1. lekce

Úvod do jazyka C++

Rozdíly mezi C a C++

Jazyk C++ je pokračovatelem jazyka C.

Hlavní rozdíly:

- **objektově orientovaný jazyk**
- dovoluje **přetěžování funkcí** (více funkcí se může jmenovat stejně, odlišnost je v parametrech)
- má datový typ **boolean** (bool, 1 nebo 0) a **string** (string)
- pro práci se stringem je třeba „includovat“ knihovnu **string**
- nový datový typ **auto**, kompilátor si automaticky zvolí typ proměnné podle hodnoty, po prvním vložení hodnoty datový typ zůstává (nelze ho více změnit)
- přibyly **konstanty** a **reference** (odkazy na proměnné)
- **streamy** (datové toky: vstup a výstup)
- změna v **dynamické alokaci paměti** (new, delete)
- **výjimky** – speciální chybové stavy programu (zachycení chyb – chybný typ proměnné, index mimo hranice, matematická funkce nedefinovaná pro určité hodnoty,...)

Reference

Reference je odkaz na proměnnou, definuje se pomocí & (amprsand).
Na rozdíl od pointeru se v kódu nemusí používat symbol *.

Např.: *int a;*
int &b=a;

Proměnná „a“ i proměnná „b“ pracují se stejným místem (adresou) v paměti – „a“ je originál, „b“ je reference (odkaz) na adresu v paměti, kde je uložená hodnota proměnné „a“.

Jestliže změníme hodnotu proměnné „a“, pak se automaticky změní hodnota proměnné „b“ a naopak. Reference je možné používat i jako výstupní parametry funkcí.

Konstanty

const - klíčové slovo, označuje proměnnou, jejíž hodnota zůstává během funkce konstantní, při opětovném spuštění funkce může mít jinou hodnotu.

Např.: *void funkce(const int &a)*

Do funkce jsme poslali adresu, kterou převezme proměnná „a“ a bude fungovat jako standardní integer (zde konstantní). Hodnota proměnné „a“ není adresa, ale konkrétní číslo!

Po dobu běhu funkce hodnota proměnné „a“ zůstane konstantní!

Při pokusu o změnu hodnoty proměnné „a“ v kódu programu dojde při kompilaci k chybě!

Dynamická alokace paměti

Nově řešíme pomocí příkazu *new*.

Např.:

Pro jednoduchou proměnnou:

*int *a=new int;*

V jazyce C je adekvátní*: int a=(int*)malloc(sizeof (int));*

Pro jednoduchou proměnnou s inicializací *(nastavení počáteční hodnoty)***:**

*int *a=new int(10);*

V jazyce C nemá přímý ekvivalent, bylo by:

```
int a=(int*)malloc(sizeof (int));
*a=10;
```

Pro pole:

*int *a=new int[počet_ prvků];*

V jazyce C je adekvátní*: int a=(int*)malloc(počet_prvků*sizeof (int));*

Uvolňování dynamicky alokované paměti

*Nově řešíme pomocí příkazu **delete**.*

Např.:

Pro způsob alokace

int *a=new int;

bude

delete a;

Pro způsob alokace

int *a=new int[10];

bude

delete[] a;

Poznámka:

Pokud naincludujeme knihovnu *cstdlib (knihovna jazyka C)*, pak můžeme používat i příkazy *malloc, realloc* a *free*, ale platí, co bylo alokováno pomocí *malloc* a *realloc*, musí být uvolněno pomocí *free* a co bylo alokováno pomocí new, musí být uvolněno pomocí *delete*.

Pokud se použije příslušná knihovna jazyka C, můžeme v jazyce C++ používat všechny příkazy patřící do příslušné knihovny jazyka C.

Přetěžování funkcí

Přetěžování funkcí spočívá v tom, že dvě nebo více funkcí mají stejné jméno, ale jiné parametry (počet nebo datový typ nebo oboje). Kompilátor si automaticky zvolí adekvátní funkci podle parametrů, kterými ji voláme.

