3. Základní řídicí struktury

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Programování v C
  - Standardní vstup a výstup
  - Výrazy a operátory
- Část 2 – Řídicí struktury
  - Větvení
  - Cykly
  - Konečnost cyklu
  - Příklady na cykly

Část I

Programování v C

I. Programování v C

Standardní vstup a výstup

Výrazy a operátory

P3.1 Standardní výstup

```c
#define PAGES 931

int main(void)
{
    printf("*%d*\n", PAGES);
    printf("*%2d*\n", PAGES);
    printf("*%10d*\n", PAGES);
    printf("*%-10d*\n", PAGES);
```

lec03/printf-field.c

```
user@machine:~/prpa/lec03$ ./printf-field
*931*
*931*
*       931*
*931       *
```

P3.2 Standardní výstup

```
5       const double RENT = 3852.99;
7       printf("*%f*\n", RENT);
8       printf("*%4.2f*\n", RENT);
9       printf("*%3.1f*\n", RENT);
10      printf("*%10.3f*\n", RENT);
11      printf("*%+4.2f*\n", RENT);
12      printf("*%010.2f*\n", RENT);
```

lec03/printf-float.c

```
user@machine:~/prpa/lec03$ ./printf-float
*3852.990000*
*3852.99*
*3853.0*
*  3852.990*
*+3852.99*
*0003852.99*
```

P3.3 Standardní výstup

```c
#include <stdio.h>

int main(void)
{
    printf("%x %X %#x\n", 31, 31, 31);
    printf("*%d*% d*% d*\n", 42, 42, -42);
    printf("*%5d*%5.3d*%05d*%05.3d*\n", 6, 6, 6, 6);

    return 0;
}
```

lec03/printf-format.c

```
user@machine:~/prpa/lec03$ ./printf-format
1f 1F 0x1f
*42* 42*-42*
*    6*  006*00006*  006*
```

P3.4 Standardní výstup

```
3  #define STRING "Testovaci retezec!"
5  int main(void)
6  {
7      printf("*%2s*\n", STRING);
8      printf("*%24s*\n", STRING);
9      printf("*%24.5s*\n", STRING);
10     printf("*%-24.5s*\n", STRING);
```

lec03/printf-format.c

```
user@machine:~/prpa/lec03$ ./printf-string
*Testovaci retezec!*
*      Testovaci retezec!*
*                   Testo*
*Testo                   *
```

P3.5 Standardní vstup

```
1  int m,n;
2  scanf("%d,%d", &m, &n);
```

```
88,121
88, 121
88,
121
```

protože `scanf` přeskakuje bílé znaky předcházející další položce vstupu (zde celému číslu), je možné za čárkou psát mezeru nebo enter

```
1  scanf("%d ,%d", &m, &n);
```

```
88,121
88  ,121
88 ,  121
```

koncept *několik bílých znaků* připouští i variantu žádný bílý znak

Trochu jinak se `scanf` chová při použití specfifikátoru `%s`

```
1  scanf("%c", &a);  // načte znak včetně bílých
2  scanf(" %c", &a); // načte první 'nebílý' znak
```

P3.6 Standardní vstup

```c
#include <stdio.h>

int main(void)
{
    int n;

    printf("Napis tri cela cisla:\n");
    scanf("%*d %*d %d", &n);
    printf("Posledni cislo je %d\n", n);

    return 0;
}
```

lec03/scanf-skip.c

```
Napis tri cela cisla:
3 4 5
Posledni cislo je 5
```

I. Programování v C

Výrazy

- Výraz předepisuje výpočet hodnoty určitého vstupu
- Výraz může obsahovat
  - **operandy** – proměnné, konstanty, volání funkcí nebo jiné výrazy
  - **operátory**
  - **závorky**
- Pořadí operací předepsaných výrazem je dáno **prioritou** a **asociativitou** operátorů

Příklad

```
1  10 + x * y  // poradi vyhodnoceni 10 + (x * y)
2  10 + x + y  // poradi vyhodnoceni (10 + x) + y
```

- Operátor `*` má vyšší prioritu než operátor `+`, operátor `+` je asociativní zleva

