

11. Úvod do programování v C++

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

- Část 1 – Procedurální programování v C++

Část I

Procedurální programování v C++

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

První program v C++

```
1  /*
2  * Hello world in C++
3  */
4  #include <iostream> // iostream nahrazuje stdio.h z C
6  using namespace std;
8  int main ( )
9  {
10     cout << "Hello World!" << endl;
11     return 0;
12 }
```

Jmenné prostory v C++

- Identifikátory v C++ se mohou odlišovat jmennými prostory
- Zápis `prostor::identifikátor` je v C++ úplným jménem proměnné

Jmenný prostor identikátoru ze `std.` knihovny má jméno `std`

- Pro zlepšení čitelnosti kódu je možné použít direktivu

```
using namespace std;
```

- Uvedená konstrukce umožňuje v dalším zdrojovém textu vynechat prefix se jménem jmenného prostoru
- Zde se jedná konkrétně o jména `std::cout` a `std::endl`
 - `cout` (výstupní proud) nahrazuje výstupní soubor `stdout` z C
 - `endl` navíc volá členskou funkci `cout.flush()`, která provede fyzický výstup bufferu

Typ `bool`

- C++ přidává primitivní datový typ `bool` pro logické hodnoty
 - V C lze použít `stdbool.h` nebo `_Bool` (C99)
- Literály typu `bool` jsou pouze dva:
 - `true` (= 1),
 - `false` (= 0).
- Pro další operace je typ `bool` kompatibilní s celočíselnými typy
- Zobrazení hodnot typu `bool` lze řídit **manipulátory**
 - `boolalpha`
 - `noboolalpha`

Typ bool

```
1  #include <iostream>
3  using namespace std;
5  int main ( ) {
6      bool a = true, b = false;
7      cout << "a=" << a << ", b=" << b << endl;
8      cout << boolalpha << "a=" << a << ", b=" << b << endl;
9      cout << "!a=" << !a << ", !b=" << !b << endl;
10     cout << "a||b=" << (a||b) << ", a&&b=" << (a&&b) << endl;
11     cout << "a+b = " << a+b << endl;
12     a = 10; b = 0;
13     cout << "a = " << a << ", b = " << b << endl;
14     cout << "a||b=" << (a||b) << ", a&&b=" << (a&&b) << endl;
15     cout << "a+b=" << a+b << endl;
16     return 0;
17 }
```


Struktury

- V C++ je identifikátor struktury (structure tag) zároveň jménem typu, takže není třeba používat `struct` nebo `typedef`

```
// C #1:
struct List { int val; struct List *next; };
struct List *head;
// C #2:
typedef struct List {
    int val; struct List *next;
} LIST;
LIST *head;
// C++:
struct LIST { int val; LIST *next; };
LIST *head;
```

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Vstup a výstup v C

```
#include <stdio.h>
int main ( void )
{
    int x;
    printf ("Napis cislo:\n");
    scanf ("%d", &x);
    printf ("Vstup byl: %d\n", x);
    return 0;
}
```

Co se stane při změně deklarace `x` na `float`?

Vstup a výstup v C++

```
#include <iostream>
using namespace std;
int main ( )
{
    int x;
    cout << "Napis cislo:" << endl;
    cin >> x;
    cout << "Vstup byl: " << x << endl;
    return 0;
}
```

Co se stane při změně deklarace `x` na `float`?

Vstup a výstup v C++

- Formátovaný vstup/výstup v C spoléhá na správný formátovací řetězec.
- Každá neshoda konverze ve formátovacím řetězci s typem parametru může způsobit chybu (pád programu).
- Proudly v C++ jsou bezpečné, neboť není třeba žádný formátovací řetězec, způsob konverze je vybrán kompilátorem podle typu parametru.
- Proudly v C++ mohou být snadno modifikovány:
 - vstup/výstup nových (uživatelských) datových typů,
 - různé zdroje/cíle proudů (soubory, buffery v paměti, sockety, ...)
- Standardní proudly:
 - `cout` – standardní výstup (stdout),
 - `cin` – standardní vstup (stdin),
 - `cerr` – standardní chybový výstup (stderr).

