X36PJC
7. přednáška

Funkce

Minulá přednáška

- Pole
  - Statická alokace
  - Dynamická alokace
- Ukazatele
  - Na typ const
  - Konstantní ukazatele
- Řetězce ve stylu C
  - Konverze na string

Funkce

- Funkce je programátorem definovaná operace.
- Funkce je základním prostředkem pro modularizaci výpočtu.
- Funkce obvykle vrací nějaký výsledek.
- Funkce má jméno a parametry.

Funkce

- Přehlednost programu.
- Zamezení opakujícího se kódu.
- Program v C:
  - tvořen funkcemi,
  - funkce se navzájem mohou volat,
  - funkci **main je předáno řízení při spuštění.**
- Program v C++:
  - obdoba C, menší význam funkcí, větší význam tříd a metod,
  - zápis metod je podobný zápisu funkcí.

Funkce

Deklarace funkce:

<typ> name (<parametry>);

- Deklarace informuje překladač, že funkce existuje, jak se jmenuje, jaké má parametry a jaký je typ návratové hodnoty.
- Deklarace se používá pro funkce, které jsou implementované "mimo" náš program - např. knihovní funkce, funkce OS, jiný modul programu.
- Deklarace funkcí se většinou umisťují do hlavičkových souborů.

Funkce

Definice funkce:

<typ> name (<parametry>) { <tělo> }

- Definice funkce je deklarace, která navíc nese vlastní implementaci.
- Definice se zpravidla neumisťují do hlavičkových souborů (patří do .c, .cc či .cpp souborů).
- Deklarací téže funkce může v daném programu existovat více, definice právě jedna.
- Definice funkcí nelze vnořovat do sebe (funkce nemůže být definována uvnitř jiné funkce).

Funkce

Parametry funkcí, návratový typ:

- Seznam **<typ> <identifikátor>, oddělený čárkou.**
- Funkce bez parametrů – prázdný seznam nebo (lépe) uvést klíčové slovo **void.**
- Parametry stejného typu – musí se rozepsat.
- Návratový typ – **void znamená proceduru.**
- Vracená hodnota – výraz za **return.**
- Pokud funkce nevrací **void, musí obsahovat return a vracet výraz správného typu.**

Funkce

- Funkci voláme pomocí operátoru *volání funkce,* což je pár závorek ()

```
int first (int a1, int a2) {
    return a1;
}
int a = first(5, 6);
```

Funkce

```
unsigned int gcd ( unsigned int a, unsigned int b )
{
  while ( a != b )
  if ( a > b )
      a -= b;
    else
      b -= a;
   return ( a );
 }

unsigned int scm ( unsigned int a, unsigned int b )
 {
   return ( a * b / gcd ( a, b ) );
 }
```

Předávání parametrů

- Parametry jsou inicializovány stejně jako proměnné.
- Pokud předávaný parametr není reference pak je zkopírován!
- Rozlišujeme tedy předávání parametrů hodnotou a odkazem.

Předávání parametrů - Ukazatelů

- Pokud je předávaným parametrem ukazatel, pak je také zkopírován.

```
void reset(int *ip) {
        *ip = 0; // změní hodnotu objektu na
                    který ip ukazuje
         ip = 0; // změní hodnotu lokálního ip
}

int i = 10;
int *g_ip = &i;
reset(g_ip);  // při volání je g_ip zkopírován
```

Předávání parametrů - Ukazatelů

- Pokud chceme funkci znemožnit změnu předávaného objektu, pak použijeme modifikátor **const.**

```
void use_ptr(const int *p) {
        *p = 10; // nepůjde přeložit, p můžeme
                 pouze číst
}
```

Předávaní parametrů

- Předávání parametrů hodnotou není vhodné:
  - pokud je parametr výstupní (funkce ho mění),
  - pokud je předávaný objekt velký,
  - pokud nelze předávaný parametr zkopírovat (např. speciální objekty reprezentující fyzické rozhraní).
- V těchto případech předáváme parametry ukazatelem nebo odkazem.

Předávání parametrů

```
// Bude pracovat správně? Proč?
   void swap(int v1, int v2) {
        int tmp = v2;
        v2 = v1;
        v1 = tmp;
   }
```

Předávání parametrů

```
// správná verze

   void swap(int &v1, int &v2) {
        int tmp = v2;
        v2 = v1;
        v1 = tmp;
   }
```

Výstupní parametry funkcí

Předávání parametrů:

- Parametry jsou předávané hodnotou.
- Volající tedy má vlastní kopii, která se po jeho ukončení odstraní.
- Takto lze předávat pouze vstupní parametry.
- Výstupní a vstupně/výstupní parametry – předávat pomocí ukazatelů nebo referencí (pouze C++).
- Vstupní parametry předávané ukazatelem či referencí – označit **const.**
- Má smysl předávat vstupní parametry jinak než hodnotou?