Výrazy a operátory

- **Výraz** se skládá z operátorů a operandů
  - Výraz sám může být operandem
  - Výraz má typ a hodnotu (Pouze výraz typu `void` hodnotu nemá.)
  - Výraz zakončený středníkem `;` je příkaz
- **Operátory** jsou vyhrazené znaky (ev. sekvence) pro zápis výrazu
  - Postup výpočtu výrazu s více operátory je dán prioritou operátoru
  - Postup výpočtu lze předepsat použitím kulatých závorek `(` a `)`
  - Obecně (mimo konkrétní případy) není pořadí vyhodnocení operandů definováno (nezaměňovat s asociativitou!)
    - Např. pro součet `f1() + f2()` není definováno, který operand se vyhodnotí jako první (tj. jaká funkce se zavolá jako první).
    - Pořadí vyhodnocení je definováno pro operandy v logickém součinu `AND` a součtu `OR`
- **Nedefinované chování** – vyhodnocení některých specifických výrazů není definováno a záleží na překladači: `i = ++i + i++;`

https://en.cppreference.com/w/c/language/eval_order

Operátory

- Binární operátory
  - Aritmetické – sčítání, odčítání, násobení, dělení
  - Relační — porovnání hodnot (menší, větší, . . .)
  - Logické — logický součet a součin
  - Operátor priřazení – na levé straně operátoru = je proměnná
- Unární operátory
  - indikující kladnou/zápornou hodnotu: + a -

    operátor - modifikuje znaménko výrazu za ním
  - modifikující proměnou: ++ a --
  - logický operátor doplněk: !
  - bitová negace (negace bit po bitu): ~
  - operátor pretypování: `(jméno typu)`
- Ternární operátor
  - podmíněné přiřazení hodnoty: ? :

http://www.tutorialspoint.com/cprogramming/c_operators.htm

Aritmetické operátory

- Operandy aritmetrických operátorů mohou být libovolného číselného typu

  Výjimkou je operátor zbytek po dělení % definovaný pro int

*	Násobení	`x*y`	
/	Dělení	`x/y`	
%	Dělení modulo	`x%y`	Zbytek po dělení `x` a `y`
+	Sčítání	`x+y`	
-	Odčítání	`x-y`	
+	Kladné zn.	`+x`	
-	Záporné zn.	`-x`	
++	Inkrementace	`++x`, `x++`	Inkrementace před/po vyhodnocení výrazu
- -	Dekrementace	`--x`, `x--`	Dekrementace před/po vyhodnocení výrazu

Unární aritmetické operátory

- Unární operátory `++` a `--` mění hodnotu svého operandu

  Operand musí být **l-hodnota**, tj. výraz, který má adresu, kde je uložena hodnota výrazu (např. proměnná)

  - lze zapsat prefixově, např. `++x` nebo `--x`

    Operace je provedena **před** vyhodnocením výrazu.

  - nebo postfixově např. `x++` nebo `x--`

    Operace je provedena **po** vyhodnocení výrazu.

  - v obou případech se však liší výsledná hodnota výrazu!

Příklad

```
1  int i = 1, a;
2  a = i++;       // i=2 a=1
3  a = ++i;       // i=3 a=3
4  a = ++(i++);   // nelze, hodnota i++ neni l-hodnota
```

Relační operátory

- Operandy relačních operátorů mohou být
  - číselného typu,
  - ukazatele shodného typu nebo
  - jeden z nich `NULL` nebo
  - typ `void`

Menší než	`x<y`	`1` pro `x` je menší než `y`, jinak `0`
Menší nebo rovno	`x<=y`	`1` pro `x` menší nebo rovno `y`, jinak `0`
Větší	`x>y`	`1` pro `x` je vetší než `y`, jinak `0`
Větší nebo rovno	`x>=y`	`1` pro `x` vetší nebo rovno `y`, jinak `0`
Rovná se	`x==y`	`1` pro `x` rovno `y`, jinak `0`
Nerovná se	`x!=y`	`1` pro `x` nerovno `y`, jinak `0`

P3.7 Relační operátory

```c
#include <stdio.h>

int main()
{
  int a = 10, b = 20, c = 30;

  // (c > b > a) vyhodnoceno jako ((c > b) > a),
  // asociativita '>' je zleva doprava.
  // takze ((30 > 20) > 10) --> (1 > 20)
  if (c > b > a)
    printf("TRUE");
  else
    printf("FALSE");

  return 0;
}
```

lec04/chained-comparison.c

Logické operátory

- Operandy mohou být číselné typy nebo ukazatele
- Výsledek 1 má význam true, 0 má význam false
- Ve výrazech `&&` a `||` se vyhodnotí nejdříve levý operand – pokud je výsledek dán levým operandem, pravý se již nevyhodnocuje

