve tam, kde sa nachádza kurzor, napíšeme nasledujúce riadky:

```
const static int HELP_TAB = 15;
const static int CONTENTS_ACTIVE = -3;
Application->HelpCommand(HELP_TAB,
CONTENTS_ACTIVE);
```

Tento kód priradí konštantné hodnoty parametrom objektu HelpCommand. Nastavenie HELP_TAB na 15 zobrazí dialógové okno pomocníka a nastavenie CONTENTS_ACTIVE na -3 zobrazí záložku Contents (Obsah).

Poznámka:
Aby sme získali Pomocníka (help) pre udalost HelpCommand, umiestníme kurzor v editore vedla HelpCommand a stlacíme F1.
Tým sme vybavili príkaz Help|Contents.

11.9.1.18 Vytvorenie obsluhy udalosti príkazu Help Index (Register pomocníka)

Na vytvorenie obsluhy udalosti príkazu **Help Index**:

1. Prejdeme do **inšpektora objektov** a zvolíme spomedzi komponentov **HelpIndex**.
2. Vyberieme záložku **Events**, zvolíme **OnExecute** a stlacíme **Ctrl+Return,** aby sme vytvorili obsluhu udalosti.
3. Práve tam, kde sa nachádza kurzor v textovom editore (pozn. prekladatela: editore kódu), napíšeme nasledujúce riadky:

```
const static int HELP_TAB = 15;
const static int INDEX_ACTIVE = -2;
Application->HelpCommand(HELP_TAB, INDEX_ACTIVE);
```

Tento kód priradí konštantné hodnoty do parametrov objektu HelpCommand. Nastavenie HELP_TAB na 15 opät zobrazí dialógové okno pomocníka a nastavenie INDEX_ACTIVE na -2 zobrazí záložku Index (Register).
A s príkazom Help|Index sme hotoví.

11.9.1.19 Vytvorenie okna About (O aplikácii)

Mnohé aplikácie obsahujú okno About, ktoré zobrazuje informácie o produkte ako sú meno, verzia, logá a môžu obsahovat aj iné právne informácie vrátane informácie o autorských právach (copyright).
Už sme nastavili príkaz HelpAbout vmanažéri akcií ci editore zoznamu akcií.
Na pridanie okna About:

1. Zvolíme **File|New|Other** (Súbor|Nový|Iné) na zobrazenie dialógového okna **Nové položky** (New Items).
2. Prejdeme na záložku **Forms** (Formuláre), zvolíme ikonu **okno About** (About Box) a stlacíme **Return**.
3. Zobrazí sa návrh formulára pre **okno About**.
4. Zvolíme samotný formulár a v **inšpektore objektov** klikneme na záložku **Properties** (vlastnosti) a zmeníme vlastnost **Caption** (nadpis) na **"About Text Editor"** (O textovom editore).
5. Prejdeme do formulára "**About**" (uvedomme si, že teraz sa volá About Text Editor). Na zmenu jednotlivých hodnôt vo formulári stlácame Tab a píšeme nové hodnoty.

   - Zmeníme Product Name (názov produktu) na " Textový editor " (Text Editor).
   - Zmeníme Version (verzia) na "Version 1.0" (Verzia 1.0).
   - Zmeníme Copyright (autorské práva) na "Copyright 2002".

6. Uložíme formulár **okna About** volbou **File|Save As** a uložíme ho ako **About.cpp**. v editore kódu prostredia C++ Builder by sme mali mat zobrazených niekolko súborov: **Unit1.cpp**, **Unit1.h**, **About.cpp**, a **ActnRes**. Nepotrebujeme síce unit **ActnRes**, no môžeme ho tam kludne ponechat.
7. Klikneme na záložku **Unit1.cpp** a presunieme sa na zaciatok v editore kódu. Pridáme príkaz include pre unit **About** do **Unit1**. Zvolíme **File|Include Unit Hdr** (Súbor|Pridat Hlv. Unitu) a potom zvolíme **About** a klikneme na **OK**. Všimnime si, že na zaciatok súboru .cpp bolo pridané `#include About.h`.
8. Stlacíme F12, aby sme sa vrátili do módu návrhu. Prejdeme do inšpektora objektov a zvolíme spomedzi našich komponentov HelpAbout.
9. Zvolíme záložku Events, potom zvolíme OnExecute a stlacíme Ctrl+Return, aby sme vytvorili obsluhu udalosti. Práve tam, kde sa kurzor nachádza v editore kódu, napíšeme nasledujúci riadok:

```
AboutBox->ShowModal();
```

Tento kód otvorí okno About potom, co používatel klikne na Help|About (Pomocník|O aplikácii). ShowModal otvorí formulár v modálnom stave (modal state), stave pocas behu programu, kedy používatel nemôže urobit nic iné, kým tento formulár nezatvorí.

10. Zvolíme **File|Save All**.

11.9.1.20 Dokoncenie našej aplikácie

Aplikácia je takmer hotová. Bohužial, ešte musíme špecifikovat niektoré položky v hlavnom formulári. Na dokoncenie aplikácie:

1. Stlacíme **F12** na prepnutie sa do hlavného formulára.
2. Prejdeme do **inšpektora objektov** a zvolíme spomedzi našich komponentov **Form1**.
3. Zvolíme záložku **Events**, vyberieme **OnCreate** a stlacíme **Ctrl+Return**.
4. Práve tam, kde sa kurzor nachádza v editore kódu, napíšeme nasledujúce riadky:

```
Application->HelpFile                                =
ExtractFilePath(Application->ExeName)                +
"TextEditor.hlp";
FileName = "untitled.txt";
StatusBar1->Panels->Items[0]->Text = FileName;
RichEdit1->Clear();
```

Tento kód inicializuje aplikáciu tak, že asociuje súbor pomocníka (Help), nastaví hodnotu FileName na untitled.txt, položí meno súboru do stavového riadku (status bar) a vycistí textovú oblast.

5. Zvolíme **File|SaveAll**, aby sme uložili naše zmeny.
6. Stlacením **F9** spustíme aplikáciu.

Gratulejeme! Práve ste dokoncili aplikáciu.

11.10 Vytváranie MDI aplikácie

Ciel tejto podkapitoly: Naucit sa co je to Viacdokumentové rozhranie (MDI, Multiple Document Interface).

