B0B99PRPA – Procedurální programování

Základy programování v C

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Programování v jazyce C
  - Charakteristika jazyka
  - Překlad
  - Proměnné
  - Základní datové typy
  - Výrazy a operátory
  - Formátovaný vstup a výstup
- Část 2 – Zadání 1. domácího úkolu (HW01)

Část I

Programování v jazyce C

I. Programování v jazyce C

Charakteristika

- Nízkoúrovňový – přístup k nízkoúrovňovým funkcím počítače

  Operace souvisí s operacemi definovanými platformou.

- Malý

  Malá množina výrazových prostředků + knihovna standardních funkcí.

- Procedurální (ne objektový)

  Programy se skládají pouze z funkcí a dat.

- Kompilovaný – překlad do strojového kódu

  Interpretované – potřebují běhové prostředí

- Staticky typovaný – kontrola typu proměnné při kompilaci

  Dynamicky typované jazyky – kontrola typu za běhu.

- Imperativní – chování programu je popsáno algoritmem

  Behaviorální – popis vstupů a výstupů.

Zápis programu

- Zdrojový kód
  - běžný textový soubor
  - zdrojové soubory zpravidla s příponou .c
  - hlavičkové soubory s koncovkou .h
- Kompilace
  - kompilací zdrojového kódu vzniká binární objektový kód
  - z objektových souborů se sestavuje spustitelný kód
- Příklad

```
1  #include <stdio.h>
2
3  int main()
4  {
5    printf("Uz programuju!\n");
6    return 0;
7  }
```

kompilace zrojového kódu

```
$ clang hello.c
```

vznikne spustitelný soubor a.out
tento soubor lze spustit, např.

```
$ ./a.out
```

program vypíše agument funkce printf

```
$ Uz programuju!
```

Struktura zdrojového kódu

```
1  /* vlozeni hlavickoveho souboru
       standardni knihovny */
2  #include <stdio.h>
3  // hlavicka funkce main()
4  int main()
5  { // hlavni funkce programu
6    /* volani funkce printf()
       definovane ve knihovne stdio.h
       */
7    printf("Uz programuju!\n");
8    /* ukonceni funkce a predani
       navratove hodnoty 0 operacnimu
       systemu */
9    return 0;
10 }
```

Identifikátory

- Jména proměnných, funkcí a typů
- Alfanumerické znaky a podtržítko, case sensitive

Komentáře

- Text umístěný mezi dvojici párových symbolů `/*` a `*/`
- Je možné i použití `//`

Odsazovače

- Mezera, tabelátor, nový řádek, ...

Závorky

- `{}`, `()`, `<>`

Klíčová slova

Žádný identifikátor nemůže mít v době překladu stejný název

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Překlad a sestavení programu

- Překlad je posloupností tří dílčích úloh, které se zpravidla provádí automaticky, ale lze je provést i individuálně.
  1. Textové předzpracování **preprocesorem**, který má vlastní makro jazyk (příkazy uvozeny znakem #)

     Odkazované hlavičkové soubory se vloží do jediného zdrojového souboru

     ```
     $ clang -E program.c -o program.i
     ```

  2. Vlastní **překlad** zdrojového souboru do objektového souboru

     Zpravidla jsou jména souboru zakoncena príponou .o

     ```
     $ clang -c program.c -o program.o
     ```

     Příkaz kombinuje volání preprocesoru a kompilátoru.

  3. Spustitelný soubor se sestaví z příslušných dílčích objektových souborů a odkazovaných knihoven, tzv. linkováním (**linker**), např.

