Moderné vzdelávanie pre vedomostnú spoločnosť / Projekt je spolufinancovaný zo zdrojov EÚ

Ing. Ján Kontuľ

Základy programovania v jazyku C++

Osvedčená pedagogická skúsenosť edukačnej praxe

Prešov

2012

Vydavateľ:	Metodicko-pedagogické centrum, Ševčenkova 11, 850 01 Bratislava
Autor OPS:	Ing. Ján Kontuľ
Kontakt na autora:	Pracovisko: Gymnázium sv. Jána Zlatoústeho, Lesná 28, 066 01 Humenné, jan.kontul@gmail.com
Názov OPS:	Základy programovania v jazyku C++
Rok vytvorenia OPS:	2012
Odborné stanovisko vypracoval:	Ing. Zuzana Tkáčová

Za obsah a pôvodnosť rukopisu zodpovedá autor. Text neprešiel jazykovou úpravou.

Táto osvedčená pedagogická skúsenosť edukačnej praxe bola vytvorená z prostriedkov projektu Profesijný a kariérový rast pedagogických zamestnanancov. Projekt je financovaný zo zdrojov Európskej únie.

Kľúčové slová

programovanie, C++, kompilácia, premenné, výrazy, vetvenie, cyklus, funkcia, pole, textový súbor

Anotácia

Práca vychádza z pedagogickej praxe a zaoberá sa základmi programovania v jazyku C++, ktorý sa vyučuje na hodinách informatiky na strednej škole.

Cieľom prvej kapitoly je podať čitateľovi základné pokyny pri práci s touto príručkou. Druhá kapitola oboznamuje čitateľa s vývojovým prostredím a ukazuje postup pri tvorbe jednoduchého programu. Tretia kapitola sa venuje premenným, čiže základným stavebným kameňom jazyka C++. Štvrtá kapitola sa zaoberá základnými príkazmi, ako aj príkazmi binárneho a viacnásobného vetvenia. Piata kapitola je venovaná významu a použitiu konštánt v jazyku C++. Cieľom šiestej kapitoly je oboznámiť čitateľa s programovými cyklami. Siedma kapitola je venovaná využitiu funkcií v programe. Ôsma kapitola popisuje problematiku jednorozmerných a viacrozmerných polí. Deviata kapitola sa venuje použitiu znakov v jazyku C++. A napokon desiata kapitola popisuje prácu s textovým súborom.

OBSAH

ÚVOD

Táto práca bola napísaná na základe pedagogickej skúsenosti z vyučovania Seminára z informatiky na strednej škole (ISCED 3). Je určená na výučbu žiakov, ktorí sa pripravujú na štúdium na vysokej škole technického zamerania. Zaoberá sa výučbou základných štruktúr programovacieho jazyka C++. Obsahuje teoretický základ pri jednotlivých preberaných pojmoch, podrobne popísané vzorové príklady, riešené cvičenia, na konci každej kapitoly úlohy na riešenie pre žiakov a v prílohách sú uvedené doplnkové materiály a dva priebežné testy, ktorými učiteľ zistí hĺbku osvojenia daného učiva žiakmi.

Motiváciou na napísanie tejto práce je to, že síce rôzneho materiálu ohľadom daného jazyka je pomerne dosť, ale učebnica jazyka C++ v takejto ucelenej forme pre učiteľov chýba. Učiteľovi tak práca poskytne základné informácie pohromade, čo preň znamená aj veľkú časovú úsporu, pretože jednotlivé materiály nemusí prácne zháňať po knižniciach, resp. na Internete. Postupnosť tém ako sú preberané nemusí korešpondovať s programátorskými príručkami, ktoré si môžeme buď kúpiť vo forme kníh, resp. nájsť na Internete. Obsah a aj postupnosť jednotlivých tém vychádza z praktických skúseností na jednotlivých hodinách, kde sa dá najlepšie zistiť, čo žiaci dokážu pochopiť rýchlo, a naopak čo žiakom robí menšie alebo väčšie problémy a podľa toho je volená aj časová dotácia pre jednotlivé kapitoly.

Hlavným cieľom práce je pomôcť učiteľom informatiky na strednej škole s prípravou na vyučovanie programovacieho jazyka C++. Pri vyučovaní jazyka C++ podľa tejto práce požadujeme, aby žiaci ovládali základné programátorské vedomosti a zručnosti z akéhokoľvek iného programovacieho jazyka.

Práca je rozdelená do desiatich kapitol a obsahuje pokyny pri práci s týmto materiálom a popis základných štruktúr programovacieho jazyka C++, ktoré by mali poskytnúť základ do ďalšieho štúdia daného jazyka.

1 ZÁKLADNÉ POKYNY PRI PRÁCI S TÝMTO MATERIÁLOM

Výukový materiál „Základy programovania v jazyku C++“ sa zaoberá základmi jedného z najpoužívanejších jazykov vo svete. Je určený ako pomôcka pre učiteľa pri výučbe programovacieho jazyka C++. Sú v ňom popísané pokyny, ako s týmto materiálom pracovať, základná teória, riešené cvičenia, príklady pre žiakov a samozrejme aj dva priebežné testy na overenie vedomostí a znalostí žiakov z prebraného učiva. Materiál je plánovaný na 24 vyučovacích hodín. Časová dotácia jednotlivých blokov je informačná a vychádza zo skúseností a pozorovania autora. Je ju možné podľa potreby modifikovať. Ďalej je nutné, aby žiaci:

- ovládali prácu s počítačom (príslušným hardvérom a softvérom),
- ovládali základy akéhokoľvek iného programovacieho jazyka.

1.1 Špecifikácia cieľovej skupiny a prehľad cieľov

Práca je určená pre žiakov strednej školy s využitím v predmete Informatika, resp. Seminár z informatiky (ISCED 3), ale primárne pre tých, ktorí sa chystajú na štúdium na vysokej škole technického zamerania. Žiaci pritom musia ovládať aspoň základy akéhokoľvek iného programovacieho jazyka.

Učiteľ by mal mať aprobáciu pre výučbu predmetu Informatika a aspoň základné skúsenosti s programovaním v jazyku C++.

Hlavným cieľom práce je pomôcť učiteľom informatiky na stredných školách s prípravou na hodiny informatiky, kde sa vyučuje programovanie v jazyku C++. Ďalším cieľom je aj budovanie medzipredmetových vzťahov informatiky a matematiky, pretože výstupné programy práce môžu slúžiť na kontrolu výsledkov riešení rôznych matematických úloh, ako je napríklad hľadanie najväčšieho spoločného deliteľa, najmenšieho spoločného násobku, pri výpočte koreňov kvadratickej rovnice a podobne.

Žiaci by mali po absolvovaní hodín programovania:

- vedieť skompilovať program a analyzovať chyby, ktoré sa môžu v programe vyskytnúť,
- vedieť vytvoriť a spustiť jednoduchý program podľa zadania a okomentovať ho,
- rozumieť práci s premennými a konštantami,
- vedieť v programe využiť jednoduché príkazy, ako aj príkazy binárneho a viacnásobného vetvenia,
- vedieť v programe správne využiť programové cykly, ako je nepodmienený cyklus *for,* tak aj podmienené cykly *while* a *do-while,*
- vedieť pracovať s funkciami v programe, teda funkciu správne zadeklarovať a následne ju v kóde aj správne použiť,
- rozumieť rozdielu medzi číselnou hodnotou a znakom a vedieť zadeklarovať premennú, do ktorej sa môžu ukladať znaky,
- vedieť pracovať s jednorozmerným aj dvojrozmerným poľom,
- vedieť pracovať s textovým súborom (čítať údaje z textového súboru, resp. zapisovať údaje do textového súboru).

Žiaci môžu zadané úlohy riešiť individuálne, ale odporúča sa vytvoriť menšie skupiny, kde je veľké pozitívum to, že žiaci medzi sebou spolupracujú, spoločne problém analyzujú a snažia sa spoločne dôjsť k správnemu riešeniu. Je to pre nich veľmi dôležitá skúsenosť pre prax, kde veľmi často na takúto spoluprácu budú odkázaní.

1.2 Štruktúra práce

Učiteľ si môže časové dotácie podľa potreby upraviť. Odporúča sa 30 až 45 minút (záleží na obšírnosti a zložitosti preberanej kapitoly) venovať teoretickým poznatkom (napr. definovaniu jednotlivých pojmov, porovnaní syntaxe príkazov jazyka C++ s jazykom, ktorý žiaci mali už pred tým, atď.) a spoločne preriešiť vzorový príklad, resp. riešené cvičenie. Ostatné hodiny sú venované samostatnej práci žiakov, kde riešia úlohy, ktoré sa nachádzajú na konci každej kapitoly. Učiteľ pritom kontroluje samostatnú prácu žiakov, koordinuje ich činnosť, poprípade sa ich snaží naviesť na správne riešenie.

Celý výukový systém je plánovaný na už vyššie uvedených 24 vyučovacích hodín, pričom sú plánované aj dva priebežné testy. Podrobnosti sú uvedené v Tabuľke 1.

Tabuľka 1 Prehľad aktivít výukového systému „Základy programovania v jazyku C++“

Kapitola	Počet vyučovacích hodín	Pomôcky
Úvod do jazyka C++	2	• počítač s príslušným softvérovým vybavením, • projektor, • premietacie plátno
Premenné	2	
Príkazy	3	
Konštanty	2	
Programové cykly	4	
Prvý priebežný test	1	Príloha 5, Príloha 6
Funkcie	2	• počítač s príslušným softvérovým vybavením, • projektor, premietacie plátno
Polia	3	
Znaky a pole znakov	2	
Vstup z textového súboru a výstup do textového súboru	2	
Druhý priebežný test	1	Príloha 7, Príloha 8

Zdroj: súkromný archív

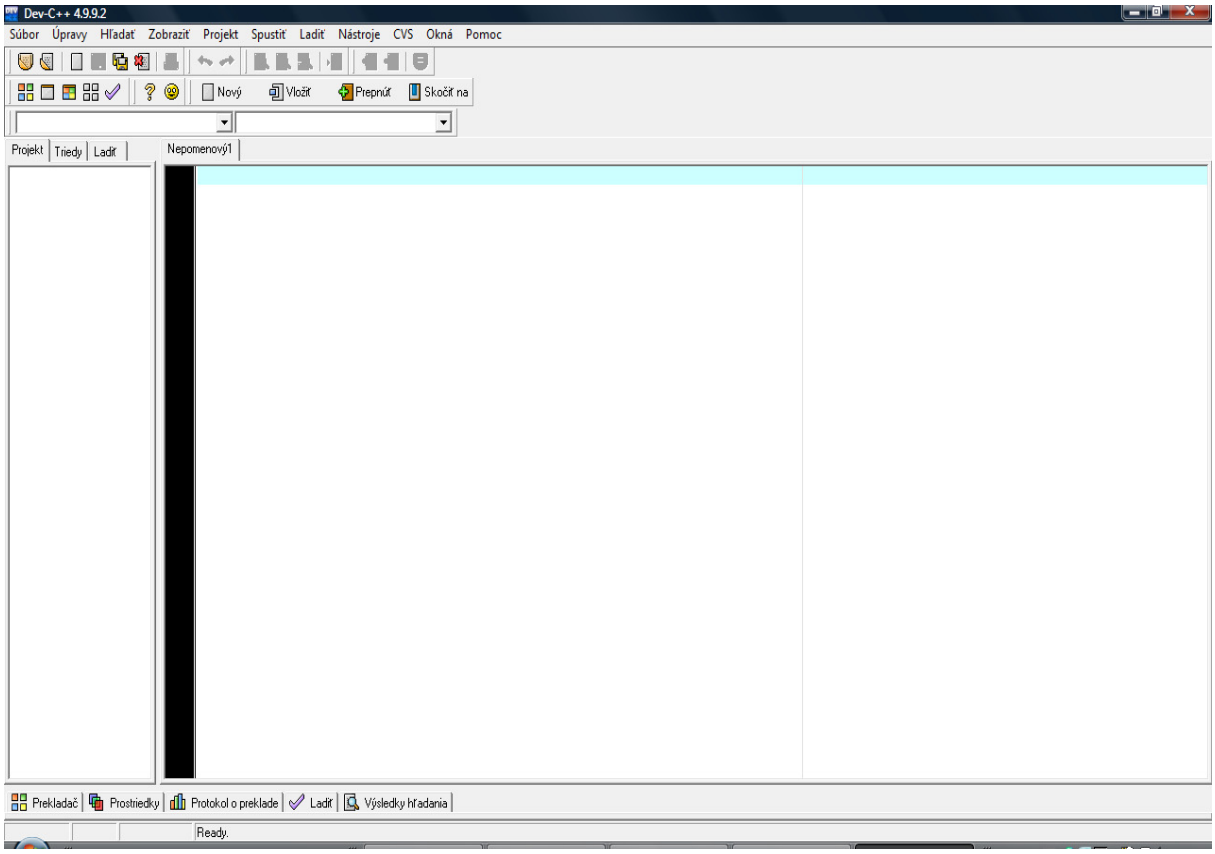
2 ÚVOD DO JAZYKA C++

Na vytvorenie programu je potrebné integrované vývojové prostredie (IDE), do ktorého nami navrhnutý zdrojový kód budeme písať, kompilovať a následne spúšťať, testovať a ladiť. Medzi profesionálnejšie a komplikovanejšie vývojové prostredia, ktoré nie sú určené len pre jazyk C++ patria napríklad Visual studio, Eclipse, resp. NetBeans. Pre naše účely však bude úplne postačovať integrované vývojové prostredie Dev-C++.

2.1 Integrované vývojové prostredie *Dev-C++*

IDE Dev-C++ je potrebné najprv nainštalovať do počítača (voľné verzie sa dajú stiahnuť z internetu). Základná práca pri vývoji, kompilácii a spustení programu potom pozostáva z nasledovných bodov:

1. spustiť aplikáciu,
2. z hlavnej ponuky vybrať: Súbor – Nový – Zdrojový kód,
3. napísať samotný zdrojový kód,
4. z hlavnej ponuky vybrať: Spustiť – Preložiť & spustiť alebo stlačiť klávesu F9,
5. vytvorí sa .exe súbor, ktorý môžeme spustiť pod ľubovoľným operačným systémom.