11.10.1 Co je to MDI aplikácia.

Teraz by sme mali vysvetlit a opísat, co je to MDI aplikácia.

Okrem Jednodokumentového rozhrania (SDI) je tiež možné vytvárat Viacdokumentové rozhranie (MDI) s viac ako jedným oknom.
Vieme, že okná používatelského rozhrania môžu byt minimalizované, zväčšené, zmenšené na ikonu a presunuté. Všetko toto je možné, ak okno leží v prostredí, ktoré obsahuje iná okná: okná na pracovnej ploche (desktop) sú asi najlepší známy príklad.
Schopnost obsahovat viac ako jedno okno tvorí rozdiel medzi SDI a MDI. SDI aplikácia nemôže obsahovat iné okná. V MDI aplikácii hlavné okno, nazývané tiež "rodicovské" okno (“parent” window), obsahuje jedno alebo viac okien "potomkov" (“child” windows).
Hlavné okno obycajne obsahuje panel ponúk (menu bar) a prípadne aj panely nástrojov (tool bars). Okno potomka sa presúva len v rámci hlavného okna. Ak sú SDI alebo MDI zmenšené na ikony, tieto ikony budú viditelné na paneli bežiacich aplikácií systému Windows (Windows application bar); ked je okno potomka v MDI zmenšené na ikonu, táto sa objaví v spodnej casti hlavného MDI okna.

11.10.2 Ako vytvorit MDI

Na vytvorenie MDI musíme zmenit vlastnost **FormStyle** formulárov ešte pocas ich vytvárania. Pre rodicovské okno zvolíme hodnotu **fsMDIForm**, pre okno potomka **fsMDIChild**.
Príklad je najlepší spôsob, ako ukázat vytvorenie MDI.

11.10.2.1 Vytvorenie MDI aplikácie

Teraz vytvoríme jednoduchú MDI aplikáciu s cielom zobrazovat a ukladat rastrové obrázky (bitmap images).
Budeme musiet urobit nasledujúce:

1. vytvorit hlavný formulár (main form; rodic MDI)
2. vytvorit jednoduchý panel ponúk (menu bar)
3. vložit dialógové okná Open (Otvorit) a Save (Uložit) tak ako sú na

paneli ponúk
4. implementovat kód položiek vyššie uvedeného panelu ponúk
5. vytvorit formulár MDI potomka
6. spustit a otestovat program

Zacneme otvorením BCB tak ako obycajne: pred nami sa objaví prázdny formulár.

1. Na vytvorenie rodicovského formulára použijeme vlastnosti štandardného prázdneho BCB formulára Form1

   - Name = “MainForm”;
   - Caption = “View bitmap”;
   - Height = 600;
   - With = 800;
   - FormStyle = “fsMDIForm” (vyberieme zo zoznamu).

2. Na vloženie panelu ponúk do rodicovského formulára
   - umiestnime komponent TmainMenu do MainForm
   - vytvoríme panel ponúk (vlastnost items) takto:
     + **&File**
     + **&Open**
     + **&Save As**
     + **&Window**
     + **&Tile**
     + **&Cascade**
     + **&Arrange All**

3. Na vloženie dialógových okien Open a Save do panelu ponúk:
   - pridáme komponent TOpenDialog do MainForm;
   - vložíme komponent TsaveDialog do MainForm;
   - modifikujeme vlastnost Name z "OpenDialog1" na “OpenBitmap”;
   - modifikujeme vlastnost Title z "OpenDialog" na “Open Bitmap”;
   - modifikujeme vlastnost Name z "SaveDialog1" na “SaveBitmap”;
   - modifikujeme vlastnost Title z "SaveDialog" na “Save Bitmap”.

4. Na implementáciu kódu položiek panelu ponúk pre každú z nich:
   - pristúpime k OI (Object Inspector, inšpektor objektov),
   - vyberieme komponent, na ktorom budeme pracovat,
   - Ctrl+Tab na otvorenie okna (pozn. prekladatela: záložky) Events (Udalosti)
   - dôjdeme kudalosti OnClick a stlacíme Ctrl+Enter, cím otvoríme okno kódu “Unit1.cpp” (kurzor sa sám nastaví na správne miesto,

kam budeme potrebovat vkladat fragmenty kódu) Na prázdnom riadku medzi zátvorkami udalosti OnClick položky ponuky Open1 vložíme nasledujúci kód:

```
if (OpenBitmap->Execute())
  {
  TChild* child = new TChild(this);
  child->Image->Picture->LoadFromFile(OpenBitmap-
>FileName);
  child->ClientWidth    =    child->Image->Picture-
>Width;
  child->ClientHeight    =    child->Image->Picture-
>Height;
  child->Caption    =    ExtractFileName(OpenBitmap-
>FileName);
  child->Show();
  }
```

Na prázdnom riadku medzi zátvorkami udalosti **OnClick** položky ponuky **SaveAs1** vložíme nasledujúci kód:

```
  TChild*                    child                    =
dynamic_cast<TChild*>(ActiveMDIChild);
  if (!child) return;
  if (SaveBitmap->Execute());
  {
  child->Image->Picture->SaveToFile(SaveBitmap-
>FileName);
  }
```

Na prázdnom riadku medzi zátvorkami udalosti **OnClick** položky ponuky **Tile1** vložíme nasledujúci kód:

```
  {
  Tile();
  }
```

Na prázdnom riadku medzi zátvorkami udalosti **OnClick** položky ponuky **Cascade1** vložíme nasledujúci kód:

```
  {
  Cascade();
```

```
}
```

Na prázdnom riadku medzi zátvorkami udalosti **OnClick** položky ponuky **ArrangeAll1** vložíme nasledujúci kód:

```
{
ArrangeIcons();
}
```

5. Na vytvorenie formulára MDI potomka:
   - vložíme nový formulár do projektu cez File\New Form (Súbor\Nový formulár);
   - priradíme hodnotu “Child” vlastnosti Name (môžeme ignorovat vlastnost Caption, kedže meno zobrazeného obrázku sa objaví automaticky pocas behu programu (runtime)).
   - priradíme hodnotu “fsMDIChild” vlastnosti FormStyle (vyberieme zo zoznamu).
   - S formulárom sa odteraz bude pracovat ako s formulárom MDI potomka.

