Ukazatele a práce s pamětí

19. prosince 2011

Programovací jazyky C a C++ umožňují provádět některé operace přímo využitím adres míst v paměti počítače, kde jsou data uložená. Například v případě používání funkcí, kdy výsledkem funkce může být jen jediná hodnota, je možné použít adresu paměti k tomu, aby funkce vrátila jako výsledek adresu místa paměti, kde je uloženo celé pole hodnot.

Pro práci s adresami se využívá speciálního datového typu, kterému se říká **ukazatel** (v angličtině pointer). Je to zvláštní nový typ proměnných, jejichž hodnotou je adresa místa v paměti počítače, kde je uložena hodnota konkrétního datového typu: `int`, `double` nebo `char`. Jednotlivé ukazatele se proto liší tím, na jaký datový typ ukazují, což je nezbytné stanovit v okamžiku jejich definice.

Definice ukazatelů `p` a `q` na datový `int` typ vypadá následovně:

```
int *p, *q;
```

Jak je vidět, definice ukazatele se od definice ostatních typů liší jen tím, že před názvem proměnné uvedeme symbol `*`.

Jak bylo řečeno, hodnotou ukazatele je adresa v paměti počítače, kde je uložen příslušný datový typ nebo jiná proměnná příslušného datového typu. Nejprve si ukážeme, jak můžeme zajistit, aby ukazatel ukazoval na jinou proměnnou (na místo v paměti, kde je uložená):

```
int a, *p;
p = &a;
```

Když definujeme proměnnou `a` typu `int`, pak symbol `a` představuje v kódu programu vždy její hodnotu. Pro získání adresy (neboli reference), kde je hodnota proměnné uložená, je třeba použít tzv. **referenční operátor** `&`, takže `&a` představuje adresu místa v paměti, kde je uložená hodnota proměnné `a`. Jinými slovy, **použitím operátoru `&` nepracujeme s hodnotou proměnné, ale s její adresou**. Příkazem `p=&a;` jsme zajistili, že ukazateli `p` byla přiřazena adresa proměnné `a`, neboli že ukazatel `p` ukazuje na proměnnou `a`.

Jelikož ukazatel `p` obsahuje informaci o místě uložení proměnné `a`, je nyní možné měnit hodnotu proměnné `a` pomocí ukazatele `p`. Následujícím příkazem nastavíme hodnotu proměnné `a` na 2:

```
*p = 2;
```

Použili jsme k tomu tzv. **dereferenční operátor** `*`. Zatímco symbol `p` představuje v kódu programu adresu místa v paměti, `*p` představuje přímo hodnotu uloženou na příslušné adrese. Jinými slovy, **použitím operátoru * nepracujeme s adresou, ale přímo s hodnotou, na kterou ukazatel ukazuje**.

Nyní si ukážeme některé přiřazovací příkazy s ukazateli:

```
p  = &a;   //do p se ulozi adresa a
a  = 3;    //do a se ulozi hodnota 3
*p = 4;    //na adresu v p se ulozi hodnota 4
a  = *p;   //do a se ulozi hodnota na adrese v p
*p = a;    //na adresu v p se ulozi hodnota a
p  = 3;    //CHYBA: p je adresa, 3 je hodnota typu int, nelze priradit
i  = p;    //CHYBA: i je typu int, p je adresa, nelze priradit
i  = &p;   //CHYBA: i je typu int, &p je adresa ukazatele, nelze priradit
```

Tři poslední uvedené příkazy jsou chybné a na jejich nesprávnost Vás upozorní přímo překladač, protože jejich nesprávnost spočívá v rozdílnosti datových typů na levé a pravé straně příkazu. Zvláštní pozornost je třeba věnovat příkazu `*p = 4;`, který je ze statického hlediska správně, to znamená, že překladač žádnou chybu neohlásí, ale přitom příkaz správně být nemusí. Např. pokud v programu napíše následující dva příkazy hned po sobě, může po spuštění program spadnout:

```
int *p;
*p = 4;
```