Výstupní parametry funkcí

Příklad (C i C++):

```
void timeToHMS ( int t, int H, int M, int S ) //  Co se stane?
 {
   H = t / 3600;
   M = ( t / 60 ) % 60;
   S = t % 60;
 }

int main ( int argc, char * argv[] )
 {
   int t, h, m, s;
   cin >> t;
   timeToHMS ( t, h, m, s );
   cout << h << ":" << m << ":" << s <<endl;
   return ( 0 );
 }
```

Výstupní parametry funkcí

Příklad (C i C++):

```
void timeToHMS ( int t, int * H, int * M, int * S ) // Co se stane?
 {
   *H = t / 3600;
   *M = ( t / 60 ) % 60;
   *S = t % 60;
 }

int main ( int argc, char * argv[] )
 {
   int t, h, m, s;
   cin >> t;
   timeToHMS ( t, &h, &m, &s );
   cout << h << ":" << m << ":" << s <<endl;
   return ( 0 );
 }
```

Výstupní parametry funkcí

Příklad (pouze C++):

```
void timeToHMS ( int t, int & H, int & M, int & S ) // Co se stane?
 {
   H = t / 3600;
   M = ( t / 60 ) % 60;
   S = t % 60;
 }

int main ( int argc, char * argv[] )
 {
   int t, h, m, s;
   cin >> t;
   timeToHMS ( t, h, m, s );
   cout << h << ":" << m << ":" << s <<endl;
   return ( 0 );
 }
```

Přetížené funkce

Přetěžování funkcí (jen C++):

- Dvě (více) funkcí může být stejného jména.
- Musí být ale odlišitelné počtem či typem parametrů.
- Nelze přetěžovat na základě návratové hodnoty.
- Pravidla pro párování typů parametrů:
  - přesná shoda,
  - roztažení,
  - standardní konverze,
  - uživatelská konverze.
- Přetěžování může být zrádné.

Přetížené funkce

Příklad přetížené funkce:

```
int abs ( int x )
 { return ( x >= 0 ? x : -x ); }

double abs ( double x )
 { return ( x >= 0 ? x : -x ); }

cout << abs ( -10 ) << " " << abs ( -10.5 );
```

Přetížené funkce

Pravidla pro párování typů parametrů:

- Přesná shoda.
- Roztažení (promotion) – nedochází ke ztrátě rozsahu ani přesnosti:
  - **char -> int**
  - **short -> int**
  - **float -> double**
- Standardní konverze – může dojít ke ztrátě přesnosti nebo rozsahu:
  - **int -> float**
  - **float -> int**
  - **int -> double**
  - **double -> int**
  - **…**
- Uživatelská konverze – konstruktorem uživ. konverze.

Přetížené funkce

- Pravidla pro párování parametrů primitivních typů:

	char	u char	short	u short	int	u int	long	u long	long long	u long long	float	double	l double
char	E	S	S	S	P	S	S	S	S	S	S	S	S
u char	S	E	S	S	P	S	S	S	S	S	S	S	S
short	S	S	E	S	P	S	S	S	S	S	S	S	S
u short	S	S	S	E	P	S	S	S	S	S	S	S	S
int	S	S	S	S	E	S	S	S	S	S	S	S	S
u int	S	S	S	S	S	E	S	S	S	S	S	S	S
long	S	S	S	S	S	S	E	S	S	S	S	S	S
u long	S	S	S	S	S	S	S	E	S	S	S	S	S
long long	S	S	S	S	S	S	S	S	E	S	S	S	S
u long long	S	S	S	S	S	S	S	S	S	E	S	S	S
float	S	S	S	S	S	S	S	S	S	S	E	P	S
double	S	S	S	S	S	S	S	S	S	S	S	E	S
l double	S	S	S	S	S	S	S	S	S	S	S	S	E

Přetížené funkce

```
void foo ( int a, double b );
void foo ( double a, int b );

foo ( 1,2 );      // chyba
foo ( 2.0, 3.0 ); // chyba
foo ( 'a', 5 );   // chyba


void bar ( float a );
void bar ( long double b );

bar ( 10 );       // chyba
bar ( 12.0 );     // chyba
bar ( 12.0f );    // ok, bar ( float )
```

Implicitní parametry funkce

- V deklaraci funkce lze nastavit výchozí hodnoty parametrů.
- Výchozí hodnoty se použijí, pokud je funkce volaná s méně skutečnými parametry.
- Implicitní parametry lze nastavovat pouze zprava.
- Výběr volané funkce – pravidla pro přetěžování + implicitní parametry.

```
void foo ( int a, int b = 3 );
foo ( 1 );    // 1, 3
foo ( 2, 4 ); // 2, 4
```

