

Štart s C++

C++, tento vzrušujúci jazyk, ktorý spája v sebe C jazyk, podporu objektovo-orientovaného programovania a generické programovanie, bol jedným z najdôležitejších programovacích jazykov 90-tych rokov a svoju pozíciu si udržiava dodnes. Jeho C predchodca prináša do C++ tradíciu **výkonného, kompaktného, rýchleho a prenositeľného** jazyka. **Objektovo-orientované** dedičstvo dodáva C++ novú programovaciu metodológiu, umožňujúcu zvládnuť stupňujúcu sa zložitost' moderných programovacích úloh. Jeho šablóny prinášajú ešte ďalšiu novú programovaciu metodológiu: **generické programovanie**. Tieto tri črty sú požehnaním, ale súčasne i prekliatím – robia jazyk veľmi výkonným, ale na druhej strane to predstavuje mnoho učenia.

Postupne prejdeme od jednoduchých základov cez objektovo-orientované programovanie po generické programovanie.

C++ spája tri samostatné programovacie kategórie:

- procedurálny jazyk, reprezentovaný jazykom C,
- objektovo-orientovaný jazyk, reprezentovaný C++ triedami, dopĺňujúcimi jazyk C, a
- generické programovanie, založené na C++ šablónach.

Samozrejme, že nebude čas prejsť všetky detaily jazyka, avšak pokúsime sa prejsť hlavné najvýznamnejšie vlastnosti jazyka a súčasne si ukážeme zaujímavé programovacie praktiky, použiteľné na riešenie i zložitejších problémov.

Začiatky C++

Počítačová technológia sa vyvíja veľmi rýchlo. Dnešné notebooky sú výkonnejšie než sálový počítač v 60-tych rokoch. A podobne sa vyvíjali i programovacie jazyky. Zmeny možno nie sú dramatické, ale sú dôležité. Väčšie, výkonnejšie počítače akceptujú väčšie, výkonnejšie programy, ktoré zasa prinášajú nové problémy, týkajúce sa správy programov a ich údržby.

V 70-tych rokoch jazyky ako C a Pascal priniesli éru štruktúrovaného programovania, filozofie, ktorá priniesla určitý poriadok a disciplínu do oblasti, ktorá to už nevyhnutne potrebovala. Jazyk C okrem prostriedkov pre štruktúrované programovanie produkoval kompaktné, rýchle programy spolu so schopnosťou adresovať hardvér, napríklad manažovať komunikačné porty a disky. Tieto danosti pomohli C jazyku stať sa dominantným programovacím jazykom 80-tych rokov. Súčasne 80-te roky sú rokmi začiatku vývoja nového programovacieho paradigmu: objektovo-orientované programovanie, OOP, ktoré bolo zaintegrované do jazykov ako Smalltalk a C++.

C jazyk

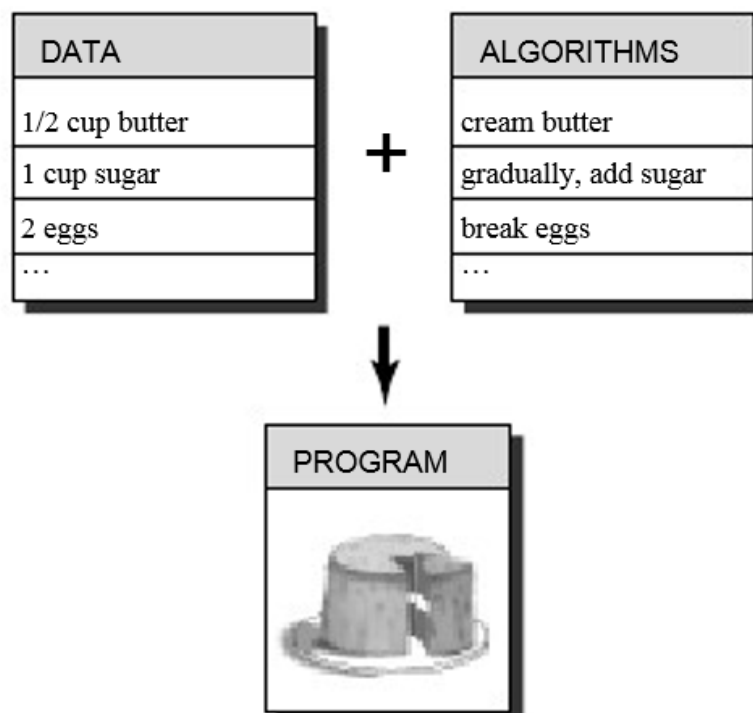
Nazačiatku 70-tych rokov Dennis Ritchie pracoval na vývoji operačného systému Unix. Na túto prácu potreboval jazyk, ktorý by bol stručný, produkoval kompaktné, rýchle programy a ktorý by umožňoval efektívne ovládať hardvér.

Tradične tieto požiadavky spĺňal tzv. jazyk symbolických adres – assembler, ktorý bol úzko previazaný na interný strojový jazyk počítača. Avšak assembler je nízko-úrovňový jazyk, t.j. pracuje priamo s hardvérom (pristupuje priamo k registrom CPU a pamäti). Okrem toho assembler je špecifický pre konkrétny počítačový procesor. Takže ak potrebujeme preniesť program v asembleri na iný druh počítača, musíme kompletne prepísať celý program používajúc iný assembler.

Ale systém Unix plánovaný pre prácu na rozmanitých typoch počítačov (resp. platformách). To vedie k použitiu vyššieho, strojovo-nezávislého jazyka. Vyšší jazyk je orientovaný na riešenie problémov a nie na špecifický hardvér. Špeciálne programy, nazývané kompilátory, prekladajú vyšší jazyk do interného jazyka konkrétného počítača. Takto môžeme vyšší programovací jazyk použiť na rôznych platformách, samozrejme s osobitným kompilátorom pre každú platformu. Ritchie chcel jazyk, ktorý by kombinoval nízkoúrovňovú výkonnosť a prístup k hardvéru so všeobecnosťou a prenositeľnosťou vyššieho jazyka. A tak na základe starších jazykov vytvoril jazyk C.



Obrázok 1 Dennis Ritchie



Obrázok 2 Dáta + Algoritmus = Program

Filozofia programovania v jazyku C

Pretože C++ transplantuje novú programovaciu filozofiu do C jazyka, pozrieme sa najskôr na staršiu filozofiu programovania jazyka C. Všeobecne povedané, počítačové jazyky narábajú s dvomi pojmami – dáta a algoritmy. Dáta vytvárajú informáciu, ktorú používa a spracováva program. Algoritmy sú metódy, ktoré program používa. Ako väčšina vtedajších hlavných jazykov i jazyk C je procedurálny jazyk. Znamená to, že dôraz kladie na algoritmickú stránku programovania. Koncepčne, procedurálne programovanie pozostáva z určenia činností, ktoré by mal počítač vykonať a potom prostredníctvom jazyka tieto činnosti implementovať. Program stanovuje počítaču množinu procedúr, podľa ktorých má počítač pracovať aby vytvoril požadovaný výstup, podobne ako recept predpisuje množinu procedúr, ktoré musí kuchár spraviť aby upiekol koláč.

Skoršie procedurálne jazyky, ako FORTRAN a BASIC, začali mať organizačné problémy so zväčšujúcim sa rozsahom programov. Napríklad programy často používajú vetviace príkazy, ktoré smerujú vykonávanie programu na rôzne množiny inštrukcii na základe výsledku nejakého testu. Mnoho starších programov mali tak zamotaný tok vykonávania („nazývaný špagetové programovanie“), že bolo prakticky nemožné program čítať a pochopiť, a modifikovať takýto program bola pozvánka pre katastrofu. Odpoveďou počítačových vedcov bolo vyvinutie disciplinovanejšieho štýlu programovania, nazývaného štruktúrované programovanie. C má vlastnosti, ktoré podporujú tento prístup. Napríklad štruktúrované programovanie limituje vetvenie (výber nasledujúcej inštrukcie) na malú množinu „vychovaných“ inštrukcii. C jazyk začleňuje tieto konštrukcie (`for` cyklus, `while` cyklus, `do while` cyklus a `if else` príkaz) do svojho slovníka.

Medzi ďalšie nové princípy patril i návrh zhora nadol. V C jazyku je hlavnou myšlienkou rozbiť rozsiahly program na menšie, ovládateľné úlohy. Ak je niektorá z týchto stále príliš rozsiahla, rozdelíme ju na ešte menšie úlohy. V tomto procese pokračujeme pokiaľ nie je celý program rozškatuľkovaný do malých, ľahko programovateľných modulov. Schéma jazyka C podporuje takýto prístup a nabáda k vyvíjaniu programových celkov, nazývaných funkcie, ktoré reprezentujú jednotlivé moduly úloh. Ako ste si všimli, techniky štruktúrovaného programovania odráža procedurálne myslenie, kde sa na program pozerá ako činnosti, ktoré sa majú vykonať.

C++ posun: Objektovo-orientované programovanie

Aj keď princípy štruktúrovaného programovania zlepšili zrozumiteľnosť, spoľahlivosť a údržbu programov, veľmi rozsiahle programy aj naďalej predstavovali zložitý problém. OOP prináša nový prístup na túto výzvu. Na rozdiel od procedurálneho programovania, ktoré kladie dôraz na algoritmy, OOP zdôrazňuje dáta. Radšej než prispôbovať problém na procedurálny prístup jazyka, OOP sa pokúša prispôbiť jazyk problému. Hlavnou myšlienkou je navrhnuť také dátové formuláre, ktoré budú zodpovedať podstatným charakteristikám problému.

