3. Základní řidící struktury

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Programování v C
  - Funkce a modularita
  - Hodnoty datových typů
  - Standardní vstup a výstup
- Část 2 – Řidící struktury
  - Řídicí struktury
  - Větvení
  - Cykly
- Část 3 – Zadání 2. domácího úkolu

Část I

Zápis programu v C

I. Zápis programu v C

Funkce a modularita

Hodnoty datových typů

Standardní vstup a výstup

Funkce

- Funkce tvoří základní stavební blok **modulárního** jazyka C

  Modulární program je složen z více modulů/zdrojových souborů.

- Každý spustitelný program v C obsahuje alespon jednu funkci a to funkci `main()`
- **Deklarace** – hlavička funkce (prototyp)

```
navratovy_typ jmeno_funkce(formalni parametry);
```

  Deklarace obsahuje informace, které vyžaduje překladač.

- **Definice** – hlavička funkce a její tělo (sekvence příkazů)

```
navratovy_typ jmeno_funkce(formalni parametry)
{
  // telo funkce
}
```

  Definice funkce bez předchozí deklarace je zároveň deklarací funkce.

Vlastnosti funkcí

- C nepovoluje funkce vnořené do jiných funkcí
- Jména funkcí se mohou exportovat do ostatních modulů (souborů)
- Formální parametry funkce jsou lokální proměnné, které jsou inicializovány skutečnými parametry při volání funkce

  Parametry se do funkce předávají hodnotou
- C dovoluje **rekurzi** – funkce může volat sebe samu
- Funkce nemusí mít žádné vstupní parametry, zapisujeme:

```
jmeno_funkce(void)
```

- Funkce nemusí vracet funkční hodnotu – návratový typ je `void`

Struktura modulu

```c
#include <stdio.h>  /* hlavickovy soubor */

#define PI 3.14     /* symbolicka konstanta - makro */

float kruh(int a);  /* hlavicka/prototyp funkce */

int main(void)      /* hlavni funkce */
{
  int v = 10;       /* definice promennych */
  float r;
  r = kruh(v);      /* volani funkce */
  return 0;         /* ukonceni hlavni funkce */
}

float kruh(int r)   /* definice funkce */
{
  float b = PI*r*r;
  return b;         /* navratova hodnota funkce */
}
```

lec03/modul.c

Zdrojové a hlavičkové soubory

- Rozsáhlejší programy je vhodné rozdělit do více souborů – modulů
- Rozdělení na zdrojové a hlavičkové soubory umožňuje rozlišit **definici** a **deklaraci**, především však podporuje
  - **Organizaci** zdrojových kódů v adresářové struktuře souborů
  - **Modularitu**
    - Hlavičkový soubor – obsahuje popis (seznam) funkcí a jejich parametrů bez konkrétní implementace (deklarace funkcí)

      Bývá zvykem do hlavičkových souborů umisťovat i definice datových typů, konstanty a makra.
    - Zdrojový soubor – konkrétní implementace funkcí modulu
    - Pokud se modul od posledního překladu nezměnil, není třeba zdrojový kód modulu znova překládat

      Optimalizace procesu překladu.
  - **Znovupoužitelnost**
    - Pro využití binární knihovny potřebuje znát její **rozhraní**, které je deklarované v hlavičkovém souboru

      Výhodné pro práci na velkých projektech nebo týmech.

I. Zápis programu v C

Funkce a modularita

Hodnoty datových typů

Standardní vstup a výstup

Literály

- Hodnoty datových typů označujeme jako literály (konstanty)
- Jazyk C má 6 typů konstant (literálů)
  - Celočíselné
  - Racionální
  - Znakové
  - Řetězcové
  - Výčtové

Datový typ enum.

  - Symbolické – `#define NUMBER 10`

Direktivy preprocesoru.