Obrázok 1 Integrované vývojové prostredie Dev-C++
Prameň: okno aplikácie Dev-C++

2.2 Kompilácia, chyby pri kompilácii

Pri kompiláci programov pracujeme s pojmami ako je samotný pojem kompilácia, potom pojmy testovanie, ladenie a krokovanie programu. Kompilácia je prevedenie zdrojového textu

do strojového kódu, ktorému rozumie počítač. Pod strojovým kódom rozumieme jednotky a nuly. Testovanie programu je činnosť, pri ktorej zisťujeme správnosť algoritmu (napr. zadávanie rozličných vstupných hodnôt) a jeho správanie sa v rôznych hraničných situáciách. Ladenie programu je proces, počas ktorého zistené chyby odstraňujeme a následne daný program opätovne testujeme. Krokovanie (trasovanie, debugovanie) je jedným z najpoužívanejších spôsobov hľadania chýb v programe. Pri krokovaní nespustíme program tak, aby sa vykonal celý, ale postupujeme po jednotlivých príkazoch. Okrem toho, že vidíme postup a jednotlivé vetvy, cez ktoré sa pri vykonávaní programu postupuje, môžeme zobraziť aj hodnoty premenných a tak zistiť, v ktorom príkaze nadobudnú nevhodný obsah.

K chybám pri kompilácii môže dôjsť z viacerých dôvodov. Často sa jedná o preklep alebo nejakú inú neúmyselnú chybu. Dobrý kompilátor nám nielen oznámi, čo sme spravili zle, ale tiež nám v kóde ukáže miesto, na ktorom k chybe došlo. Najlepšie kompilátory nám dokonca ponúknu aj možnosť, ako danú chybu opraviť.

Chyby v programe delíme na:

1. syntaktické (syntax errors) – príčinou týchto chýb je neznalosť alebo nepozornosť pri písaní programu. Vznikajú vtedy, keď nedodržíme pravidlá (syntax) pri písaní príkazov, napr. vynechanie bodkočiarky za príkazom, používanie premennej bez toho, aby sme ju vopred deklarovali, chybne napísaným príkazom (napr. napíšeme *cut* namiesto *cout*) a pod.,
2. sémantické – máme splnené všetky podmienky pre zápis programu, no napriek tomu pomocou neho správny výsledok nezískame. Túto kategóriu chýb možno rozdeliť na:
   a) chyby vznikajúce počas behu programu (run-time errors) – sa objavia vtedy, keď počítač nie je schopný pokračovať v práci, pretože nastal stav, ktorý môže zapríčiniť nesprávnosť výsledku. Sú to chyby spôsobené tým, že program pracuje s chybnými dátami, obvykle vloženými užívateľom programu. Patrí sem napr. delenie nulou, otváranie neexistujúceho súboru, výpočet väčšieho čísla, aké je počítač schopný spracovať, atď.,
   b) logické chyby (logical errors) – sa hľadajú najťažšie. Nie sú zapríčinené nesprávnym prepisom algoritmu do programovacieho jazyka, ale nesprávnosťou samotného algoritmu. V tomto prípade program robí niečo úplne iné (pracuje inak, počíta niečo iné), ako to, čo sa od neho vyžaduje. Program zvyčajne beží bezproblémovo, no v niektorých (prípadne všetkých) prípadoch vracia zlé hodnoty. (Napríklad *X = A – 1;* Premenná *X* obsahuje nesprávnu hodnotu, lebo výraz je chybne napísaný, pretože správne malo byť *X = A + 1;*). Na odhalenie tohto typu chýb potrebujeme mať určité skúsenosti alebo aspoň dobrého poradcu, pretože inak by ich nájdenie mohlo trvať veľmi dlho. Zvyčajne je potrebné program analyzovať a krokovať po jednotlivých príkazoch.

2.3 Jednoduchý program

Na úvod si popíšme a zanalyzujme jednoduchý program, ktorý môže vyzerať takto:

```
0:      #include <iostream>
1:
2:      int main()
3:      {
4:              std::cout << "Cau!\n";
5:              return 0;
6:      }
```

Výstup: Cau!

Analýza programu: Na riadku 0 je do aktuálneho súboru zahrnutý súbor *iostream.* Ako program funguje? Prvým znakom je *#,* čo je signálom pre preprocesor. Preprocesor sa spustí pri každom spustení kompilátora. Prejde si zdrojový kód a nájde riadky, ktoré začínajú symbolom kríža (#). S týmito riadkami pracuje ešte pred spustením samotného kompilátora. Príkaz *include* je inštrukcia pre preprocesor, ktorá mu hovorí: „To, čo nasleduje, je názov súboru. Nájdi tento súbor a vlož jeho obsah priamo na toto miesto programu.“ Lomené zátvorky okolo názvu súboru preprocesoru hovoria, aby tento súbor hľadal na obvyklých miestach. Ak je kompilátor správne nastavený, lomené zátvorky znamenajú, že preprocesor bude súbor *iostream* hľadať v adresári, ktorý pre náš kompilátor uchováva všetky súbory, ktoré sú obvykle predmetom operácie zahrnutia. Súbor *iostream* (Input-Output-Stream, prúd vstupu a výstupu) používa objekt *cout,* ktorý zaisťuje písanie na obrazovku. Výsledkom riadku 0 bude zahrnutie súboru *iostream* do tohto programu, ako keby sme jeho obsah do programu napísali sami. Na riadku 2 začína vlastný program, a to funkciou s názvom *main().* Každý program v C++ obsahuje túto funkciu. Pod funkciou rozumieme úsek kódu, ktorý vykonáva jednu alebo viac akcií. Funkcie sú obvykle spúšťané alebo volané inými funkciami, ale funkcia *main()* má zvláštny význam. Je volaná automaticky po spustení programu hneď ako prvá. Podobne ako v prípade iných funkcií, musí byť aj u funkcie *main()* stanovené, aký typ hodnoty bude vracať. Typ vrátenej hodnoty funkcie *main()* v tomto programe je *int,* čo znamená, že táto funkcia vráti operačnému systému po svojom ukončení celočíselnú hodnotu. V našom prípade vracia celočíselnú hodnotu 0, ako vidíme na riadku 5. Vrátenie hodnoty operačnému systému je relatívne nepodstatná a málo používaná vlastnosť, ale štandard C++ vyžaduje, aby bola funkcia *main()* deklarovaná vyššie uvedeným spôsobom.

Dobrá rada: Niektoré kompilátory dovoľujú funkciu *main()* deklarovať tak, že vracia hodnotu *void* (bezrozmerná hodnota). To však v C++ už nie je naďalej prípustné. Preto ako návratový typ funkcie *main()* budeme používať typ *int* a na poslednom riadku funkcie budeme vracať hodnotu 0. Niektoré operačné systémy umožňujú testovať hodnotu vrátenú programom. Podľa dohody je návratová hodnota 0 signálom normálneho ukončenia programu.

Pokračujme však ďalej v analýze programu. Všetky funkcie sú ohraničené počiatočnou (*{*) a koncovou (*}*) zloženou zátvorkou. Na riadkoch 3 a 6 sú zátvorky k funkcii *main().* Všetko, čo sa nachádza medzi týmito zátvorkami, sa považuje za súčasť dotyčnej funkcie. Najpodstatnejšia časť celého programu je na riadku 4. Na výpis hodnôt (napr. na obrazovku) sa používa objekt *cout.* Na načítanie hodnôt (napr. z klávesnice) sa používa objekt *cin.* Objekt *cout* sa nachádza v štandardnej knižnici. Pod knižnicou rozumieme zbierku tried (podprogramov) a štandardná knižnica je štandardná zbierka, ktorá je súčasťou každého kompilátora spĺňajúceho štandard ANSI[1]. Špecifikátor oboru názvov *std* kompilátoru oznamuje, že objekt *cout,* ktorý chceme použiť, je súčasťou štandardnej knižnice, ktorej názov je práve *std.* Pretože v rôznych knižniciach alebo v programových moduloch môžu existovať objekty s rovnakým názvom, rozdeľuje C++ všetko do tzv. „oborov názvov“ (resp. názvových priestorov – anglicky *namespace*). Kompilátoru sa jeho použitím oznamuje: „Nasledujúce meno *cout* je meno, ktoré je súčasťou štandardného oboru názvov *std* a žiadneho iného oboru názvov.“ Syntax vyzerá tak, že pred *cout* sú umiestnené znaky *std* a za nimi nasledujú dve dvojbodky. Ďalej sa o oboroch názvov dozvieme viac. Objekt *cout* sa používa takto: Napíšeme slovo *cout* a zaň operátor presmerovania výstupu <<. Čokoľvek nasleduje za operátorom presmerovania výstupu, objaví sa na obrazovke. Ak chceme, aby sa

[1] http://sk.wikipedia.org/wiki/ANSI, 13. 11. 2012

vypísal reťazec znakov, musíme ho dať do úvodzoviek ("), ktoré si môžeme všimnúť v riadku 4. Pod textovým reťazcom rozumieme postupnosť znakov, ktoré je možné zobraziť na obrazovke výstupu alebo zapísať do textového súboru. Posledné dva znaky \n objektu *cout* značia zalomenie riadka za slovom *Cau!*

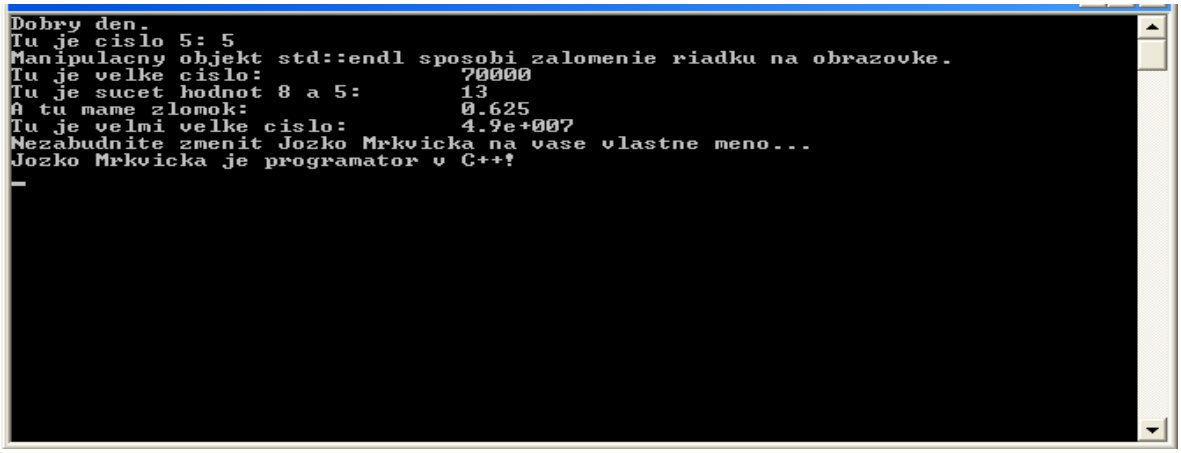
Dobrá rada: Po spustení programu sa výstupná obrazovka mihne tak rýchlo, že výsledky nestačíme ani prečítať. Jedným zo spôsobov odstránenia daného problému je pridať medzi riadok *std::cout << "Ahoj!\n";* a riadok *return 0;* nasledujúci kód:
char reakcia;
std::cin >> reakcia;
To spôsobí, že sa program zastaví a bude čakať, pokiaľ nezadáme nejakú hodnotu. Ak teda chceme program ukončiť, zadáme číslo alebo nejaký znak a potvrdíme klávesom *Enter.*

2.4 Objekt *cout*

Na konkrétnych príkladoch sa teraz pozrime na využitie objektu *std::cout*:

```
0: // Výpis – používanie objektu std::cout
1: #include <iostream>
2: int main()
3: {
4:       std::cout << "Dobry den.\n";
5:       std::cout << "Tu je cislo 5: " << 5 << "\n";
6:       std::cout << "Manipulacny objekt std::endl ";
7:       std::cout << "sposobi zalomenie riadku na obrazovke.";
8:       std::cout << std::endl;
9:       std::cout << "Tu je velke cislo:\t\t" << 70000;
10:      std::cout << std::endl;
11:      std::cout << "Tu je sucet hodnot 8 a 5:\t";
12:      std::cout << 8+5 << std::endl;
13:      std::cout << "A tu mame zlomok 5/8:\t\t";
14:      std::cout << (float) 5/8 << std::endl;
15:      std::cout << "Tu je velmi velke cislo:\t";
16:      std::cout << (double) 7000 * 7000 << std::endl;
17:      std::cout << "Nezabudnite zmenit Jozko Mrkvicka ";
18:      std::cout << "na vase vlastne meno...\n";
19:      std::cout << "Jozko Mrkvicka je programator v C++!\n";
20:      char reakcia;
21:      std::cin >> reakcia;
22:      return 0;
23: }
```

Výstup:



Obrázok 2 Výstupná obrazovka – využitie objektu *std::cout*

Zdroj: súkromný archív

Analýza programu: Príkaz *#include <iostream>* v 1 riadku spôsobí, že sa súbor *iostream* pripojí k zdrojovému súboru. To je nutné vždy, keď sa v programe používa objekt *cout* a s ním súvisiace funkcie. Riadok 4 ukazuje najjednoduchšie použitie objektu *cout*: výstup reťazca, čiže postupnosti znakov na obrazovku. Symbol *\n* je zvláštny znak pre formátovanie, ktorý objekt *cout* interpretuje tak, že na obrazovku vypíše znak pre nový riadok. V riadku 5 sa objektu *cout* predajú tri hodnoty, ktoré sú navzájom oddelené operátorom vloženia. Prvá hodnota je reťazec "Tu je cislo 5: ". Za dvojbodkou je medzera, ktorá je súčasťou reťazca. Potom sa predá operátoru vloženia hodnota 5 a znak nového riadka (vždy musí byť v apostrofoch, resp. úvodzovkách). Tým dôjde k výstupu riadku: „Tu je cislo 5: 5“ na obrazovku. Pretože za prvým reťazcom nie je žiadny znak nového riadku, bude sa ďalšia hodnota vypisovať bezprostredne za daným reťazcom. Označujeme to ako zreťazenie hodnôt. Na riadku 6 je vypísaná informačná správa a potom sa použije manipulačný objekt *endl.* Jeho úlohou je vypísať na obrazovku nový riadok. Objekt *endl* je tiež súčasťou štandardnej knižnice.