Výstupní manipulátory

- Řídí formátování výstupu, deklarovány v `<iomanip>`:
 - `endl` – nový řádek + flush,
 - `flush` – synchronizace bufferu proudu s fyzickým výstupem,
 - `setw (x)` – šířka výstupního pole,
 - `setfill (c)` – výplňkový znak,
 - `right/left` – zarovnání doprava / doleva,
 - `setprecision (x)` – počet desetinných míst,
 - `fixed/scientific` – formát bez / s exponentem (semilog),
 - `hex/oct/dec` – základ číselné soustavy 16, 8, 10,
 - `showbase/noshowbase` – vy/nevy-pisovat 0x - hex, resp. 0 - oct,
 - `boolalpha/noboolalpha` – true, false / 1, 0.

Příklad

```
int x = 10;
cout << "dekadicky " << x << endl;
cout << "sirka 10 znaku " << setw ( 10 ) << x << endl;
cout << "sestnactkove " << hex << x << endl;
cout << "opet dekadicky " << dec << x << endl;
```

Vstupní manipulátory

- řídí formátování vstupu:
 - `ws` – extrahuje bílé znaky,
 - `hex/oct/dec` – základ číselné soustavy 16, 8, 10,
 - `skipws/noskipws` – přeskokování bílých znaků při dalších operacích,
 - `boolalpha/noboolalpha` – vstup true, false / 1, 0,
 - `setw (n)` – omezení délky načítaného řetězce.

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Reference – motivace

- Práce s ukazateli v C může být nepohodlná
 - potenciální nebezpečí přepisu paměti
 - komplikovaný zápis

Lze řešit pomocí konstatních ukazatelů.

```
void swapC (int *px, int *py) {  
    int tmp = *px;  
    *px = *py;  
    *py = tmp;  
}  
  
...  
int a, b;  
swapC (&a, &b);
```

Reference – motivace

```
int readResize (int ** data, int * nr, int * max) {
    int x, res = scanf ("%d", &x);
    if (res != 1) return res;
    if (*nr >= *max) {
        *max += 100;
        *data = (int*) realloc (*data, *max * sizeof (**data));
    }
    (*data)[(*nr)++] = x; // !!!
    return 1;
}

...
int dataNr = 0, dataMax = 0, *data = NULL;
while (readResize (&data, &dataNr, &dataMax) == 1) {
    ...
}
```

Reference – motivace

- Nepřehlednost ukázkového kódu je dána mnoha rolemi, které má v C programech ukazatel:
 - dynamická alokace paměti,
 - realizace výstupních a vstupně-výstupních parametrů funkce,
 - předávání pole,
 - předávání vstupního parametru bez jeho kopírování
- Například velké struktury.
- C programátor pomocí ukazatelů říká, jak se má kód přeložit do strojového kódu. Zápis ale neříká, co má ukazatel za roli.
 - Reference v C++ přebírá některé role ukazatele, kde je ukazatel příliš obecný a kde zápis pomocí ukazatelů pouze prodlužuje kód.
 - Reference se v C++ použije zejména pro:
 - realizaci výstupních a vstupně-výstupních parametrů funkce a
 - předávání vstupního parametru bez jeho kopírování.

Reference – motivace

- Ukázkový kód přepsaný s referencemi:

```
void swapCPP ( int & x, int & y ) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
...  
int a, b;  
swapCPP ( a, b );
```

Reference – motivace

- Ukázkový kód přepsaný s referencemi:

```
int readResize (int *& data, int &nr, int &max) {
    int x, res = scanf ( "%d", &x );
    if (res != 1) return res;
    if (nr >= max) {
        max += 100;
        data = (int*) realloc ( data, max * sizeof (*data) );
    }
    data[nr++] = x;
    return 1;
}
...
int dataNr = 0, dataMax = 0, *data = NULL;
while ( readResize ( data, dataNr, dataMax ) == 1 ) {
    ...
}
```