Toto je nejčastější chyba v používání ukazatelů a je na ní velice nepříjemné, že chování výsledného programu není zcela předvídatelné a proto bývá problematické tuto chybu objevit. Jak jsme si říkali již v dřívější přednášce, při definici proměnné se pro proměnnou alokuje místo v paměti, kde ovšem může být uložena nějaká hodnota z přechozího užívání této paměti. Proto při definici ukazatele `int *p;` má tento ukazatel hodnotu nějaké neznámé adresy a nevíme, zda je paměť na této adrese možné používat či nikoliv. Následujícím příkazem `*p = 4;` se pokoušíme do této paměti zapsat hodnotu 4, což se tedy někdy může zdařit, někdy program může spadnout. V každém případě se pokoušíme zapisovat do paměti, kterou nemáme plně pod kontrolou, což je chybné.

Hodnotu ukazatele je možné nastavit také na jakousi *nulovou* hodnotu. Pro tento účel se používá speciální konstanta `NULL`.

```
int *p;
p = NULL;
```

Pokud si chceme nějakou konkrétní adresu vypsat, použijeme ve funkci `printf()` formát `%p`. Následující příkaz vypíše adresu proměnné `a` a hodnotu ukazatele `p`:

```
printf( "Adresa a je: %p a hodnota p je: %p\n", &a, p );
```

Abychom mohli pomocí ukazatele ukládat hodnoty do jiné paměti, než kde jsou uložené již existující proměnné, použijeme příkaz pro alokaci nové paměti. Tomuto typu alokace paměti se říká dynamická alokace. Následující příkaz je součástí jazyka C++, nikoli však jazyka C, proto je pro překlad pomocí překladače Borland 5.5 třeba ukládat zdrojový kód s příponou `.cpp`.

```
p = new int;
```

V jazyce C se pro dynamickou alokaci paměti používá jiných funkcí, které jsou mimo rámec tohoto kurzu. Uvedeným příkazem se v paměti vyhledá volné místo o velikosti jedné proměnné typu `int`. Na toto místo je teď možné ukládat požadovanou hodnotu výše zmíněným příkazem, např. `*p = 4;`.

Dynamická alokace paměti se od statické alokace liší. Hlavní rozdíl se projeví zejména při použití dynamické alokace uvnitř funkce. Paměť obsazená staticky alokovanými proměnnými se po ukončení funkce automaticky uvolní. Dynamicky alokovaná paměť však nikoliv. Tu musí programátor uvolnit sám ve chvíli, kdy už ji nepotřebuje. K uvolnění paměti je třeba znát její adresu, tzn. hodnotu ukazatele, kterého jsme pro alokaci použili. Uvolnění se provádí následujícím příkazem:

```
delete p;
```

V uvedeném příkladu je třeba mít na paměti, že dynamicky je alokovaná pouze paměť, na kterou ukazatel `p` ukazuje. Samotný ukazatel `p` je alokován staticky, takže jeho hodnota obsahující adresu dynamicky alokované paměti bude po ukončení funkce ztracena, pokud tuto hodnotu nepoužijeme jako návratovou hodnotu funkce.

Na závěr si jen připomeneme, kde jsme až do teď ukazatele používali, aniž bychom o tom věděli:

- Ve druhé přednášce jsme se setkali s funkcí `scanf()`, kterou používáme pro načtení hodnot z konzole a jejich uložení do zvolených proměnných. Tyto proměnné do funkce posíláme jako její parametry a právě proto, aby funkce mohla změnit jejich hodnotu, posíláme do této funkce ve skutečnosti adresy proměnných pomocí referečního operátoru `&`.
- V přechozí přednášce jsme se setkali s proměnnou, s jejíž pomocí se odkazujeme na datový soubor, ze kterého chceme číst, či do něho zapisovat. Tuto proměnnou ve skutečnosti definujeme jako ukazatel na soubor a proto při definici používáme symbolu `*`.