Příklad 1:

int funkce (int a, int b) { return a+b; }

int funkce (double a, double b) { return a-b; }

...

```
funkce(1,2);      // zavolá první funkci, protože 1 a 2 jsou integery, výsledek je 3.
funkce(1.5,2.3);  // zavolá druhou funkci, protože 1.5 a 2.3 jsou double, výsledek je -0.8,
                  // ale návratová hodnota bude 0 (protože je int!).
funkce(1.5,2);    // toto vyvolá chybu, protože není definovaná funkce s prvním
                  // parametrem double a druhým integer a kompilátor hledá jiné funkce
                  // se stejným jménem a zjišťuje, zda nemůže parametry přetypovat.
                  // Kompilátor umí přetypovat integer na double a double na integer, ale
                  // protože do obou funkcí je potřeba stejný počet konverzí pro obě
                  // funkce nedokáže rozhodnout, kterou funkci má spustit.
funkce('a','b');  // zavolá první funkci, protože 'a' a 'b' jsou chary, které kompilátor umí
                  // přetypovat na integer, ale ne na double.
```

Příklad 2:

int funkce (int a, int b, int c) { return a+b+c; }

int funkce (double a, double b, double c) { return a-b-c; }

funkce(1,2,1.5); *// zavolá první funkci, protože u jejího spuštění je méně konverzí,*
// výsledek bude 4, protože 1.5 je po vložení do integeru rovna 1!

Vstupní a výstupní streamy

- jsou v knihovně *iostream* (knihovny v C++ nemají koncovku „.h“)
- vstupní stream je *cin* (v jazyce C je ekvivalent *scanf*)
- výstupní stream je *cout* (v jazyce C je ekvivalent *printf*)
- cin a cout nemají formátovací řetězce!

Příklad použití

C++	C
int a; *char c;* *cin >> a >> c;*	*int a;* *char c;* *scanf(“%d%c“,&a,&c);*
int a=10; *cout<<”Vysledek je ”<<a<<endl;* *//endl* je konec řádku	*int a=10;* *printf(”Vysledek je %d\n”,a);*

Výjimky

- umožňují ošetření chyb bez fatálního ukončení programu
- výjimka se vyvolá příkazem *throw parametr,* parametr může být jakéhokoli datového typu
- parametr může obsahovat
  - zprávu pro uživatele
  - objekt, ve kterém došlo k chybě s možností následné opravy uživatelem
  - hodnotu k dalšímu zpracování
- výjimky se zachycují pomocí *try* a *catch*
- za *catch* je parametr, který definuje, které výjimky bude příkaz *try* a *catch* zachycovat
  (viz další příklady)

Příklad:

*try {testovaný blok programu} catch(const char * vyjimka) {co se stane při výjimce}*

- tato konstrukce zachytí všechny výjimky vyvolané příkazem *throw "nějaký text";*

```
try
{
      if (a==10) throw "Máš tam chybu!";
}
catch(const char * vyjimka)
{
      cout<<vyjimka;
}
```

*const char * vyjimka* je ukazatel na řetězec charů
(pro konstantní text se vždy používá *const char *)*

V případě, že hodnota proměnné „a" je 10, pak se vyhodí (vytvoří) výjimka (chybový stav určitého datového typu). V našem příkladu se stane, že se pole znaků *(const char *)* obsahující text *"Máš tam chybu!"* vloží do proměnné „vyjimka" a hodnota proměnné vyjimka se zobrazí příkazem cout, to znamená, že se zobrazí věta "Máš tam chybu!"

try { } catch(int vyjimka) { }

- tato konstrukce zachytí všechny výjimky vyvolané příkazem *throw cele_cislo;*

try { } catch(...) { }

- tato konstrukce zachytí zcela všechny výjimky vyvolané příkazem *throw* jakýkoli_datový_typ
- se zachycenou chybou nemůžeme pracovat, protože ji nemáme uloženou v žádné proměnné
- tímto zajistíme, že chod programu neskončí fatální chybou, ale nemůžeme s chybou nijak pracovat

Struktura programu

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

int main (void)
{
}
```

Knihovny:

iostream	- umožňuje vstupy a výstupy
string	- umožňuje práci se stringem (již ne pole charů, ale proměnná typu string)
cstring	- knihovna z jazyka C pro práci s polem charů
cmath	- knihovna jazyka C pro práci s matematickými funkcemi *(ta samá je pro C++)*
ctime	- knihovna pro práci s časem a datem

using namespace std;	definujeme, že pracujeme se standardním namespacem namespace – jmenný prostor, který obsahuje klíčová slova jazyka C++ Pokud tento řádek nepoužijeme, pak se před každou standardní funkci musí napsat *std::klíčové slovo*

Kompilace kódu programu:

g++ -std=c++11 -Wall -pedantic jmeno_souboru.cpp

Přepínače kompilátoru:

-std=c++11	- umožňuje používat funkce ve standardu 2011
-Wall	- výpis všech varování (typicky nepoužitá proměnná)
-pedantic	- další podrobná varování, důkladnější kontrola kódu