Reference

- C++ reference je prostředek, kterým lze vytvořit odkaz na již existující proměnnou:
 - odkaz má všechny vlastnosti původní proměnné,
 - reference musí být při vytvoření inicializovaná proměnnou, na kterou odkazuje,
 - odkazovanou proměnnou nelze po dobu existence reference změnit,
 - reference se nejčastěji vytváří a inicializuje při volání funkce (parametry funkce v ukázce).
- Jak kompilátor implementuje referenci?
 - vnitřní implementace reference je starost autorů kompilátoru,
 - kompilátor referenci může interně implementovat jako ukazatel nebo nijak (pouhým proházením proměnných – například pokud by v ukázce provedl překlad funkce inline).

Reference

- Proměnná typu reference může existovat i uvnitř funkce, má však jen omezené použití:

```
void foo ( ) {  
    int x;  
    int &y = x; // deklarace + inicializace  
    // x a y jsou dve ruzna jmena pro tu samou promennou  
    cin >> x;  
    cout << y;  
    // zmena x je viditelna i v y  
    // referenci by kompilator nejspise optimalizoval:  
    // identifikator y by nahradil x  
}
```

Reference

- Reference uvnitř funkce pro zpřehlednění zápisu:

```
struct TCoord { int m_X, m_Y; };  
void foo ( TCoord ** matrix, ... ) {  
    for ( int i = 0; i < ...; i ++ )  
        for ( int j = 0; j < ...; j ++ ) {  
            const TCoord &x = matrix[i][j];  
            if ( sqrt ( x . m_X * x . m_X +  
                x . m_Y * x . m_Y ) > ... ) ...  
  
            // FIXME  
        }  
}
```


Konstanty

- Primitivní typy s `const` kvalifikátorem jsou v C++ konstanty.
- Na rozdíl od C maker se C++ konstanty řídí pravidly pro rozsah platnosti. Je tedy možno deklarovat konstanty lokálně v modulu, funkci nebo třídě.

```
const int MAX = 100;
int array[MAX]; // ok

int main(void) {
    int *p;
    const int *q;

    MAX = 10; // error
    p = &MAX; // error
    q = &MAX; // ok
    return 0;
}
```

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Implicitní hodnoty parametrů

- V deklaraci funkce může být uvedena implicitní hodnota parametru.
- Odpovídající parametr může být při volání funkce vynechán.
- Protože v C/C++ jsou poziční parametry, lze implicitní parametry deklarovat jen "na konci" seznamu parametrů.

```
void print ( int x, int y = 0, int z = 0 ) {  
    cout << "x=" << x << endl;  
    cout << "y=" << y << endl;  
    cout << "z=" << z << endl;  
}  
  
int main( void ) {  
    print ( 10, 20, 30 ); // 10, 20, 30  
    print ( 10, 20 ); // 10, 20, 0  
    print ( 10 ); // 10, 0, 0  
    return 0;  
}
```

Implicitní hodnoty parametrů

- Nedovolené použití implicitních parametrů:

```
void f ( int x = 1, int y ); // error
void g ( int x, int y = 10 );
void g ( int x );
// Pretizena funkce g
// Pretizen funkce povoleno, ale ne timto zpusobem
// Neexistuje zpusob, jak zavolat druhou funkci
g ( 20 ); // viceznacne
g ( 10, 40 ); // ok
```

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Inline funkce

- Volání funkce má jistou režii:
 - příprava rámce zásobníku,
 - zneplatnění cache paměti procesorem,
 - uložení/obnovení registrů procesoru, ...
- Tuto režii zpravidla nemusíme uvažovat. Významněji se uplatní pouze pro často volané triviální funkce.
- C++ umožňuje vytvořit funkce, které se nevolají. Místo toho kompilátor v místě volání vloží kód takové funkce. To má dvě výhody:
 - zdrojový kód není opakován (vytváření opakovaného kódu je zajišťováno kompilátorem na úrovni strojového kódu),
 - eliminuje se režie spojená s voláním podprogramu.
- Inline funkce se zavádějí pomocí klíčového slova `inline`.
- Kompilátor nemusí uposlechnout inline deklaraci:
 - funkce je příliš dlouhá,
 - funkce je rekurzivní,
 - nastaven příliš nízký stupeň optimalizace.