Trieda (`class`) v C++ predstavuje špecifikáciu takéhoto nového dátového formulára a objekt je konkrétna dátová štruktúra, vytvorená podľa tohto plánu. Vo všeobecnosti trieda definuje aké dáta sú použité na reprezentáciu objektu a operácie, ktoré sa môžu s dátami vykonávať.

OOP prístup k návrhu programu je najskôr navrhnuť triedy, ktoré presne reprezentujú tie podmienky, s ktorými sa program zaoberá. Až potom prejdeme na návrh programu, používajúc objekty navrhnutých

tried. Proces, pri ktorom postupujeme z nižšej úrovne, ako sú triedy, na vyššiu úroveň, ako je návrh programu, sa nazýva programovanie zdola nahor.

OOP nie je len spojenie dát a metód do definície triedy. OOP napríklad umožňuje tvorbu znovu použiteľného kódu, čím dokáže ušetriť mnoho práce. Skrývanie informácie chráni dáta pred nevhodným zásahom. Polymorfizmus dovoľuje vytvárať viacnásobné definície operátorov a funkcií, pričom ich použitie určuje kontext programu. Dedičnosť dovoľuje odvodzovať nové triedy zo starších tried. Ako vidíme, OOP zavádza mnoho nových myšlienok a znamená odlišný prístup k programovaniu ako k nemu pristupuje procedurálne programovanie. Namiesto koncentrovania sa na úlohy sa sústreďujeme na reprezentovanie predstáv. Namiesto programovania zhora nadol občas použijeme prístup zdola nahor.

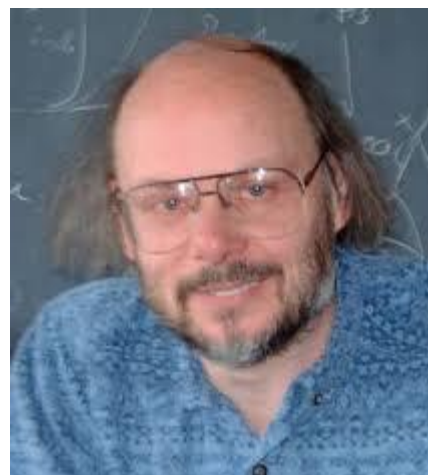
Návrh užitočnej, spoľahlivej triedy môže byť zložitá úloha. Našťastie OOP jazyky to zjednodušujú začlenením existujúcich tried do vlastného programu. Predajcovia dodávajú rozmanité užitočné knižnice tried, vrátane knižníc tried, určených na zjednodušenie tvorby programov pre rôzne operačné systémy. Jednou z naozajstných výhod C++ je, že dovoľuje ľahko znova používať a adaptovať existujúci, otestovaný kód.

C++ generické programovanie

Generické programovanie je ďalšia programovacia paradigma, ktorú jazyk C++ podporuje. Spolu s OOP sa podieľa na celi zjednodušiť znovu použitie kódu a spôsobe abstrakcie všeobecných myšlienok. Zatiaľ čo OOP kladie dôraz na dátový aspekt programovania, generické programovanie zdôrazňuje nezávislosť od konkrétneho dátového typu. A jeho zámer je iný. OOP je nástroj na správu rozsiahlych projektov, zatiaľ čo generické programovanie poskytuje nástroje na realizáciu všeobecných úloh, ako je napríklad triedenie dát alebo zlučovanie zoznamov. Pojem generické označuje typovo nezávislý kód. C++ reprezentácie dát prinášajú mnoho typov – celé čísla, reálne čísla, znaky, reťazce znakov a užívateľsky definované zložené typy. Ak potrebujeme napríklad usporiadať dáta týchto rôznych typov, normálne by sme museli vytvoriť samostatnú triediacu funkciu pre každý typ. Generické programovanie rozširuje jazyk tak, že môžeme napísať funkciu pre generický (t.j. nešpecifikovaný) typ iba raz a použiť ju pre rozmanité konkrétne typy. C++ šablóny poskytujú mechanizmus na takúto prácu.

Genéza C++

Podobne ako jazyk C, C++ začalo svoj Bellových laboratóriách, kde Bjarne Stroustrup vyvinul tento jazyk na začiatku 80-tych rokov. Jeho hlavným cieľom bolo uľahčiť a spríjemniť písanie dobrých programov. Stroustrup viac zaujímalo, aby C++ bolo užitočné, než aby presadzoval konkrétnu programovaciu filozofiu alebo štýl. Potreby reálneho programovania sú dôležitejšie než teoretická čistota pri determinovaní jazykových črt C++. Stroustrup založil C++ na jazyku C kvôli jeho stručnosti, vhodnosti na systémové programovanie, širokú dostupnosť a jeho úzku väzbu na operačný systém Unix. OOP vlastnosti boli inšpirované jazykom pre počítačovú simuláciu Simula67.



Obrázok 3 Bjarne Stroustrup

Stroustrup pridal OOP vlastnosti a podporu pre generické programovanie do C jazyka bez významnej zmeny C zložky. Takže C++ je nadmnožina C, čo znamená, že akýkoľvek platný C program je i platným C++ programom. Existujú nejaké minoritné odlišnosti, ale nie sú kľúčové. C++ programy môžu využívať existujúce C softvérové knižnice. Knižnice sú kolekcie programových modulov, ktoré môžeme volať z programu. Poskytujú osvedčené riešenia mnohých bežných programovacích problémov, čím ušetrí mnoho času a úsilia. Toto pomohlo k rozšíreniu C++.

Meno C++ vychádza z inkrementačného operátora ++ jazyka C. A preto C++ korektne naznačuje rozšírenú verziu C.

Počítačový program interpretuje problém zo skutočného života do postupnosti činností, ktoré má počítač vykonať. OOP aspekt jazyka C++ dáva jazyku schopnosť popisovať pojmy problému a C zložka jazyka C++ poskytuje jazyku schopnosť priblížiť sa k hardvéru. Takáto kombinácia schopností umožnila rozšírenie C++.

Až keď C++ dosiahol určitý úspech Stroustrup doplnil šablóny, ktoré umožnili generické programovanie.

Prenositel'nosť a štandardy

Predstavme si, že vytvárame program pre Windows a ešte skôr ako ho dokončíme, zadávateľ zmení požiadavky na operačný systém na Unix. Môžeme spúšťať program na novej platforme? Samozrejme musíme program prekompilovať C++ kompilátorom, určeným pre novú platformu. Ale musíme zmeniť i kód, ktorý sme už napísali? Ak program preložíme bez nutnosti nejakých zmien a program bez problémov rekompilujeme, hovoríme, že **program je prenositeľný**.

Existuje niekoľko prekážok prenositeľnosti, pričom prvou je hardvér. Program, ktorý je hardvérovo špecifický bude pravdepodobne neprenositeľný. Táto prekážka sa dá čiastočne obísť tak, že všetok hardvérovo-závislý kód lokalizujeme do funkcionálnych modulov a pre každú platformu prepíšeme iba tieto špecifické moduly.

Druhou prekážkou sú odchýlky jazyka. Aj keď väčšina tvorcov sa snaží vytvárať navzájom kompatibilné verzie C++, je to ťažké bez štandardu, ktorý by presne popisoval ako má jazyk fungovať. A preto americký normalizačný úrad vytvoril v roku 1990 výbor, ktorý mal vytvoriť normu jazyka C++. K nim sa pridala i medzinárodná organizácia pre štandardy (ISO) aby spoločne vytvorili normu C++.

Po niekoľkých rokoch práce bola prijatá v roku 1998 norma jazyka C++. Táto norma, nazývaná C++98 nielen prečistila popis už existujúcich vlastností jazyka C++, ale rozšírila jazyk o výnimky, identifikáciu typov za chodu programu (RTTI), šablóny a Standard Template Library (STL).

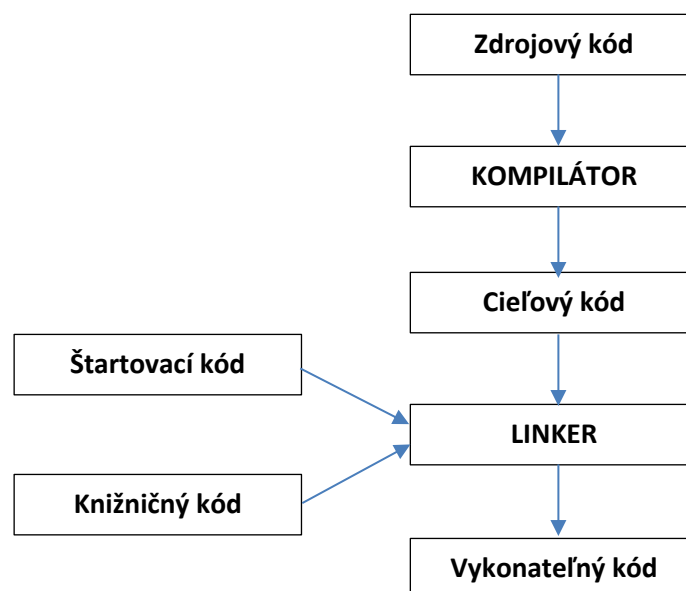
Rok 2003 priniesol publikáciu druhého vydania C++ normy. Toto vydanie normy predstavovala iba technickú revíziu, t.j. upratilo prvú verziu – zorganizovalo typy, redukovalo nejednoznačnosti, ale nemenilo vlastnosti jazyka. Toto vydanie je označované ako C++03.

