5. Ukazatele

B0B99PRPA – Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Adresování

- Každá proměnná v paměti má tři hlavní charakteristiky
  - Jméno (identifikátor)
  - Obsah (hodnota)
  - Adresa – jednoznačná identifikace obsazeného místa v paměti

Vyjímkou jsou proměnné v paměťové třídě register.

- **Přímé adresování** – přístup k obsahu na adrese prostřednictvím jména proměnné
- **Nepřímé adresování** – přístup prostřednictvím jiné proměnné, která obsahuje adresu – tzv. reference
  - nepřímé adresování umožňuje funkcím přistupovat k paměti alokované mimo tělo funkce
  - nepřímé adresování umožňuje funkcím mít vic než jednu návratovou hodnotu (viz `scanf`)

Ukazatel (pointer)

- Ukazatel (pointer) je proměnná, jejíž hodnota je adresa v paměti
- Pointer může být inicializován adresou jiné proměnné

  Odkazuje na oblast paměti, kde je uložena hodnota proměnné.

- Ukazatel má typ proměnné, na kterou může ukazovat

  Důležité pro ukazatelovou aritmetiku

- Ukazatel může odkazovat na
  - proměnné všech typů – primitivní i strukturované
  - funkce
  - jiné ukazatele
- Ukazatel může být též bez typu (void)
  - Velikost proměnné nelze z vlastnosti ukazatele určit
  - Pak může obsahovat adresu libovolné proměnné
- Prázdná adresa ukazatele je definovaná hodnotou konstanty `NULL`

Definice ukazatele

- Deklarace ukazatele

```
1  // ukazatel na promennou typu char
2  char *p1;
3  // ukazatel na promennou typu int
4  int *p2;
```

- Inicializace ukazatele

```
1  char a;
2  // ukazatel p1 inicializovan adresou promenne a
3  p1 = &a;
4  int b;
5  // ukazatel p2 inicializovan adresou promenne b
6  p2 = &b;
```

- Referenční operátor `&` – vrací adresu paměti, kde je uložena hodnota proměnné, před kterou je uveden

Dereferenční operátor

- Vrací l-hodnotu (l-value) odpovídající hodnotě na adrese ukazatele
- Umožňuje číst a zapisovat hodnotu na adrese dané obsahem ukazatele

Příklad lec05/pointer.c

```c
#include <stdio.h>

int main()
{
  int a = 10;
  int *p = &a;
  printf("a = %i\n", a);
  *p = 20;
  printf("a = %i\n", a);

  return 0;
}
```

```
a = 10
a = 20
```

Ukazatele, proměnné a jejich hodnoty

- Proměnné jsou názvy adres, kde jsou uloženy hodnoty příslušného typu
- Kompilátor pracuje přímo s adresami

V případě kompilace se zpravidla jedná o adresy relativní.

- Ukazatel je proměnná, ve které je uložena adresa. Na této adrese se pak nachází hodnota nějakého typu (např. int).
- Ukazatele realizují tzv. nepřímé adresování
- Dereferenční operátor přistupuje na proměnnou adresovanou hodnotou ukazatele
- Operátor `&` vrací adresu, kde je uložena hodnota proměnné

Ukazatele a kódovací styl

- Symbol * lze zapisovat u identifikátoru proměnné nebo typu
- Preferujeme zápis u proměnné, abychom předešli omylům

```
char* a, b, c;    // ukazatel je pouze a
char *a, *b, *c;  // vsechny promenne jsou ukazatele
```

- Ukazatel na ukazatel: `char **a;`
- Typ ukazatel: `char*`, `char**`
- Neplatná adresa má symbolické jmého `NULL`
  - makro preprocesoru
  - v C99 lze i 0
  - proměnné v C nejsou automaticky inicializovány a ukazatele tak mohou odkazovat na neplatnou paměť

Předávání parametrů funkcím

- V C jsou **parametry funkce předávány hodnotou**
- Parametry jsou lokální proměnné funkce (alokované v zásobníku), které jsou inicializované na hodnotu předávanou funkci

```
1  void fce(int a, char *b) {
2      /*  a - lokalni promena typu int (v zasobniku)
3          b - lokalni promena typu ukazatel na promenou typu
4              char (hodnota je adresa, take v zasobniku)  */
5  }
```

- Lokální změna hodnoty proměnné neovlivnuje hodnotu proměnné vně funkce
- Při předání ukazatele máme přístup na adresu původní proměnné, kterou můžeme měnit
- Ukazatelem tak realizujeme **volání odkazem**

Předávání parametrů funkcím – příklad

```c
void fce(int a, char* b) { // a - volání hodnotou, b - volání odkazem
  a += 1;
  (*b)++;
}
//--
int a = 10;
char b = 'A';
printf("%13s a=%d b=%c\n", "Pred volanim:", a, b);
fce(a, &b);
printf("%13s a=%d b=%c\n", "Po volani:", a, b);
```

lec05/function-call.c

```
Pred volanim: a=10 b=A
   Po volani: a=10 b=B
```