     ```
     $ clang program.o -o program
     ```

I. Programování v jazyce C

Schéma překladu a sestavení programu

Vývoj programu v C

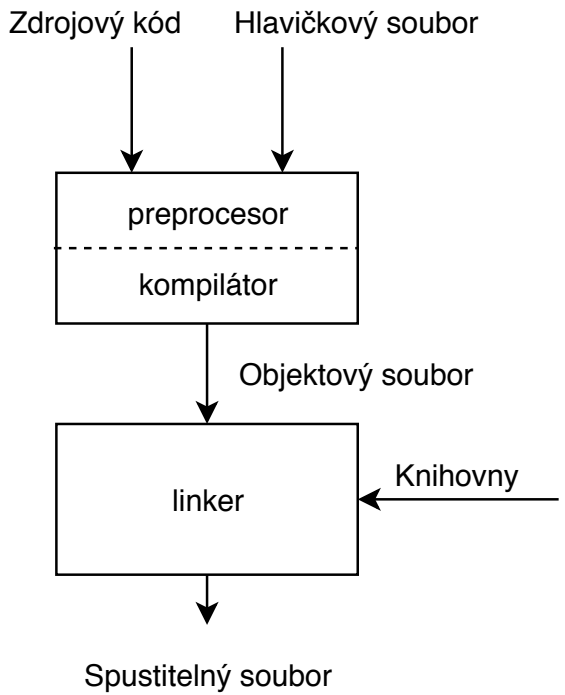
- editace zdrojových souborů

  (.c a .h)

- kompilace dílčích zdrojových souborů do objektových souborů

  (.o nebo .obj)

- linkování přeložených souborů do spustitelného programu
- spouštění a ladění aplikace
- opětovná editace zdrojových souborů



Dílčí kroky překladu a sestavení programu

- **preprocesor** – umožňuje definovat makra a tím přizpůsobit překlad aplikace kompilačnímu prostředí

  Výstupem je zdrojový soubor – text.

- **kompilátor** – překládá zdrojový (textový) soubor do strojově čitelné (a spustitelné) podoby

  Nativní (strojový) kód platformy, bytecode, případně assembler.

- **linker** – sestavuje program z objektových souborů do podoby výsledné aplikace

  Stále může odkazovat na knihovní funkce (dynamické kni- hovny linkované při spuštění programu), může též obsahovat volání OS (knihovny).

Dílčí části (preprocesor, kompilátor a linker) jsou zpravidla součástí jednoho programu (clang, gcc), který lze spouštět s různými parametry.

Překladače jazyka C

- V rámci předmětu budeme používat především překladače
  - **gcc** – GNU Compiler Collection

    https://gcc.gnu.org
  - **clang** – C language family frontend for LLVM

    http://clang.llvm.org

  Základní použití (přepínače a argumenty) je u obou překladačů stejné.
- Pro win* platformy existují odvozená prostředí cygwin nebo MinGW

  https://www.cygwin.com/, http://www.mingw.org/

Příklad

```
$ gcc -c program.c -o program.o
$ gcc program.o -o program
```

I. Programování v jazyce C

Príklad součtu dvou hodnot

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int sum; /* definice lokalni promenne typu int */
6
7    sum = 100 + 43; /* hodnota vyrazu se ulozi do sum */
8    printf("Soucet 100 a 43 je %i\n", sum);
9    /* %i formatovaci prikaz pro tisk celeho cisla */
10   return 0;
11 }
```

- Proměnná `sum` typu `int` reprezentuje celé číslo, jehož hodnota je uložena v paměti
- `sum` je námi zvolené symbolické jméno místa v paměti, kde je uložena celočíselná hodnota (typu `int`)

Příklad součtu hodnot dvou proměnných

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int var1;
6    int var2 = 10; /* inicializace hodnoty promenne */
7    int sum;
8
9    var1 = 13;
10   sum = var1 + var2;
11
12   printf("The sum of %i and %i is %i\n", var1, var2, sum);
13   return 0;
14 }
```

- Proměnné `var1`, `var2` a `sum` reprezentují tři různá místa v paměti (automaticky přidělené), ve kterých jsou uloženy tři celočíselné hodnoty.

I. Programování v jazyce C

Celočíselné datové typy

int, short, long, char

char – celé číslo v rozsahu jednoho bajtu nebo také znak

- velikost paměti alokované příslušnou (celo)číselnou proměnnou se může lišit dle architektury počítače nebo překladače
- typ int má zpravidla velikost 4 bajty a to i na 64-bitových systémech
- aktuální velikost paměťové reprezentace lze zjistit operátorem sizeof(), kde argumentem je jméno typu nebo proměnné.

```
1 int i;
2 printf("%lu\n", sizeof(int));
3 printf("ui size: %lu\n", sizeof(i));
```

Neceločíselné datové typy

float, double

- Umožňují zobrazit aproximace racionálních čísel v určitém rozsahu a s určitou přesností
- Jsou dané implementací, většinou dle standardu IEEE-754-1985
  - `float` – 32-bit IEEE 754, reálné číslo (floating number)
  - `double` – 64-bit IEEE 754, číslo s dvojnásobnou přesností
- Reprezentace reálných čísel $n = m \times b^e$
- IEEE reprezentace 32-bit (4B) čísla float má
  - 1 znaménkový bit (znaménko mantisy)
  - exponent – 8 bitů, je v rozsahu $\langle -126; 127 \rangle$, ale není uložen se znaménkem, nýbrž posunut o hodnotu 127, takže exponent -126 je uložen jako hodnota 1, exponent 127 se stane 254 atd.
  - mantisa – 23 bitů, normalizována do intervalu $\langle 1.0; 2.0 \rangle$.