C++ sa rozvíjalo ďalej a ISO výbor schválil novú normu v auguste 2011, neformálne nazývanú C++11. Podobne ako C++98 i C++11 pridala do jazyka mnoho vlastností. Okrem toho jej cieľom bolo odstránenie nezrovnalostí a uľahčenie naučenia sa a použitia jazyka.

Mechanizmus tvorby programu

Predpokladajme, že už máme napísaný program v jazyku C++. Ako ho donútime pracovať? Presný postup závisí od počítačového prostredia a konkrétneho C++ kompilátora, ktorý použijeme, avšak základné kroky sú rovnaké:

1. Pomocou textového editora napíšeme program a uložíme ho do súboru. Tento súbor predstavuje **zdrojový kód programu**.
2. Zdrojový kód preložíme (skompilujeme). Znamená to, že spustíme program, ktorý preloží zdrojový kód do interného jazyka, nazývaného strojový kód, používaného hostiteľským počítačom. Súbor, obsahujúci preložený program predstavuje **cieľový kód** (object code) programu.
3. Cieľový kód spojíme s ďalším kódom. Napríklad C++ programy bežne používajú knižnice. C++ knižnica obsahuje cieľový kód množiny počítačových rutín, nazývaných funkcie. Spájanie (linkovanie) spája cieľový kód s cieľovým kódom týchto funkcií a s nejakým štandardným štartovacím kódom, aby sa vytvorila spustiteľná verzia programu. Súbor, obsahujúci tento finálny produkt sa nazýva **vykonateľný kód**.

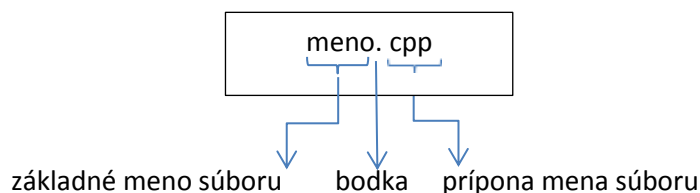


Obrázok 4 Postup programovania

Tvorba súboru so zdrojovým kódom

Tvorba zdrojového kódu závisí od nástrojov, ktoré má programátor k dispozícii. Mnoho implementácií C++, medzi nimi i Microsoft Visual C++, poskytuje integrované vývojové prostredie, ktoré umožňuje organizovať všetky kroky vývoja programu, vrátane editácie, v rámci jedného hlavného programu. Niektoré iné implementácie, napríklad GNU C++ pre Unix a Linux, manažujú iba kompiláciu a spájanie (linkovanie) a očakávajú zadávanie príkazov z príkazového riadku. V takomto prípade môžeme na tvorbu a modifikáciu zdrojového kódu použiť ľubovoľný textový editor.

Pri pomenúvaní súboru musíme použiť vhodnú príponu, ktorá bude súbor identifikovať ako C++ resp. C zdrojový súbor. Táto prípona nebude oznamovať iba nám, že sa jedná o zdrojový súbor, ale i samotnému kompilátoru. Prípona pozostáva z bodky, za ktorou nasleduje znak alebo skupina znakov.



Použitie prípony závisí od implementácie C++. Napríklad Unix rozlišuje malé a veľké písmená a prípona veľké C označuje meno zdrojového C++ súboru. Malé písmeno c označuje C program.

Tabuľka 1 Prípony mena zdrojového súboru

C++ implementácia	Prípona zdrojového súboru
Unix	C, cc, cxx, c
GNU C++	C, cc, cxx, cpp, c++
Microsoft Visual C++	cpp, cxx, cc

Systémy, ktoré nerozlišujú malé a veľké písmená v názve súboru, používajú na rozlíšenie typu zdrojového súboru dopĺňujúce písmená.

Kompilácia a linkovanie (spájanie)

Pôvodne Stroustrup implementoval C++ s kompilátorom z jazyka C++ do jazyka C namiesto vývoja priameho prekladača z C++ do cieľového kódu. Tento program (`cfront`) prekladal C++ zdrojový kód do C zdrojového kódu, ktorý sa následne prekladal štandardným C kompilátorom. Tento prístup zjednodušil zavedenie C++ do komunity. Podobne i ďalšie implementácie použili tento prístup pri zavádzaní C++ na iné platformy. Postupne ako sa vývoj C++ vzrastal, mnoho tvorcov kompilátora C++ vytváralo kompilátory, ktoré priamo kompilovali C++ zdrojový kód do cieľového kódu. Tento priamy prístup zrýchlil proces kompilácie a zvýraznil skutočnosť, že C++ je samostatný, aj keď podobný jazyk.

Unix a Linux požívajú na preklad najčastejšie prekladač GNU C++, ktorý sa štartuje príkazovým riadkom, napríklad:

```
g++ mojprogram.cpp
```

Výsledkom príkazu je vykonateľný súbor `a.out`.

GNU C++ kompilátor je v systéme Windows. Výsledkom príkazu je vykonateľný súbor `a.exe`.

Windows kompilátory

Medzi najznámejšie Windows kompilátory sa v súčasnosti radí Microsoft Visual C++. Napriek rôznemu dizajnu majú Windows kompilátory mnoho spoločných črt. Zvyčajne musíme vytvoriť

projekt a do tohto projektu pridať súbor, resp. súbory, tvoriace program. Po zostavení projektu môžeme kompilovať a linkovať program.

Nástup do programovania

Keď stavíme jednoduchý dom, začíname základmi a nosnou konštrukciou. Ak od začiatku nemáme pevnú štruktúru, neskôr pri riešení detailov budeme mať problémy. Podobne, keď sa učíme počítačový jazyk mali by sme začať základnou štruktúrou programu a až neskôr prejsť na detaily ako sú cykly a objekty.

Začnime klasicky s jednoduchým C++ programom, ktorý zobrazí nejaký oznam. C++ je citlivý na veľkosť písmen, t.j. rozlišuje medzi znakmi malých a veľkých písmen a preto treba dávať pozor na správny zápis identifikátorov.

C++ program

```
// prvy.cpp - zobrazí oznam

#include <iostream>

int main()
{
    using namespace std;
    cout << "Ahoj C++.";
    cout << endl;
    cout << "Zaciname!" << endl;
    return 0;
}
```

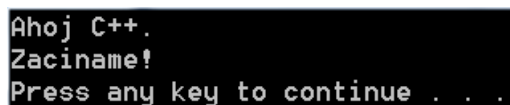
C program

```
// prvy.c - zobrazí oznam

#include <stdio.h>

int main()
{
    printf("Ahoj C++.");
    printf("\n");
    printf("Zaciname!\n");
    return 0;
}
```

Výstup

A screenshot of a terminal window showing the output of the C++ program. The text displayed is: "Ahoj C++.", "Zaciname!", and "Press any key to continue . . .".

```
Ahoj C++.
Zaciname!
Press any key to continue . . .
```


Program vytvárame zo stavebných blokov, nazývaných funkcie. Typicky program zostavujeme tak, že najskôr definujeme hlavne úlohy a potom navrhujeme samostatné funkcie, ktoré tieto úlohy riešia. Uvedený program (C++) je veľmi jednoduchý a na riešenie stačí jediná funkcia nazvaná `main`. Príklad obsahuje nasledujúce prvky:

- Komentár, indikovaný prefixom `//`
- Direktívu preprocesora `#include`
- Hlavičku funkcie `int main()`
- Direktívu `using namespace`
- Telo funkcie, ohraničené znakmi `{ a }`

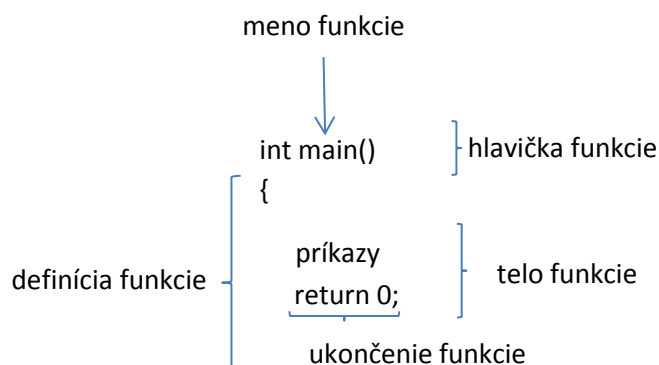
Príklady, ktoré obsahujú vlastnosti objektu `cout` na zobrazenie oznamu, resp. funkciu `printf` (C program). Pozrime sa bližšie na tieto rozmanité prvky programu.

Vlastnosti funkcie `main()`

Ak odstránime *ozdoby*, tak vzorový program má nasledujúcu základnú štruktúru:

```
int main()
{
    príkazy
    return 0;
}
```

Tieto riadky vyjadrujú, že existuje funkcia nazvaná `main()` a popisujú ako má funkcia fungovať. Dohromady predstavujú definíciu funkcie. Definícia má dve časti: prvý riadok `int main()`, ktorý nazývame hlavička funkcie a časť, uzatvorenú v zložených zátvorkách (`{ a }`), ktorá tvorí telo funkcie. Hlavička funkcie predstavuje rozhranie funkcie pre zvyšok programu a telo funkcie reprezentuje príkazy pre počítač, ktoré má funkcia vykonať. V C++ sa celá inštrukcia nazýva príkaz. Každý príkaz musí byť ukončený bodkočiarkou.