AND	`x&&y`	`1` pokud `x` ani `y` není rovno `0`, jinak `0`
OR	`x||y`	1 pokud alespoň jeden z `x`, `y` není rovno `0`, jinak `0`
NOT	`!x`	`1` pro `x` rovno `0`, jinak `0`

Část II

Řídicí struktury

Řídicí struktury

- Části programu, které předepisují způsob provedení dílčích příkazů
- Základní druhy řídících struktur
  - **posloupnost** – postupné provedení příkazů – složený příkaz a blok
  - **větvení** – provedení příkazů v případě splnění podmínky
  - **cyklus** – opakované provedení, dokud je splněna podmínka
- Podmíněné řízení běhu programu
  - Podmíněný příkaz: `if ()` nebo `if () ... else`
  - Programový přepínač: `switch () case ...`
- Cykly
  - `for ()`
  - `while ()`
  - `do ... while ()`
- Nepodmíněné větvení programu
  - `continue`, `break`, `return` a `goto`

Složený příkaz a blok

- **Složený příkaz** – posloupnost příkazů

```
{
  // vymezení složeného příkazu složenými závorkami
  printf("No. of steps \%i\n", steps);
}
```

- **Blok** – posloupnost definic proměnných a příkazů
  - ovlivňuje rozsah platnosti proměnných
  - pojmenovaný blok – základ procedur a funkcí

```
{
  int steps = 10;
  printf("No. of steps \%i\n", steps);
}
steps += 1; //nelze - mimo rozsah platnosti bloku
```

Definice – alokace paměti podle konkrétního typu proměnné. Rozsah platnosti proměnné je **lokální** v rámci bloku.

II. Řídicí struktury

Podmíněný příkaz if

- Umožňuje větvení programu na základě podmínky
- Možné tvary:

```
if (podminka) prikaz
if (podminka) prikaz1 else prikaz2
if (podminka1) prikaz1 else if (podminka2) prikaz2 else prikaz3
```

- `podminka` – logický výraz, jehož hodnota je logického typu

false (hodnota 0) nebo true (hodnota různá od 0)

- `prikaz` – příkaz, složený příkaz nebo blok

Příkaz je zakončen středníkem ;

Příklad – varianty zápisu zjištění menší hodnoty z `x` a `y`:

```
1  int min = y;
2  if (x < y) min = x;
```

```
1  int min = y;
2  if (x < y)
3    min = x;
```

```
1  int min = y;
2  if (x < y) {
3    min = x;
4  }
```

Podmíněný příkaz if

- Podmíněné příkazy mohou být vnořené a můžeme je řetězit

Příklad

```
1  int max;
3  if (a > b) {
4    if (a > c) {
5      max = a;
6    }
7  }
8
```

```
1  if (a > b) {
2    // ...
3  } else if (a < c) {
4    // ...
5  } else if (a == b) {
6    // ...
7  } else {
8    // ...
9  }
10
```

Podmíněný příkaz if

- Jestliže v případě splnění či nesplnění podmínky má být provedeno více příkazů, je třeba z nich vytvořit složený příkaz nebo blok.

Příklad jestliže `x < y`, vyměňte hodnoty těchto proměnných

```
1  if (x < y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
```

Co se stane, když za příkazem větvení nebude blok?

```
1  if (x < y)
2    int tmp = x;
3  x = y;
4  y = tmp;
```

P3.8 Podmíněný příkaz if

- Do proměnné `min` uložte menší z čísel `x` a `y` a do proměnné `max` uložte větší z čísel.

```
if (x < y) {
  min = x;
  max = y;
} else {
  min = y;
  max = x;
}
```

- Špatné řešení:

```
if (x < y)
  min = x;
max = y;
else min = y; //unexpected token else
  max = x;
```

Programový přepínač switch

- Příkaz `switch` (přepínač) umožnuje větvení programu do více větví na základě ruzných hodnot výrazu výčtového (celočíselného) typu, jako jsou např. `int`, `char`, `short`, `enum`.
- Tvar příkazu

```
switch (vyraz) {
  case konstanta1:
    prikazy1; break;
  case konstanta2:
    prikazy2; break;
  // ---
  case konstantaN:
    prikazyN; break;
  default:
    prikazydef;
}
```

