

B0B99PRPA – Procedurální programování

Základní řídicí struktury

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Programování v C
 - Zdrojové soubory
 - Funkce
 - Hodnoty datových typů
 - Interakce s uživatelem
- Část 2 – Řídící struktury
 - Řídící struktury
 - Větvení
 - Cykly
- Část 3 – Zadání 2. domácího úkolu

Část I

Zápis programu v C

I. Zápis programu v C

Zdrojové soubory

Funkce

Hodnoty datových typů

Interakce s uživatelem

Zdrojové soubory programu v C

- zdrojový soubor s koncovkou .c
- hlavičkový soubor s koncovkou .h
- Zdrojové soubory jsou překládány do binární podoby překladačem a vznikají objektové soubory (.o)
- Z objektových souborů (object files) se sestavuje výsledný program, ve kterém jsou již všechny funkce známy a relativní adresy se nahradí absolutními

Zdrojové a hlavičkové soubory

- 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 co modul nabízí, tj. popis (seznam) funkcí a jejich parametrů bez konkrétní implementace (deklarace funkcí)
- Konkrétní implementace je pak ve zdrojovém souboru
- 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

Zdrojové soubory

Funkce

Hodnoty datových typů

Interakce s uživatelem

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(seznam parametru);
```

Deklarace obsahuje informace, které vyžaduje překladač.

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

```
navratovy_typ jmeno_funkce(seznam parametru)
{
    // 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 – lokální proměnné jsou pro každé jednotlivé volání zakládány znovu na zásobníku
- Funkce nemusí mít žádné vstupní parametry, zapisujeme:

```
jmeno_funkce(void)
```

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

Struktura modulu

```
1  #include <stdio.h>  /* hlavickovy soubor */
2
3  #define PI 3.14     /* symbolicka konstanta - makro*/
4
5  float kruh(int a);  /* hlavicka/prototyp funkce */
6
7  int main(void)     /* hlavni funkce */
8  {
9      int v = 10;    /* definice promennych */
10     float r;
11     r = kruh(v);   /* volani funkce */
12     return 0;     /* ukonceni hlavni funkce */
13 }
14
15 float kruh(int r)  /* definice funkce */
16 {
17     float b = PI*r*r;
18     return b;     /* navratova hodnota funkce */
19 }
```

I. Zápis programu v C

Zdrojové soubory

Funkce

Hodnoty datových typů

Interakce s uživatelem

Literály

- Hodnoty datových typů označujeme jako literály (konstanty)
- Jazyk C má 6 typů konstant (literálů)
 - Celočíselné
 - Racionální
 - Znakové
 - Řetězcové
 - Výčtové
- Symbolické – `#define NUMBER 10`

Datový typ enum.

Direktivy preprocesoru.

Literály celočíselných datových typů

- Zápis čísel

- dekadický 123 450932
- šestnáctkový (hexadecimální) 0x12 0xFAFF (začíná 0x nebo 0X)
- osmíkový (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 literálů:
 - S řádovou tečkou – `13.1`
 - Mantisa a exponent – `31.4e-3` nebo `31.4E-3`
- Typ racionálního literálu:
 - `double` – pokud není explicitně určen
 - `float` – přípona `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 řídicí 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 oddělovací (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'
-----	-----	-----	-----	------

Konstanty výčtového typu

- Formát

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

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

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

- Typ – výčtová konstanta typu `int`

- Hodnotu konstanty můžeme 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
```

Většinou používáme velká písmena.

- Symbolické konstanty mohou vyjadřovat konstantní výraz

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

- Symbolické konstanty mohou 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

Zdrojové soubory

Funkce

Hodnoty datových typů

Interakce s uživatelem

Standardní vstup a výstup

- Program spuštěný v prostředí operačního systému má přístup ke znakově orientovanému standardnímu vstupu (`stdin`) a výstupu (`stdout`)

Jsou to prostředky operačního systému.

- Prostřednictvím `stdout` a `stdin` program komunikuje s uživatelem

U MCU probíhá komunikace jinými prostředky (např. sériový port).

- Funkce definované v knihovně `<stdio.h>`
 - `putchar()` – výstup (tisk) znaku
 - `getchar()` – vstup (načtení) znaku
 - `printf()` – výstup formátovaného textu
 - `scanf()` – vstup formátovaného textu

Funkce jsou součástí standardní knihovny, nikoliv jazyka C.

Formátovaný výstup – printf()

- Na rozdíl od `putc()` umí tisknout obsah proměnných různých datových typů
- Argumentem funkce je textový řídicí řetězec formátování výstupu
 - Řídicí řetězec formátu je uvozen znakem `'%'`
 - Znakové posloupnosti (nezačínající `%`) se vypíše tak jak jsou uvedeny
 - Důležité řídicí řetězce pro výpis hodnot proměnných

<code>char</code>	<code>%c</code>
<code>_Bool</code>	<code>%i, %u</code>
<code>int</code>	<code>%i, %x, %o</code>
<code>float</code>	<code>%f, %e, %g, %a</code>
<code>double</code>	<code>%f, %e, %g, %a</code>

- je možné specifikovat počet vypsaných míst, zarovnání, atd.

Formátovaný výstup – příklad

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6      int i = 'a';
7      int h = 0x61; /* šestnactkova soustava */
8      int o = 0141; /* osmickova soustava */
9
10     printf("i: %i h: %i o: %i c: %c\n", i, h, o, i);
11     printf("oct: \141 hex: \x61\n");
12
13     return 0;
14 }
```

Formátovaný vstup – scanf()

- Číselné hodnoty ze standardního vstupu lze načíst funkcí `scanf()`
- Argumentem je textový řídicí řetězec

podobně jako `printf()`

- Je nutné předat adresu paměťového místa pro uložení hodnoty

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i;
6
7     printf("Zadej hodnotu int: ");
8     scanf("%i", &i); /* operator & vraci adresu promenne i */
9
10    printf("Zadal jsi %in", i);
11
12    return 0;
13 }
```


Část II

Řídicí struktury

II. Řídicí struktury

Řídicí struktury

Větvení

Cykly

Řídicí struktury

- Části programu, které předepisují způsob provedení dílčích příkazů
- Základní druhy řídicích 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`:

(a)

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

(b)

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

(c)

```
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 řídicí proměnné
while (i<=n) {  // řídicí výraz
    f = f*i;    // tělo cyklu
    i = i+1;    // změna řídicí 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 místo 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 programové 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: 27.10.2018, 23:59:59**



Shrnutí přednášky



Diskutovaná témata

- Programování v jazyce C
 - Interakce s uživatelem
 - Funkce
 - Hodnoty proměnných
- Řídící struktury
 - Větvení
 - Cykly
- Příště: řídicí struktury podrobněji



Diskutovaná témata

- Programování v jazyce C
 - Interakce s uživatelem
 - Funkce
 - Hodnoty proměnných
- Řídící struktury
 - Větvení
 - Cykly
- Příště: řídicí struktury podrobněji