   Teraz zistíme, že **OI (inšpektor objektov)** nám umožnuje pracovat iba s aktívnym formulárom a jeho komponentami; aby sme sa presunuli z formulára na iný:

   - otvoríme View\Project Manager (Zobrazit\Manažér projektu);
   - zo stromu ponuky si vyberieme formulár, s ktorým chceme dalej narábat
   - stlacíme Enter

6. Na vloženie a nastavenie komponentu **TImage** do formulára “**Child**” (potomok; tento komponent bude zobrazovat bitmapy)
   - Vložíme komponent Timage do formulára Child;
   - priradíme hodnotu "Image" vlastnosti Name;
   - priradíme hodnotu "True” vlastnosti NameStretch
   - priradíme hodnotu "alClient" vlastnosti Align

   Teraz vykonáme nasledujúce operácie:

   - zvolíme File\Save (Súbor\Uložit) a uložíme unit formulára pod menom "MDIChild";
   - prejdeme do editora kódu (F12), stlacením Ctrl+Tab zvolíme stránku prislúchajúcu Unit1.cpp (MainForm);

- otvoríme File\Include Unit Hdr... (Súbor\Vložit Hlv. Unitu...), vyberieme MDIChild a stlacíme Enter (kompilátor bude odteraz schopný referencovat TChild).

7. Teraz môžeme uložit celý projekt; prvky, ktoré doposial neboli uložené, si ponechajú svoje štandardné mená; stlacíme F9, cím skompilujeme a spustíme program. Teraz môžeme skúsit otvárat a ukladat rastrové obrázky a používat volby (options) na zobrazovanie okien.

Vyššie uvedený program ukazuje myšlienku funkcnosti MDI aplikácie. Bohužial nesie v sebe niektoré problémy. Napríklad ak sa pokúsime otvorit negrafický súbor, bude sa generovat nepovolený prístup (access violation); naviac prázdne okno potomka sa otvorí pri štarte programu; takisto ak zatvoríme okno potomka, tak bude vlastne iba zmenšené na ikonu.

Kapitola 12: Nadstavbový modul 2 Vývoj databáz v prostredí Borland C++ Builder

Ciel tejto kapitoly: Naucit sa co je to relacná databáza a ako vytvárat databázu v prostredí Borland C++ Builder 5.0

12.1 Koncepcia relacných databáz

Ciel tejto podkapitoly: Naucit sa nieco o princípoch relacných databáz. Naucit sa navrhovat databázy pocínajúc analýzou reálnych objektov a ich rozvrhnutím do tabuliek.

Naucíme sa navrhovat relacnú databázu ako obraz reálnych objektov a vztahov medzi nimi. Naucíme sa o primárnom (primary), sekundárnom (secondary) a cudzom (foreign) klúci, o ich funkciách pri triedení a o jedinecnosti záznamov vtabulke. Tiež uvidíme, že existujú 3 spôsoby ako prepájat tabulky, nazývané vztahy 1:m, m:m, a 1:1.

12.1.1 Definícia databáz

- Co je to databáza, tabulka (table), záznam (record)?
- Tabulky, požiadavky (queries), formuláre (forms), zostavy (reports)
- Polia, typy údajov
- Co sú entity a atribúty

12.1.2 Definícia relacných databáz

- Definícia relacnej databázy (master-detail, nadradený-podradený)
- Rozdiel medzi relacnými a hierarchickými databázami
- Príklady relacných databáz
- Navrhovanie relacných databáz
- Normalizácia

12.1.3 Klúce

- Využitie klúcov (keys)
- Typy klúcov
- Ako definovat klúce

12.1.4 Typy prepojení

- Úcel vztahov

- Typy vztahov
- Referencná integrita

12.2 Dalšie cvicenia

1. Navrhnite databázu telefónnych císel a e-mailových adries. Osoba môže mat lubovolné telefónne císla a e-mailové adresy.
2. Navrhnite databázu na požiciavanie hudobných CD (zákazníci, CD-cka, požiciavanie)
3. Navrhnite databázu pre predaj hudobných CD (zákazníci, CD-cka, objednávky, oodrobnosti o objednávkach, úcty)
4. Navrhnite databázu, ktorá obsahuje informácie o škole (ucitelia, žiaci, triedy, predmety, …)

12.3 ODBC, BDE Administrator a Database-Desktop

Ciel tejto podkapitoly: Naucit sa ako vytvárat databázy dostupné pre aplikácie vytvorené v prostredí Borland C++ Builder 5 (BCB5). Naucit sa pracovat s BDE Administrator. Taktiež sa naucit ako vytvárat databázu s Database-Desktop. Naucit sa ako vytvárat pripojenia na databázu cez ODBC.

V tejto kapitole zacneme používat Database-Desktop a vytvárat tabulky a klúce. Potom vytvoríme pripojenie k týmto databázam s použitím BDE Administrator. Na vytváranie pripojení kdatabázam budeme používat aj ODBC.

12.3.1 Vytváranie databáz

- Spoznávanie aplikácie Database-Desktop.
- Vytváranie tabuliek, definovanie polí, nastavovanie typu údajov.
- Definovanie klúcov
- Nastavovanie referencnej integrity medzi tabulkami

12.3.2 Vytváranie pripojenia k databázam

- Spoznávanie aplikácie BDE Administrator.
- Vytváranie nového pripojenia k databáze

12.3.3 Vytváranie pripojení k databázam s použitím ODBC

- Definícia ODBC-ovládacov
- Porovnanie pôvodných (native) a ODBC ovládacov
- Vytvorenie nového ODBC-pripojenia s použitím ovládacieho panelu ODBC-Administrator.

12.4 Dalšie cvicenia

5. Použite cvicenia z predchádzajúcej kapitoly a vytvorte tabulky a pripojenia k databázam.

12.5 Prístup k databáze

Ciel tejto podkapitoly: Naucit sa vytvárat prístupy k databázam v prostredí Borland C++ Builder 5 (BCB5). Zoznámit sa s komponentmi, ktoré sa používajú na definovanie prístupu k databáze s jednou tabulkou aj k relacným databázam.

Zacneme s prehladom komponentov BCB5, používaných na prístup k údajom. Pozrieme sa na prístup cez komponenty TTable alebo TQuery a TDataSource.
Nakoniec si ukážeme ako zobrazovat údaje vo formulári.