- Přepínač `switch(vyraz)` větví program do `N` větví
- Hodnota `vyraz` je porovnávána s `N` konstantními výrazy typu `int` příkazy `case konstantai: ...`
- Hodnota `vyraz` musí být celočíselná a hodnoty konstant musí být navzájem různé
- Pokud je nalezena shoda, program pokračuje od tohoto místa dokud nenajde příkaz `break` nebo konec příkazu
- Pokud shoda není nalezena, program pokračuje nepovinnou sekcí `default`

Sekce **default** se zpravidla uvádí jako poslední

- Příkazy switch mohou být vnořené

P3.9 Programový přepínač switch

```c
switch (n)
{
  case 1:
    printf("*"); break;
  case 2:
    printf("**"); break;
  case 3:
    printf("***"); break;
  case 4:
    printf("****"); break;
  default:
  printf("---");
}
```

```c
if (n == 1) {
  printf("*");
}
else if (n == 2) {
  printf("**");
}
else if (n == 3) {
  printf("***");
}
else if (n == 4) {
  printf("****");
}
else printf("---");
```

- Co se vypíše, pokud ve větvích nebudou příkazy `break` a n=3?

II. Řídicí struktury

Cykly while a do-while

- Tvar příkazu **while**

```
while (podminka)
  prikaz
```

- Tvar příkazu **do-while**

```
do
  prikaz
while (podminka)
```

Příklad

```
1  q = x;
2  while (q >= y) {
3    q = q - y;
4  }
```

```
1  q = x;
2  do {
3    q = q - y;
4  } while (q >= y);
```

- Jaká je hodnota proměnné `q` po skončení cyklu?

Cyklus while

- Tvar příkazu

```
while (podminka) prikaz
```

- Průběh cyklu
  1. Vyhodnotí se výraz `podminka`
  2. Pokud `podminka != 0`, provede se příkaz `prikaz`, jinak cyklus končí
  3. Opakování vyhodnocení výrazu `prikaz`
- Řídicí výraz
  - vyhodnocení na začátku cyklu

    Cyklus se nemusí provést ani jednou
  - aktualizace v těle cyklu, jinak je cyklus nekonečný

Cyklus do-while

- Tvar příkazu

```
do prikaz while (podminka);
```

- Průběh cyklu
  1. Provede se příkaz `prikaz`
  2. Vyhodnotí se výraz `podminka`
  3. Pokud `podminka != 0`, cyklus se opakuje
- Řídicí výraz
  - vyhodnocení na konci cyklu

    Cyklus se provede vždy alespoň jednou
  - aktualizace v těle cyklu, jinak je cyklus nekonečný

P3.10 Cyklus while

- Program na výpočet faktoriálu přirozeného čísla

$$n! = \prod_{k=1}^{n} k$$

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int i = 1, f = 1, n;
  printf("zadejte prirozene cislo: ");
  scanf("%d", &n);
  while (i<n) {
    i = i+1;
    f = f*i;
  }
  printf("%d! = %d\n", n, f);
  return 0;
}
```

Cyklus for

- Cyklus je často řízen proměnnou, pro kterou je stanoveno:
  - jaká je počáteční hodnota
  - jaká je koncová hodnota (podmínka pro provedení těla cyklu)
  - jak změnit hodnotu proměnné po každém provedení těla cyklu

Příklad

```
1  int f = 1;
2  int i = 1;          // počáteční hodnota řídící proměnné
3  while (i<=n) {      // řídící výraz
4      f = f*i;        // tělo cyklu
5      i = i+1;        // změna řídící proměnné
6  }
```

- Cykly tohoto druhu lze zkráceně předepsat příkazem `for`:

```
1  int f = 1;
2  for (int i=1; i<=n; i=i+1) f=f*i;
```

Cyklus for

- Tvar příkazu

  `for (inicializace; podminka; zmena) prikaz`

- Odpovídá cyklu while ve tvaru:

```
inicializace;
while (podminka) {
  prikaz;
  zmena;
}
```

Příklad

```
1  for (int i = 0; i < 10; ++i) {
2    printf("i = %i\n", i);
3  }
```

Změnu řídicí proměnné lze zapsat operátorem inkrementace `++` nebo dekrementace `--`, lze též použít zkrácený zápis přiřazení, např. `+=`.

- Výrazy `inicializace` a `zmena` mohou být libovolného typu
- Libovolný z výrazů lze vynechat
- **break** – cyklus lze nuceně opustit příkazem `break`
- **continue** – část těla cyklu lze vynechat příkazem `continue`
- Při vynechání řídicího výrazu `podminka` se cyklus bude provádět nepodmíněně

Cyklus for – příklady

- Správný zápis

