Y36PJC Programování v jazyce C/C++

Příkazy, I/O

Ladislav Vagner

Dnešní přednáška

- Příkazy v C/C++.
- Řídící konstrukce v C/C++:
  - větvení: **`if/else`**, **`switch`**,
  - cykly: **`while`**, **`for`**, **`do/while`**.
- I/O operace v C/C++:
  - C funkce,
  - C++ streamy,
  - práce se soubory.
- Časté chyby.

Minulá přednáška

- Datové typy.
- Deklarace.
- Operátory.
- Výrazy.

Příkazy

- Výraz ukončený středníkem je příkaz.
- Aby měl příkaz smysl, musí mít výraz vedlejší efekt:
  - změna obsahu proměnné (std. operátory **=**, **++**, **--**, **+=**,…, příp. přetížené operátory),
  - nebo musí obsahovat volání funkce.

```
c = a + b;
c++;
foo ();
cout << "abc";
c+4;          // vysledek se neulozi
foo;          // neni volani funkce
c << 5;       // vysledek se neulozi
```

Řídící příkazy

Podmínky - **`if`**:

- Obecný tvar: **`if (<podmínka>) <příkaz>;`**
- Nepovinná část: **`else <příkaz>;`**
- Podmínka je libovolný výraz, testovaný na nenulovost (splněno) či nulovost (nesplněno). Nemusí být **`bool`**.

Příklady:

```
int a, b, c;
if (a > b) a ++;
if (a > b) c = a; else { c = b; }
if (a) …
if (a = b) … // !!! ==
```

Řídící příkazy

Cykly s podmínkou na začátku - **`while`**.

- Obecný tvar: **`while (<podmínka>) <příkaz>;`**
- Podmínka opět nula/nenula.
- Nekonečný cyklus: např. **`while (1) { … }`**

Příklad:

```
int a, b;
a = ...; b = ...;
while ( a != b )
 if ( a > b )
   a -= b;
  else
   b -= a;
```

Řídící příkazy

Cykly s testem na konci - **do-while**.

- Obecný tvar: **do <příkaz> while (<podmínka>);**
- Tělo se vždy provede alespoň jednou.
- Vhodné např. pro testování vstupních hodnot.

Příklad:

```
int x;
do
 {
   cout << "Zadej cislo od 0 do 100" <<endl;
   cin >> x;
 } while (x < 0 || x > 100);
```

Řídící příkazy

Cyklus **`for`**.

- Obecný tvar: **`for (<v1>;<v2>;<v3>) <příkaz>;`**
  - výraz **`<v1>`** – inicializace (případně i deklarace) řídící proměnné cyklu,
  - výraz **`<v2>`** – podmínka (test nenulovosti),
  - výraz **`<v3>`** – inkrement (proveden po každém průchodu tělem cyklu).
- Libovolný z výrazů lze vynechat.
- Nekonečný cyklus např.: **`for (;;) <příkaz>;`**

Příklad:

```
for ( i = 0; i < 10; i ++ )
 cout << i << endl;
```

Řídící příkazy

Přepínač - **`switch`**.

- Rozdělení do více větví dle hodnoty výrazu.
- Varianty označeny **`case <hodnota>:`**
- Větev pro ostatní: **`default:`**
- "Propadávání" - nutnost používat **`break;`**

Příklad:

```
switch (a)
 {
   case 0:  cout << "nula";       break;
   case 1:  cout << "jedna";      break;
   default: cout << "neco jine"; break;
 }
```

Řídící příkazy

Ostatní:

- **`break;`** ukončuje cyklus / blok **`switch`**.
- **`continue;`** ukončí tuto iteraci tělem cyklu a případně spustí novou iteraci.
- **`return <výraz>;`** předá řízení zpět volající funkci.
- **`goto <návěští>;`** předá řízení na dané místo programu (ale všichni víme, jak je to s goto).