12.5.1 Prístup k databáze cez TTable a TDataSource

- Ktoré komponenty potrebujeme použit na sprístupnenie databázy
- Pridávanie Ttable komponentu do formulára.
- Vytváranie pripojenia k databáze.
- Nastavovanie vlastností (properties) objektu Table.
- Pridávanie objektu TdataSource do formulára.
- Vytváranie pripojenia k objektu Table.
- Nastavovanie vlastností (properties) objektu DataSource.
- Vytváranie vypocítaných položiek (calculated fields) a vyhladávacích položiek (lookup-fields) (dvojklik na tabulke alebo objekte query)

12.5.2 Zobrazovanie údajov vo formulári

- Otestovanie ovládacích prvkov pre údaje (data-controls)
- Použitie komponentov TDBText, TDBEdit, TDBMemo na zobrazenie údajov.
- Použitie komponentov TDBListbox a TDBCombobox na zobrazenie údajov.
- Použitie komponentov TDBRadioGroup a TDBCheckBox na zobrazenie údajov.
- Použitie komponentu TDBGrid na zobrazenie údajov.
- Navigácia medzi záznamami.
- Pridávanie, úprava a mazanie záznamov.

12.5.3 Použitie dátového modulu (Data-Module)

- Definícia dátového modulu
- Použitie dátového modulu s viacerými formulármi

12.6 Dalšie cvicenia

Použite cvicenia z predchádzajúcich kapitol na vytvorenie jednoduchých tabuliek.

12.7 Vytváranie po□iadaviek s pou□itím TQuery

Ciel tejto podkapitoly: Naucit sa vytvárat požiadavky v BCB. Tak isto sa naucíme ako vytvárat SQL požiadavky.

V tejto kapitole budeme pracovat s komponentom TQuery, ktorý sa používa na vytváranie požiadaviek. Pretože vsúvislosti skomponentom TQuery sa používa SQL, zacneme s krátkym úvodom do SQL a naucíme sa príkaz select (vyber).

12.7.1 Definície

- Definícia SQL
- Použitie TQuery na zavedenie SQL-výrazov do BCB

12.7.2 Úvod do SQL: SELECT

- Syntax jazyka SQL.
- Jednoduché požiadavky s použitím select.
- Klúcové slová WHERE a ORDER.
- Použitie JOIN na definovanie požiadaviek s viacerými tabulkami.
- Použitie parametrov v požiadavkách.

12.7.3 Použitie TQuery na vytváranie požiadaviek

- Pridávanie komponentu TQuery do formulára.
- Vlastnosti (properties) objektu query.
- Definovanie SQL príkazov v query-objekte.
- Použitie TQuery na definovanie "master-detail" formulárov (nadradený-podradený).

12.8 Dalšie cvicenia

Použite cvicenia z predchádzajúcich kapitol na vytváranie SQL-príkazov a "master-detail" formulárov.

12.9 Návrh zostavy (report)

Ciel tejto podkapitoly: Naucit sa vytlacit údaje z jednej alebo viacerých tabuliek, triedit ich a zoskupovat.

V tejto kapitole uvidíme ako pripravit údaje na vytlacenie s použitím komponentu report (zostava). Budeme pracovat s nejakými dalšími komponentmi, aby sme mohli riadit zobrazenie údajov v zostave.

12.9.1 Definície

- Definícia zostavy a jej použitie.
- Komponenty treba použit pri zostave.

12.9.2 Komponent TQuickReport

- Pridávanie komponentu TQuickReport do formulára.
- Vlastnosti (properties) objektu quick-report.
- Komponent TQRPreview na zobrazenie náhladu
- Komponent TQRPrinter na tlac

12.9.3 Komponenty na zobrazovanie údajov v zostave

- Umiestnenie komponentov v zostave: TQRBand
- Zobrazenie pevného textu: TQRLabel and TQRMemo
- Zobrazenie obsahu polí: TQRDBText
- Výpocty v zostave: TQRDBCalc
- Císla strán, cas a dátum: TQRSysData
- Databázy "master-detail": TQRDetailLink
- Zoskupovanie (grouping) záznamov v zostave: TQRGroup

12.10 Dalšie cvicenia

Použite cvicenia z predchádzajúcich kapitol na vytváranie zostáv.

12.11 Aktualizácia SQL

Ciel tejto podkapitoly: Naucit sa, ako menit údajev tabulke so vztahmi "master-detail" (nadradený-podradený).

Najprv sa naucíme o SQL príkazoch update, delete a insert a ako ich používat. Potom sa zoznámime s objektom updateSQL, ktorý sa používa pri práci s týmito SQL príkazmi.

12.11.1 Príkaz UPDATE (aktualizovat), DELETE (vymazat), INSERT (vložit)

- Použitie SQL-príkazov UPDATE, DELETE a INSERT

12.11.2 Komponent TUpdateSQL

- Použitie TUpdateSQL na zmenu, mazanie alebo vkladanie údajov
- Pridávanie komponentu TUpdateSQL do formulára
- Definovanie SQL-príkazu

12.12 Dalšie cvicenia

Použite cvicenia z predchádzajúcich kapitol na aktualizáciu údajov vo vztahoch "master-detail" (nadradený-podradený).

Kapitola 13: Zadania projektov po základnej casti 1

Zadania na skúšky:
Všeobecné pokyny:

- Zadania sú rôznej zložitosti
- Niektoré zadania majú dalšie funkcie, ktoré môžu byt v definícii problému pridané alebo z nej vynechané.