```
for (i = 0; i < 10; i++) {}
for (; a < 4.0; a += 0.2) {}
for (; i < 10;) {}
for (;; i++) {}  // nekonečný cyklus
for (;;) {}      // nekonečný cyklus, ekv.  while(1)
for (i = 1; i < 10; printf("%i\n", i), i++);
```

- Nesprávné použití

```
for () {}                    // chybí středníky
for (i = 1, i == x, i++) {} // čárky místo středníků
for (x < 4) {}               // chybí středníky
```

Příkaz návratu na vyhodnocení řídicího výrazu continue

- Příkaz `continue` lze použít pouze v těle cyklu
  - `for ()`, `while ()`, `do-while ()`
- Příkaz způsobí přerušení vykonávání těla cyklu a nové vyhodnocení řídicího výrazu

Příklad

```c
int i;
for (i = 0; i < 15; ++i) {
  if (i % 2 == 0) {
    continue;
  }
  printf("i: %2d\n", i);
}
```

```
i:  1
i:  3
i:  5
i:  7
i:  9
i: 11
i: 13
```

lec04/loop-continue.c

Příkaz nuceného ukončení cyklu break

- Příkaz `break` lze použít pouze v těle řídících struktur
  - `for()`, `while()`, `do...while()`, `switch()`
- Program pak pokračuje následujícím příkazem.

Příklad

```c
int i = 10;
while (i > 0) {
  if (i == 5) {
    printf("opoustim cyklus\n");
    break;
  }
  printf("i: %2d", i--);
}
printf("konec cyklu i: %d\n", i);
```

```
i: 10
i: 9
i: 8
i: 7
i: 6
opoustim cyklus
konec cyklu i: 5
```

lec04/loop-break.c

Příkaz nepodmíněného lokálního skoku goto

- Syntax: `goto navesti;`
- Příkaz předá řízení na místo určené návěštím `navesti`, lze použít pouze v těle funkce
- Skok nesmí směřovat dovnitř bloku, který je vnořený do bloku, kde je příslušné `goto` umístěno

Příklad

```
1  int test = 3;
2  for (int i = 0; i < 10; i++) {
3    if (i == test) {
4      goto OUT;
5    }
6    printf ("i = %i\n", i);
7  }
8  return 0;
9  OUT: return -1;
```

II. Řídicí struktury

Konečnost cyklu

- Konečnost algoritmu – pro přípustná data skončí v konečné době
- Aby byl algoritmus konečný, musí každý cyklus v něm uvedený skončit po konečném počtu kroků
- Jedním z důvodu neukončení programu je **zacyklení**
- **Zacyklení** – program opakovaně vykonává cyklus, jehož podmínka ukončení není nikdy splněna

```
1  while (i != 0) {
2    j = i - 1;
3  }
```

- Cyklus se neprovede ani jednou,
- nebo neskončí.
- Záleží na hodnotě řídicí proměnné `i` před voláním cyklu

Základní pravidlo konečnosti cyklu

- Provedením těla cyklu se musí změnit hodnota proměnné použité v podmínce ukončení

```
for (int i = 0; i < 5; ++i) {
// --
}
```

- Uvedené pravidlo konečnost cyklu nezaručuje

```
int i = -1;
while ( i < 0 ) {
  i = i - 1;
}
```

- Konečnost cyklu závisí na hodnotě proměnné před vstupem do cyklu.

Konečnost cyklu

```
1  while (i != n) {
2    // příkazy nemenici hodnotu promenne i
3    i++;
4  }
```

- Vstupní podmínka konečnosti uvedeného cyklu
  - $i \leq n$ pro celá čísla

Jak by vypadala podmínka pro proměnné typu double?

- Splnění vstupní podmínky konečnosti cyklu musí zajistit příkazy předcházející cyklu
- Zabezpečený program testuje přípustnost vstupních dat

II. Řídicí struktury

P3.11 Je číslo palindrom?

- **Palindrom** – symetrický útvar, stejný význam při čtení zleva i zprava
- Příklady: 131, 5896985, ...
- Možné řešení: výpočet reverzního čísla

```
1  int n, num, rev;
3  scanf("%d", &n);
5  num = n;
7  while(n != 0) {
8    rev = (rev * 10) + (n % 10);
9    n  /= 10;
10 }
12 if (rev == num) printf("cislo %d je palindrom", n);
13 else printf("cislo %d neni palindrom", n);
```