

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) řídicí 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 splněno
```

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-cyklů:

```
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 stále platí deklarace i z předchozího for
    for ( i = 0; i < 20; i ++ ) { ... }
}
```

Dle nové normy C++:

```
void foo ( void )
{
    for ( int i = 0; i < 10; i ++ ) { ... }
    // zde už neplatí deklarace i z předchozího 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 - neformátovaný

- Reprezentace dat v paměti a na vstupu/výstupu je stejná.
- Čtecí/zápisové operace nezasahují do zpracovávané posloupnosti bajtů.
- Příklad – celá čísla:

Hodnota	v paměti	v souboru
1	00 00 00 01 (4B)	00 00 00 01 (4B)
1000	00 00 03 E8 (4B)	00 00 03 E8 (4B)
1000000	00 0F 42 40 (4B)	00 0F 42 40 (4B)

Vstup a výstup - formátovaný

- Paměťová (binární) reprezentace odlišná od textové reprezentace na vstupu/výstupu.
- Zápisové operace provádí konverzi z paměťové (binární) reprezentace na text.
- Čtecí operace převádějí textovou reprezentaci zpět na paměťovou binární.
- Ošetření chyb při konverzi.
- Příklad – celá čísla:

Hodnota	v paměti	v souboru (např.)
1	00 00 00 01	31 (1B)
1000	00 00 03 E8	31 30 30 30 (4B)
1000000	00 0F 42 40	46 34 32 34 30 (5B)

Vstup a výstup

Binární a textové soubory:

- znak konce řádku v souboru:
 - LF ($\backslash n = 0x0A$) UNIX,
 - CR LF ($\backslash r \backslash n = 0x0D 0x0A$) Windows, DOS,
 - CR ($\backslash r = 0x0D$) 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, jinak dojde k poškození obsahu.

Vstup a výstup v C

Formátovaný 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 v C

Formátovaný vstup – funkce `scanf`:

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

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


Vstup a výstup v 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 v C++

Datový proud (stream) – tok dat od zdroje k cíli, s možnou konverzí:

- Vstupní proud (`istream`) – data jsou vyčítána ze zdroje a ukládána do paměti (proměnných). Zdroje:
 - klávesnice (`cin`),
 - soubor (instance `ifstream`),
 - paměťový prostor (instance `istringstream`).
- Výstupní proud (`ostream`) – data jsou z proměnných zasílána k cíli:
 - konzole (`cout`),
 - soubor (instance `ofstream`),
 - paměťový prostor (instance `ostringstream`).

Vstup a výstup v C++

Formátovaný výstup:

- rozhraní dáno předkem – třídou `ostream`,
- přetížený operátor `<<`,
- knihovna zajišťuje přetížení pro standardní datové typy,
- odbourává potřebu formátovacího řetězce.

Příklad:

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

Vstup a výstup v C++

Manipulátory:

- sada nástrojů pro řízení vzhledu formátovaného 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 soustavě
cout << setw ( 20 ) << b;    // sirka 20 znaku
cout << setprecision ( 3 ) << b; // 3 des. místa
```

Vstup a výstup v C++

Číselná soustava:

`hex`

`oct`

`dec`

Tvar výstupu:

`setw(x)`

`setprecision(x)`

`setfill(c)`

`fixed`

`scientific`

Odřádkování:

`endl`

Vstup a výstup v C++

Formátovaný vstup:

- rozhraní dáno předkem – třídou `istream`,
- přetížený operátor `>>`,
- knihovna zajišťuje přetížení pro standardní datové typy,
- 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 v C++

Neformátovaný vstup:

- `get()` – čtení bajtu,
- `getline()` – čtení řádky,
- `read()` – čtení bloku binárních dat do paměti,
- `gcount()` – počet bajtů přečtený voláním `read()`,
- `seekg()` – přesun čtecího ukazatele,
- `tellg()` – zjištění pozice čtecího ukazatele.

Vstup a výstup v C++

Neformátovaný výstup:

- `put()` – zápis bajtu,
- `write()` – zápis bloku binárních dat,
- `seekp()` – přesun zápisového ukazatele,
- `tellp()` – zjištění pozice zápisového ukazatele,
- `flush()` – vyprázdnění vyrovnávacích pamětí.

Vstup a výstup v C++

Ošetření chyb (formátovaný i neformátovaný vstup/výstup):

- Příznaky chyb:
 - `fail()` chyba při poslední operaci,
 - `bad()` neopravitelná chyba proudu,
 - `eof()` pokus o čtení za koncem souboru,
 - `good()` `!eof() && !fail() && !bad()`
- Datový proud neprovádí žádné další I/O operace, pokud je ve stavu `fail()` nebo `bad()`.
- Potvrzení chyby (nulování příznaku chyby): metoda `clear()`.

Vstup a výstup v C++

```
cout << "Zadejte cele kladne cislo:" << endl;
while ( 1 )
{
    cin >> x;
    if ( ! cin . good ( ) )
    {
        cin . clear ( );
        cin . ignore ( INT_MAX, '\n' );
        cout << "Nebylo zadane cele cislo." << endl;
    }
    else if ( x <= 0 )
        cout << "Nebylo zadane kladne cislo." << endl;
    else
        break;
}
```

Vstup a výstup v C++

Práce se soubory:

- souborové datové proudy: `ifstream` a `ofstream`,
- při vytváření se určí jméno souboru,
- indikace (ne)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;  
if ( is . fail () ) { ... }  
is . close ();
```

Vstup a výstup v 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 );
```

Vstup a výstup v C++

Paměťové proudy:

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

Příklad:

```
ostream os;  
int a;
```

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

Dotazy ...

Děkuji za pozornost.