Posledný príkaz funkcie `main()`, nazývaný príkaz návratu, funkciu ukončuje. Príkaz reprezentuje činnosť, ktorá sa má vykonať. Príkaz začína a kde končí. C++ i C jazyk používajú ako ukončenie príkazu znak bodkočiarky, ktorá je súčasťou príkazu (na rozdiel od oddeľovača príkazov).

Hlavička funkcie ako rozhranie

Vo všeobecnosti C++ funkcia je aktivovaná, alebo volaná, inou funkciou. Hlavička funkcie popisuje rozhranie medzi volanou funkciou a funkciou, ktorá ju volá. Časť, ktorá sa nachádza pred menom funkcie sa nazýva **návratový typ**. Popisuje informáciu, ktorá ide z volanej funkcie do funkcie, ktorá ju volá. Časť v zátvorkách za menom funkcie sa nazýva zoznam argumentov alebo parametrov. Popisuje informáciu, ktorá ide z volajúcej funkcie do volanej funkcie. Tento všeobecný popis je trochu mätúci, keď ho aplikujeme na funkciu `main()`, pretože normálne túto funkciu nevoláme z inej časti programu. Avšak `main` je volaná štartovacím kódom, ktorý kompilátor pripája k nášmu programu ako prepojenie medzi programom a operačným systémom.

C++ funkcia, volaná inou funkciou, môže vracať hodnotu aktivujúcej (volajúcej) funkcie. Táto hodnota sa nazýva *návratová hodnota*. V prípade funkcie `main()` sa vracia hodnota typu `int`, ako to naznačuje kľúčové slovo `int`. Ďalej si všimnime prázdne zátvorky. Vo všeobecnosti C++ funkcia môže posilať informáciu do volanej funkcie. Túto informáciu popisuje časť, uzatvorená okrúhlymi zátvorkami. V našom prípade funkcia `main()` nedostáva žiadnu informáciu, alebo inak povedané nemá žiadne argumenty.

Tiež môžeme použiť variant:

```
int main(void)
```

Použitie kľúčového slova `void` v zátvorkách je explicitný spôsob vyjadrenia, že funkcia nemá žiaden argument.

Niektorí programátori používajú i variant:

```
void main()
```

Avšak tento variant, ktorý hovorí, že funkcia nevracia žiadnu hodnotu, nie je definovaný v norme jazyka C++ a preto na niektorých systémoch nemusí fungovať. Pre tých, ktorým sa nechcem písať príkaz `return` na konci funkcie `main` ISO norma C++ definuje, že ak kompilátor narazí na koniec funkcie `main` bez príkazu `return`, znamená to to isté ako `return 0`; Platí to však iba pre funkciu `main()`.

Funkcia `main()` je vstupnou funkciou programu. Pokiaľ nedefinujete funkciu s takýmto menom, nemáme kompletný program a kompilátor nás na to upozorní.

C++ komentáre

Dvojité lomítko uvádza C++ komentár. Komentár je poznámka programátora pre čitateľa, ktorá zvyčajne identifikuje časť programu alebo vysvetľuje nejaký aspekt kódu. Kompilátor komentáre ignoruje. C++ poznámka začína od `//` a končí koncom riadku. Poznámka môže byť na svojom vlastnom riadku alebo v tom istom riadku ako kód. Komentáre sa používajú na dokumentovanie programu. Čím zložitejší program, tým cennejšie sú komentáre. Pomáhajú pochopiť čo kód robí nielen iným, ale i samotnému programátorovi.

C komentáre

C++ pozná i C komentáre, ktoré sa uzatvárajú medzi symboly `/*` a `*/`:

```
#include <iostream> /* a C-komentár */
```

Pretože C komentár je ukončený znakmi `*/` a nie koncom riadku, môžu byť písané vo viacerých riadkoch. Norma jazyka C99 pridáva `//` komentár i do jazyka C.

C++ preprocesor a iostream súbor

Ak program potrebuje využívať bežný C++ vstup/výstup, musí obsahovať nasledujúce riadky (C++98):

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

(V prípade C programu to je: `#include <stdio.h>`)

C++, podobne ako C, používa preprocesor. Je to program, ktorý spracováva zdrojový súbor pred vykonaním samotnej kompilácie. Na vyvolanie preprocesora nie je potrebné nič robiť, automaticky sa aktivuje pri kompilácii programu. Direktívy preprocesora začínajú znakom `#`.

Direktíva

```
#include <iostream>
```

Spôsobí, že preprocesor pridá obsah súboru `iostream` k programu. To je typická činnosť preprocesora: pridanie alebo náhrada textu v zdrojovom kóde pred kompiláciou.

Prečo by sa mal pridať obsah súboru `iostream` k programu? C++ vstup/výstup schéma obsahuje niekoľko definícií, ktoré sa nachádzajú v súbore `iostream`. Prvý program potrebuje tieto definície, aby mohol využívať prostriedky `cout` na zobrazenie správy. Direktíva `#include` spôsobí, že obsah súboru `iostream` sa odošle spolu s obsahom zdrojového súboru do kompilátora. V podstate obsah súboru `iostream` nahradí riadok `#include <iostream>` v zdrojovom programe.

Hlavičkové súbory

Súbory ako je `iostream` sa nazývajú **včleňované súbory** (pretože sa včleňujú do iných súborov) alebo **hlavičkové súbory** (pretože sa včleňujú na začiatku súboru). C++ kompilátor dodáva mnoho hlavičkových súborov, pričom každý obstaráva osobitnú skupinu vlastností. Tradične, z jazyka C, hlavičkové súbory používajú pre hlavičkové súbory príponu `h`, aby sa jednoduchým spôsobom identifikoval typ súboru. Napríklad hlavičkový súbor `math.h` obstaráva rozmanité C matematické funkcie. Spočiatku to prevzal i jazyk C++. Napríklad hlavičkový súbor, podporujúci vstup a výstup bol pomenovaný `iostream.h`. Ale neskôr sa spôsob pomenovania v C++ zmenil. V súčasnosti je prípona `h` rezervovaná pre staré C hlavičkové súbory (ktoré C++ môže stále používať), zatiaľ čo C++ hlavičkové súbory sú bez prípony. Existujú však i C hlavičkové súbory, ktoré boli skonvertované do C++ hlavičkových súborov. Tieto súbory boli premenované tak, že sa vypustila prípona `h` a pridala sa k menu súboru prípona `c` (indikujúca, že je to z C). Napríklad C++ verzia hlavičkového súboru `math.h` je `cmath`. Niekedy sú C a C++ verzie hlavičkových súborov identické, ale v niektorých prípadoch sa líšia.

Tabuľka 2 Konvencie pomenovania hlavičkového súboru

Druh hlavičky	Konvencia	Príklad	Poznámka
C++ starý štýl	Končí príponou .h	iostream.h	Použiteľné v C++ programoch
C starý štýl	Končí príponou .h	math.h	Použiteľné v C a C++ programoch
C++ nový štýl	Bez prípony	iostream	Použiteľné v C++ programoch, používa namespace std
Konvertované C	C prefix, bez prípony	cmath	Použiteľné v C++ programoch, môžu používať nie-C vlastnosti

Namespace (menopriestor)

Ak použijeme súbor `iostream`, musíme použiť nasledujúcu direktívu, aby program mohol používať definície z tohto súboru:

```
using namespace std;
```

Toto sa nazýva using direktíva. Menopriestory sú určené na zjednodušenie písania rozsiahlych programov a programov, ktoré kombinujú existujúci kód od rôznych tvorcov. V tomto duchu triedy, funkcie a premenné, ktoré sú štandardnou súčasťou C++ kompilátorov, sú umiestnené v menopriestore `std`. Znamená to, že premenná `cout` a `endl` sú v skutočnosti `std::cout` a `std::endl`. Direktíva `using` sprístupňuje všetky názvy v menopriestore `std`.

C++ výstup s cout

Na zobrazenie oznamu používa C++ program príkaz:

```
cout << "Ahoj C++.";
```

Časť uzatvorená v úvodzovkách je tlačенý oznam. V C++ sa každá postupnosť znakov, uzatvorená v úvodzovkách nazýva znakový reťazec. Notácia `<<` indikuje, že príkaz posiela reťazec do `cout`. Čo je `cout`? Je to preddefinovaný objekt, ktorý dokáže zobrazovať rôzne veci, vrátane reťazcov, čísel a jednotlivých znakov.

Prvý pohľad na preťažovanie operátorov

Tí čo už programovali v jazyku C si všimli, že operátor vkladania (`<<`) vypadá ako operátor bitového posunu doľava. Toto je príklad preťaženia operátora, ktoré umožňuje aby ten istý symbol operátora mal rôzne významy. Ktorý význam sa použije zistí kompilátor z kontextu.

Manipulátor endl

Ďalší príkaz programu je:

```
cout << endl;
```

endl je špeciálna C++ notácia, ktorá reprezentuje pojem začiatku riadku. Vloženie endl do výstupného prúdu spôsobí, že obrazovkový kurzor sa presunie na začiatok riadku. Špeciálne notácie ako je endl, ktoré majú osobitný význam pre cout sa prezývajú manipulátory.