Dobrá rada: Manipulátor *endl* znamená koniec riadku (z anglického *end line*). V programoch je lepšie používať *endl* namiesto *\n*, pretože manipulátor *endl* sa prispôsobí používanému operačnému systému, zatiaľ čo *\n* nemusí byť na niektorých platformách dostačujúcim znakom pre správne vytvorenie nového riadka.

V riadku 9 sa po prvý krát stretávame s formátovacím znakom *\t*. Ten vloží na výstup znak tabulátora a používa sa v riadkoch 9 až 15, aby bol výstup zarovnaný. Riadok 9 ukazuje, že je možné okrem malých celočíselných hodnôt vypísať aj veľké čísla. Riadok 12 demonštruje, že objektu *cout* je možné predložiť aj jednoduché sčítanie. Objektu *cout* je predaná hodnota 8+5, ale na obrazovku sa vypíše hodnota súčtu 13. V riadku 14 sa do objektu cout vloží 5/8. Výraz *(float)* hovorí, že táto hodnota má byť vyhodnotená ako desatinné číslo, a tak na výstupe vidíme hodnotu zlomku. V riadku 16 je objektu *cout* predaná hodnota 7000*7000 a výraz *(double)* spôsobí, že ju objekt *cout* vyhodnotí ako číslo s plávajúcou desatinnou čiarkou. V riadku 19 máme doplniť svoje meno a na výstupe tak potvrdiť, že sme skutočne programátorom, či programátorkou v C++.

Dobrá rada: Niektoré kompilátory môžu považovať za chybu, ak okolo výrazu pre súčet nedáme okrúhle zátvorky. V takomto prípade by sa riadok 12 zmenil takto:

std::cout << (8+5) << std::endl;

Možno po čase zistíme, že používať *std::* pred každým výskytom *cout, cin,* resp. *endl* začína byť trochu únavné. Aj keď sa jedná o presný spôsob stanovenia oboru názvov, môže nás ich neustále zadávanie zdržiavať. Štandard ANSI ponúka dve riešenia tohto drobného problému. Prvou možnosťou je kompilátoru na začiatku programu oznámiť, že budeme používať objekty *cout, cin* a *endl* zo štandardnej knižnice. Všeobecne kód bude potom vyzerať napríklad takto:

```
#include <iostream>
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;
    príkazy;
    return 0;
}
```

Druhou možnosťou, ktorou sa môžeme vyhnúť nepraktickému písaniu *std::* pred objekty *cout, cin,* resp. *endl* spočíva v špecifikácii celého štandardného oboru názvov. To znamená, že pri každom objekte, pokiaľ nebude určené inak, sa predpokladá , že pochádza zo štandardného oboru názvov. Na to použijeme príkaz *using namespace std;* Kód potom môže vyzerať takto:

```
#include <iostream>
int main()
{
    using namespace std;
    príkazy;
    return 0;
}
```

2.5 Komentáre

Keď program píšeme, tak v tej chvíli je nám úplne jasné, čo činíme, program sa nám zdá v tej chvíli úplne zrozumiteľný a jasný. Po nejakom čase, keď sa k programu opäť vrátime, sa nám už taký jasný a čitateľný zdať nemusí. Aby sme sa tomu vyhli a aby sme mohli pomôcť aj ostatným porozumieť nášmu kódu, je potrebné program okomentovať a teda do programu zaradiť komentáre. Jedná sa o text, ktorý kompilátor ignoruje, ale ktorý čitateľa informuje o tom, čo sa v tom ktorom mieste programu deje.

V jazyku C++ používame jednoriadkové a viacriadkové komentáre. Jednoriadkové komentáre sa vytvárajú pomocou dvoch lomiek *//*. Kompilátor bude potom ignorovať všetko, čo nasleduje za týmto komentárom až do konca riadku. Viacriadkové komentáre začínajú lomkou, za ktorou nasleduje hviezdička /*. Táto značka komentára vraví kompilátoru, aby preskočil všetko ďalšie až po znaky komentára **/*.

Použitie komentárov si vysvetlíme na nasledujúcom vzorovom príklade:

```
#include <iostream>
int main()
{
	using std::cout;
	using std::cin;

	/* toto je komentár
	a siaha až po ukončujúcu
	značku hviezdičky a lomítka */
	cout << "Nazdar ludia!\n";
	// tento komentár končí na konci riadku
	cout << "Komentar skoncil!\n";

	// komentáre s dvojitou lomkou môžu byť na samostatnom riadku
	/* takisto ako komentáre s lomkou a hviezdičkou */
	char reakcia;
	cin >> reakcia;
	return 0;
}
```

Podstatou komentárov je to, že komentáre by nemali hovoriť čo sa deje, ale prečo sa to deje.

Úloha 2.1: Nájdite chyby v danom kóde:

```
include <iosteam>

int main()
{
	std:cout >> "Cau!;
	char reakcia;
	std::cin >> reakcia
	return 0;
{
```

Úloha 2.2: Napíšte program, ktorý na obrazovku pod seba vypíše tri programovacie jazyky, ktoré poznáte. Každý riadok kódu okomentujte.

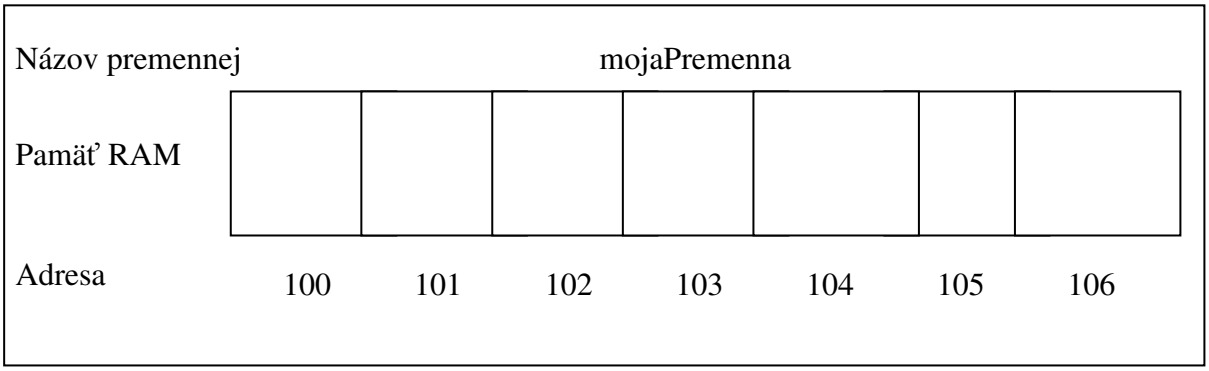
3 PREMENNÉ

Premenná je v programe C++ miesto, kde sa uchováva informácia. Rozumieme tým miesto v pamäti počítača, na ktoré môžeme hodnotu uložiť a z ktorého ju môžeme neskôr znovu získať. Premenné poskytujú len dočasné uchovanie informácií. Keď počítač vypneme, tieto hodnoty sa stratia. Trvalé uchovávanie hodnôt je už iná záležitosť a obvykle sa realizuje v databáze alebo sa hodnoty uložia do súboru. Základné typy premenných sú uvedené v Prílohe 3.

3.1 Reprezentácia dát v pamäti počítača

Pamäť počítača si môžeme predstaviť napr. ako rad priečinkov, ktoré sú zoradené vedľa seba. Každému priečinku – umiestneniu v pamäti – je postupne priradené číslo. Týmto číslam hovoríme adresy v pamäti. Premenná si vyhradí jeden alebo aj viac priečinkov, do ktorých môže uložiť hodnotu.

Pod názvom premennej (napr. mojaPremenna) rozumieme menovku na priehradke, vďaka ktorej ju môžeme ľahko nájsť bez toho, aby sme museli poznať skutočnú adresu v pamäti. Na obrázku 3 je schématické znázornenie pamäti. Všimnime si, že premenná *mojaPremenna* začína na pamäťovej adrese s číslom 103. Podľa veľkosti potom môže premenná *mojaPremenna* zabrať jednu alebo aj viac pamäťových adries.



Obrázok 3 Schematické znázornenie pamäte

Prameň: Liberty Jesse, 2007, s. 58

3.2 Vymedzenie pamäte

Keď v C++ definujeme premennú, musíme kompilátoru oznámiť, o aký druh premennej sa jedná (či je to celé číslo, znak a pod.). Podľa tejto informácie vyhradí kompilátor v pamäti príslušné miesto a zároveň vie, aký druh hodnoty chceme v premennej uchovávať.

Kompilátor má tiež možnosť nás varovať alebo zobraziť chybu, keď sa pokúšame uložiť do určitej premennej hodnotu nesprávneho typu (programovací jazyk s touto charakteristikou sa označuje ako „silno typový“).

Každý priečinok má veľkosť 1 bajt. Ak typ premennej, ktorý vytvoríme, bude mať veľkosť 4 bajty, vyhradia sa 4 bajty v pamäti, čiže 4 priečinky. Prostredníctvom typu premennej (napr. celé číslo) kompilátoru oznámime, koľko pamäte (koľko priečinkov) si má pre túto premennú vyhradiť.

Veľkosť celého čísla určuje procesor (32 bitový alebo 64 bitový) a kompilátor, ktorý používame. Nasledujúci program nám vypíše, aká je v našom prípade presná veľkosť jednotlivých dátových typov. K tomu použijeme operátor *sizeof(),* ktorý je súčasťou kompilátora a vracia veľkosť objektu, ktorý mu predáme ako parameter.

```
// Určenie veľkostí typov premenných na našom počítači
#include <iostream>
int main()
{
        using std::cout;
        using std::cin;

        cout << "Velkost typu int je:\t\t"
                << sizeof(int) << " bajtov.\n";
        cout << "Velkost typu short je:\t\t"
                << sizeof(short) << " bajtov.\n";
        cout << "Velkost typu long je:\t\t"
                << sizeof(long) << " bajtov.\n";
        cout << "Velkost typu char je:\t\t"
                << sizeof(char) << " bajtov.\n";
        cout << "Velkost typu float je:\t\t"
                << sizeof(float) << " bajtov.\n";
        cout << "Velkost typu double je:\t"
                << sizeof(double) << " bajtov.\n";
        cout << "Velkost typu bool je:\t\t"
                << sizeof(bool) << " bajtov.\n";
        char reakcia;
        cin >> reakcia;
        return 0;
}
```

3.3 Celé čísla so znamienkom (*signed*) a bez znamienka (*unsigned*)

U všetkých celočíselných typov rozlišujeme dve varianty: čísla so znamienkom (*signed*) a bez znamienka (*unsigned*). Podstata spočíva v tom, že za istých okolností môžeme potrebovať pracovať so zápornými číslami, inokedy nie. Celé čísla všetkých typov (*short* aj *long*), ktoré nie sú explicitne označené ako *unsigned,* teda bez znamienka, sa považujú za *signed,* teda so znamienkom.

3.4 Definovanie premennej

Keď deklarujeme premennú, dôjde k alokácii (vyhradeniu) pamäte pre túto premennú. Premenná sa zavádza uvedením jej typu, za ktorým nasleduje jedna alebo viac medzier, názov premennej a bodkočiarka. Napríklad nasledujúci príkaz definuje celočíselnú premennú s názvom *mojVek*:

int mojVek;

Vytvorenie viac premenných naraz:

unsigned int mojVek, mojaHmotnost; // dve celočíselné premenné bez znamienka
long int plocha, sirka, dlzka; // tri premenné typu long

Jazyk C++ rozlišuje medzi malými a veľkými písmenami (je to case-sensitive jazyk). Inými slovami, premenná s názvom *vek* nie je totožná s premennou s názvom *Vek* a tá je zase iná, ako premenná *VEK*.

V jazyku C++ sú niektoré slová vyhradené a my ich nesmieme používať ako názvy premenných. Patria sem napr. *if, while, for, main,* atď.

Hodnota sa do premennej priradzuje pomocou operátora priradenia (=). Premennej s názvom *Sirka* priradíme hodnotu 5 takto:
unsigned short Sirka;
Sirka = 5;
Obidva tieto kroky môžeme spojiť a premennú Sirka inicializovať priamo v definícii:
unsigned short Sirka = 5;

Poznámka: *long* je skrátená verzia pre *long int* a *short* je skrátená verzia pre *short int.*

Inicializácia sa veľmi podobá priradeniu a u celočíselných premenných je rozdiel medzi nimi minimálny. Neskôr, keď budeme preberať konštanty, uvidíme, že niektoré hodnoty je nutné inicializovať, pretože nie je možné do nich neskôr priradiť hodnotu. Rozdiel spočíva hlavne v tom, že k inicializácii dochádza v tom istom okamihu, keď premenná vznikne.
// vytvorenie viacerých premenných typu long a ich inicializácia
long sirka = 5, dlzka = 7;
V tomto prípade sa inicializuje celočíselná premenná *sirka* typu *long* na hodnotu 5 a celočíselná premenná *dlzka* typu *long* na hodnotu 7. Je dokonca možné aj kombinovať definície s inicializáciami:
int mojVek = 25, tvojVek, jehoVek = 18;
V tomto prípade sa vytvoria tri premenné typu *int* a prvá a tretia sa zároveň aj inicializujú.