Řídící příkazy – časté chyby

Výrazy v podmínce:

```
if ( x = 'a' ) { … }  // vždy splneno
```

správně:

```
if ( x == 'a') { … }
```

Chceme-li skutečně přiřadit a testovat na nenulovost:

```
while ( x = nextChar () ) { … }
```

lépe:

```
while ( (x = nextChar ()) != 0 ) { … }
```

Řídící příkazy – časté chyby

Vnořování if:

```
if ( a % 2 == 0 )
  if ( a > 100 )
    cout << "a je sude a velke";
 else
  cout << "a je liche";  // patri k poslednimu if
```

else se vždy váže k nejbližšímu if. Správně:

```
if ( a % 2 == 0 )
  {
    if ( a > 100 )
     cout << "a je sude a velke";
  }
 else
  cout << "a je liche";
```

Řídící příkazy – časté chyby

Míchání znaménkových a neznaménkových datových typů:

```
unsigned int u;
int v;
u = 10; v = -1;
if ( u < v )
 cout << u << " < " << v << endl;
```

Správně - stejný datový typ nebo podle kontextu přetypovat:

```
if ( (int)u < v ) // if (u<(unsigned int)v)
 cout << u << " < " << v << endl;
```

Řídící příkazy – časté chyby

Propadávání u switch:

```
switch ( hodnota )
 {
   case 0:
    cout << "varianta 0";     // !!
   case 1:
    cout << "varianta 1";     // !!
   default:
    cout << "jina varianta"; // !!
 }
```

Řídící příkazy – časté chyby

Středník za cyklem:

```
sum = 0;
while ( a > 0 )  ;   // prazdne telo
 { sum += a % 10; a /= 10; }
```

Chceme-li skutečně prázdné tělo cyklu:

```
for (sum=0; a > 0; sum += a % 10, a /=10);
```

lépe:

```
for (sum=0; a > 0; sum += a % 10, a /=10) {}
```

Řídící příkazy – časté chyby

Datový typ při porovnání:

```
unsigned int i;
…
while ( i >= 0 )      // !! always true
 {
   cout << "Hi" << endl;
   i --;
 }
```

Řídící příkazy – časté chyby

Inkrement for-cyklù:

```
int i;

for ( i = 0; i < 10; i ++ )
 cout << i << endl;

for ( i = 9; i >= 0; i -- )
 cout << i << endl;

for ( i = 9; i >= 0; i ++ ) // !!
 cout << i << endl;
```

Řídící příkazy – časté chyby

Dle staré normy C++:

```
void foo ( void )
 {
   for ( int i = 0; i < 10; i ++ ) { … }
   // zde stale plati deklarace i z predchoziho for
   for ( i = 0; i < 20; i ++ ) { … }
 }
```

Dle nové normy C++:

```
void foo ( void )
 {
   for ( int i = 0; i < 10; i ++ ) { … }
   // zde uz neplati deklarace i z predchoziho for
   for ( int i = 0; i < 20; i ++ ) { … }
 }
```

Řídící příkazy – časté chyby

Univerzálně:

```
void foo ( void )
 {
   int i;
   for ( i = 0; i < 10; i ++ ) { … }
   for ( i = 0; i < 20; i ++ ) { … }
 }
```

Nebo:

```
void foo ( void )
 {
   for ( int i = 0; i < 10; i ++ ) { … }
   for ( int j = 0; j < 20; j ++ ) { … }
 }
```

Vstup a výstup - C

Výstup – funkce **`printf`**:

- formátovací řetězec:
  - text zobrazený beze změny,
  - substituce (uvozené znakem `%`).
- parametry pro substituce,
- substituce ve formátovacím řetězci musí typově odpovídat parametrům.

Příklad:

```
int           a = 10;
double        b = 2.5;
const char *  c = "Hello";
printf ( "a= %d, b= %f, c= %s\n", a, b, c );
```

Vstup a výstup - C

Vstup – funkce **`scanf`**:

- formátovací řetězec:
  - substituce (uvozené znakem `%`),
- adresy proměnných pro uložení hodnot,
- substituce ve formátovacím řetězci musí typově odpovídat parametrům.

Příklad:

```
int            a;
double         b;
char           c [40];
scanf ( "%d %f %s\n", &a, &b, c );
scanf ( "%d %f %39s\n", &a, &b, c );
```

Vstup a výstup - C

Nevýhody funkcí **`printf`** a **`scanf`**:

- formátovací řetězec musí odpovídat parametrům,
- překladač nemá obecně možnost kontrolovat, zda skutečně odpovídá (omezeně se snaží gcc),
- chyba v párování způsobí chybu konverze či dokonce pád programu.

Výhody funkcí **`printf`** a **`scanf`**:

- trochu kompaktnější zápis,
- trochu snazší lokalizace (lze lépe oddělit řetězce od programu).