Literály celočíselných datových typů

- Zápis čísel
  - dekadický 123 450932
  - šestnáctkový (hexadecimální) 0x12 0xFAFF (začíná `0x` nebo `0X`)
  - osmickový (oktalový)) 0123 0567 (začíná `0`)
  - unsigned 12345U (přípona `U` nebo `u`)
  - long 12345L (přípona `L` nebo `l`)
  - unsigned long 12345ul (přípona `UL` nebo `ul`)
- Není-li přípona uvedena, jde o literál typu `int`

Literály neceločíselných datových typů

- Neceločíselné datové typy jsou dané implementací, většinou se řídí standardem IEEE-754-1985
- Formát zápisu racionálních literálu:
  - S řádovou tečkou – `13.1`
  - Mantisa a exponent – `31.4e-3` nebo `31.4E-3`
- Typ racionálního literálu:
  - `double` – pokud není explicitne urcen
  - `float` – prípona `F` nebo `f`
  - `long double` – přípona `L` nebo `l`

Znakové literály

- Formát – znak v jednoduchých apostrofech

  `'A'`, `'B'` nebo `'\n'`

- Hodnota – znakový literál má hodnotu odpovídající kódu znaku

  `'0'` 48, `'A'` 65

  Hodnota znaku mimo ASCII (větší než 127) závisí na překladači.

- Typ – znaková konstanta je typu `int`
- Specifikace číselné hodnoty v tabulce znaků (ASCII)
  - hodnota v osmičkové nebo šestnáctkové soustavě, uvozena apostrofy, před vlastní hodnotu zpětné lomítko

    Pokud neuvedeme na začátku 0, je hodnota interpretována jako číslo v osmičkové soustavě tj. `'\32'` je totéž jako `'\032'`, tedy hodnota 32 v osmičkové soustavě (26 v desítkové)

    Příklad řídící znaky. `'\110'`, `'\07'`, `'\0xA3'`

Řetězcové literály

- Formát – posloupnost znaků a řídicích znaků (escape sequences) uzavřená v uvozovkách

  `"Retezcova konstanta s koncem radku\n"`

  - Řetězcové konstanty oddělené oddělovaci (white spaces) se sloučí do jediné, např.

    `"Retezcová konstanta" "s koncem radku\n"`

    se sloučí do

    `"Retezcova konstanta s koncem radku\n"`

- Typ – řetězcová konstanta je uložena v poli typu char a zakoncená znakem `'\0'`

'a'	'h'	'o'	'j'	'\0'

Konstanty výčtového typu

- Formát
  - Implicitní hodnoty konstanty výčtového typu zacínají od 0 a každý další prvek má hodnotu o jedničku vyšší
  - Hodnoty mužeme explicitne predepsat

```
enum {
  SPADES,
  CLUBS,
  HEARTS,
  DIAMONDS
};
```

```
enum {
  SPADES = 10,
  CLUBS, /* hodnota je 11 */
  HEARTS = 3,
  DIAMONDS = 20
};
```

- Typ – výčtová konstanta typu `int`
  - Hodnotu konstanty mužeme použít pro iteraci v cyklu

```
enum {SPADES, CLUBS, HEARTS, DIAMONDS, NUM_COLORS};
for (int i = SPADES; i < NUM_COLORS; ++i) {
...
}
```

Symbolické konstanty

- Formát – konstanta je založena příkazem preprocesoru `#define`
  - Tzv. neparametrické makro
  - Každý `#define` musí být na samostatném řádku

```
#define ZAPOCET 1
```

Vetšinou používáme velká písmena.

- Symbolické konstanty mohou vyjadřovat konstantní výraz

```
#define MAX_1 ((10*6) - 3)
```

- Symbolické konstanty mohu být vnořené

```
#define MAX_2 (MAX_1 + 1)
```

  - Je-li hodnota výraz, jsou kulaté závorky nutné pro správné vyhodnocení výrazu, např. pro 5*MAX_1 s vnějšími závorkami je 5*((10*6) - 3)=285 vs. 5*(10*6) - 3=297.

Proměnné s konstantní hodnotou

- Uvedením klíčového slova (modifikátoru) `const` lze označit proměnnou jako konstantu
- Překladač kontroluje přiřazení a nedovolí hodnotu proměnné nastavit znovu.