Cvičenie 3.1: Napíšte program, ktorý vypočíta obsah obdĺžnika, ak sú dané dĺžky jeho strán.
Riešenie:

```
#include <iostream>
int main()
{
        using std::cout; using std::cin; using std::endl;
        unsigned short int Sirka = 5, Dlzka;
        Dlzka = 10;
        unsigned short int Plocha = (Sirka * Dlzka);
        cout << "Sirka: " << Sirka << "\n";
        cout << "Dlzka: " << Dlzka << endl;
        cout << "Plocha: " << Plocha << endl;
        char reakcia;
        cin >> reakcia;
        return 0;
}
```

Výstup:
Sirka: 5
Dlzka: 10
Plocha: 50

3.5 Kľúčové slovo *typedef*

Opakované písanie napr. *unsigned short int* v programe môže byť únavné, a čo je najdôležitejšie, náchylné k chybám. Jazyk C++ dokáže pre celú túto frázu vytvoriť pomocou kľúčového slova *typedef* skratku (alias). Slovo *typedef* pochádza z anglického *type definition* (definícia typu).

Prakticky tak môžeme vytvárať synonymá a je dôležité, aby sme túto operáciu rozlišovali od tvorby nových typov. Za kľúčovým slovom *typedef* napíšeme daný použitý názov typu a potom nový názov. Výraz sa ukončí bodkočiarkou. Napríklad:
typedef unsigned short int USHORT;
Vznikne tu nový názov typu *USHORT,* ktorý budeme môcť použiť všade tam, kde by sme inak museli písať *unsigned short int.* Môžeme si to ukázať na nasledujúcom príklade:

```
#include <iostream>

typedef unsigned short int USHORT;

int main()
{
        using std::cout;
        using std::cin;
        using std::endl;

        USHORT Sirka = 5;
        USHORT Dlzka;
        Dlzka = 10;
        USHORT Plocha = (Sirka * Dlzka);
        cout << "Sirka: " << Sirka << "\n";
        cout << "Dlzka: " << Dlzka << endl;
        cout << "Plocha: " << Plocha << endl;
        char reakcia;
        cin >> reakcia;
        return 0;
}
```

Úloha 3.1: Vytvorte program, ktorý vypočíta objem a povrch rotačného valca, ak užívateľ na vstupe zadá polomer podstavy valca a výšku valca.
Úloha 3.2: Napíšte program, ktorý si od užívateľa vypýta rok jeho narodenia a následne vypíše rok, v ktorom bude mať užívateľ 50 rokov.

4 PRÍKAZY

Príkazy v C++ dovoľujú riadiť vykonávanie kódu, vyhodnocovať výrazy alebo nerobiť vôbec nič (prázdny príkaz). Všetky príkazy v C++ sú zakončené bodkočiarkou, dokonca aj prázdny príkaz, čo je vlastne bodkočiarka a nič iného. Jeden z najbežnejších jednoduchých príkazov, príkaz priradenia má nasledujúcu podobu:
x = a+b;

Všade tam, kde môžeme použiť jeden príkaz, je možné vložiť aj zložený príkaz, ktorý sa označuje pojmom *blok*. Blok začína ľavou zloženou zátvorkou *{* a končí pravou zloženou zátvorkou *}*. Aj keď každý príkaz v bloku musí byť ukončený bodkočiarkou, blok sám sa bodkočiarkou neukončuje. Pozrime si príklad:

```
{
        pom = a;
        a = b;
        b = pom;
}
```

Tento blok sa vykoná ako jeden príkaz, pričom sa v ňom vymenia hodnoty v premenných *a* a *b*.

Cvičenie 4.1: Napíšte program, ktorý vymení hodnoty v dvoch ľubovoľných celočíselných premenných.
Riešenie:

```
#include <iostream>
using namespace std;

int main()
{
 int a,b,pom;
 cout << "Zadaj hodnotu do premennej a: ";
 cin >> a;
 cout << "Zadaj hodnotu do premennej b: ";
 cin >> b;
 cout << endl;

 pom = a;
 a = b;
 b = pom;

 cout << "Vpremennej a je hodnota: " << a << "\n";
 cout << "Vpremennej b je hodnota: " << b << "\n";

   char reakcia;
   cin >> reakcia;
   return 0;
 }
```

Všetko, čo sa vyhodnotí na nejakú hodnotu, sa v C++ označuje pojmom *výraz.* Hovoríme, že výraz vracia hodnotu. Napríklad 3+2 vracia hodnotu 5, preto sa jedná o výraz. Všetky výrazy sú zároveň príkazy.

Úsek kódu, ktorý môžeme označiť ako výraz, nás možno prekvapí. Uvedieme tri príklady:

```
3.2             // vracia hodnotu 3.2
PI              // konštanta typu float, ktorá vracia hodnotu 3.14
SekundZaMinutu  // konštanta typu int, ktorá vracia hodnotu 60
```

V predchádzajúcich troch príkladoch predpokladáme, že PI je konštanta, ktorá bola už skôr vytvorená a inicializovaná na hodnotu 3.14, a *SekundZaMinutu* je konštanta, ktorej hodnota je rovná číslu 60. Za týchto okolností sú na všetkých troch riadkoch správne výrazy.
Trochu komplikovanejší je výraz:
x = a + b;
v ktorom sa nielen sčítajú hodnoty *a* a *b* a výsledok tohto súčtu sa priradí do premennej *x,* ale tiež je aj vrátená hodnota tohto priradenia (hodnota *x*). Pretože sa jedná o výraz, môže byť tým pádom sám na pravej strane operátora priradenia:
y = x = a + b;
Tento riadok sa vyhodnotí v nasledujúcom poradí:

1. sčíta sa hodnota premenných *a* a *b*,
2. výsledok výrazu *a + b* sa priradí do premennej *x*,
3. výsledok výrazu priradenia *x = a + b* sa priradí do premennej *y*.

Ak sú premenné *a, b, x* a *y* všetky typu *int* a ak *a* má hodnotu 2 a *b* má hodnotu 5, bude obom premenným *x* aj *y* priradená hodnota 7. Tento príklad je ilustrovaný v nasledujúcom programe:

```
0:  // Vyhodnocovanie zložitejších výrazov
1:
2:  #include <iostream>
3:  int main()
4:  {
5:      using std::cout;
6:      using std::cin;
7:      using std::endl;
8:
9:      int a=0, b=0, x=0, y=35;
10:     cout << "a: " << a << " b: " << b;
11:     cout << " x: " << x << " y: " << y << endl;
12:     a = 9;
13:     b = 7;
14:     y = x = a+b;
15:     cout << "a: " << a << " b: " << b;
16:     cout << " x: " << x << " y: " << y << endl;
17:     char reakcia;
18:     cin >> reakcia;
19:     return 0;
20: }
```

Analýza programu: Na riadku 9 sa inicializujú a zadeklarujú štyri premenné. Na riadkoch 10 a 11 sa na obrazovku vypíšu ich hodnoty. Na riadku 12 sa do premennej *a* priradí hodnota 9

a na riadku 13 sa do premennej *b* priradí hodnota 7. Na riadku 14 sa sčítajú hodnoty premenných *a* a *b* a výsledok tohto súčtu sa priradí do premennej *x*. Tento výraz (*x = a + b)* sa vyhodnotí na hodnotu (súčet *a + b*) a táto hodnota sa potom priradí do premennej *y*.

4.1 L-hodnoty a p-hodnoty

Operand, ktorý môže byť umiestnený na ľavej strane operátora priradenia sa nazýva l-hodnota. Operand, ktorý sa môže vyskytovať na pravej strane operátora priradenia sa nazýva p-hodnota (alebo aj r-hodnota).
Konštanty sú p-hodnoty, ktoré nemôžu byť l-hodnotami. Môžeme teda napísať:

```
x = 48;          // v poriadku
```

ale nemôžeme napísať

```
48 = x;          // chyba, nejedná sa o l-hodnotu!
```

L-hodnota je operand, ktorý môže byť na ľavej strane výrazu. P-hodnota je operand, ktorý môže byť na pravej strane výrazu. Môžeme si všimnúť, že všetky l-hodnoty môžu byť aj p-hodnotami, ale nie všetky p-hodnoty môžu byť aj l-hodnotami. Príklad p-hodnoty, ktorá nie je l-hodnotou: môžeme napísať *x = 5;*, ale nemôžeme napísať *5 = x;* (x môže byť l-hodnotou aj p-hodnotou, ale 5 môže byť len p-hodnotou).

4.2 Celočíselné delenie a delenie so zvyškom

S celočíselným delením sa stretneme, keď máme napr. číslo 21 vydeliť číslom 4 (21/4). Odpoveď znie, že výsledok je 5 (so zvyškom). Ak chceme spočítať, aký je zvyšok po celočíselnom delení čísla 21 číslom 4 (21%4), získame výsledok 1.

Dobrá rada: Pri delení 5/3 dostávam výsledok 1. Čo nie je v poriadku?
Odpoveď: Ak delíme jedno celé číslo iným celým číslom, dostaneme ako výsledok celé číslo. Preto 5/3 je 1. Skutočná odpoveď by znela, že výsledok je 1 so zvyškom 2. Ak chceme dostať výsledok v tvare desatinného čísla, musíme používať premenné typu *float*. 5.0/3.0 nám vráti hodnotu 1.66667. Ak je čitateľ alebo menovateľ číslom s pohyblivou rádovou čiarkou, vygeneruje kompilátor podiel, ktorý bude číslom s pohyblivou rádovou (desatinnou) čiarkou.

4.3 Kombinácia operátora priradenia s matematickými operátormi

V programovaní nie je neobvyklé, ak sa k hodnote premennej pridá iná hodnota a výsledok sa potom priradí späť do pôvodnej premennej. Ak máme napríklad premennú *mojVek* a chceme zvýšiť jej hodnotu o 2, môžeme napísať:

```
int mojVek = 5;
int zvys;
zvys = mojVek + 2;    // súčet 5 + 2 a uloženie do premennej zvys
mojVek = zvys;        // vloženie späť do mojVek
```

V predchádzajúcom prípade sa však jedná o značne komplikovanú a neúspornú metódu. V C++ je preto možné dať tú istú premennú na obe strany operátora priradenia a v našom prípade potom dôjde k nasledujúcej zmene:

```
mojVek = mojVek + 2;
```

To je už omnoho lepšie vyjadrenie. V algebre by bol tento výraz považovaný za nezmyselný, ale v C++ sa bude čítať ako „k hodnote v premennej *mojVek* pričítaj 2 a výsledok priraď do premennej *mojVek*“.

V C++ však existuje aj omnoho jednoduchšia forma zápisu, ktorá sa však už tak dobre nečíta:

mojVek += 2;
Operátor += sám pripočíta p-hodnotu k l-hodnote a potom výsledok znova priradí do l-hodnoty. Názov tohto operátora sa vyslovuje „plus-rovná sa“. Príkaz by sa potom čítal „mojVek plus-rovná sa 2“. Keby bola hodnota premennej *mojVek* rovná 4, potom by sme po vykonaní tejto operácie dostali výsledok 6.

Existujú takisto obdoby tejto operácie pre odčítanie (-=), delenie (/=), násobenie (*=) a delenie so zvyškom (% =).

4.4 Povaha pravdivosti

Štandard ANSI pri posudzovaní pravdivosti, resp. nepravdivosti výrazov zaviedol typ *bool.* Tento typ môže nadobúdať len dve hodnoty: *true* (pravda) a *false* (nepravda).

Každý výraz je možné vyhodnotiť z hľadiska jeho pravdivosti či nepravdivosti. Výrazy, ktoré sa matematicky vyhodnotia na nulu, budú vracať hodnotu false. Všetky ostatné výrazy budú vracať hodnotu *true.*

Tabuľka 2 Relačné operátory

Názov	Operátor
Rovná sa	==
Nerovná sa	!=
Väčší než	>
Väčší nanajvýš rovný	>=
Menší než	<
Menší nanajvýš rovný	<=

Prameň: Liberty Jesse, 2007, s. 89

4.5 Inkrementácia a dekrementácia

Najbežnejšou hodnotou, ktorá sa pričítava alebo odčítava a potom znova priradzuje premenným je číslo 1. V C++ sa zvýšenie hodnoty o 1 nazýva inkrementácia a zníženie hodnoty o 1 dekrementácia. Tieto operácie sa môžu vykonávať pomocou zvláštnych operátorov.

Operátor inkrementácie (++) zvýši hodnotu premennej o 1 a operátor dekrementácie (--) zníži hodnotu premennej o 1. Ak máme napríklad premennú *C* a chceme jej hodnotu zvýšiť o 1, môžeme to vykonať pomocou nasledujúceho príkazu:
C++; // vezme premennú C a zvýši jej hodnotu o 1
Ekvivalent tohto príkazu je
C = C + 1;
alebo tiež s obmenou tohto príkazu
C += 1;

Operátory inkrementácie a dekrementácie majú dve formy: prefixovú a postfixovú. Prefixový variant sa píše pred názvom premennej *(++mojVek*), zatiaľ čo postfixový variant sa píše za názvom premennej (*mojVek++*).

Pri jednoduchom príkaze nebude príliš záležať na tom, ktorý z týchto variantov použijeme. V prípade zložitého príkazu, keď budeme zvyšovať (alebo znižovať) hodnotu premennej o 1

a potom budeme chcieť výsledok priradiť do ďalšej premennej, na tom už záleží veľmi. Prefixový operátor sa vyhodnocuje pred priradením, zatiaľ čo postfixový operátor sa vyhodnocuje po priradení.

Prefixová sémantika je nasledovná: Zvýši sa hodnota o 1 a potom sa vráti. Postfixová sémantika je iná: Vráti sa najprv hodnota a až potom sa hodnota premennej zvýši o 1.
Napríklad ak je *x* celočíselná premenná a jej hodnota je 5, a napíšeme
int a = ++x;
tak kompilátoru oznámime, aby hodnotu premennej *x* zvýšil o 1 (na 6) a potom túto hodnotu vzal a priradil ju do premennej *a.* Preto *a* je teraz 6 a *x* je teraz tiež 6.
Potom ak napíšeme
int b = x++;
oznámime kompilátoru, aby vzal hodnotu v premennej *x* (6), priradil ju do premennej *b*, potom sa vrátil späť a zvýšil hodnotu *x* o jeden. Preto hodnota v premennej *b* je teraz 6, ale hodnota v premennej *x* je teraz 7.