13.1 Zadanie 1: Aké je tvoje nové meno?

Nasleduje výnatok z knihy pre deti "Captain Underpants and the Perilous Plot of Professor Poopypants", ktorú napísal Dave Pilkey:

Použite tretie písmeno z vášho krstného mena, na urcenie vášho nového krstného mena:
A = poopsie
B = lumpy
C = buttercup
D = gidget
E = crusty
F = greasy
G = fluffy
H = cheeseball
I = chim-chim
J = stinky
K = flunky
L = booger
M = pinky
N = zippy
O = goober
P = doofus
Q = slimy
R = loopy
S = snotty
T = tulefel
U = dorkey
V = squeezit
W = oprah
X = skipper
Y = dink
Z = zsa zsa

Použite druhé písmeno z vášho priezviska, na urcenie prvej polovice vášho nového priezviska:
A = diaper
B = toilet
C = giggle
D = burger
E = girdle
F = barf
G = lizard
H = waffle
I = cootie
J = monkey
K = potty
L = liver
M = banana
N = rhino
O = bubble
P = hamster
Q = toad
R = gizzard
S = pizza
T = gerbil
U = chicken
V = pickle
W = chuckle
X = tofu
Y = gorilla
Z = stinker

Použite štvrté písmeno z vášho priezviska, na urcenie druhej polovice vášho nového priezviska:
A = head
B = mouth
C = face
D = nose
E = tush
F = breath
G = pants
H = shorts
I = lips
J = honker

K = butt
L = brain
M = tushie
N = chunks
O = hiney
P = biscuits
Q = toes
R = buns
S = fanny
T = sniffer
U = sprinkles
V = kisser
W = squirt
X = humperdinck
Y = brains
Z = juice

Úlohy:

1. Vytvorte formulár, do ktorého bude možné zadat svoje meno.
2. Po kliknutí na tlacidlo bude vytvorené nové meno. Po každom kliknutí na to isté tlacidlo môže byt pridaná dalšia cast nového mena.
3. Formulár môže obsahovat tlacidlo “nové”, aby niekto mohol zadat nové meno.
4. Môže tu byt aj druhý formulár, ktorý sa môže otvárat stlacením tlacidla v prvom formulári. V tomto formulári môžu byt v troch zoznamových rámikoch zobrazené zoznamy vyššie uvedených hodnôt.
5. Okrem tlacidiel môžu byt všetky príkazy dostupné v ponuke a skratkovými klávesmi.

13.2 Zadanie 2: Šibenica (Hangman)

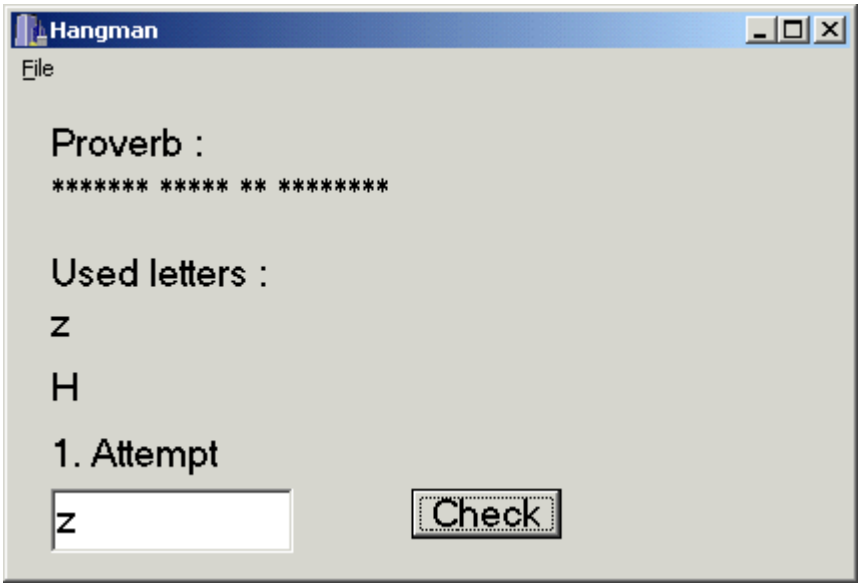
Šibenica je hra na hádanie slov alebo prísloví (proverb):
Slovo alebo príslovie je náhodne vybrané zo zoznamu. Všetky písmená sú zmenené na bodky a každá medzera zostane medzerou. Tieto bodky a medzery sú zobrazené v políčku vo formulári. Potom niekto môže napísat písmeno do editacného políčka. Po kliknutí na tlacidlo Skontroluj! (check) sa zistuje, ci písmeno patrí do hladaného slova alebo príslovia. Ak tam patrí, potom sú korešpondujúce bodky v prvom políčku zmenené na toto písmeno. Ak tam písmeno nepatrí, môžeme zobrazit správu a do slova “HANGMAN” pridat jedno písmeno.
Hra koncí, ak je uhádnuté slovo, alebo ak je slovo hangman kompletné a

políčko s hádaným slovom stále obsahuje nejaké bodky.
Príkazy “Nová hra” a “Koniec” môžu byt vyvolané z ponuky alebo skratkovými klávesmi.

Poznámka: Pri výbere slova alebo príslovia použite funkcie randomize() a random(limit).
Dalšie funkcie:

- Program pocíta pokusy a zobrazuje ich vo formulári.
- Zobrazuje sa zoznam písmen, ktoré už boli použité (used letters).
- Písmeno nemôže byt použité druhý krát. Ak to niekto urobí, bude zobrazené varovanie. Tento pokus (attempt) môže byt pocítaný ako správny alebo nesprávny.

Nasleduje obrázok, ktorý ukazuje, ako môže vyzerat formulár:



13.3 Zadanie 3: Hanoiské ve□e

V tejto hre máme 3 veže zostavené z diskov rôznej velkosti (v našich zadaniach ich môžeme reprezentovat císlami). Disky môžete presúvat z jednej veže na inú, ale nikdy nemôžete položit väcší disk na menší (v našom prípade väcšie císlo nad menšie). Pri jednom tahu môžete presúvat len jeden disk.
Môžete presunút celú vežu diskov z jedného miesta na iné.
Úlohy:

- Vytvorte formulár s troma políckami (napr. zoznamovými rámikmi).
- Na zaciatku hry sa program opýta, aká môže byt veža vysoká.
- Pomocou tlacidiel môžete po jednom presúvat “disky” z jednej veže na druhú.

- Ak presun nie je možný, môže byt zobrazené varovanie.
- Na konci môže byt zobrazené okno so správou o úspešnom ukoncení hry.
- Dalšia funkcia:
  Pocítadlo môže pocítat cas, ktorý bol potrebný na dokoncenie hry. Pri tejto funkcii použite TTimer-komponent.

13.4 Zadanie 4: Kalkulátor

Úlohou je navrhnút kalkulátor:
Úlohy:

- Pridávanie císel tlacidlami alebo skratkovými klávesmi
- Políčko, ktoré zobrazuje čísla a výpocet
- Výpocet pre scítanie, odcítanie, násobenie a delenie
- Pocítanie s percentami.
- Tlacidlo nový výpocet (C-tlacidlo)
- Tlacidlo na mazanie displeja, pretože bolo zadané zlé číslo. Výpocet nebude pozastavený (CE-tlacidlo)
- Tlacidlo na odstránenie naposledy pridanej cifry.