Příklad

```
const float pi = 3.14159265;
```

- Na rozdíl od symbolické konstanty mají konstantní proměnné typ a překladač tak může provádět **typovou kontrolu**.

```
#define PI 3.14159265
```

I. Zápis programu v C

Funkce a modularita

Hodnoty datových typů

Standardní vstup a výstup

Standardní výstup – příklad 1/4

```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-format
*931*
*931*
*       931*
*931       *
```

Standardní výstup – příklad 2/4

```
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*
```

Standardní výstup – příklad 3/4

```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*
```

Standardní výstup – příklad 4/4

```
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                  *
```

Standardní vstup – příklad

```
int m,n;
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

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

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

koncept "několik bíých znaků" připouští i variantu žádný bílý znak

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

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

Standardní vstup – příklad

```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
```

Část II

Řídicí struktury

II. Řídicí struktury

Řídicí struktury

Větvení

Cykly

Řídicí struktury

- Části programu, které předepisují způsob provedení dílčích příkazů
- Základní druhy řídících struktur
  - **posloupnost** – postupné provedení příkazů
  - **větvení** – provedení příkazů v případě splnění podmínky
  - **cyklus** – opakované provedení, dokud je splněna podmínka
- Budeme používat
  - složený příkaz a blok pro posloupnost
  - příkazy **if** a **switch** (přepínač) pro větvení
  - příkazy **while**, **do** a **for** pro cyklus

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

Řídicí struktury

Větvení

Cykly

Podmíněný příkaz `if`

- Umožňuje větvení programu na základě podmínky
- Má dva tvary:

```
if (podminka) prikaz1 else prikaz2
if (podminka) prikaz
```

- `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`:

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

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

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

II. Řídicí struktury

Řídicí struktury

Větvení

Cykly

Příkaz cyklu `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

Příkaz cyklu `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

Příkaz cyklu `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:

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

Cykly tohoto druhu lze zkráceně předepsat příkazem `for`:

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

Příkaz cyklu `for`

- Tvar příkazu

```
for (inicializace; podminka; zmena) prikaz
```

- Odpovídá cyklu while ve tvaru:

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

Příklad

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

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

Příkaz cyklu `for` – příklady

- Správný zápis

```
for (i = 0; i < 10; i++)
for ( ; a < 4.0; a += 0.2)
for ( ; i < 10; )
for ( ; ; i++) /* Nekonecny cyklus */
for ( ; ; )    /* Nekonecny cyklus, ekv. while(1) */
```

- Nesprávné použití

```
for ( )      /* Chybi stredniky */
for (i = 1, i == x, i++) /* Carky misto stredniku */
for ( x < 4 )     /* Chybi stredniky */
```

Část III

Zadání domácího úkolu

Zadání 2. domácího úkolu (HW02)

Téma: První cyklus

- **Motivace:** Automatické načítání vstupních dat
- **Cíl:** Využití cyklu jako základní programového konstrukce pro hromadné zpracování dat.
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw02`
  - Zpracování libovolně dlouhé posloupnosti celých čísel
  - Výpis načtených čísel
  - Výpis statistiky vstupních čísel
    - Počet načtených čísel; Počet kladných a záporných čísel a jejich procentuální zastoupení na vstupu
    - Četnosti výskytu sudých a lichých čísel a jejich procentuální zastoupení na vstupu
    - Průměrná, maximální a minimální hodnota načtených čísel
- **Termín odevzdání:** **19.10.2019, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

- Programování v jazyce C
  - Interakce s uživatelem
  - Funkce
  - Hodnoty proměnných
- Řidící struktury
  - Větvení
  - Cykly

- Příště: řidící struktury podrobněji

Diskutovaná témata

- Programování v jazyce C
  - Interakce s uživatelem
  - Funkce
  - Hodnoty proměnných
- Řidící struktury
  - Větvení
  - Cykly

- Příště: řidící struktury podrobněji