4.6 Príkaz *if*

Normálne sa program vykonáva riadok po riadku, a to v poradí, v akom sa riadky vyskytujú v zdrojovom kóde. Príkaz *if* umožňuje testovanie podmienky (napríklad či sa dve premenné rovnajú) a rozvetvenie kódu na rôzne časti podľa jej výsledku. Jedna z najjednoduchších podôb príkazu *if* vyzerá takto:

```
if (výraz)
    príkaz;
```

Výraz v zátvorke môže byť ľubovoľný, ale obvykle sa jedná o výraz, ktorý obsahuje relačný operátor. Ak má výraz hodnotu nepravda, príkaz sa preskočí. Ak sa hodnota výrazu vyhodnotí ako pravda, príkaz sa vykoná.

Ak chceme do príkazu *if* zahrnúť viac príkazov, musíme ich zahrnúť do bloku, a teda dané príkazy ohraničiť zátvorkami *{* a *}*:

```
if (výraz)
{
    príkaz1;
    príkaz2;
    príkaz3;
}
```

Často je však potrebné program vetviť v dvoch smeroch. Jedným spôsobom, keď je podmienka pravdivá a druhým spôsobom, keď je podmienka nepravdivá. Na to použijeme vetvenie s použitím nepovinnej vetvy začínajúcej slovom *else*:

```
if (výraz)
    príkaz1;
else
    príkaz2;
```

Cvičenie 4.2: Napíšte program, ktorý vypočíta podiel dvoch čísel. Pri riešení zohľadnite aj možnosť, že užívateľ zadá do menovateľa nulu.

Riešenie:

```
#include <iostream>
using namespace std;

int main()
{
 float a,b;
 cout << "Zadaj citatel: ";
 cin >> a;
 cout << "Zadaj menovatel: ";
 cin >> b;
 cout << endl;

 if (b!=0)
       cout << "Podiel je: " << a/b << "\n";
else
       cout << "Nedovolene delenie nulou " << "\n";

   char reakcia;
   cin >> reakcia;
   return 0;
 }
```

Stojí za pozornosť, že v klauzule *if* alebo *else* je možné použiť akýkoľvek príkaz, dokonca aj ďalší príkaz *if* alebo *else.* Môžeme teda vytvoriť vnorené príkazy *if,* ako napríklad tento:

```
if (výraz1)
{
      if (výraz2)
         príkaz1;
      else
      {
         if (výraz3)
            príkaz2;
         else
            príkaz3;
      }
}
else
      príkaz4;
```

Tento ťažkopádny príkaz *if* si môžeme vysvetliť takto: Ak je *výraz1* a *výraz2* pravdivý, potom sa vykoná *príkaz1.* Ak je *výraz1* pravdivý, ale *výraz2* nepravdivý a súčasne je *výraz3* pravdivý, vykoná sa *príkaz2.* Ak je *výraz1* pravdivý, ale *výraz2* a *výraz3* sú nepravdivé, vykoná sa *príkaz3.* A nakoniec, ak je *výraz1* nepravdivý, vykoná sa *príkaz4.*

4.7 Príkaz *switch*

Na rozdiel od príkazu *if,* ktorý vyhodnocuje len jednu hodnotu, sú výrazy *switch* schopné vetvenia podľa niekoľkých hodnôt. Všeobecná podoba príkazu *switch* je nasledovná:

```
switch (výraz)
{
case prváHodnota:    príkaz;
                     break;
case druháHodnota:   príkaz;
                     break;

case ntáHodnota:     príkaz;
                     break;
default:             príkaz;
                     break;
}
```

Výraz je ľubovoľný platný výraz C++, ktorý sa vyhodnotí (resp. je ho možné jednoznačne previesť) na celočíselnú hodnotu, a príkazy sú ľubovoľné platné príkazy, resp. bloky príkazov v C++. Môžeme si všimnúť, že vyhodnocovanie sa týka len rovnosti. Nie je tu teda možné použiť ani relačné operátory a ani logické operácie.

Pokiaľ sa jedna z hodnôt za kľúčovým slovom *case* bude zhodovať s hodnotou výrazu, preskočí vykonávanie programu k jeho príkazom a bude pokračovať až na koniec bloku *switch*, až pokiaľ nenarazí na príkaz *break.* Ak ale nedôjde k žiadnej zhode, vykonávanie programu skočí k nepovinnému príkazu *default.*

V prípade, že neexistuje žiadna odpovedajúca hodnota a nie je definovaný ani *default,* vykonávanie programu zvyšok príkazu *switch* preskočí a pokračuje za ním.

Dobrá rada: Takmer vždy je vhodné, aby súčasťou príkazu *switch* bol aj príkaz *default.* Aj keď nie je potrebný, môže často slúžiť ako test pre hypoteticky nemožný prípad a výstup pre chybové hlásenie, čo sa môže ukázať ako zásadná pomoc aj pri ladení programu.

Je dôležité poznamenať, že ak na konci príkazu *case* nebude žiadny príkaz *break,* prejde vykonávanie programu k príkazom za ďalším *case.* To môže byť niekedy nevyhnutné, ale obvykle sa jedná o chybu. Ak sa ponechá vykonávanie programu prejsť k ďalšiemu príkazu *case,* je nutné, aby bolo zrejmé, že príkaz *break* bol vynechaný úmyselne a že nejde o opomenutie. K tomu stačí pripojiť krátky komentár. Ako funguje príkaz *switch* si ukážeme na nasledujúcom príklade:

```
#include <iostream>
int main()
{
	using namespace std;
	signed short int cislo;
	cout << "Vlozte cislo z intervalu 0 az 5: ";
	cin >> cislo;
	switch (cislo)
	{
	case 0:		cout << "Prilis male cislo!";
				break;
	case 1:		cout << "Dobra praca!\n";
				break;
	case 2:		cout << "Pekna volba!\n";
				break;
	case 3:		cout << "Vynikajuco!\n";
				break;
	case 4:		cout << "Skvele!\n";
				break;
	case 5:		cout << "Neuveritelne!\n";
				break;
	default:	cout << "Cislo nespada do rozmedzia!\n";
				break;
	}
	cout << "\n\n";
	char reakcia;
	cin >> reakcia;
	return 0;
}
```

Úloha 4.1: Napíšte program, ktorý nájde väčšie číslo z dvoch zadaných čísel. Uvažujte aj s alternatívou, že užívateľ zadá obe čísla rovnaké.
Úloha 4.2: Napíšte program, ktorý zistí, či sa dá alebo nedá zostrojiť trojuholník. Ako vstupné hodnoty budú zadané strany trojuholníka.
Úloha 4.3: Vytvorte program, ktorý vypočíta korene kvadratickej rovnice $ax^2+bx+c=0$.
Úloha 4.4: Pomocou príkazu *switch* zostavte kalkulačku. Použite operácie sčítania, odčítania, násobenia a delenia.

5 KONŠTANTY

Podobne ako je to u premenných, tak aj konštanty slúžia ako miesta, kde sa uchovávajú dáta. Ako už samotný názov napovedá, konštanty, na rozdiel od premenných, nemôžu zmeniť svoju hodnotu. Keď definujeme konštantu, musíme ju inicializovať a neskôr jej už nemôžeme priradiť inú hodnotu. V jazyku C++ poznáme dva typy konštánt: literálne a symbolické.

Pod literálnou konštantou rozumieme hodnotu, ktorá sa zadá priamo do programu. Napríklad:
int mojVek = 20;
Premenná *mojVek* je typu *int* a 20 je literálna (doslovná) konštanta.

Pod symbolickou konštantou rozumieme konštantu reprezentovanú názvom, podobne ako je názvom reprezentovaná premenná. Na rozdiel od premennej však nemôžeme hodnotu konštanty po jej inicializácii zmeniť. Ak má napríklad program jednu celočíselnú premennú s názvom *studenti* a ďalšiu premennú s názvom *triedy,* môžeme vypočítať, koľko máme pri danom počte tried študentov. Tu predpokladajme, že každá trieda má 20 študentov.
*studenti = triedy * 20;*
V tomto prípade je 20 literálnou konštantou. Náš kód by sa však lepšie čítal a udržiaval, keby sme túto hodnotu nahradili symbolickou konštantou:
*studenti = triedy * studentiNaTriedu;*
Takýto spôsob definovania konštanty je výhodný najmä preto, lebo keby sme sa neskôr rozhodli počet študentov v každej triede zmeniť, stačilo by to urobiť len raz v mieste definovania konštanty *studentiNaTriedu.* Nie je teda nutné vykonávať akékoľvek zmeny pri ďalšom možnom výskyte tejto konštanty v programe.

Symbolickú konštantu môžeme v C++ definovať dvoma spôsobmi:

1. Definícia konštanty pomocou *#define*
   Je to tradičný spôsob, ktorý je dnes už trochu zastaraný. Používa direktívu preprocesora *#define.* Pretože túto direktívu používa mnoho už existujúcich programov, musíme dobre pochopiť jej skutočný spôsob použitia. Ak chceme konštantu definovať tradičným spôsobom, musíme zadať:
   #define studentiNaTriedu 20
   Všimnime si, že pri konštante *studentiNaTriedu* nie je určený žiadny typ (*int, char* a podobne). Príkaz *#define* znamená len obyčajnú substitúciu textu. Vždy, keď preprocesor narazí na slovo *studentiNaTriedu,* vloží do textu namiesto neho hodnotu 20. Pretože preprocesor beží pred kompilátorom, kompilátor nikdy žiadnu konštantu neuvidí, narazí len na číslo 20.
2. Definícia konštanty pomocou *const*
   Aj keď konštanty definované pomocou *#define* fungujú dobre, existuje v C++ aj lepší spôsob:
   const unsigned short int studentiNaTriedu = 20;
   V tomto prípade sa opäť deklaruje symbolická konštanta s názvom *studentiNaTriedu,* tentoraz sa však jedná o konštantu typu *unsigned short int.* Výhodou tejto metódy je jednoduchšia údržba kódu a zlepšená prevencia chýb. Najväčším rozdielom je však to, že konštanta má pevný typ a kompilátor si môže vynútiť, aby sa používala v súlade so svojim typom.

5.1 Vymenované konštanty

Vymenované konštanty dovoľujú vytvoriť nové typy a premenné, ktorých hodnoty sa obmedzujú len na množinu uvedených hodnôt. Môžeme napríklad vymenovaním deklarovať typ *FARBA,* ktorý bude nadobúdať nasledujúce hodnoty: *CERVENA, MODRA, ZELENA, BIELA a CIERNA.*

Pre syntax vymenovaných konštánt platí, že začínajú kľúčovým slovom *enum,* za ktorým nasleduje názov typu, ľavá zložená zátvorka, povolené hodnoty oddelené čiarkou a nakoniec pravá zložená zátvorka za ktorou je bodkočiarka. Napríklad:
enum FARBA { CERVENA, MODRA, ZELENA, BIELA, CIERNA };
Tento príkaz spôsobí:

1. vytvorenie nového vymenovania *FARBA,* čím dôjde k vytvoreniu nového typu,
2. vytvorenie symbolickej konštanty *CERVENA* s hodnotou 0, symbolickej konštanty *MODRA* s hodnotou 1, symbolickej konštanty *ZELENA* s hodnotou 2, atď.

Každá vymenovaná konštanta má celočíselnú hodnotu. Ak neurčíme inak, bude mať prvá konštanta hodnotu 0 a každá ďalšia bude vždy o jednotku väčšia. Každú z týchto konštánt je však možné inicializovať na konkrétnu hodnotu a tým, ktoré inicializované nie sú, sa priradí hodnota podľa predchádzajúcich. Napr.:
enum FARBA {CERVENA=100, MODRA, ZELENA=500, BIELA, CIERNA=700};
Konštanta *CERVENA* bude mať hodnotu 100, *MODRA* bude mať hodnotu 101, *ZELENA* bude mať hodnotu 500, *BIELA* hodnotu 501 a nakoniec *CIERNA* hodnotu 700.
Je teda dôležité si uvedomiť, že vymenované premenné sú v skutočnosti typu *unsigned int* a že vymenované konštanty sú vlastne celé čísla. Využitie vymenovaných konštánt si ukážeme na nasledujúcom príklade:

```
#include <iostream>
int main()
{
        enum Dni { Pondelok, Utorok, Streda,
                Stvrtok, Piatok, Sobota, Nedela };

        Dni dnes;
        dnes = Pondelok;

        if ( dnes == Sobota || dnes == Nedela )
                std::cout << "\nMam rad vikend!\n";
        else
                std::cout << "\nHura do skoly!\n";
        char reakcia;
        std::cin >> reakcia;
        return 0;
}
```

Analýza programu: V programe sme si vytvorili typ *Dni,* ktorý nadobúda hodnoty *Pondelok, Utorok, Streda, Stvrtok, Piatok, Sobota, Nedela.* Potom deklarujeme premennú *dnes,* ktorá je typu *Dni.* Ak do premennej *dnes* vložíme hodnotu *Sobota* alebo *Nedela,* potom program

vypíše správu „Mam rad vikend!“. Ak tam vložíme jednu z hodnôt *Pondelok, Utorok, Streda, Stvrtok* alebo *Piatok,* program vypíše správu „Hura do skoly“. Keďže v našom programe je v premennej *dnes* hodnota *Pondelok,* program nám vypíše „Hura do skoly“.