Dalšie funkcie:

- Výpocty so zátvorkami.
- Pocítanie mocnín a odmocnín.
- Pamät na čísla

13.5 Zadanie 5: Prehrávanie hudby

- Máte zoznam skladieb, reprezentujúci *.wav súbory (niektoré z nich sú inštalované pocas inštalácie windows).
- Niekto si vyberie skladbu zo zoznamu. Máme tlacidlá na zacatie, zastavenie, a pozastavenie prehrávania skladby.
- Tak isto tu môže byt možnost presunút sa na dalšiu alebo predchádzajúcu skladbu.
- Na prehrávanie skladieb použite komponent TMediaPlayer.
- Príkazy môžu byt aktivované tlacidlami z ponuky alebo skratkovými klávesmi.

Dalšie funkcie:

- Možnost vytvorit zoznam existujúcich skladieb.
- Náhodné prehrávanie: Prehrávanie skladieb náhodne. Použite funkciu randomize() a random(limit).
- Opakovanie: Prehrávanie skladby znovu a znovu.

13.6 Zadanie 6: Vyslovovanie čísel

Najprv nahrajte niekoho, ked hovorí čísla a slová, napr. jeden, dva, tri, …, osemnást, devätnást, dvadsat, tridsat, …, sto, tisíc, milión atd. ako wav súbory.
Do editacného pola vo formulári môže používatel zapísat císlo. Po kliknutí na tlacidlo bude císlo vyslovené, prehrávaním korešpondujúcich .wav súborov.
Na prehrávanie císel použite komponent TMediaPlayer.

Kapitola 14: Zadania projektov po základnej casti 2

14.1 Zadanie 1: Databáza CD-ciek

Vytvorte prehrávac, ktorý si ukladá do súboru názvy hudobných CD a názvy stôp (skladieb).
Funkcie:

- Identifikácia hudobného CD
- Nacítanie poctu skladieb
- Prehrávanie skladby, stop, pauza, dalšia skladba, predchádzajúca skladba
- Náhodné prehrávanie skladieb, používatelom nadefinovaný zoznam skladieb, opakovanie skladby, opakovanie celého hudobného CD
- Pridávanie a úprava názvu a interpreta CD.
- Ukladanie týchto informácií do súboru
- Pri použití hudobného CD, má prehrávac automaticky skontrolovat, ci už “databáza” neobsahuje informácie o tomto CD

Príkazy súvisiace s prehrávaním jednej alebo viacerých skladieb implementujte ako triedu. Do dalšej triedy implementujte všetky príkazy súvisiace s “databázou” názvov CD, interpretov a názvami stôp (skladieb), vrátane súborových príkazov otvorit (open), uložit (save) a zatvorit (close).

14.2 Zadanie 2: Databáza adries

Vytvorte “databázu” ludí a ich adries. Na ukladanie adries použite jednoduchý textový súbor.

Funkcie:

- Vytvorte formulár na zobrazovanie, úpravu, mazanie a pridávanie nových adries.
- Vytvorte prístup k súboru na ukladanie adries.
- Používatel môže používat rôzne súbory s adresami.

Prístup k súboru implementujte ako triedu. Táto trieda tiež môže identifikovat adresu v súbore a zobrazit ju vo formulári.

Dalšia funkcia:

- Jedna osoba môže mat viac ako jednu adresu. Takže mená a adresy musia byt uložené v rôznych súboroch. Na urcenie, ktorá adresa

patrí ku ktorej osobe, použite identifikacné císla.
- Prístup k súborom implementujte ako základnú triedu. Vytvorte 2 odvodené triedy, jednu na prístup k súboru ludí a jednu na prístup k súboru adries.

14.3 Zadanie 3: Kalendár

Naprogramujte jednoduchý kalendár. Kalendár môže zobrazovat celý mesiac. Používatelovi môže byt umožnené vybrat si rok a mesiac. Tak isto mu môže byt umožnené posúvat sa na predchádzajúci alebo nasledujúci mesiac. Kalendár môže byt prednastavený tak, že zobrazuje aktuálny mesiac aktuálneho roku.
Kalendár môže zobrazovat soboty a nedele inou farbou, ako ostatné pracovné dni.

Vytvorte kalendár s použitím komponentu StringGrid.
Výpocet roka, mesiaca, sobôt a nediel implementujte ako triedu.
Dalšie funkcie:
- Používatel môže zadávat informácie o schôdzkach, ktoré sú uložené v súbore. Schôdzka môže obsahovat zaciatok, koniec a predmet. Schôdzky môžete zobrazovat v dalšej mriežke, ktorá bude aktualizovaná vždy, ked používatel vyberie iný den v mriežke kalendára. Mriežka kalendára by mala zobrazovat informáciu o pocte schôdzok na aktuálny den.
  Takisto môžete zobrazovat schôdzky v dalšom formulári, ktorý bude otvorený ak používatel dvojklikne na den alebo stlací Ctrl+Enter.
- Implementujte sviatky. Máme rôzne druhy sviatkov: cirkevné sviatky ciastocne závisia od Velkej noci. Existujú funkcie na výpocet víkendu, kedy bude v konkrétnom roku Velká noc. Informácie o nich sú publikované na internete. Ostatné cirkevné sviatky a prázdniny majú presne urcené dni. Umožnite používatelom vytvárat nové sviatky a ukladat ich do súboru.
  Prístup k súborom implementujte ako triedu. Vypocítavanie sviatkov implementujte ako triedu, odvodenú zo základných tried, na základné “kalendárové” výpocty a prístup k súborom.

