Ukazatele - Pointery

Pavel Čeleda

celeda@liberouter.org

Kurz jazyka C - přednáška č. 5

Ukazatele - Pointery

- pointer představuje adresu v paměti a na této adrese se ukrývá příslušná hodnota,

hodnota	25 →	18
symbolická adresa	`p_i`	`*p_i`
absolutní adresa	87	25

  - proměnná `p_i` je pointer,
  - hodnota `p_i` je 25,
  - číslo 25 představuje absolutní adresu v paměti,
  - na absolutní adrese je v paměti hodnota 18,
  - `p_i` ukazuje na hodnotu 18, ale sám má hodnotu 25.
- * označujeme jako *dereferenční operátor*,
- & označujeme jako *referenční operátor*.

Ukazatele - Pointery

- získání obsahu na adrese `i = *p_i;`
- zapsání hodnoty na adresu `*p_i = 5;`
- pointer je vždy svázán s datovým typem, správné označení „`pointer na typ ...`“,
- definice proměnné typu pointer na typ **int** je `int *p_i;`
- získání libovolné proměnné pomocí referenčního operátoru `&`
  např. `int i, *p_i=&i;` nebo `p_i=&i;`
- příklady přiřazení pointerů

```
*p_i = 1;
*(p_i+3) = 5;
p_i = &i;
```

- přiřazení paměti pro přímý přístup do paměti (mikropočítače)

```
unsigned char *p_mem;
p_mem = (unsigned char *) 0x80;
```

Použití pointerů v přiřazovacích příkazech

- *statické* - přiřazení je správně v době překladu
  levá strana musí být stejného typ jako pravá strana
- *dynamické* - přiřazení je správně v době překladu i při běhu programu
  - levá strana musí být stejného typ jako pravá strana
  - přiřazovat jen přes inicializované nebo správně nastavené pointery

Použití pointerů v přiřazovacích příkazech

- platí `int i, *p_i;`

a) staticky správné:

`i = 3;`	do `i` dá hodnotu 3
`*p_i = 4;`	na adresu v `p_i` dá hodnotu 4
`i = *p_i;`	do `i` dá obsah z adresy v `p_i`
`*p_i = i;`	do `p_i` dá obsah `i`
`p_i = &i;`	naplní `p_i` adresou `i`

b) staticky nesprávné:

`p_i = 3;`	do `p_i` je dána hodnota 3 (absolutní adresa 3)
`i = p_i;`	do `i` se dá obsah `p_i` (adresa)
`i = &p_i;`	do `i` se dá adresa `p_i`

Použití pointerů v přiřazovacích příkazech

- platí `int i, *p_i;`

a) dynamicky správné:

`p_i = &i; *p_i = 4;` je totéž jako `i = 4;`

b) staticky nesprávné:

`*p_i = 4;` 4 je přiřazeno na náhodnou adresu v `p_i`

Priority

- operátor * má vyšší prioritu než operátor +
  `i = *p_i + 1;` → `i = (*p_i) + 1;`
- operátor ++ má stejnou prioritu jako operátor *
  `i = *p_i++` → `i = *p_i; p_i++;`
  tato operace se často využívá při práci s řetězci
- vypsání adresy která je uložena v pointeru (hodnota pointeru)

```
int i, *p_i = &i;
printf("Adresa promenne i je %p, hodnota p_i je
%p", &i, p_i);
```

Nulový pointer NULL

- symbolická konstanta definovaná v `stdio.h`
  `#define NULL 0` nebo `#define NULL ((void *)0)`
- lze přiřadit bez přetypování všem typům pointerů,
- používá se pro označení, že tento pointer neukazuje momentálně na nic, tzn. nemá přidělenou paměť

Konverze pointerů

- typické použití při dynamickém přidělování paměti

```
char *p_c;
int *p_i;
p_c = p_i; /* nevhodne */
p_c = (char *) p_i; /* lepsi */
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);

void vymen(int *p_x, int *p_y)
{




}
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);

void vymen(int *p_x, int *p_y)
{
   int pom;




}
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);

void vymen(int *p_x, int *p_y)
{
   int pom;

   pom = *p_x;


}
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);

void vymen(int *p_x, int *p_y)
{
   int pom;

   pom = *p_x;
   *p_x = *p_y;

}
```

Pointery a funkce - volání odkazem

```
int i=5, j=3;
vymen(&i, &j);

void vymen(int *p_x, int *p_y)
{
   int pom;

   pom = *p_x;
   *p_x = *p_y;
   *p_y = pom;
}
```

Pointery na fce a fce jako parametry funkcí

- fce vracející pointer na datový typ
  `char *najdi_adresu_znaku(char c)`
- proměnná jako pointer na funkci vracející nějaký typ
  `double (*p_fd)();`
- POZOR na podobnost
  - `double (*p_fd);` je totožné s `double *p_fd;` $\longrightarrow$ pointer na double, nikoliv pointer na fci vracející double
  - `double *p_fd();` deklarace fce **p_fd**, která vrací pointer na double

Jak číst komplikované definice - I.

- `int x;` ... `x` je typu **int**
- `float *y;` ... `y` je pointer na typ **float**
- `double *z();` ... `z` je funkce vracející pointer na **double**
- `int *(*v)();` ... `v` je pointer na funkci vracející pointer na **int**

Pointerová aritmetika

- Platné operace s pointery
  - součet pointeru a celého čísla
  - rozdíl pointeru a celého čísla
  - porovnání pointerů stejných typů
  - rozdíl dvou pointerů stejných typů
- operátor **sizeof()** - `i = sizeof(*p_i);`
  `i` bude obsahovat počet bajtů nutných pro uložení objektu na který ukazuje `p_i`

Dynamické přidělování paměti

- přidělování paměti za chodu programu (hromada angl. heap),
- přidělení paměti `malloc()`, `calloc()`
- uvolnění paměti `free()`
- funkce jsou uvedeny v souboru `stdlib.h`

Přidělení paměti - malloc()

- `void *malloc(size_t size);`
- vrací adresu prvního přiděleného prvku
- neníli v paměti dost místa vrací NULL !!!
- ```
  int *p_i;
  if((p_i = (int*) malloc(100)) == NULL)
  {
   printf("Malo pameti!");
   exit(1);
  }
  ```
  ```

Přidělení paměti - calloc()

- `void *calloc(size_t n, size_t size);`
- slouží pro alokaci **n** prvků o velikosti **size**
- odpovídá volání fce `malloc(n * size)`
- ```
  int *p_i;
  if((p_i = (int*) calloc(100, sizeof(int))) == NULL)
  {
   printf("Malo pameti!");
   exit(1);
  }
  ```
  ```

Uvolnění paměti - free()

- `void free(void *ptr);`
- vracejte nepotřebnou paměť zpět
- ```
  free((void *)p_c);
  p_c = NULL;
  ```
  ```