Cvičenie 5.1: Prerobte predchádzajúci kód tak, že vymenovaný typ nahradíte radom celočíselných konštánt.
Riešenie:

```
#include <iostream>
int main()
{
	const int Pondelok = 0;
	const int Utorok = 1;
	const int Streda = 2;
	const int Stvrtok = 3;
	const int Piatok = 4;
	const int Sobota = 5;
	const int Nedela = 6;

	int dnes;
	dnes = Pondelok;

	if ( dnes == Sobota || dnes == Nedela )
		std::cout << "\nMam rad vikend!\n";
	else
		std::cout << "\nHura do skoly!\n";
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

Analýza programu: Výsledok bude rovnaký, ako v predchádzajúcom programe. Tentoraz sa ale každá z konštánt explicitne definuje (*Pondelok, Utorok*, atď.) a žiadny vymenovaný typ *Dni* tu nenájdeme.

Úloha 5.2: Napíšte program, kde nadefinujete vymenovanú konštantu s názvom *Mesiace.* Potom nech si program vypýta číselnú hodnotu z intervalu 1 až 12. Program nech následne vypíše mesiac, ktorý prislúcha danej hodnote. Napr. 1 – január, 2 – február, atď.
Úloha 5.1: Napíšte program, ktorý vypočíta obvod a obsah kruhu, ak užívateľ na vstupe zadá veľkosť polomeru kruhu. Ludolfovo číslo (pí), ktoré je pri výpočte potrebné, deklarujte ako konštantu a priraďte jej hodnotu 3,1416.

6 PROGRAMOVÉ CYKLY

6.1 Cyklus *while*

Prostredníctvom cyklu *while* sa opakovane vykonáva sekvencia príkazov, pokiaľ je počiatočná podmienka pravdivá. Syntax cyklu je nasledovná:
while (podmienka)
príkaz;
Podmienka je ľubovoľný výraz C++ a *príkaz* je ľubovoľný platný príkaz alebo blok príkazov C++. Ak sa podmienka vyhodnotí ako pravdivá, vykoná sa príkaz a potom sa znovu otestuje podmienka. To sa opakuje tak dlho, až prestane podmienka platiť. V tom okamihu sa cyklus *while* ukončí a vykonávanie programu bude pokračovať prvým riadkom nasledujúcim za príkazom *príkaz.*

Cyklus *while* si ilustrujeme na nasledujúcom vzorovom príklade:

```
#include <iostream>
int main()
{
	int citac = 0;		// inicializácia podmienky
	while (citac < 5)	// test, či je podmienka stále pravdivá
	{
		citac++;		// telo cyklu
		std::cout << "citac: " << citac << "\n";
	}

	std::cout << "Hotovo, citac: " << citac << ".\n";
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

6.2 Cyklus *do-while*

V cykle *do-while* sa telo cyklu vykoná pred tým, než sa otestuje jeho podmienka. Tým zaistíme, že sa telo cyklu vykoná stále aspoň raz. Syntax príkazu vyzerá takto:
do
príkaz;
while (podmienka);
Najprv sa vykoná príkaz a až potom sa vyhodnotí podmienka. Ak je podmienka pravdivá, cyklus sa zopakuje. V opačnom prípade sa cyklus ukončí. Príkazy a podmienky sú inak identické s príkazmi a podmienkami cyklu *while.* Cyklus *do-while* si ilustrujeme na nasledujúcom príklade:

```
#include <iostream>
int main()
{
	using namespace std;
	int pocet;
	cout << "Kolko pozdravov mam vypisat? ";
	cin >> pocet;
	do
	{
		cout << "Ahoj\n";
		pocet--;
	}  while (pocet > 0);
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

6.3 Cyklus *for*

V cykle *for* sa v jednom príkaze kombinujú tri kroky: inicializácia, testovanie a zvýšenie hodnoty.

Prvý príkaz je inicializácia. Je možné sem umiestniť ľubovoľný platný príkaz C++, ale obvykle tu dochádza k vytvoreniu a inicializácii premennej – čítača. Druhý príkaz slúži na testovanie a je tu možné použiť ľubovoľný platný výraz C++. Úloha druhého príkazu zodpovedá podmienke v cykle *while*. Tretí príkaz predstavuje akciu. Pre tento príkaz je typické, že tu dochádza ku zvýšeniu alebo zníženiu hodnoty, aj keď aj sem je možné dať ľubovoľný platný príkaz C++.

Stojí za povšimnutie, že prvý a tretí príkaz môžu byť ľubovoľné platné príkazy C++, ale druhý príkaz musí byť výrazom C++, ktorý vracia hodnotu. Syntax príkazu je:
for (inicializácia; test; akcia)
príkaz;

Príkaz *inicializácia* sa používa na inicializáciu stavu čítača alebo k podobnému úvodnému nastaveniu cyklu. *Test* je ľubovoľný výraz C++, ktorý sa pri každom prechode cyklom vyhodnocuje. Ak je test pravdivý, vykoná sa telo cyklu *for* a potom sa vykoná *akcia* v hlavičke (obvykle tu dôjde ku zvýšeniu hodnoty čítača).
Cyklus *for* si ukážeme na nasledujúcom príklade:

```
#include <iostream>
int main()
{
	int citac;
	for (citac = 0; citac < 5; citac++)
		std::cout << "Cyklus! ";
	std::cout << "\nCitac: " << citac << ".\n";
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

Úloha 6.1: Napíšte program, ktorý vypočíta najväčší spoločný deliteľ (NSD) dvoch prirodzených čísel.
Úloha 6.2: Napíšte program, ktorý vypočíta najmenší spoločný násobok (nsn) dvoch prirodzených čísel.
Úloha 6.3: Vytvorte program pre hru „Hádaj číslo“. Úlohou užívateľa je uhádnuť číslo z intervalu 0 až 100. Hra končí uhádnutím čísla. Program nech na konci vypíše aj počet pokusov, ktoré hráč potreboval na uhádnutie správneho čísla.
Úloha 6.4: Napíšte program, ktorý nám vypočíta n – tú mocninu zadaného čísla.
Úloha 6.5: Zostavte program na výpočet váženého aritmetického priemeru. Užívateľ na vstupe najprv zadá počet známok a následne aj jednotlivé známky a aj ich váhy.
Úloha č. 6.6: Napíšte program, ktorý vypočíta hodnotu faktoriálu zadaného prirodzeného čísla. Počítajte aj s alternatívou, že užívateľ na vstupe zadá zápornú hodnotu čísla.

7 FUNKCIE

Aj keď je *main()* funkciou, jedná sa o nezvyčajnú funkciu. Ak má byť nejaká funkcia užitočná, musí byť možné ju v priebehu programu zavolať. Funkciu *main()* však volá samotný operačný systém. Riadky programu sú postupne vykonávané v poradí, v akom sa objavujú v zdrojovom kóde, pokiaľ nie je zavolaná funkcia. V tej chvíli sa program rozvetví, aby funkciu vykonal. Keď je vykonávanie funkcie dokončené, vráti sa riadenie na riadok kódu, ktorý nasleduje bezprostredne za volaním tejto funkcie. Toto ilustruje nasledujúci príklad:

```
#include <iostream>

// funkcia IlustracnaFunkcia
// vytlačí veľmi užitočnú správu
void IlustracnaFunkcia()
{
        std::cout << "Vo vnutri Ilustracnej funkcie\n";
}

// funkcia main - vytlačí správu, potom zavolá
// ilustračnú funkciu a nakoniec vytlačí
// druhú správu
int main()
{
        std::cout << "Vo funkcii main\n";
        IlustracnaFunkcia();
        std::cout << "Spat vo funkcii main\n";
        char reakcia;
        std::cin >> reakcia;
        return 0;
}
```

Výstup:
Vo funkcii main
Vo vnutri Ilustracnej funkcie
Spat vo funkcii main

Funkcia vracia buď skutočnú hodnotu, alebo prázdnu hodnotu typu *void,* teda nič. Funkcia, ktorá má napríklad sčítať dve celočíselné hodnoty, vracia hodnotu súčtu zadaných čísel a mala by byť deklarovaná s celočíselnou návratovou hodnotou. Funkcia, ktorá má len vytlačiť správu na výstup, nemá čo vrátiť a mala by byť deklarovaná tak, že vráti hodnotu typu *void.*

Funkcia sa skladá z hlavičky a tela. Hlavička obsahuje návratový typ, názov funkcie a parametre, ktoré tejto funkcii prislúchajú. Parametre umožňujú predanie hodnôt funkcii. Ak by teda mala funkcia sčítať dve čísla, boli by sčítance parametrami funkcie. Tu je príklad typickej hlavičky funkcie: *int Sucet(int a, int b).*

Pod parametrom rozumieme deklaráciu typu hodnoty, ktorá sa bude funkcii predávať. Skutočná hodnota predávaná funkcii pri zavolaní sa nazýva argument. Mnoho programátorov termíny argument a parameter používa ako synonymá. Iní sú si naopak odlišnosťou medzi nimi vedomí.

Telo funkcie sa skladá zo začiatočnej zloženej zátvorky, žiadneho alebo viac príkazov a koncovej zloženej zátvorky. Príkazy tvoria samotnú činnosť funkcie. Funkcia môže vracať špecifickú hodnotu, k čomu slúži príkaz *return.* Tento príkaz zároveň funkciu ukončuje. Ak vo funkcii príkaz return vynecháme, bude pri ukončení automaticky vrátená hodnota *void.* Vrátená hodnota musí byť rovnakého typu, aký je deklarovaný v hlavičke funkcie.

Rôzne programovacie jazyky a rôzne programovacie metodiky môžu označovať funkcie odlišným termínom. Jedným z často používaných označení je metóda. Metóda je len iným označením pre funkciu, ktorá je súčasťou nejakej triedy.

Cvičenie 7.1: Napíšte program, ktorý ilustruje funkciu, ktorá prijíma dve celočíselné parametre a vracia celočíselnú hodnotu – súčet zadaných parametrov.
Riešenie:

```
#include <iostream>
int Sucet(int x, int y)
{
	std::cout << "Vo funkcii Sucet(), prijate " << x << " a " << y << "\n";
	return (x+y);
}

int main()
{
	using std::cout;
	using std::cin;

	cout << "Vo funkcii main()!\n";
	int a, b, c;
	cout << "Vlozte dve cisla: ";
	cin >> a;
	cin >> b;
	cout << "\nVolanie funkcie Sucet()\n";
	c=Sucet(a,b);
	cout << "\nSpat vo funkcii main().\n";
	cout << "c bolo nastavene na " << c;
	cout << "\nKoniec...\n";
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

Úloha 7.1: Doplňte Cvičenie 7.1 tak, aby zahŕňalo aj funkciu odčítania. Nazvite ju *Rozdiel()* a použite ju podobným spôsobom ako funkciu *Sucet().*
Úloha 7.2: Vytvorte funkciu, ktorá vypočíta ciferný súčet ľubovoľného prirodzeného čísla.

8 POLIA

Pole je pomenovaná skupina premenných rovnakého typu, uložených v pamäti bezprostredne za sebou. Tieto premenné označujeme ako prvky poľa. Polia delíme na jednorozmerné a viacrozmerné. Pole deklarujeme zadaním typu, za ktorým nasleduje názov poľa a subskript (alebo aj index). Subskript je počet prvkov poľa uzatvorený v hranatých zátvorkách. Napríklad:
long PoleCisel[25];
deklaruje pole 25 celých čísel nazvané *PoleCisel.* Keď kompilátor vidí túto deklaráciu, vyhradí si dostatok pamäte pre uchovanie všetkých 25 prvkov. Pretože každý prvok typu *long* vyžaduje 4 bajty, táto deklarácia vyhradzuje 100 za sebou nasledujúcich bajtov pamäte.

8.1 Prvky poľa

K jednotlivým prvkom poľa pristupujeme prostredníctvom zadania odsadenia od názvu poľa. Prvky poľa sa počítajú od nuly. Preto je prvým prvkom poľa *nazovPola[0]*. V príklade *PoleCisel[25]* je prvým prvkom poľa *PoleCisel[0]*, druhým prvkom *PoleCisel[1]*, atď. To môže byť trochu mätúce. Pole *NejakePole[3]* má tri prvky. Sú to *NejakePolel[0]*, *NejakePole[1]* a *NejakePolel[2]*. Všeobecne teda platí, že *NejakePole[n]* má *n* prvkov, ktoré sú očíslované od *NejakePole[0]* až po *NejakePole[n-1]*. Opäť však musíme myslieť na to, že index je odsadením, takže prvý prvok poľa je nula úložných miest od začiatku poľa, druhý je vzdialený jedno úložné miesto, atď. Preto je *PoleCisel[25]* číslované od *PoleCisel[0]* po *PoleCisel[24]*. Nasledujúci výpis ukazuje, ako deklarovať pole piatich celých čísel a každú pozíciu zaplniť nejakou hodnotou:

Cvičenie 8.1: Napíšte program, ktorý si od užívateľa do jednorozmerného poľa načíta 5 prirodzených čísel a následne ich vypíše na obrazovku.
Riešenie:

```
#include <iostream>
int main()
{
	int mojePole[5];
	int i;
	for ( i=0; i<5; i++) // premenná i bude nadobúdať hodnoty 0 – 4 (je to 5 hodnôt)
	{
		std::cout << "Hodnota mojePole[" << i << "]: ";
		std::cin >> mojePole[i];
	}
	for (i = 0; i<5; i++)
		std::cout << i << ": " << mojePole[i] << "\n";
	char reakcia;
	std::cin >> reakcia;
	return 0;
}
```

Dobrá rada: Niektorí programátori označujú *NazovPola[0]* za nultý prvok. Na to si ale radšej nezvykajme. Ak je *NazovPola[0]* nultým prvkom poľa, akým je potom *NazovPola[1]*? Prvým? Ak áno, potom keď uvidíme *NazovPola[24]*, uvedomíme si, že sa nejedná o prvok

24, ale až o prvok 25? Omnoho lepšie je hovoriť, že *NazovPola[0]* je na nultom odsadení a že sa jedná o prvý prvok poľa.

8.2 Inicializovanie polí

Jednoduché pole zabudovaných typov, ako sú celé čísla a znaky, môžeme inicializovať už pri prvej deklarácii poľa. Za názov poľa umiestníme znamienko „rovná sa“ (=) a v zložených zátvorkách uvedieme zoznam hodnôt oddelených čiarkami. Napríklad zápis:
int PoleCelychCisel[5] = { 10, 20, 30, 40, 50 };
deklaruje *PoleCelychCisel[5]* ako pole piatich čísel. Priradzuje *PoleCelychCisel[0]* hodnotu 10, *PoleCelychCisel[1]* hodnotu 20, atď.

Ak vynecháme veľkosť poľa, potom bude mať pole takú veľkosť, aby presne obsiahlo svoju inicializáciu. Ak teda zadáme:
int PoleCelychCisel[] = { 10, 20, 30, 40, 50 };
vytvoríme vlastne to isté pole ako v predchádzajúcom príklade.

Nemôžeme inicializovať viac prvkov, ako sme pre pole deklarovali. A preto:
int PoleCelychCisel[5] = { 10, 20, 30, 40, 50, 60 };
spôsobí chybu kompilácie, pretože sme deklarovali pole s piatimi členmi a inicializovali šesť hodnôt. Je však možné zapísať:
int PoleCelychCisel[5] = { 10, 20 };
V tomto prípade sme zadeklarovali pole s piatimi členmi, ale inicializovali sme len prvé dva, konkrétne *PoleCelychCisel[0]* a *PoleCelychCisel[1]*.

8.3 Deklarovanie polí

Pri deklarovaní počtu elementov nemusíme používať len literály, ale môžeme využiť tiež konštanty alebo vymenovania. V skutočnosti je použitie takýchto hodnôt výhodnejšie, pretože tým potom získame jediné miesto, odkiaľ môžeme počet elementov poľa ovládať. Tvorbu nejakého počtu elementov alebo aj rozmeru poľa pomocou vymenovania ukazuje nasledujúci výpis:

```
0: #include <iostream>
1: int main()
2: {
3:       enum DniTyzdna { Ned, Pon, Uto, Str, Stv, Pia, Sob, DniVTyzdni };
4:       int PoleTyzdna[DniVTyzdni] = { 10, 20, 30, 40, 50, 60, 70 };
5:       std::cout << "Hodnota v utorok je: " << PoleTyzdna[Uto];
6:       char reakcia;
7:       std::cin >> reakcia;
8:       return 0;
9: }
```

Výstup: Hodnota v utorok je: 30
Analýza programu: Riadok 3 vytvára vymenovanie nazvané *DniTyzdna.* To má osem členov. Nedeľa je rovná hodnote 0 a *DniVTyzdni* sa rovná hodnote 7. Na riadku 4 sa deklaruje *PoleTyzdna,* ktoré má *DniVTyzdni,* čiže sedem členov. Riadok 5 aplikuje konštantu vymenovania *Uto* ako odsadenie v poli. Pretože *Uto* sa vyhodnotí ako 2, je vrátený tretí prvok poľa, *PoleTyzdna[2],* a vytlačí sa na riadku 5.

8.4 Viacrozmerné pole

Je možné mať aj pole, ktoré má viac, ako len jeden rozmer. Každý rozmer je predstavovaný ako určitý subskript (index) v poli. Preto má dvojrozmerné pole dva indexy, trojrozmerné pole má tri indexy, atď. Polia môžu mať ľubovoľný počet rozmerov, ale napriek tomu bude mať väčšina našich polí jeden alebo maximálne dva rozmery.

Dobrým príkladom dvojrozmerného poľa je šachovnica. Jeden rozmer predstavuje osem riadkov a druhý rozmer predstavuje osem stĺpcov.

Predpokladajme, že máme triedu nazvanú *STVOREC*. Deklarovanie poľa nazvaného *Sachovnica,* ktorá ju predstavuje, bude vyzerať takto:
STVOREC Sachovnica[8][8];

Tie isté dáta by sme mohli reprezentovať tiež aj jednorozmerným poľom so 64 štvorčekmi. Napríklad:
STVOREC Sachovnica[64];

8.5 Inicializácia viacrozmerných polí

Viacrozmerné polia je možné aj inicializovať. Postupne priradzujeme zoznam hodnôt prvkom poľa, pričom posledný index poľa sa mení, zatiaľ čo predchádzajúci zostáva nezmenený. Ak máme teda pole:
int Pole[5][3];
potom sa prvé tri prvky uložia do *Pole[0]*, druhé tri do *Pole[1]*, atď.
Uvedené pole inicializujeme zadaním:
int Pole[5][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
Kvôli zreteľnosti môžeme zoskupiť inicializácie zloženými zátvorkami. Napríklad:

```
int Pole[5][3] = {    {1, 2, 3},
                      {4, 5, 6},
                      {7, 8, 9},
                      {10, 11, 12},
                      {13, 14, 15}   };
```

Kompilátor síce vnútorné zložené zátvorky ignoruje, ale takýto zápis nám pomáha v pochopení rozdelenia čísel vo viacrozmernom poli. Jednotlivé hodnoty musia byť oddelené čiarkami bez ohľadu na zložené zátvorky. Celá inicializácia musí byť v zložených zátvorkách a musí sa končiť bodkočiarkou.

Nasledujúci výpis vytvára dvojrozmerné pole. Prvým rozmerom je množina čísel od 0 po 4. Druhý rozmer je tvorený dvojnásobkom jednotlivých hodnôt v prvom rozmere:

```
0: // Vytvorenie viacrozmerného poľa
1:
2: #include <iostream>
3: using namespace std;
4:
5: int main()
6: {
7:       int nejakePole[5][2] = { {0,0}, {1,2}, {2,4}, {3,6}, {4,8}};
8:       for (int i = 0; i<5; i++)
9:                for (int j=0; j<2; j++)
10:               {
11:                       cout << "nejakePole[" << i << "][" << j << "]: ";
12:                       cout << nejakePole[i][j]<< endl;
13:               }
14:   char reakcia;
15:   cin >> reakcia;
16:   return 0;
17: }
```

Výstup:

Obrázok 4 Výstupná obrazovka – viacrozmerné pole

Zdroj: súkromný archív

Analýza programu: Riadok 7 deklaruje *nejakePole* ako dvojrozmerné pole. Prvý rozmer sa skladá z piatich celých čísel a druhý rozmer je tvorený dvoma celými číslami.

Úloha 8.1: Prerobte Cvičenie 8.1 tak, aby hodnoty, ktoré do poľa zadá užívateľ, program následne vypísal v opačnom poradí.
Úloha 8.2: S využitím poľa napíšte program, ktorý prevedie zadané číslo z dekadického tvaru do binárneho tvaru.
Úloha 8.3: Napíšte program, ktorý načíta maticu s rozmermi $n \times n$ a vypíše prvky hlavnej a aj vedľajšej diagonály.

9 ZNAKY A POLE ZNAKOV

Premenná typu znak (typ *char*) má zvyčajne veľkosť 1 bajt, čo je dosť na to, aby dokázala prijať 1 hodnotu. Premennú typu *char* môžeme interpretovať ako malé číslo (0 až 255) alebo ako prvok súboru ASCII (American Standard Code for Information Interchange). Súbor znakov ASCII a ich obdoba ISO (International Standards Organization) ponúka spôsob kódovania všetkých písmen, číslic a interpunkčných znamienok.

Ak by sme však chceli do ľubovoľnej premennej vložiť viac, ako jeden znak, môžeme vytvoriť aj takzvané „pole znakov“, ktoré vytvoríme nasledovne:
char Slovo[20];
V tomto prípade sme vytvorili premennú s názvom *slovo,* do ktorej môžeme vložiť maximálne 20 znakov.

Počítače však nepoznajú písmená, vety alebo interpunkčné znamienka. Jediné, čomu rozumejú sú čísla. V skutočnosti to, čomu naozaj rozumejú je, či v konkrétnom mieste kríženia spojov je dostatočné množstvo elektrickej energie. Ak je tomu tak, bude tento stav symbolizovať hodnota 1, opačný stav symbolizuje hodnota 0. Podľa zoskupenia jednotiek a núl je počítač schopný generovať vzory, ktoré je možné interpretovať ako čísla a tým sa potom môžu priradiť písmená a interpunkčné znamienka.

Podľa kódu ASCII sa napr. hodnote 97 priradzuje písmeno *a.* Podobne sú všetky malé a veľké písmená, všetky čísla a interpunkčné znamienka priradené hodnotám medzi 1 až 128. Zostávajúcich 128 značiek a symbolov je vyhradených pre výrobcu počítača, aj keď dnes sa stal už takmer štandardom rozšírený súbor znakov od IBM.

Keď do premennej typu *char* vložíme napr. hodnotu *a,* bude tam v skutočnosti číslo medzi 0 až 255. Kompilátor však vie medzi sebou znaky (reprezentované apostrofom, potom písmenom, číslicou alebo interpunkčným znamienkom, nasledovaným opäť apostrofom) a hodnoty súboru ASCII tam aj späť prekladať.

Funkcia prevádzajúca hodnoty na písmená a späť môže byť ľubovoľná. Neexistuje žiadny zvláštny dôvod, prečo sa malému písmenu *a* priradzuje práve hodnota 97. Ak sa na tom všetci (klávesnica, kompilátor aj obrazovka) zhodnú, nedôjde k žiadnym problémom. Je však nutné si uvedomiť, že medzi hodnotou 5 a znakom “5“ je veľký rozdiel. Skutočná hodnota znaku “5“ je 53, podobne ako písmeno *a* má hodnotu 97. Ilustruje to nasledujúci program:

```
#include <iostream>
int main()
{
        for (int i = 32; i<128; i++)
                std::cout << (char) i;
        char reakcia;
        std::cin >> reakcia;
        return 0;
}
```

Analýza programu: Tento jednoduchý program vytlačí hodnoty znakov pre celé čísla od 32 do 127. Výpis využíva na dosiahnutie tohto efektu celočíselnú premennú *i.* Číslo, uložené v premennej *i* sa vytlačí na obrazovku vo forme znaku (*char*).

Cvičenie 9.1: Napíšte program, ktorý sa užívateľa opýta na jeho meno, ktoré si uloží do premennej. Potom program užívateľa pozdraví a popraje mu pekný deň.
Riešenie:

```
#include <iostream>
int main()
{
   char meno[20];
   std::cout << "Ako sa volas?";
   std::cout << std::endl;
   std::cin >> meno;
   std::cout << "Ahoj, " << meno << ". Prajem Ti pekny den." << std::endl;
         char reakcia;
         std::cin >> reakcia;
         return 0;
}
```

Kompilátor C++ rozoznáva aj niekoľko špeciálnych znakov určených na formátovanie kódu. Najznámejšie z nich sú uvedené v Prílohe 4.

Úloha 9.1: Napíšte program, ktorý si od užívateľa najprv vypýta jedno prirodzené číslo. Následne nech užívateľ dostane na výber: ak stlačí klávesu „m“ – program vypočíta druhú mocninu daného čísla, ak stlačí „o“ – program vypočíta druhú odmocninu daného čísla a napokon, ak stlačí „f“ – program vypočíta faktoriál daného prirodzeného čísla (Pomôcka: pri riešení využite príkaz *switch*).

10 VSTUP Z TEXTOVÉHO SÚBORU A VÝSTUP DO TEXTOVÉHO SÚBORU

Aby bolo možné začať zapisovať do súboru a čítať zo súboru, musíme najprv vytvoriť objekt *ofstream,* resp. *ifstream* a potom priradiť tento objekt určitému súboru na disku. Aby sme mohli využívať dané objekty, musíme do svojho programu zahrnúť *fstream.h.*

Ak chceme otvoriť súbor *mojsubor* objektom *ofstream,* musíme deklarovať inštanciu objektu *ofstream* a ako parameter predáme názov súboru:
ofstream fout(mojsubor);

Otvorenie toho istého súboru pre vstup funguje podobným spôsobom, len namiesto *ofstream* použijeme *ifstream*:
ifstream fin(mojsubor);

Ďalšou dôležitou funkciou pri práci so súbormi je funkcia *close().* Používa sa na zatvorenie súboru, len čo skončíme s čítaním zo súboru, resp. zápisom do súboru.

Cvičenie 10.1: Zostavte program, ktorý si od užívateľa vypýta názov textového súboru a následne doň vloží súčet dvoch zadaných celých čísel.
Riešenie:

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
 char nazovSuboru[15];
 cout << "Zadajte nazov suboru: ";
 cin >> nazovSuboru;

 ofstream fout(nazovSuboru);  // otvoriť pre zápis
 int a,b,c;
 cout << "Zadaj dve cisla: ";
 cin >> a >> b;
 c=a+b;

 fout << "Sucet cisel " << a << " a " << b << " je " << c << "." << "\n";
 fout.close();                          // zatvorenie súboru

   char reakcia;
   cin >> reakcia;
   return 0;
 }
```

Úloha 10.1: Vytvorte program, ktorý zašifruje text v textovom súbore “sprava.txt” pomocou Cézarovej šifry (posun znakov o 3 doprava). Súbor “sprava.txt” si treba predtým vytvoriť a napísať doň nejaký text. Zašifrovanú správu uložte do súboru “sprava2.txt”.

ZÁVER

Táto práca má slúžiť hlavne ako pomôcka pre učiteľa. Učiteľ ju môže použiť ako celok, alebo si z nej môže vybrať len nejaké kapitoly. Podobne si môže zmeniť aj časovú dotáciu jednotlivých kapitol v závislosti na šikovnosti žiakov.

Cieľom tejto práce je podeliť sa so skúsenosťami autora pri vyučovaní základov programovacieho jazyka C++ na hodinách informatiky na strednej škole. Je veľmi dôležité, aby sa s týmto jazykom oboznámili žiaci už na strednej škole, pretože im to pomôže v ich ďalšom štúdiu na vysokej škole, resp. programátorskej praxi.

Súčasťou práce je osem príloh – vývojový diagram, ktorý ilustruje postup práce na vývoji programu v jazyku C++, tabuľkovo spracované informácie týkajúce sa priority operátorov, základných typov premenných, či špeciálnych znakov určených na formátovanie kódu v jazyku C++. Okrem toho v prílohách sú zaradené aj dva priebežné testy a aj ich vzorové riešenia. Testy majú slúžiť na overenie vedomostí žiakov.