14.4 Zadanie 4: Šibenica (Hangman)

Šibenica je hra na hádanie slov alebo prísloví (proverb):
Slovo alebo príslovie je náhodne vybrané zo zoznamu. Všetky písmená sú zmenené na bodky a každá medzera zostane medzerou. Tieto bodky a medzery sú zobrazené v polícku vo formulári. Potom niekto môže napísat

písmeno do editacného políčka. Po kliknutí na tlacidlo Skontroluj! (check) sa zistuje, ci písmeno patrí do hladaného slova alebo príslovia. Ak tam patrí, potom sú korešpondujúce bodky v prvom políčku zmenené na toto písmeno. Ak tam písmeno nepatrí, môžeme zobrazit správu a do slova "HANGMAN" pridat jedno písmeno.
Hra koncí, ak je uhádnuté slovo, alebo ak je slovo hangman kompletné a políčko s hádaným slovom stále obsahuje nejaké bodky.
Príkazy "Nová hra" a "Koniec" môžu byt vyvolané z ponuky alebo skratkovými klávesmi.

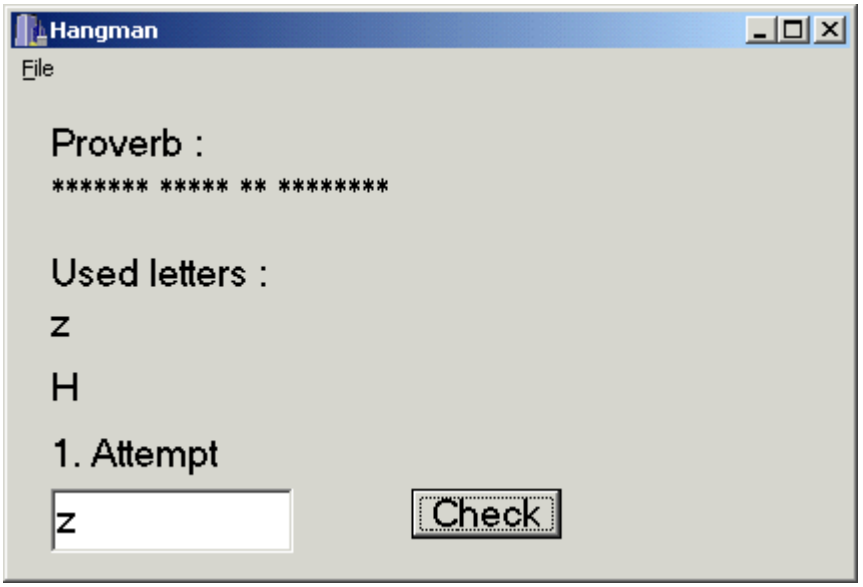
Poznámka: Pri výbere slova alebo príslovia použite funkcie randomize() a random(limit).
Dalšie funkcie:

- Program pocíta pokusy a zobrazuje ich vo formulári.
- Zobrazuje sa zoznam písmen, ktoré už boli použité (used letters).
- Písmeno nemôže byt použité druhý krát. Ak to niekto urobí, bude zobrazené varovanie. Tento pokus (attempt) môže byt pocítaný ako správny alebo nesprávny.
- Slová alebo príslovia môžu byt uložené v súbore. Ak sa zacne nová hra, môže sa otvorit súbor a náhodne vybrat jedno zo slov alebo prísloví.
- V druhom formulári môže používatel upravovat slová alebo príslovia. Tieto sú zobrazené v zozname. Ak používatel jedno z nich oznací, môže ho upravit a uložit zmeny. Taktiež môže pridávat nové slová a príslovia. Pri zatvorení formulára sú zmeny uložené do súboru.

Hlavné úlohy:

- Prístup k súboru implementujte ako triedu: cítanie, zápis, prístup k súboru, atd.
- Funkcie súvisiace s hrou implementujte ako triedu: nová hra, skontrolovat písmeno, zoznam použitých písmen, pocet pokusov, atd.

Nasleduje obrázok, ktorý ukazuje, ako môže vyzerat formulár:



14.5 Zadanie 5: Kalkulátor

Úlohou je navrhnút kalkulátor:
Úlohy:

- Pridávanie císel tlacidlami alebo skratkovými klávesmi
- Políčko, ktoré zobrazuje čísla a výpocet
- Výpocet pre scítanie, odcítanie, násobenie a delenie
- Pocítanie s percentami.
- Tlacidlo nový výpocet (C-tlacidlo)
- Tlacidlo na mazanie displeja, pretože bolo zadané zlé číslo. Výpocet nebude pozastavený (CE-tlacidlo)
- Tlacidlo na odstránenie naposledy pridanej cifry.
- Dalšie funkcie:
- Výpocty so zátvorkami.
- Pocítanie mocnín a odmocnín.
- Pamät na čísla

Vstup, výpocty a pamät implementujte do troch rôznych tried.

Kapitola 15: Zadania záverecného projektu pre nadstavbový modul 1 (VCL)

15.1 Zadanie 1: Jednoduchý textový editor

Navrhnite textový editor. Editor by mal byt schopný editovat jednoduchý text.
Crty editora:

- Otvorit (Open) a zatvorit (Close) jeden textový súbor (SDI).
- Uložit (Save) a Uložit ako (Save As).
- Vybranie (Select) textu
- Vymazávanie, kopírovanie a presun textu (vystrihnút (cut), kopírovat (copy), prilepit (paste))
- Nastavenie dlžky riadku.

Implementujte prístup k súboru (file-access) a editovanie súboru (file-edit) ako rôzne triedy
Doplnkové crty:

- Zobrazenie pozície kurzora podla riadku a stlpca
- Výpocet poctu slov a znakov v súbore
- Operácia spät (undo) pre posledný editacný príkaz
- Tlac súboru (Print)
- Nastavenie strany (Page setup)

Implementujte tlac ako triedu.

15.2 Zadanie 2: Zlo□itý textový editor

Navrhnite textový editor. Editor by mal byt schopný editovat jednoduchý text.
Crty editora:

- Otvorit (Open) a zatvorit (Close) viac ako jeden súbor (MDI).
- Uložit (Save) a Uložit ako (Save As).
- Vybranie (Select) textu
- Vymazávanie, kopírovanie a presun textu (vystrihnút (cut), kopírovat (copy), prilepit (paste)), a to aj z jedného otvoreného súboru do iného.

- Nastavenie dlžky riadku.

Implementujte prístup k súboru (file-access) a editovanie súboru (file-edit) ako rôzne triedy
Doplnkové crty:

- Zobrazenie pozície kurzora podla riadku a stlpca
- Výpocet poctu slov a znakov v súbore
- Operácia späť (undo) pre posledný editacný príkaz
- Tlac súboru (Print)
- Nastavenie strany (Page setup)

Implementujte tlac ako triedu.