Neformátovaný výstup (binární), práce se soubory v C – viz popis funkcí **`fopen, fclose, fread, fwrite,`** …

Vstup a výstup - C++

Vstupní a výstupní proudy:

- datový proud (stream) - tok dat od zdroje k cíli, s možnou konverzí,
- formátovaný vstup – data jsou z textové reprezentace (posloupnost znaků) transformována do vnitřní binární reprezentace,
- formátovaný výstup – data jsou z vnitřní binární reprezentace transformovaná na posloupnost znaků,
- neformátovaný vstup/výstup – data jsou předána beze změny jako posloupnost bajtů.

Vstup a výstup - C++

Standardní výstup:

- přetížený operátor **<<**,
- knihovna zajišťuje přetížení pro standarní datové typy,
- instance **`cout`** – datový proud pro standardní výstup,
- odbourává potřebu formátovacího řetězce.

Příklad:

```
int a = 10;
double b = 2.5
const char * c = "Hello";
cout << a << " " << b << " " << c;
```

Vstup a výstup - C++

Standardní vstup:

- přetížený operátor **>>**,
- knihovna zajišťuje přetížení pro standarní datové typy,
- instance **cin** – datový proud pro standardní vstup,
- odbourává potřebu formátovacího řetězce.

Příklad:

```
int    a;
double b;
char   c [40];
cin  >> a >> b >> c;
cin  >> a >> b >> setw (sizeof ( c )) >> c;
```

Vstup a výstup - C++

Manipulátory:

- sada nástrojů pro řízení vzhledu výstupu,
- poslány do proudu před proměnnou ovlivní její zobrazení,
- deklarovány v hlavičkovém souboru **<iomanip>**.

Příklad:

```
int    a = 3;
double b = 1.0 / 3.0;

cout << hex << a;                    // v 16 soustave
cout << setw ( 20 ) << b;            // sirka 20 znaku
cout << setprecision ( 3 ) << b; // 3 des. mista
```

Vstup a výstup - C++

Číselná soustava:

- **`hex`**
- **`oct`**
- **`dec`**

Tvar výstupu:

- **`setw(x)`**
- **`setprecision(x)`**
- **`setfill(c)`**

Odřádkování:

- **`endl`**

Vstup a výstup - C++

Práce se soubory:

- souborové datové proudy: **`ifstream`** a **`ofstream`**,
- při vytváření se určí jméno souboru,
- indikace úspěšnosti operace - volání **`fail()`**,
- deklarovány v hlavičkovém souboru **`<fstream>`**,
- po použití zavřít voláním **`close()`**.

Příklad:

```
ifstream is ( "filename.ext" );
int a;

if ( is . fail () ) { … }
is >> a;
is . close ();
```

Vstup a výstup - C++

Binární a textové soubory:

- znak konce řádku v souboru:
  - LF (\n = 0x0A) UNIX,
  - CR LF (\r \n = 0x0D 0x0A) Windows, DOS,
  - CR (\r = 0x0D) Mac,
- v paměti je konec řádku vždy LF,
- C/C++ knihovna na Windows/Mac provádí automatickou konverzi:
  - při čtení nahradí posloupnost CR LF (CR) znakem LF,
  - při zápisu nahradí znak LF znaky CR LF (CR).
- automatickou konverzi je potřeba vypnout pro binární soubory,
- parametr konstruktoru – způsob otevření.

Vstup a výstup - C++

Textový soubor:

```
ifstream it ( "fn.ext" );
ofstream ot ( "fn.ext" );
```

Binární soubor:

```
ifstream ib ( "fn.ext", ios::binary | ios::in );
ofstream ob ( "fn.ext", ios::binary | ios::out );
```

Práce s binárními soubory:

- viz popis metod **`read()`**, **`write()`**, **`seekg()`**, **`tellg()`**, **`gcount()`**, …

Vstup a výstup - C++

Paměťové proudy:

- proudy **`istringstream`** a **`ostringstream`**,
- deklarovány v hlavičkovém souboru **`<sstream>`**,
- přístup ke zformátovanému řetězci – volání **`str()`**,
- použití stejné jako u souborových datových proudů.

Příklad:

```
ostringstream os;
int a;

os << setw ( 20 ) << setfill ( '0' ) << a;
cout << os . str ();
```

Dotazy …

Děkuji za pozornost.