Znak nového riadku

C++ má ešte ďalší, starší spôsob, ako indikovať výstup nového riadku – C notáciu \n. Rozdiel je v tom, že endl zaručí odoslanie výstupu na obrazovku pred vykonaním ďalšieho príkazu.

Formátovanie kódu

Niektoré jazyky sú riadkovo orientované (FORTRAN), kde na jednom riadku môže byť jeden príkaz. V C++ však bodkočiarka označuje koniec príkazu. Znamená to, že jeden príkaz môže byť v vo viacerých riadkoch alebo naopak v jednom riadku viac príkazov. Hoci C++ poskytuje slobodu formátovania, programy budú čitateľnejšie, ak budú dodržiavať praktický štýl:

- Jeden príkaz na riadok
- Otvárajúca zložená zátvorka a uzatvárajúca zložená zátvorka funkcie majú svoj vlastný riadok
- Príkazy funkcie sú odsadené
- Medzi menom funkcie a zátvorkami žiadne medzery

Dodržiavanie prvých troch pravidiel má jednoduchý účel udržiavať kód čistý a čitateľný. Posledné pravidlo pomáha odlíšiť funkcie od niektorých zabudovaných C++ štruktúr (napríklad cykly).

C++ príkazy

C++ program je kolekcia funkcií a každá funkcia je kolekcia príkazov. C++ obsahuje niekoľko druhov príkazov.

```
#include <iostream>
```

```
int main()
{
    using namespace std;

    int koleso;    // deklarácia, definícia
    koleso = 4;    // priradenie
    koleso = koleso - 2;    // modifikácia premennej
    cout << koleso;
    return 0;
}
```

Deklaračné príkazy a premenné

Počítače sú presné, systematické stroje. Ak chceme do počítača uložiť informáciu, musíme určiť miesto v pamäti a súčasne koľko priestoru v pamäti informácia vyžaduje. Jediným spôsobom ako to v C++ urobiť, je použiť deklaračný (definičný) príkaz, ktorý udáva typ ukladanej informácie a súčasne i túto pridelenú pamäť pomenúva. Napríklad príkaz:

```
int koleso;
```

konkrétne, že program požaduje pamäťový priestor na uchovanie celého čísla (označované v C++ ako int). O detaily alokácie sa postará kompilátor. C++ pracuje s niekoľkými typmi dát a typ int patrí medzi základné dátové typy.

Pomenovanie pamäťového miesta je druhou úlohou príkazu. V tomto prípade deklaračný príkaz deklaruje, že odteraz bude program používať meno koleso na identifikáciu hodnoty, uloženej v pridelenom pamäťovom mieste. Koleso nazývame premenná, pretože môže meniť svoju hodnotu. V C++ musíme deklarovať všetky premenné. Ak by sme deklaráciu vynechali, kompilátor by hlásil chybu pri ďalšom použití premennej koleso.

Vo všeobecnosti deklarácia indikuje typ ukladaných dát a meno, ktoré program použije pre tieto uložené dáta. Uvedený príkaz sa nazýva definujúci deklaračný príkaz, skrátene definícia. Znamená to, že prikáže kompilátoru alokovať pamäťový priestor pre premennú. V zložitejších situáciách môžeme používať referenčné deklarácie, ktorá počítaču oznamuje, že môže používať premennú, ktorá už bola definovaná niekde inde. Vo všeobecnosti deklarácia nemusí byť aj definíciou.

Flexibilné deklarácie

Deklarácie premenných v mnohých jazykoch musia byť na začiatku funkcie alebo procedúry. Avšak C++ (a aj C99) nemá takéto obmedzenie. Premennú môžete deklarovať bezprostredne pred jej použitím. Na druhej strane nevýhodou je, že nemusíme hneď vedieť, aké premenné funkcia používa.

Prečo musíme premenné deklarovať

Niektoré jazyky (BASIC) vytvárajú nové premenné vždy, keď použijeme nové meno bez nutnosti explicitnej deklarácie. Toto sa zdá byť priateľskejšie pre používateľa, avšak iba zdanlivo. Ak nepozorne skomolíme meno premennej, nechtiac vytvoríme novú premennú.

Priradovacie príkazy

Priradovací príkaz priraduje hodnotu do pamäťového miesta. Napríklad uvedený príkaz priraduje hodnotu 4 do pamäťového miesta reprezentovaného premennou koleso.

```
koleso = 4;
```

Operátor = sa nazýva priradovací operátor. Priradovací operátor môžeme v C++ i C reťaziť, t.j.

```
int pocet;  
int hodnota;  
int celkom;  
pocet = hodnota = celkom = 0;  
Príkaz sa vyhodnocuje sprava doľava.
```

Druhý príkaz demonštruje ako môžeme meniť hodnotu premennej:

```
koleso = koleso - 2;
```

Výraz vpravo od priraďovacieho operátora je príkladom aritmetického výrazu. Počítač odpočíta 2 od 4, čím hodnota premennej `koleso` bude 2.

Funkcie

Funkcie sú základné moduly, z ktorých je C++ program vybudovaný. C++ funkcie rozdeľujeme do dvoch typov:

- S návratovou hodnotou
- Bez návratovej hodnoty

Funkcia s návratovou hodnotou

Funkcia, ktorá má návratovú hodnotu, vytvára hodnotu, ktorú môžeme priradiť do premennej alebo použiť v inom výraze. Napríklad štandardná C/C++ knižnica obsahuje funkciu, nazývanú `sqrt`, ktorá vracia druhú odmocninu čísla. Môžeme napísať nasledujúci príkaz:

```
x = sqrt(6.25);
```

Výraz `sqrt(6.25);` aktivuje (volá) funkciu `sqrt()`. Výraz `sqrt(6.25);` označujeme ako volanie funkcie, aktivovaná funkcia sa nazýva volaná funkcia a funkcia, obsahujúca volanie funkcie sa označuje ako volajúca funkcia.

Hodnota v okrúhlych zátvorkách (6.25) je informácia, ktorá sa posiela do funkcie; hovoríme, že sa prenáša do funkcie. Hodnota, ktorá sa posiela do funkcie takýmto spôsobom, sa nazýva argument alebo parameter. Funkcia `sqrt()` vypočíta odpoveď a pošle ju späť do volajúcej funkcie. Hodnota, posielená naspäť sa nazýva návratová hodnota funkcie. Návratovú hodnotu funkcie si môžeme predstaviť ako niečo, čo nahradí volanie funkcie v príkaze, keď funkcia ukončí svoju prácu. Takže argument je informácia odosielaná do funkcie a návratová hodnota je hodnota, posielená naspäť z funkcie.

Prototyp funkcie

Skôr ako C++ kompilátor funkciu použije, musí vedieť aké typy argumentov používa a aký je typ návratovej hodnoty. Ak jej chýbajú tieto informácie, kompilátor nevie ako má interpretovať zadané hodnoty. C++ oznamuje tieto informácie prostredníctvom príkazu prototypu funkcie.

Prototyp funkcie robí pre funkcie to, čo deklarácia pre premenné: udáva o aké typy sa jedná. Napríklad prototyp funkcie pre `sqrt()` vypadá nasledovne:

```
double sqrt(double); // prototyp funkcie
```

Počiatočné `double` znamená, že `sqrt()` vracia hodnotu typu `double`. `double` v zátvorkách znamená, že `sqrt()` vyžaduje argument typu `double`.

Bodkočiarka na konci prototypu identifikuje, že sa jedná o príkaz, predstavujúci prototyp funkcie a nie hlavičku funkcie. AK by sme bodkočiarku vynechali, kompilátor by riadok interpretoval ako hlavičku funkcie a očakával by, že bude nasledovať telo funkcie, ktoré bude funkciu definovať.

Ak v programe použijeme `sqrt()`, musíme dodať i prototyp. Môžeme to spraviť dvomi spôsobmi:

- Prototyp funkcie napíšeme do zdrojového kódu sami
- Včleníme do zdrojového kódu hlavičkový súbor `cmath` (`math.h`), ktorý tento prototyp obsahuje.

Druhý spôsob je lepší, pretože je pravdepodobnejšie, že obsahuje správny tvar prototypu. Každá funkcia C++ knižnice má prototyp v jednom alebo viacerých hlavičkových súboroch.

Nezamieňajme si prototyp funkcie s definíciou funkcie. Prototyp iba popisuje rozhranie funkcie. Definícia obsahuje kód funkcie.

Prototyp funkcie musíme umiestniť pred prvé použitie funkcie. Zvyčajne sa prototypy umiestňujú pred definíciu funkcie `main`:

```
#include <iostream>
#include <cmath> // alebo math.h

int main()
{
    side = sqrt(area);
    return 0;
}
```

Knižničné funkcie

C++ knižničné funkcie sú uložené v knižničných súboroch. Keď kompilátor kompiluje program, musí prehľadávať knižničné súbory aby našiel funkcie, ktoré sú v programe použité. Jednotlivé kompilátory sa líšia v tom, ktoré knižničné súbory sa prehľadávajú automaticky. Ak sa nejaká knižnica neprehľadáva automaticky, musí byť k programu pripojená explicitne (`-lm`).

Niektoré funkcie vyžadujú viac ako jednu informáciu. Tieto funkcie používajú viac argumentov, oddelených čiarkami. Napríklad prototyp funkcie `pow` je nasledujúci:

```
double pow(double, double);
```

a volanie vypadá nasledovne:

```
vysledok = pow(5.0, 8.0);
```

Niektoré funkcie nevyžadujú žiaden argument. Napríklad funkcia `rand()` má prototyp:

```
int rand(void);
```

Slovo `void` explicitne indikuje, že funkcia nemá žiaden argument. V C++ môžeme slovo `void` vynechať.