`http://www.h-schmidt.net/FloatConverter`

Znaménkové celočíselné typy

- Specifikace, zda typ zobrazuje pouze s kladná čísla, nebo i záporná
  - **signed** (implicitní)

    `signed char` – rozsah -128 - 127
  - **unsigned**

    `unsigned char` – rozsah 0 - 255

nemůže zobrazit záporné číslo

```
1 unsigned char A = 127;
2 char B = 127;
3 printf("Promenne A=%i a B=%i\n", A, B);
4 A = A + 2;
5 B = B + 2;
6 printf("Promenne A=%i a B=%i\n", A, B);
```

Datový typ char - znak

Reprezentuje celé číslo (byte) nebo znak (gr. symbol)

Kódování ASCII – American Standard Code for Information Interchange.

- znaménkový, implicitní specifikátor záleží na kompilátoru
- lze s nimi používat aritmetické nebo relační operátory a pod.
- hodnotu znaku lze zapsat jako tzv. znakovou konstantu

```
1 char C = 'a';
2 printf("Hodnota C=%i nebo znak '%c'\n", C, C);
3 C = C + 1;
4 printf("Hodnota C=%i nebo znak '%c'\n", C, C);
```

Řídící znaky

- \n – new line, \t – tabular, \r – carriage return
- \a – beep, \b – backspace, \f – form feed, \v – vertical space

Logický datový typ

- Ve verzi **C99** je zaveden logický datový typ `_Bool`
- Jako hodnota true je libovolná hodnota typu `int` různá od 0
- Lze využít hlavičkového souboru `<stdool.h>`, kde je definován typ `bool` a hodnoty true a false

```
#define false 0
#define true 1
#define bool _Bool
```

- **ANSI C** explicitní datový typ pro logickou hodnotu nedefinuje.
- Běžně se používá podobná definice jako v `<stdbool.h>`

```
#define FALSE 0
#define TRUE 1
```

Rozsahy celočíselných typů

- Rozsahy celočíselných typů v C nejsou dány normou, ale implementací
  - Mohou se lišit implementací a prostředím 16 bitů vs 64 bitů
  - Bývá zvykem, že int je 4 bytový
- Norma garantuje, že pro rozsahy typů platí
  - `short` $\leq$ `int` $\leq$ `long`
  - `unsigned short` $\leq$ `unsigned` $\leq$ `unsigned long`
- Pokud potřebujeme specifickou velikost, lze použít datové typy definované např. v `<stdint.h>`

IEEE Std 1003.1-2001

`int8_t`	`uint8_t`
`int16_t`	`uint16_t`
`int32_t`	`uint32_t`

I. Programování v jazyce C

Výraz

- Výraz předepisuje výpočet hodnoty určitého vstupu
- Struktura výrazu obsahuje operandy, operátory a závorky
- Výraz může obsahovat
  - proměnné
  - konstanty
  - operátory
  - závorky
  - volání funkcí
- Pořadí operací předepsaných výrazem je dáno prioritou a asociativitou operátorů

Operátory

- Znaky (nebo posloupnost znaků) vyhrazené pro zápis výrazů
- 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 promenou: `++` a `--`
  - logický operátor doplněk: `!`
  - operátor pretypování: `(jméno typu)`
- Ternární operátor
  - podmíněné přiřazení hodnoty

Proměnné a přiřazení

- Proměnné definujeme uvedením typu a jména proměnné
  - Jména proměnných volíme malá písmena
  - Víceslovná jména zapisujeme s podtržítkem (nebo **CamelCase**)
  - Proměnné definujeme na samostatném řádku

```
int n;
int number_of_items;
int LongNameOfVariable;
```