Implicitní parametry funkce

```
void foo ( double a, int b = 10 );
void foo ( long int a );

foo ( 12 );   // chyba - nejednoznacne
foo ( 'z' );  // chyba - nejednoznacne
foo ( 15.3 ); // foo ( double, int )
foo ( 3.0f ); // foo ( double, int )
```

Funkce s proměnným počtem parametrů

- V C lze deklarovat funkci s proměnným počtem parametrů (variadic function).
- Po povinných parametrech následuje výpustka **… .**
- Volání – překladač umožní s libovolným počtem skutečných parametrů na místě výpustky.
- Přístup k parametrům výpustky – makra v hlavičkovém souboru **<cstdarg> (<stdarg.h>).**
- Interpretace parametů – povinnost programátora.
- Počet skutečně předaných parametrů nelze zjistit.
- Překladač nekontroluje, zda je s parametry nakládáno "správně".

Funkce s proměnným počtem parametrů

```
#include <cstdarg>
using namespace std;
int sum_all ( int cnt, … ) {
        va_list arg;
        int     i, val, sum = 0;
        va_start ( arg, cnt );
        for ( i = 0; i < cnt; i ++ )
        {
                val = va_arg ( arg, int );
                sum += val;
        }
        va_end ( arg );
        return ( sum );
}
```

Funkce s proměnným počtem parametrů

```
int main ( int argc, char * argv [] )
{
        cout << sum_all ( 3, 10, 20, 30 ) << endl;

        cout << sum_all ( 3, 10, 20, 30, 40 ) << endl;
        // nesouhlasi udany skutecny pocet
        cout << sum_all( 5, 10, 20, 30 ) << endl;
        // nesouhlasi udany skutecny pocet
        cout << sum_all( 3, 10, 20, 5.5 ) << endl;
        // nesouhlasi datovy typ, bude spatne
        // interpretovan.
        return ( 0 );
}
```

Funkce s proměnným počtem parametrů

- Lze použít jak v C, tak v C++.
- V C++ je bezpečnější použít přetížené funkce a implicitní hodnoty parametrů:
- pro výpustku si musí programátor vybudovat svoji vlastní metodu, jak zjistit typ a počet skutečně předaných parametrů (např. dodatečný parametr),
- kompilátor není schopen detekovat prakticky žádné chyby ve volání a použití parametrů.
- Výpustku nelze použít pro předávání instancí tříd.
- Výpustka je používaná některými knihovními funkcemi:
  - **printf, scanf, syslog, exec, …**

Inline funkce

- Volání funkce představuje nenulovou režii (příprava záznamů na zásobníku, ukládání registrů CPU, zneplatnění cache, rozpracovaných instrukcí, ...).
- Citelné zejména u krátkých a často volaných funkcí.
- Optimalizace – rozpis instrukcí funkce v místě použití.
- Klíčové slovo **inline.**
- Inline funkce nesmí být rekurzivní.
- Inline funkce se zpravidla umisťují do hlavičkových souborů.

```
inline int max2 ( int a, int b )
 {
      return a > b ? a : b;
}
```

Funkce main

- Vstupní bod programu (předáno řízení po spuštění).
- Návratová hodnota - signalizace úspěchu programu:
  - 0 ok,
  - 1, 2, 3, … rozlišení příčiny neúspěchu.
- Vstupní parametry:
  - **argc počet parametrů na příkazové řádce+1,**
  - **argv pole parametrů z příkazové řádky**
  - **argv[0] jméno spuštěného programu**
  - **argv[1] první parametr,**
  - …
  - **argv[argc-1] poslední parametr.**

Ukazatele na funkce

- Ukazatel, který ukazuje na funkci.
- Jeho hodnotou je adresa funkce.
- Deklarace:

  <typ> (*name) (<parametry>);
- Typ je návratový typ funkce na kterou může ukazetel ukazovat.
- Name je název ukazatele.

Ukazatele na funkce

Příklad

int gcd(int a, int b); // deklarace funkce

// ukazatel fp na funkci vracející int, se dvěma parametry int

int (*fp)(int, int);

fp = gcd; // do fp uložíme adresu funkce gcd

int *abc (int, int); // deklarace funkce vracející int * !!!

Ukazatele na funkce

- Pro přehlednost je lépe definovat typ ukazatele na funkci pomocí typedef.

typedef int (*FP)(int, int);

FP function_pointer;

//FP2 je ukazatel na funkci vracející int *

typedef int *(*FP2)(int *, int *);

Ukazatele na funkce

- Jak definovat funkci vracející ukazatel na funkci?

```
// toto je jedna možnost
int (* get_pf(int a))(int, int) {
        return gcd;
}
// nebo pomocí typedef
FP get_pf(int a) {
        return gcd;
}
```

Literatura

- C++ Primer (4th Edition) by Stanley B. Lippman, kapitola 7
- C++ Primer Plus (5th Edition) by Stephen Prata, kapitola 7

Děkuji Vám za pozornost