Volanie funkcie je nasledovné:

```
nahodneCislo = rand();
```

Všimnime si, že na volanie funkcie musíme použiť zátvorky aj keď funkcia nemá žiaden parameter.

Existujú i funkcie, ktoré nevracajú žiadnu hodnotu. Sú to funkcie, ktoré niečo vykonajú a nie je nutné aby volajúcej funkcii vracali nejaký výsledok. Toto zapisujeme v prototyp funkcii pomocou kľúčového slova `void`.

```
void fun(double);
```

Pretože `fun` nevracia žiadnu hodnotu, nemôžeme ju použiť ako súčasť priradovacieho príkazu alebo nejakého iného výrazu. Môžeme iba použiť príkaz volania funkcie:

```
fun(10.3);
```

V niektorých jazykoch sa funkcie, ktoré nevracajú hodnotu, nazývajú procedúry alebo podprogramy. V C++ sa pojem funkcia používa pre obidva typy.

Funkcie definované programátorom

Štandardná knižnica C jazyka poskytuje viac ako 140 preddefinovaných funkcií. Ak použitie takejto funkcie vyhovuje a postačuje, treba ju použiť a nevytvárať vlastnú.

Programovanie však spočíva vo vytváraní vlastných funkcií. Jednu vlastnú funkciu sme už definovali `main()`. Každý C++ program musí obsahovať `main()` funkciu, definovanú programátorom. Postup použitia vlastnej funkcie je podobný knižničným funkciám. Funkciu voláme menom, pred použitím funkcie musíme zabezpečiť prototyp funkcie a navyše musíme definovať telo funkcie.

```
void mojaFun(int); // prototyp funkcie

int main()
{
    mojaFun(3); // volanie funkcie
    return 0;
}

void mojaFun(int n) // definícia funkcie
{
    // ... kód
}
```

Jazyk C i C++ nedovoľuje vnorené definície funkcie, t.j. definíciu funkcie v tele inej funkcie. Všetky funkcie sú na jednej úrovni.

Kľúčové slová

Kľúčové slová tvoria slovník počítačového jazyka. Pretože C++ kľúčové slová majú špeciálnu úlohu v C++, nemôžeme ich použiť na iné účely.

Konvencie pomenovania

Spôsob pomenovania funkcií závisí od tímu. Mal by byť však jednotný.

Práca s dátami

Podstatou objektovo-orientovaného programovania je návrh a rozširovanie vlastných dátových typov. Návrh vlastných dátových typov reprezentuje úsilie nájsť typovú zhodu s dátami. Ak typy navrhujeme správne, tým jednoduchšie sa bude potom s dátami pracovať. Ale skôr ako začneme navrhovať vlastné dátové typy, potrebujeme poznať typy, ktoré sú zabudované do C++, pretože tieto typy budú našimi stavebnými blokmi.

C++ rozlišuje dve skupiny zabudovaných typov: základné typy a zložené typy. Základné typy reprezentujú celé a reálne čísla. Znie to ako iba dva typy, avšak C++ pripúšťa, že iba jeden celočíselný typ a jeden reálny typ nebude stačiť na všetky programátorské požiadavky, takže poskytuje niekoľko variant týchto dát. Zložené typy sú vybudované zo základných typov a zaraďujeme sem polia, reťazce, smerníky a štruktúry.

Samozrejme program potrebuje nástroj na identifikáciu uložených dát. Jednou z metód je použitie premenných.

Jednoduché premenné

Program potrebuje uchovávať informácie. Aby program mohol uchovávať informáciu v počítači, musí sledovať tri základné vlastnosti:

- Kde je informácia uložená
- Akú hodnotu uchováva
- O aký druh informácie sa jedná

Predpokladajme napríklad takéto príkazy:

```
int pocet;  
pocet = 5;
```

Tieto príkazy prikazujú programu, že má uchovávať celé číslo a meno pocet reprezentuje celočíselnú hodnotu, v našom prípade 5. V podstate program vymedzí kus pamäti, postačujúci na uchovanie celého čísla, označí miesto a nakopíruje na toto miesto hodnotu 5. Tieto príkazy programátorovi nehovoria, kde v pamäti je hodnota uložená, ale program si túto informáciu sleduje. Na získanie pamäťovej adresy premennej pocet môžeme použiť operátor &.

Názvy premenných

C++ vyzýva a podporuje použitie zmysluplných názvov pre premenné. Ak premenná reprezentuje počet študentov, mala by sa nazývať pocet_studentov alebo pocetStudentov a nie x alebo y. Pri pomenovaní by sme mali dodržiavať niekoľko nasledujúcich jednoduchých pravidiel:

- V menách by sme mali používať písmena abecedy, číslice a podtržítka (_).
- Prvým znakom mena nesmie byť číslica.
- Malé a veľké písmena sa rozlišujú.
- Ako názov nemôžeme použiť kľúčové slovo jazyka C++.

- Názvy, začínajúce podtržítom alebo dvomi podtržítkami sú rezervované pre implementačné použitie, t.j. kompilátorom a prostriedkami, ktoré používa. Názvy, začínajúce jedným podtržítom sú rezervované pre globálne identifikátory, definované v implementácii kompilátora.
- C++ neohraničuje dĺžku názvu a všetky znaky mena sú významné. Avšak niektoré platformy môžu mať svoje vlastné limity.

ANSI C99 sa líši od C++ tým, že C jazyk garantuje len 63 znakov mena je významných, t.j. dve mená, ktorých prvých 63 znakov sa zhoduje, sa považujú za identické.

```
int pes;
int Pes;
int PES;
```

Ak chceme meno vytvoriť z dvoch alebo viacerých slov, zvyčajne sa každé slovo oddeľuje podtržítkom alebo prvé písmeno každého slova okrem prvého je veľké. Niektorí programátori vkladajú do názvu premennej doplňujúcu informáciu – prefix, ktorý indikuje typ alebo obsah premennej. Napríklad npocet, kde písmeno n indikuje celočíselnú hodnotu (maďarská notácia).

Celočíselné typy

Celé čísla sú čísla bez zlomkovej časti, avšak žiadna pamäť počítača nedokáže reprezentovať všetky celé čísla, počítač dokáže reprezentovať iba podmnožinu všetkých celých čísel. C++ poskytuje niekoľko typov celých čísel, čo nám dáva možnosť vybrať si taký celočíselný typ, ktorý spĺňa konkrétne požiadavky programu.

Rozmanité C++ celočíselné typy sa líšia v rozsahu pamäti, ktorú používajú na uchovanie celého čísla. Väčší blok pamäti môže reprezentovať väčší interval celých čísel. Niektoré typy (znamienkové) dokážu reprezentovať kladné i záporné čísla. Rozsah pamäti použitej pre celé číslo sa označuje ako šírka. Čím viac pamäti hodnota používa, tým je širšia.

- C++ (C) pozná nasledujúce celočíselné typy (v poradí rastúcej šírky): `char`, `short`, `int`, `long` a C++11 `long long`. Každý typ má znamienkovú a bezznamienkovú verziu.

Celočíselné typy `short`, `int`, `long` a `long long`

Počítačová pamäť sa skladá z bitov. Použitím rôzneho počtu bitov na uchovanie hodnoty C++ typy `short`, `int`, `long` a `long long` môže reprezentovať až štyri rôzne šírky celého čísla. Bolo by výhodné, keby každý typ mal vždy nejakú konkrétnu šírku vo všetkých systémoch, napríklad `short` by bolo vždy 16 bitov, `int` zasa 32 bitov, atď. Avšak život nie je taký jednoduchý. Nie každá možnosť je vhodná pre všetky koncepcie počítačov. C++ ponúka flexibilnú normu, ktorá zaručuje minimálne rozsahy, ktoré preberá z jazyka C:

- `short` má minimálnu šírku 16 bitov
- `int` je minimálne taký veľký ako `short`
- `long` má minimálnu šírku 32 bitov a je minimálne taký veľký ako `int`
- `long long` má minimálnu šírku 64 bitov a je minimálne taký veľký ako `long`

Bity a byty

Základnou počítačovou jednotkou je bit. Byte je zvyčajne reprezentovaný ako 8-bitová jednotka pamäti. Avšak C++ definuje byte odlišne. C++ byte pozostáva z najmenej toľkých bitov, koľko je potrebné na uloženie základnej implementačnej znakovnej množiny.

Takéto definície typov umožňujú rôznym implementáciám C++ použiť rôzne šírky pre rovnaké typy, čo môže robiť problémy pri prenose programu z jedného prostredia do iného.

Na zisťovanie šírky môžeme použiť operátor `sizeof`, ktorý môžeme použiť s typom alebo menom premennej:

```
int pocet;

cout << "int je " << sizeof (int) << " bytov.\n";
cout << "short je " << sizeof pocet << " bytov.\n";
```

Pri použití názvu premennej nemusíme písať zátvorky.

Symbolické konštanty – preprocesorový spôsob

Hlavičkový súbor `climits` (`limits.h`) obsahuje definície symbolických konštánt, ktoré obsahujú hraničné hodnoty jednotlivých typov. `INT_MAX` obsahuje najväčšiu hodnotu typu `int`.