- Přiřazení je nastavení hodnoty proměnné, tj. uložení definované hodnoty na místo v paměti, kterou proměnná reprezentuje
- Tvar přiřazovacího operátoru: **proměnná** = **výraz**

  Výraz je literál, proměnná, volání funkce,...
- **Příkaz přiřazení** se skládá z operátoru přiřazení = a ;
  - Levá strana přiřazení musí být **l-value** – location-value, left-value

    Tj. musí reprezentovat paměťové místo pro uložení výsledku.
  - Přiřazení je výraz a můžeme jej použít všude, kde je dovolen výraz příslušného typu

Aritmetické operátory

- Pro operandy číselných typů jsou definovány operátory
  - unární operátor změna znaménka -
  - binární sčítání +, odčítání −, násobení ∗ a dělení /
- Pro operandy celočíselných typů pak dále
  - binární operátor zbytek po dělení %
- Pro oba operandy stejného typu je výsledek aritmetické operace stejného typu
- V případě kombinace typu **int** a **double**, se **int** převede na **double** a výsledek je hodnota typu double.

Implicitní typová konverze.

Příklad – Aritmetické operátory 1/2

```
1  int a = 10;
2  int b = 3, c = 4;
3  int d = 5, result;
4
5  result = a - b; // rozdil
6  printf("a - b = %i\n", result);
7  result = a * b; // nasobeni
8  printf("a * b = %i\n", result);
9  result = a / b; // celociselne deleni
10 printf("a / b = %i\n", result);
11
12 result = a + b * c; // priorita operatoru
13 printf("a + b * c = %i\n", result);
14
15 printf("a * b + c * d = %i\n", a * b + c * d); // -> 50
16 printf("(a * b) + (c * d) = %i\n", (a * b) + (c * d)); // -> 50
17 printf("a * (b + c) * d = %i\n", a * (b + c) * d); // -> 350
```

Příklad – Aritmetické operátory 2/2

```
1  int x1 = 1;
2  double y1 = 2.2357;
3  float x2 = 2.5343f;
4  double y2 = 2;
5
6  printf("P1 = (%i, %f)\n", x1, y1);
7  printf("P1 = (%i, %i)\n", x1, (int)y1);
8  // operator pretypovani (double)
9  printf("P1 = (%f, %f)\n", (double)x1, (double)y1);
10 printf("P1 = (%.3f, %.3f)\n", (double)x1, (double)y1);
11
12 printf("P2 = (%f, %f)\n", x2, y2);
13 // implicitni konverze na float, resp.  double
14 double dx = (x1 - x2);
15 double dy = (y1 - y2);
16
17 printf("(P1 - P2)=(%.3f, %0.3f)\n", dx, dy);
18 printf("|P1 - P2|^2=%.2f\n", dx * dx + dy * dy);
```

I. Programování v jazyce C

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 vypíší tak jak jsou uvedeny
  - Důležité řídicí řetězce pro výpis hodnot proměnných

char	`%c`
_Bool	`%i`, `%u`
int	`%i`, `%x`, `%o`
float	`%f`, `%e`, `%g`, `%a`
double	`%f`, `%e`, `%g`, `%a`

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

Formátovaný vstup – `scanf()`

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

podobně jako 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

Zadání domácího úkolu

Zadání 1. domácího úkolu (HW00)

Téma: Načítání vstupu, výpočet a výstup

- **Motivace:** Získat představu o interakci uživatele s programem
- **Cíl:** Program pro načítání vstupu, formátovaného výstupu a základní posloupnosti příkazů
- **Zadání:** `https://cw.fel.cvut.cz/wiki/courses/b0b99prpa/hw/hw01`
  - Načítání celých čísel ze standardního vstupu
  - Výpis čísel v dekadické a šestnáckové soustavě
  - Provedení základní aritmetických operací s načtenými čísly
  - Dodržení správného formátování výstupu
- **Termín odevzdání:** **20.10.2018, 23:59:59**

Shrnutí přednášky

Diskutovaná témata

- Programování v jazyce C
  - Charakteristika
  - Překlad
- Proměnné
- Základní datové typy
- Výrazy a operátory
- Formátovaný vstup a výstup

- Příště: Základní řídící struktury

Diskutovaná témata

- Programování v jazyce C
  - Charakteristika
  - Překlad
- Proměnné
- Základní datové typy
- Výrazy a operátory
- Formátovaný vstup a výstup

- Příště: Základní řídící struktury