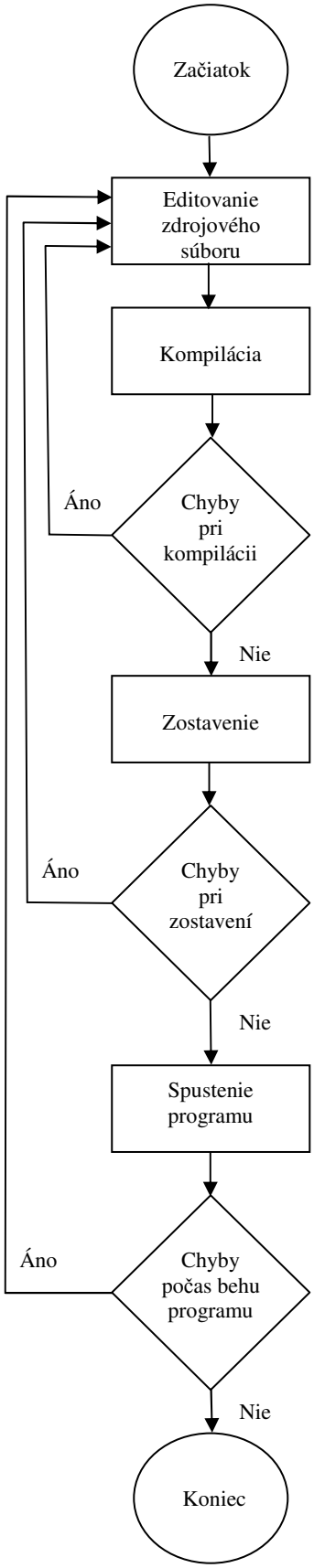
Dúfam, že táto práca bude pre pedagógov užitočnou pomôckou pri výučbe základov programovacieho jazyka C++, ktorý je v dnešnej dobe veľmi populárny a samozrejme sa aj naďalej mení a rozvíja.

ZOZNAM BIBLIOGRAFICKÝCH ZDROJOV

1. Eckel, B – Allison, Ch. 2006. Myslíme v jazyku C++. 2 díl – knihovna zkušeného programátora. Praha. Grada Publishing. ISBN: 80-247-1015-3
2. Glassborow, F. 2005. Naučte se programovat!. Podrobný průvodce programováním v C++. Praha. Grada Publishing. ISBN: 80-247-1243-1
3. Liberty, J. 2007. Naučte se C++ za 21 dní. 2 aktualizované vydání. Brno. Computer Press. ISBN: 978-80-251-1583-1
4. Virius, M. 2006. Jazyky C a C++. Kompletní kapesní průvodce programátora. Praha. Grada Publishing. ISBN: 80-247-1494-9

ZOZNAM PRÍLOH

Príloha 1 Vývojový cyklus programu C++



Prameň: Liberty Jesse, 2007, s. 36

Príloha 2 Priorita operátorov v C++

Základné priority operátorov (vyššie zapísané majú vyššiu prioritu):

::	stanovenie oboru platnosti
( )	zátvorky
! + - ++ --	negácia, unárne + a -, operátor inkrementácie a dekrementácie (napríklad i--)
* / %	násobenie, delenie, delenie so zvyškom
+ -	binárne + a – (napríklad –5 je unárne mínus, ale 4–5 je binárne mínus)
<< >>	bitový posun (doľava a doprava)
< <= > >=	porovnanie
== !=	rovnosť, nerovnosť
&	bitové AND
^	bitové XOR
|	bitové OR
&&	logické AND
||	logické OR
= *= /= += -=	priradenie
,	operátor čiarka

Prameň: Liberty Jesse, 2007, v prílohe

Príloha 3 Základné typy premenných v C++

Typ	Veľkosť	Hodnoty
bool	1 bajt	true, resp. false
unsigned short int	2 bajty	0 až 65 535
short int	2 bajty	-32 768 až 32 767
unsigned long int	4 bajty	0 až 4 294 967 295
long int	4 bajty	-2 147 483 648 až 2 147 483 647
int (16 bitov)	2 bajty	-32 768 až 32 767
int (32 bitov)	4 bajty	-2 147 483 648 až 2 147 483 647
unsigned int (16 bitov)	2 bajty	0 až 65 535
unsigned int (32 bitov)	4 bajty	0 až 4 294 967 295
char	1 bajt	256 znakových hodnôt
float	4 bajty	1,2e-38 až 3,4e38
double	8 bajtov	2,2e-308 až 1,8e308

Prameň: Liberty Jesse, 2007, s. 61

Poznámka: Veľkosť premenných sa môže od hodnôt uvedených v tabuľke líšiť, a to v závislosti od typu kompilátora a počítača, ktorý používame.

Príloha 4 Špeciálne znaky určené na formátovanie kódu v C++

Znak	Význam
\a	pípnutie
\b	zmazanie predchádzajúceho znaku (backspace)
\f	posun o stránku
\n	nový riadok
\r	návrat vozíka
\t	tabulátor
\v	vertikálny tabulátor
\'	apostrof
\"	úvodzovky
\?	otáznik
\\	spätná lomka
\000	oktálový (osmičkový) zápis
\xhhh	hexadecimálny (šestnástkový) zápis

Prameň: Liberty Jesse, 2007, s. 71-72

Prvý priebežný test

Meno a priezvisko žiaka: .....
Trieda: .....

1. Vysvetlite rozdiel medzi testovaním a ladením programu.

2. Vymenujte a popíšte chyby, ktoré sa môžu vyskytnúť v programe.

3. Napíšte program, ktorý vypočíta objem kvádra, ak sú zadané dĺžky jeho hrán a, b, c. Uložte ho do súboru „VasePriezvisko31.cpp“ na pracovnú plochu počítača.

4. Napíšte program, ktorý vypíše najväčšie číslo z troch zadaných čísel. Uložte ho do súboru „VasePriezvisko41.cpp“ na pracovnú plochu počítača.

5. Napíšte program, ktorý na obrazovku počítača vypíše postupnosť čísel *1, 2, ..., n*, pričom každé číslo bude napísané na samostatnom riadku obrazovky. Najväčšie číslo zadá užívateľ a jeho hodnota musí byť v rozsahu <1,20>. Overte aj možnosť, že užívateľ nezadá hodnotu z daného intervalu. Program uložte do súboru „VasePriezvisko51.cpp“ na pracovnú plochu počítača.

Prvý priebežný test

Meno a priezvisko žiaka: .....
Trieda: .....

1. Vysvetlite rozdiel medzi testovaním a ladením programu.

Testovanie je činnosť, pri ktorej zisťujeme správnosť algoritmu a jeho správanie sa v rôznych hraničných situáciách a ladenie je proces, počas ktorého chyby, ktoré sme testovaním zistili, odstraňujeme.

2. Vymenujte a popíšte chyby, ktoré sa môžu vyskytnúť v programe.

Syntaktické – vznikajú neznalosťou alebo nepozornosťou pri písaní programu. Napr. Vynecháme bodkočiarku, chybne napíšeme príkaz atď.
Sémantické – syntakticky je program napísaný správne, ale správny výsledok napriek tomu nedostaneme. Tieto chyby rozdeľujeme na: a/ chyby vznikajúce počas behu programu – počítač nie je schopný pokračovať v práci. Príčinou môže byť napr. delenie nulou, otváranie neexistujúceho súboru, atď., b/ chyby logické – v tomto prípade je nesprávny samotný algoritmus. Napr. vo výraze vynecháme potrebnú zátvorku, atď.

3. Napíšte program, ktorý vypočíta objem kvádra, ak sú zadané dĺžky jeho hrán a, b, c. Uložte ho do súboru „VasePriezvisko31.cpp“ na pracovnú plochu počítača.

```
#include <iostream>
using namespace std;
int main()
{
 float a,b,c,objem;
 cout << "Zadaj dlzku hrany a: ";
 cin >> a;
 cout << "Zadaj dlzku hrany b: ";
 cin >> b;
 cout << "Zadaj dlzku hrany c: ";
 cin >> c;
 cout << endl;
 objem = a*b*c;
 cout << "Objem kvadra je: " << objem << "\n";
    char reakcia;
    cin >> reakcia;
    return 0;
}
```

4. Napíšte program, ktorý vypíše najväčšie číslo z troch zadaných čísel. Uložte ho do súboru „VasePriezvisko41.cpp“ na pracovnú plochu počítača.

```
#include <iostream>
using namespace std;
int main()
{
 float c1,c2,c3,max;
 cout << "Zadaj prve cislo: ";
 cin >> c1;
 cout << "Zadaj druhe cislo: ";
 cin >> c2;
 cout << "Zadaj tretie cislo: ";
 cin >> c3;
 cout << endl;
 if (c1>c2)
    max = c1;
    else max = c2;
 if (c3>max)
    max = c3;
 cout << "Najvacsie cislo je: " << max << "\n";
    char reakcia;
    cin >> reakcia;
    return 0;
}
```

5. Napíšte program, ktorý na obrazovku počítača vypíše postupnosť prirodzených čísel *1,2,...,n*, pričom každé číslo bude napísané na samostatnom riadku obrazovky. Najväčšie číslo zadá užívateľ a jeho hodnota musí byť v rozsahu <1,20>. Uvažujte aj s možnosťou, že užívateľ nezadá hodnotu z daného intervalu. Program uložte do súboru „VasePriezvisko51.cpp“ na pracovnú plochu počítača.

```
#include <iostream>
using namespace std;
int main()
{
 int n,i;
          do
          {   cout << "Vyberte hornu hranicu z rozsahu 1-20:\n";
                    cin >> n;
          } while ((n<1) || (n>20));
          cout << endl;
 for (i = 1; i <= n; i++)
                    cout << i << "\n";
   char reakcia;
   cin >> reakcia;
   return 0;
}
```

Poznámka: Pri tvorbe programov uvádzame len jeden z možných spôsobov riešenia. Žiaci daný problém môžu vyriešiť aj iným spôsobom a riešenie môže byť správne.

Druhý priebežný test

Meno a priezvisko žiaka: .....
Trieda: .....

1. Z ktorých dvoch častí sa skladá funkcia v programe? Popíšte ich.

2. Odhaľte chybu, ktorá sa nachádza v nasledujúcej časti kódu.

```
unsigned short Pole[5][4];
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
        Pole[i][j] = i+j;
```

3. Napíšte program, ktorý si od užívateľa vypýta názov programovacieho jazyka, ktorý používa. Keď ho užívateľ zadá, program nech vypíše, že zadaný programovací jazyk je skvelý (napr. „C++ je skvelý“). Program uložte do súboru „VasePriezvisko32.cpp“ na pracovnú plochu počítača.

4. Napíšte program, ktorý si od užívateľa postupne vypýta štyri celé čísla a uloží si ich do jednorozmerného poľa. Následne nech program vypíše tieto užívateľom zadané čísla v opačnom poradí. Program uložte do súboru „VasePriezvisko42.cpp“ na pracovnú plochu počítača.

5. Napíšte program, ktorý vypočíta obvod všeobecného trojuholníka, ak sú dané dĺžky jeho strán. Výsledok dajte vypísať do textového súboru s názvom „trojuholnik.txt“. Program uložte do súboru „VasePriezvisko52.cpp“ na pracovnú plochu počítača.

Druhý priebežný test

Meno a priezvisko žiaka: .....
Trieda: .....

1. Z ktorých dvoch častí sa skladá funkcia v programe? Popíšte ich.

Funkcia sa skladá z hlavičky a tela. Hlavička obsahuje návratový typ, názov funkcie a parametre, ktoré tejto funkcii prislúchajú. Telo funkcie sa skladá zo začiatočnej zloženej zátvorky, žiadneho alebo viac príkazov a koncovej zloženej zátvorky.

2. Odhaľte chybu, ktorá sa nachádza v nasledujúcej časti kódu.

```
unsigned short Pole[5][4];
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
        Pole[i][j] = i+j;
```

Zadané pole je 5 prvkov krát 4 prvky. Kód však inicializuje pole 4x5.

3. Napíšte program, ktorý si od užívateľa vypýta názov programovacieho jazyka, ktorý používa. Keď ho užívateľ zadá, program nech vypíše, že zadaný programovací jazyk je skvelý (napr. „C++ je skvelý“). Program uložte do súboru „VasePriezvisko32.cpp“ na pracovnú plochu počítača.

```
#include <iostream>
using namespace std;
int main()
{
   char jazyk[15];
   cout << "Ktory jazyk pouzivas pri svojom programovani?\n";
   cin >> jazyk;
   cout << std::endl;

   cout << "Jazyk " << jazyk << " je skvely.";
        char reakcia;
        cin >> reakcia;
        return 0;
}
```

4. Napíšte program, ktorý si od užívateľa postupne vypýta štyri celé čísla a uloží si ich do jednorozmerného poľa. Následne nech program vypíše tieto užívateľom zadané čísla v opačnom poradí. Program uložte do súboru „VasePriezvisko42.cpp“ na pracovnú plochu počítača.

```
#include <iostream>
using namespace std;
int main()
{
	int cislo[4];
	int i;
	for ( i=1; i<5; i++)
	{
		cout << "Zadaj prvu hodnotu:";
		cin >> cislo[i];
	}
	cout << endl;
	for (i = 4; i>0; i--)
		cout << cislo[i] << endl;
	char reakcia;
	cin >> reakcia;
	return 0;
}
```

5. Napíšte program, ktorý vypočíta obvod všeobecného trojuholníka, ak sú dané dĺžky jeho strán. Výsledok dajte vypísať do textového súboru s názvom „trojuholnik.txt“. Program uložte do súboru „VasePriezvisko52.cpp“ na pracovnú plochu počítača.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
 ofstream fout("trojuholnik.txt");
 float a,b,c,obvod;
 cout << "Zadaj dlzku strany a: ";
 cin >> a;
 cout << "Zadaj dlzku strany b: ";
 cin >> b;
 cout << "Zadaj dlzku strany c: ";
 cin >> c;
 obvod=a+b+c;
 cout << endl;
 cout << "Vysledok je zapisany v subore ´trojuholnik.txt´ na disku pocitaca.";
 fout<<"Obvod trojuholnika so stranami: " << a << ", " << b << ", " << c << " je " <<obvod<< ".";
 fout.close();
   char reakcia;
   cin >> reakcia;
   return 0;
 }
```

Poznámka: Pri tvorbe programov uvádzame len jeden z možných spôsobov riešenia. Žiaci daný problém môžu vyriešiť aj iným spôsobom a riešenie môže byť správne.