Konštanty sú definované nasledovným spôsobom:

```
#define INT_MAX 2147483647
```

`#define` je direktíva preprocesora, ktorá prikazuje preprocesoru asi nasledovné: vyhľadaj v programe všetky výskyty `INT_MAX` a každý výskyt nahraď `2147483647`. Túto direktívu môžeme používať i na definovanie vlastných symbolických konštánt. Avšak `#define` je prežitok z C jazyka. C++ má lepší spôsob ako vytvárať symbolické konštanty (použitím kľúčového slova `const`).

Inicializácia

Inicializácia spája priradenie z deklaráciou. Napríklad nasledujúci príkaz deklaruje premennú `pocet` a nastavuje ju na maximálnu hodnotu typu `int`:

```
int pocet = INT_MAX;
```

Tento spôsob inicializácie je prevzatý z jazyka C. C++ zavádza i ďalší spôsob inicializácie (rýchlejší):

```
int pocet (INT_MAX);
```

Ak neinicializujeme premennú definovanú vo vnútri funkcie, premenná neurčená, t.j. obsahuje hodnotu, ktorá bola v pamäti pred vytvorením premennej.

Inicializácia C++11

Existuje ešte ďalší formát inicializácie, ktorá sa používa pre polia a štruktúry, ale možno ju použiť pre jednoduchotové premenné:

```
int pocet = {24};
```

Použitie inicializácie pomocou zložených zátvoriek na inicializáciu jednoduchotovej premennej nie príliš bežné, ale C++11 norma ju rozširuje.

Predovšetkým môžeme ju používať bez znamienka =:

```
int pocet {24};
```

Ďalej zátvorky môžeme nechať prázdne, čím premennú inicializujeme na 0:

```
int pocet {};
```

Po tretie, poskytuje väčšiu ochranu pred chybami konverzie typov (neskôr).

Čo je dôvodom toľkých rôznych spôsobov inicializácie? Akokoľvek neuveriteľné sa to môže zdať, dôvodom je zjednodušenie zvládnutia C++ pre nováčikov. V minulosti mal jazyk C++ rôzne formy inicializácie pre rôzne typy a forma inicializácie triednych premenných (objektov) bola odlišná od formy, používanej pre obyčajné štruktúry, a tá bola odlišná od formy, používanej pre jednoduché premenné. C++ pridalo zátvorkovú formu inicializácie aby sa inicializácie obyčajných premenných vykonávali ako inicializácie triednych premenných. C++11 umožňuje používať syntax so zloženými zátvorkami pre všetky typy – univerzálna inicializačná syntax. Zvyšné spôsoby inicializácie zostali pre zachovanie spätnej kompatibility.

Bezznamienkové (unsigned) typy

Každý z uvedených štyroch typov má i svoju unsigned variantu, ktorá nedokáže uchovávať záporné hodnoty. Naopak však výhodou je zvýšenie maximálnej hodnoty, ktorú môže premenná uchovávať. Samozrejme, unsigned typy môžeme používať iba pre veličiny, ktoré nie sú nikdy záporné. Na vytváranie unsigned verzii základných celočíselných typov sa používa kľúčové slovo unsigned:

```
unsigned short malyPocet;  
unsigned int vacsiPocet;  
unsigned vacsiPocet2;  
unsigned long velkyPocet;  
unsigned long long najvacsiPocet;
```

Celočíselné konštanty

Celočíselná konštanta je konštanta, ktorú môžeme zapísať explicitne, napríklad 212, 1176. C++ ako i C dovoľuje písať celé čísla v troch číselných základoch: základ 10 (favorit), základ 8 (starší Unix favorit), základ 16 (hardvérový favorit). Na identifikáciu základu C++ používa prvú alebo prvé dve číslice číselnej konštanty. Ak je prvá číslica v interval 1-9, tak číslo je so základom 10. Ak je prvá číslica 0, a druhá v rozsahu 1-7, číslo je so základom 8 (042). Ak sú prvé dva znaky čísla 0x alebo 0X, číslo je

hexadecimálne. Písmena a-f alebo A-F predstavujú hexadecimálne číslce zodpovedajúce hodnotám 10-15.

Ako sa C++ rozhoduje o aký typ konštanty sa jedná

Deklarácie oznamujú programu typ konkrétnej celočíselnej premennej. Ale ako je to s konštantami? Predpokladajme, že v programe použijeme konštantu:

```
cout << "Rok = " << 2014 << "\n";
```

Ako sa uloží 2014, `long` alebo `int`, alebo ako nejaký iný typ? Odpoveď znie, že C++ uchováva celočíselné konštanty ako typ `int`, pokiaľ nie je dôvod urobiť to inak. Existujú dva dôvody – použitie prípony, ktorá indikuje konkrétny typ alebo hodnota je príliš veľká na typ `int`.

Prípony sú písmena, pridávané na koniec numerickej konštanty aby indikovali typ. Prípona `l` alebo `L` znamená, že celočíselná konštantá je typu `long`, prípona `u` alebo `U` indikuje konštantu typu `unsigned int` a `ul` (ľubovoľná kombinácia poradia a malých a veľkých písmen) indikuje konštantu typu `unsigned long`. C++11 poskytuje prípony `ll` a `LL` pre typ `long long` a `ull`, `Ull`, `uLL` a `ULL` pre typ `unsigned long long`.

Typ char: znaky a malé celé čísla

Typ `char` je najmenší celočíselný typ. Ako už naznačuje názov, tento typ je určený na uchovávanie znakov, t.j. písmen a čísl. Zatiaľ čo uchovávanie čísel nie je pre počítač zložité, avšak uchovávanie písmen je niečo iné. Programovacie jazyky sa s tým vysporiadávajú jednoducho tak, že pre písmena používajú kódy. Takže typ `char` je ďalší celočíselný typ. Je zaručené, že je dostatočne veľký, aby reprezentoval celý interval základných symbolov – všetky písmena, číslce, interpunkcia, atď. – cieľového počítačového systému.

Najpoužívanejšou množinou symbolov v USA je ASCII znaková množina. Číselný kód (ASCII kód) reprezentuje každý znak množiny. Napríklad 65 je kód písmena A.

```
int main()
{
    char ch = 'A';
    cout << ch << endl;
    return 0;
}
```

Skutočnosť, že programovací jazyk, reprezentuje ako celé čísla, má svoje výhody najmä pri konverziách.

Existujú znaky, ktoré sa nedajú priamo zadať do programu z klávesnice (napr. nový riadok). Ďalšie znaky sa nedajú zadať, pretože ich jazyk používa na špeciálne účely. Napríklad úvodzovky (“) ohraničujú reťazec znakov. C++ používa špeciálnu notáciu, nazývanú escape postupnosti pre tieto znaky. Napríklad `\n` predstavuje znak nového riadku. Postupnosť “`\`” predstavuje úvodzovky ako znak a nie ako oddeľovač.

Názov znaku	ASCII symbol	C++ kód	Desiatkový/Hexa kód
Nový riadok	NL (LF)	<code>\n</code>	10/0xA

Horizontálny tabelátor	HT	\t	9/0x9
Vertikálny tabelátor	VT	\v	11/0xB
Krok späť	BS	\b	8/0x8
Návrat vozíka	CR	\r	13/0xD
Výstraha	BEL	\a	7/0x7
Opačné lomítko	\	\\	92/0x5C
Otáznik	?	\?	63/0x3F
Jednoduchá úvodzovka	'	\'	39/0x27
Dvojitá úvodzovka	"	\"	34/0x22

signed char a unsigned char

Na rozdiel od int, typ char nie je signed. Ani nie je ani unsigned. Výber je ponechaný na C++ implementácii aby mohol tvorca kompilátora využiť vlastnosti hardvéru. Samozrejme môžeme používať typy signed char a unsigned char ako číselné typy.

Typ wchar_t

Programy môžu pracovať so znakovými množinami, pre ktoré nestačí jeden 8-bitový byte. C++ rieši tento problém niekoľkými spôsobmi.

- Ak je znaková množina základnou množinou implementácie, môže byť typ char definovaný ako 16-bitový byte alebo väčší.
- Implementácia môže podporovať obidve znakové množiny. Typ char reprezentuje základnú znakovú množinu a ďalší typ wchar_t rozšírenú znakovú množinu. Typ wchar_t je celočíselný typ s postačujúcim priestorom na reprezentovanie najväčšej rozširujúcej znakovej množiny systému.

Na indikovanie rozšíreného znaku sa používa písmeno L.

```
wchar_t pismeno = L'A';
```

Nové typy C++11: char16_t a char32_t

Veľkosť wchar_t sa môže líšiť v závislosti od implementácie C++. A tak C++11 zavádza typy char16_t, ktorého rozsah je 16 bitov a je unsigned. Podobne char32_t je 32 bitové a tiež unsigned.

Na indikovanie rozšíreného znaku typu char16_t sa používa písmeno u.

```
char16_t pismeno = u'A';
```

Na indikovanie rozšíreného znaku typu char32_t sa používa písmeno U.

```
char32_t pismeno = U'A';
```

Typ bool

ANSI/ISO C++ norma doplnila nový typ, nazývaný bool. Hodnoty premennej tohto typu môžu nadobúdať hodnoty true alebo false. Každá nenulová hodnota sa interpretuje ako true a nula ako false.

Kvalifikátor const

Vráťme sa k téme symbolických názvov konštánt. Symbolický názov konštanty môže naznačovať význam konštanty. Zároveň ak program používa konštantu na viacerých miestach a potrebujeme zmeniť jej hodnotu, stačí to spraviť iba raz v definícii.