Inline funkce

- Inline funkce jsou bezpečnější než makra

Dají se lépe debuggovat – menší náchylnost k chybám.

```
inline int max ( int x, int y ) {  
    return x>=y ? x : y;  
}  
  
#define MAX(x,y) ((x)>=(y) ? (x) : (y))  
  
int main ( ) {  
    int a = 10;  
    cout << max ( a++, 4 ) << endl; // 10  
    cout << a << endl;             // 11  
    cout << MAX ( a++, 4 ) << endl; // 12  
    cout << a << endl;             // 13  
    return 0;  
}
```

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Přetěžování funkcí

- Počet a typy parametrů mohou být využity pro odlišení funkcí stejného jména.

```
int cube ( int x ) {  
    return x * x * x;  
}
```

```
double cube ( double x ) {  
    return x * x * x;  
}
```

```
int main ( void ) {  
    int a = 5;  
    float b = 4.2;  
    cout << a << "^3 = " << cube ( a ) << endl;  
    cout << b << "^3 = " << cube ( b ) << endl;  
    return 0;  
}
```

Přetěžování funkcí

- Při volání přetížené funkce se kompilátor rozhoduje podle nejlepší shody parametrů.
- Porovnání parametrů má čtyři úrovně:

přesná shoda – typy skutečných a formálních parametrů jsou stejné,

roztážení (promotion) – zachová rozsah i přesnost:

char → int, enum → int, enum → int, float → double

standardní konverze – přesnost či rozsah mohou být ztraceny:

int → double, double → int, unsigned → int, int → long, ...

uživatelská konverze – konverze zavedená uživatelem definovaným konstruktorem nebo přetíženým operátorem přetypování (cast).

Přetěžování funkcí

- Pro výběr přetížené funkce se porovnávají všichni kandidáti:
 - kandidáty jsou všechny funkce daného jména volatelné s daným počtem parametrů.
- Vítězná funkce musí mít porovnávací kategorii stejnou nebo lepší, než ostatní kandidáti. To musí platit pro všechny parametry.
- Pokud neexistuje právě jeden vítěz (funkce s nejlepší shodou ve všech parametrech), porovnávací algoritmus ohlásí chybu.
- Takto nastavená pravidla jsou striktní (vítěz musí mít nejlepší konverzi ve všech parametrech), přesto dokáží překvapit.
- Je rozumné se vyhnout nadměrnému přetěžování funkcí.

I. Procedurální programování v C++

První program

Vstup a výstup

Reference

Implicitní parametry

Inline funkce

Přetěžování funkcí

Alokace paměti

Dynamická alokace paměti

- Dynamickou alokaci provádí operátor `new`:
 - výsledek operace `new` má správný typ, nemusí se přetypovávat (`cast`),
 - velikost je dána v počtu prvků (nikoli v bajtech),
 - pro objektové datové typy volá operátor `new` konstruktor.
- Paměť alokovaná použitím operátoru `new` musí být uvolněna pomocí operátoru `delete`.
- Nelze mixovat C a C++ alokaci a uvolňování paměti:
 - blok alokovaný použitím `malloc` musí být uvolněn použitím `free`,
 - objekt alokovaný použitím `new` musí být uvolněn použitím `delete`,
 - pole alokované použitím `new []` musí být uvolněno použitím `delete []`.

Dynamická alokace paměti

```
int *p = new int; // alokuje promennou typu int
struct S {
    int a;
    char b;
};
S *q = new S; // alokuje strukturu typu S
// Pozn.: C++ zde nevyžaduje klicove slovo struct
int *a = new int[1000]; // alokuje pole
delete p; // uvolnuje jednoduchou promennou
delete q;
delete [] a; // uvolnuje pole, bez [] je to chybne
a = p + 1;
delete a; // chybne, uvolnit lze jen to,
// co bylo vytvoreno pomoci new
```