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
  - Funkce a modularita
  - Hodnoty datových typů
  - Standardní vstup a výstup
- Část 2 – Řídicí 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) {
  // tělo funkce
  return hodnota_navratoveho_typu;
}
```

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:

```
navratovy_typ jmeno_funkce (void) {
  // tělo funkce
  return hodnota_navratoveho_typu;
}
```

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

I v takovém případě může obsahovat **return**, ovšem bez argumentu

Struktura modulu

lec03/modul.c

```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 */
}
```

Zdrojové a hlavičkové soubory

- Rozsáhlejší programy je vhodné rozdělit do více souborů – **modulů**
- Rozdělení na zdrojové a hlavičkové soubory umožňuje rozlišit **definici** a **deklaraci**
- Dělením do modulů je podporována
  - **Organizace** zdrojových kódů v adresářové struktuře souborů
  - **Modularita**
    - **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

      Optimalizace překladu – pokud se modul od posledního překladu nezměnil, není třeba zdrojový kód modulu znova překládat
  - **Znovupoužitelnost**
    - Pro využití **binární knihovny** je třeba znát její **rozhraní** 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 numerických datových typů

- Celá čísla, pokud není uvedena přípona, jde o literál typu `int`
  - 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`)
  - binární `0b00110011` (začíná `0b` nebo `0B`)

  Není součástí standardu, jedná se o rozšíření (implementované jak v gcc, tak v clang)
- Racionální čísla, pokud není uvedena přípona, kde o literál typu `double`
  - `float` – prípona `F` nebo `f`
  - `long double` – přípona `L` nebo `l`
- Formát zápisu racionálních literálu:
  - s řádovou tečkou – `13.1`
  - mantisa a exponent – `31.4e-3` nebo `31.4E-3`

Znakové literály

- Typ – znaková konstanta je typu `int`
- 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.

- Specifikace číselné hodnoty v tabulce znaků (ASCII)
  - hodnota v oct nebo hex 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'`

- Zajímavosti v ASCII tabulce

dec	bin	hex	ascii
32	00100000	0x20	`' '`
48	00110000	0x30	`'0'`
57	00111001	0x39	`'9'`
65	01000001	0x41	`'A'`
97	01100001	0x61	`'a'`

Změnou bitu lze změnit velikost znaku [A-Za-z]

Odečtením konstanty lze získat číselnou hodnotu

Řetězcové literály

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

  `"Retezcova konstanta s koncem radku\n"`

  - Řetězcové konstanty oddělené oddělovači (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 zakončená znakem `'\0'`

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

- Velikost pole potřebná pro uložení literálu může být automaticky určena (včetně ukončovacího znaku)

```
1  char a[] = "Ahoj";
2  printf("Rezezcovy literal: %s\n", a);
3  printf("Velikost pole s literalem: %ul\n", sizeof(a));
```

Konstanty výčtového typu

- Formát
  - Implicitní hodnoty konstanty výčtového typu začínají od 0 a každý další prvek má hodnotu o jedničku vyšší
  - Hodnoty můžeme explicitně předepsat

```
enum {
  SPADES,   // 0
  CLUBS,    // 1
  HEARTS,   // 2
  DIAMONDS  // 3
};
```

```
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, pro přehlednost používáme velká písmena

```
#define ZAPOCET 1
```

- 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 pro správné vyhodnocení nutné kulaté závorky
  - `5*MAX_1` s vnějšími závorkami je 5*((10*6)-3) = 285
  - `5*MAX_1` bez závorek je 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


```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       *
```

Standardní výstup – příklad


```c
    const double RENT = 3852.99;

    printf("*%f*\n", RENT);
    printf("*%4.2f*\n", RENT);
    printf("*%3.1f*\n", RENT);
    printf("*%10.3f*\n", RENT);
    printf("*%+4.2f*\n", RENT);
    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


```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


```c
#define STRING "Testovaci retezec!"

int main(void)
{
    printf("*%2s*\n", STRING);
    printf("*%24s*\n", STRING);
    printf("*%24.5s*\n", STRING);
    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 bílých 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 – 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

Řídicí struktury

Větvení

Cykly

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 stredníkem ;

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

```
int max;

if (a > b) {
  if (a > c) {
    max = a;
  }
}

```

```
if (a > b) {
  // ...
} else if (a < c) {
  // ...
} else if (a == b) {
  // ...
} else {
  // ...
}

```

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

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

Co se stane, když za příkazem větvení nebude blok?

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

Podmíněný příkaz if – příklad

- 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

Programový přepínač switch – příklad

```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

Řídicí struktury

Větvení

Cykly

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

Cyklus while – příklad

- Program na výpočet faktoriálu přirozeného čísla

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

```
1  #include <stdio.h>
2  #include <stdlib.h>

4  int main(void) {
5    int i = 1, f = 1, n;
6    printf("zadejte prirozene cislo: ");
7    scanf("%d", &n);
8    while (i<n) {
9      i = i+1;
10     f = f*i;
11   }
12   printf("%d! = %d\n", n, f);
13   return 0;
14 }
```

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;
```

Část III

Zadání domácího úkolu

Zadání 3. domácího úkolu (HW03)

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/hw03`
  - 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:** **17.10.2020, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

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

- Příště: řídicí struktury podrobněji, operátory

Diskutovaná témata

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

- Příště: řídicí struktury podrobněji, operátory