Ako sme už spomínali pri príkaze #define, C++ má lepší spôsob ako definovať symbolické konštanty. Tým spôsobom je použitie kľúčového slova const, ktorým modifikujeme deklaráciu a inicializáciu premennej. Napríklad:

```
const int Mesiace = 12; // Mesiace je symbolická konštanta s hodnotou 12
```

Odteraz môžeme v programe používať Mesiace namiesto 12. Po inicializácii konštanty kompilátor už nedovolí zmeniť jej hodnotu. Inicializácia konštanty musí byť súčasťou definície. Kľúčové slovo const sa nazýva kvalifikátor, pretože kvalifikuje význam deklarácie.

Prečo je const lepšie?

- Dovoľuje explicitne definovať typ.
- Použitím pravidiel rozsahu môžeme konštanty ohraničiť na konkrétne funkcie alebo súbory.
- const môžeme použiť aj pre zložitejšie typy ako sú polia a štruktúry.

ANSI C tiež dovoľuje používať const kvalifikátor, ktorý prevzalo z C++. Avšak C++ verzia je trochu iná. Jeden rozdiel sa týka pravidiel rozsahu. Hlavný rozdiel však spočíva v tom, že C++ dovoľuje const hodnotu použiť na deklarovanie rozsahu poľa.

Čísla s pohyblivou rádovou bodkou

Čísla v pohyblivej rádovej čiarkke predstavujú druhú hlavnú skupinu základných C++ typov. Tieto čísla nám umožňujú reprezentovať čísla so zlomkovou časťou. Tento typ poskytuje omnoho väčší rozsah hodnôt.

Počítač uchováva tieto hodnoty v dvoch častiach. Jedna časť predstavuje hodnotu a druhá časť zväčšuje alebo zmenšuje hodnotu. Súčiniteľ veľkosti slúžia na posun desatinnej bodky, a preto pojem pohyblivá rádová čiarka (bodka).

Zápis čísel v pohyblivej rádovej bodke

C++ pozná dva spôsoby zápisu týchto čísel. Prvá metóda je štandardný zápis čísla s desatinnou bodkou:

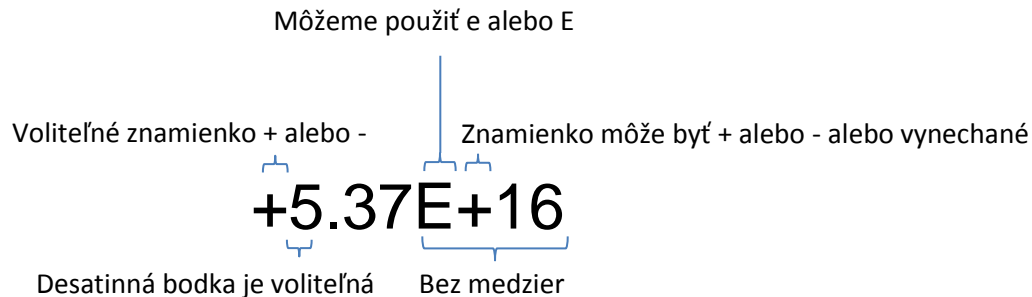
```
12.34 // plávajúce číslo
```

Druhý spôsob sa nazýva E notácia:

3.45E6 // plávajúce číslo

Znamená to, že číslo 3.45 sa vynásobí číslom 1000000. E6 znamená 10 na 6-u. 6 sa nazýva exponent a 3.45 je mantisa.

E notácia je užitočná pre veľmi veľké a veľmi malé čísla.



Obrázok 6 E notácia

Typy pohyblivej rádovej bodky

Podobne ako ANSI C i C++ má tri typy pohyblivej rádovej bodky: `float`, `double` a `long double`. Tieto typy sú určené počtom významných číslic, ktoré dokážu reprezentovať.

V skutočnosti typ `float` vyžaduje 32 bitov, `double` 48 bitov a nie menej ako `float`, `long double` najmenej ako `double`. Všetky tri môžu mať rovnaký rozsah. Typicky však `float` je 32 bitov, `double` je 64 bitov a `long double` je 80, 96 alebo 128 bitov.

Konštanty v pohyblivej rádovej bodke

Keď zapíšeme do programu konštantu v pohyblivej rádovej bodke, uloží sa ako typ `double`. Ak chceme konštantu typu `float`, musíme použiť príponu `f` alebo `F`. Pre typ `long double` sa používa prípona `l` alebo `L`.

```
1.234f           // float konštanta
2.45E20F         // float konštanta
2.345324E28      // double konštanta
2.2L             // long double konštanta
```

Výhody a nevýhody čísel v pohyblivej rádovej bodke

Čísla v pohyblivej rádovej bodke majú v porovnaní s celými číslami dve výhody:

- dokážu reprezentovať čísla medzi celými číslami
- dokážu reprezentovať omnoho väčší rozsah hodnôt

Na druhej strane operácie s týmito číslami sú zvyčajne pomalšie ako operácie s celými číslami a môžeme stratiť presnosť.

Celočíselné typy spolu s typmi v pohyblivej rádovej bodke nazývame aritmetické typy.

C++ aritmetické operátory

C++ má päť základných aritmetických operátorov:

- operátor + pre sčítanie
- operátor – pre odčítanie
- operátor * pre násobenie
- operátor / pre delenie. Ak sú obidva operandy celočíselného typu, výsledkom je celočíselná časť podielu a zlomková časť sa zahodí
- operátor % pre výpočet modula, t.j., zvyšku po delení. Obidva operandy musia byť celočíselného typu

Poradie operácií a priorita operátorov

Aký bude výsledok výrazu:

```
int a = 3 + 4 * 5; // 35 alebo 23
```

Na vyhodnocovanie výrazu C++ aplikuje rovnaké prioritné pravidlá, ako platia v algebre, t.j. najskôr sa násobí a delí a potom sčítava a odčítava. Samozrejme na zmenu pravidiel môžeme použiť zátvorky. V prípade rovnakej úrovne priority sa výraz vyhodnocuje zľava doprava:

```
int a = 120/4 * 5; // 150 alebo 6
```

Ak vo výraze použijeme premenné rovnakého typu, výsledok bude tiež zodpovedať použitému typu. Ak vo výraze použijeme rôzne typy, prekladač aplikuje tzv. automatickú konverziu (pokiaľ je to možné).

Operátor modulo

Operátor modulo vracia zvyšok po delení. V spolupráci s celočíselným delením je vhodný na riešenie problémov, ktoré si vyžadujú delenie hodnoty na rôzne integrované skupiny, napríklad mincovník, cyklický pohyb indexu v poli s zadaným rozmerom, atď.

```
int rozsah = 10;  
int i = 0;  
  
i = (i + 1) % rozsah;
```

Konverzie typov

Množstvo C++ typov umožňuje vybrať si najvhodnejší typ podľa potreby. Na druhej strane však prináša i komplikácie. 11 celočíselných typov a 3 typy pohyblivej rádovej čiarky prinášajú množstvo

kombinácií, ktoré musí počítač riešiť, najmä ak začneme typy miešať. C++ robí mnohé konverzie typov automaticky. Pre konverzie všeobecne platí:

- C++ konvertuje hodnoty keď priraďujeme hodnotu jedného aritmetického typu do premennej iného aritmetického typu.
- C++ konvertuje hodnoty keď vo výraze kombinujeme rôzne typy
- C++ konvertuje hodnoty pri prenose argumentov do funkcií

Automatické konverzie sú síce dobré, ale je potrebné dávať si na pozor a všímať si prípadné varovania prekladača.

Konverzie pri inicializácii a priradovaní

C++ je značne liberálny pri priradovaní číselnej hodnoty jedného typu do premennej iného typu. Zakaždým sa hodnota skonvertuje na typ prijímajúcej premennej.

```
short s;  
long l;  
l = s; // priradenie short do long
```

Program zoberie hodnotu premennej s (16 bitov) a rozšíri ju na hodnotu typu long (32 bitov). Priradenie hodnoty do typu s väčším rozsahom hodnôt zvyčajne nie je problém, hodnota sa nezmení. Avšak priradenie long hodnoty (2111222333) do premennej typu float zvyčajne končí stratou presnosti, pretože float má iba 6 platných miest, takže hodnota sa môže zaokrúhliť na 2.11122E9. Potencionálne problémy, ktoré môžu vzniknúť pri konverzii (platí aj pre tradičnú inicializáciu):

Typ konverzie	Potencionálny problém
Väčší typ pohyblivej rádovej bodky na menší typ pohyblivej rádovej bodky	Strata presnosti (platné číslice) <ul style="list-style-type: none">- hodnota môže byť mimo rozsah cieľového typu (výsledok nedefinovaný)
Typ pohyblivej rádovej bodky na celočíselný typ	<ul style="list-style-type: none">- Strata zlomkovej časti- Pôvodná hodnota môže byť mimo rozsah cieľovej hodnoty (výsledok nedefinovaný)
Väčší celočíselný typ do menšieho celočíselného typu	Pôvodná hodnota môže byť mimo rozsah cieľového typu (zvyčajne sa priradia iba spodné bity hodnoty)

Nulová hodnota priradená do premennej typu bool sa konvertuje na false a nenulová na true. To umožňuje v C++ písať i na pohľad nelogické zápisy podmienených príkazov.