15.3 Zadanie 3: Jednoduchý tabul kový procesor

Navrhnite tabulkový procesor. Procesor by mal umožnovat editovanie jednoduchých vstupov, identifikovat číselné a textové vstupy, vykonávat jednoduché výpocty (scítanie, odcítanie, násobenie, delenie).
Crty tabulkového procesora:

- Otvorit (Open) a zatvorit (Close) jeden súbor (SDI).
- Uložit (Save) a Uložit ako (Save As).
- Vložit a editovat císla a text v bunke
- Vykonávat jednoduché výpocty: scítanie, odcítanie, násobenie, delenie
- Ukazovat chyby vo vzorcoch ako chyby (errors).
- Vymazávanie, kopírovanie a presun textu (vystrihnút (cut), kopírovat (copy), prilepit (paste)).

Použite komponent StringGrid ako tabulku.
Implementujte prístup k súboru (file-access), editovanie súboru (file-edit) a výpocet (calculation) ako rôzne triedy
Doplnkové crty:

- Zobrazenie pozície kurzora podla riadku a stlpca (teda podla bunky)
- Operácia späť (undo) pre posledný editacný príkaz
- Tlac súboru (Print)
- Nastavenie strany (Page setup)
- Definujte funkciu "Sum", ktorá vie scítat obsahy zvolených buniek

Implementujte tlac ako triedu.

15.4 Zadanie 4: Zlo□itý tabul kový procesor

Navrhnite tabulkový procesor. Procesor by mal umožnovat editovanie jednoduchých vstupov, identifikovat číselné a textové vstupy, vykonávat jednoduché výpocty (scítanie, odcítanie, násobenie, delenie).
Crty tabulkového procesora:

- Otvorit (Open) a zatvorit (Close) viac ako jeden súbor (MDI).
- Uložit (Save) a Uložit ako (Save As).
- Vložit a editovat císla a text v bunke
- Vykonávat jednoduché výpocty: scítanie, odcítanie, násobenie, delenie
- Ukazovat chyby vo vzorcoch ako chyby (errors).
- Vymazávanie, kopírovanie a presun textu (vystrihnút (cut), kopírovat (copy), prilepit (paste)) zo súboru do súboru

Použite komponent StringGrid ako tabulku.
Implementujte prístup k súboru (file-access), editovanie súboru (file-edit) a výpocet (calculation) ako rôzne triedy
Doplnkové crty:

- Zobrazenie pozície kurzora podla riadku a stlpca (teda podla bunky)
- Operácia spät (undo) pre posledný editacný príkaz
- Tlac súboru (Print)
- Nastavenie strany (Page setup)
- Definujte funkciu "Sum", ktorá vie scítat obsahy zvolených buniek

Implementujte tlac ako triedu.

Kapitola 16: Zadania projektov po nadstavbovom module 2 (Databázy)

16.1 Zadanie 1: Databáza CD-ciek

Vytvorte prehrávac, ktorý môže obsahovat názvy hudobných CD a názvy stôp/skladieb v súbore.
Funkcie:

- Identifikácia hudobného CD
- Nacítanie poctu skladieb
- Prehrávanie skladby, stop, pauza, dalšia skladba, predchádzajúca skladba
- Náhodné prehrávanie skladieb, používatelom nadefinovaný zoznam skladieb, opakovanie skladby, opakovanie celého hudobného CD
- Pridávanie a úprava názvu a interpreta CD.
- Ukladajte tieto informácie do databázy, pozostávajúcej z 2 súborov, jeden obsahuje názvy CD a interpretov a druhý obsahuje názvy skladieb.
- Použite formát dBase, Paradox alebo Access.
- Urobte návrh databázy.
- Vytvorte tabulky.
- Vytvorte formulár na zobrazovanie obsahu databázy.
- Pri používaní hudobného CD, môže prehrávac automaticky skontrolovat, ci “databáza” obsahuje informácie o tomto CD.

Príkazy súvisiace s prehrávaním jednej alebo viacerých skladieb implementujte ako triedu.
Prístup do databázy implementujte ako dalšiu triedu.

16.2 Zadanie 2: Databáza adries

Vytvorte databázu ludí a ich adries. Ludia môžu mat viac ako jednu adresu, tá istá adresa môže byt priradená k viac ako jednej osobe.
Použite formát dBase, Paradox alebo Access.
Funkcie:

- Používatel môže vytvárat vztahy medzi osobami a adresami.
- Urobte návrh databázy.
- Vytvorte tabulky.

- Vytvorte formulár.

Prístup k databáze, vrátane úpravy záznamov, implementujte ako triedu.

16.3 Zadanie 3: Zoznam úloh

Vytvorte databázu úloh. Úloha môže obsahovat predmet, zaciatok, koniec, dokoncenie, dátum dokoncenia, zostávajúci cas do dokoncenia úlohy, využitý cas, percentuálne splnenie úlohy atd.
Funkcie:

- Vytvorte formulár na zobrazovanie, úpravu, mazanie a pridávanie nových úloh.
- Úlohy môžu byt rozdelené do kategórií úlohy v budúcnosti, aktuálne a dokoncené úlohy.
- Používatel si môže vybrat zobrazenie úloh na dnešný den, tento týžden, tento mesiac alebo všetky
- Dokoncené úlohy môžu byt zobrazené len v špeciálnom zozname “dokoncené”.
- Použite formát dBase, Paradox alebo Access.
- Urobte návrh databázy.
- Vytvorte tabulky.

Prístup do databázy, vrátane úprav záznamov, implementujte ako triedu.

16.4 Zadanie 4: Hudobná zbierka

Vytvorte databázu na spravovanie hudobnej zbierky. Popri názve, skladbách a interpretoch tu môžu byt polícka s typom média (CD, LP, MC, atd.), s hudobným štýlom (klasická, pop, country, atd.) a s podobnými kategóriami.
Funkcie:

- Vytvorte formulár na zobrazovanie, úpravy, mazanie a pridávanie novej hudby.
- Všetky zoznamy kategórií môžu byt editovatelné.
- Použite formát dBase, Paradox alebo Access.
- Urobte návrh databázy.
- Urobte tabulky.

Prístup k databázam, vrátane úpravy záznamov, implementujte ako jednu alebo viac tried (základné a odvodené triedy).