

PB111 Nízkoúrovňové programování (shrnutí jazyka C)

P. Ročkai

Část 1: Výpočetní stroj 1 | Část 2: Lokální proměnné, řízení toku 2 |

Část 1: Výpočetní stroj

Tato kapitola jazyk C nepoužívá.

Část 2: Lokální proměnné, řízení toku

Počínaje touto kapitolou budeme většinu programů psát ve zjednodušené verzi jazyka C. V tomto kurzu budeme psát programy do jednoho souboru, který bude sestávat z definic typů (uvidíme později) a podprogramů. Na diskusi o sémantice podprogramů zatím nejsme připraveni, proto je budeme chápat jako syntaktickou obálku pro kód, který budeme psát.

Program bude typicky vypadat takto:

```
int podprogram( int parametr1, int parametr2 )
{
    ...
}

int main()
{
    assert( podprogram( 1, 2 ) == 3 );
    ...
}
```

Podprogram s názvem `main` bude v tomto kurzu vždy obsahovat testy, které ověřují základní funkcionalitu ostatních podprogramů. Můžete si do něj vždy přidat svoje vlastní testy. Zápis `podprogram(1, 2)` je volání (použití) podprogramu – prozatím jej nebudeme mimo testy potřebovat, protože jediné podprogramy, které budeme moct v tomto předmětu použít, jsou ty, které si sami napíšeme.

1 Hodnoty, objekty a proměnné Proměnné znáte již z kurzu IB111 – proměnné v jazyce C mají s těmi v Pythonu mnoho společného, ale mají také důležité odlišnosti. Prvním, v zásadě syntaktickým, rozdílem je, že v jazyce C musíme každou proměnnou **deklarovat** – to provedeme zápisem `typ jméno;` případně `typ jméno = výraz;`. První forma proměnnou pouze deklaruje, ale její počáteční hodnotu ponechá neurčenu – tuto hodnotu **není dovoleno použít**.

Typ proměnné určuje, jakých hodnot může nabývat – k dispozici máme proza tím tyto zabudované typy:

- `unsigned` – celé číslo v rozsahu 0 až 65535,¹
- `int` – celé číslo v rozsahu -32768 do 32767,²
- `bool` – celé číslo, 0 nebo 1, které typicky reprezentuje pravdivostní hodnotu – 0 pro `false`, 1 pro `true`,
- `signed char` – celé číslo v rozsahu -128 až 127,

- `unsigned char` – celé číslo v rozsahu 0 až 255,
- `char` – typ se stejným rozsahem jako jeden z předchozích dvou (který z nich je určeno implementací), ale přesto z pohledu kontroly typů od obou odlišný.

Proměnná je v jazyce C pevně svázaná³ s **objektem**. Objekt je **abstrakce paměti** – reprezentuje entitu, která je schopna pamatovat si **hodnotu**, již můžeme z objektu **přečíst** nebo do objektu **uložit** novou (a tím tu předchodí přepsat). Objekt tak můžeme chápat jako dvojí zobecnění paměťové buňky:

- místo jednoho bajtu si pamatuje **hodnotu** (která může mít potenciálně složitou vnitřní strukturu, i když takové zatím neumíme v jazyce C sestrojít),
- místo adresy má **identitu** – objekt můžeme „uchopit“ a pracovat s ním – obvykle tak, že tento objekt svážeme s proměnnou.

Realizace objektů je důležitým prvkem implementace programovacího jazyka a může se případ od případu lišit. Zejména není pravda, že by byl objekt pevně svázan s nějakou adresou nebo registrem – překladač může objekt transparentně přesouvat dle potřeby výpočtu.⁴

2 Živost a rozsah platnosti Objekt, který je s proměnnou svázaný, vznikne právě deklarací, a zanikne opuštěním rozsahu platnosti této proměnné. Čtení objektu je implicitní – provede se kdykoliv proměnnou použijeme jako hodnotu ve výrazu, zápis do objektu pak provedeme operátorem přiřazení (viz také další sekce).

Podobně jméno proměnné je platné počínaje deklarací, a konče pravou složenou závorkou, která ukončuje nejbližší uzavírající blok (složený příkaz nebo tělo funkce – podrobněji rozebereme dále). Například:

```
{
    // zde x ještě není deklarováno
    int x;
    {
        int y;
        ... // zde můžeme použít jak x tak y
    } // zde končí rozsah platnosti y
    ... // zde již y není lze použít
} // zde končí rozsah platnosti x
```

³ Na rozdíl od jazyka Python, kde je možné vazbu proměnné na objekt změnit přiřazením. To v jazyce C možné není.

⁴ Překladač jazyka C například běžně přesouvají objekty mezi registry a zásobníkem podle aktuální situace. Tentýž objekt může být tedy v různých fázích výpočtu fyzicky uložen na různých místech.

U proměnných je tak syntakticky zaručeno, že jsou svázaný s živým objektem – kdykoliv můžeme jméno proměnné použít, objekt, který tato proměnná pojmenovává, existuje.

3 Výrazy Na úrovni jazyka C je základní jednotkou výpočtu **výraz** – podobně jako v jazyce Python můžeme výrazy tvořit induktivně. Jsou-li:

- $e_1, e_2 \dots e_n$ výrazy,
- `var` jméno proměnné,
- `lit` číselný literál (konstanta),

existují také výrazy tvaru:⁵

1. `lit` (konstanta) je výraz,
2. `var` (jméno proměnné) je výraz,
3. použití aritmetického operátoru (binární v infixovém zápisu, unární v prefixovém):
 - $e_1 + e_2, e_1 - e_2,$
 - $e_1 * e_2, e_1 / e_2, e_1 \% e_2$ (modulo)
 - unární mínus $-e_1,$
4. relační operátory:
 - $e_1 == e_2$ (rovnost), $e_1 != e_2$ (nerovnost)
 - $e_1 <= e_2, e_1 >= e_2, e_1 < e_2, e_1 > e_2$
5. bitové logické operace a posuvy:
 - binární $e_1 \& e_2$ (and), $e_1 | e_2$ (or), $e_1 \wedge e_2$ (xor),
 - unární $\sim e_1$ – bitová negace,
 - bitové posuvy zapisujeme $e_1 >> e_2, e_1 << e_2,$
6. operátory přiřazení (pozor na změnu oproti jazyku Python – v jazyce C je přiřazení výraz, nikoliv příkaz):
 - jednoduché `var = e1,`
 - složené `var += e1, var -= e1,`
 - dále `var *= e1, var /= e1, var %= e1,`
 - s bitovým posuvem `var <<= e1, var >>= e2,`
 - s bitovou operací `var &= e1, var ^= e1, var |= e1,`
7. operátory zvýšení a snížení proměnné o jedničku:
 - prefixové `++var, --var,`
 - postfixové `var++, var--,`
8. operátor čárka, $e_1, e_2,$
9. booleovské logické operace:
 - binární $e_1 \&\& e_2$ (and), $e_1 || e_2$ (or),
 - unární $!e_1,$
 - ternární $e_1 ? e_2 : e_3,$

⁵ S dalšími operátory se setkáme v pozdějších kapitolách.

¹ Pro typy `int` a `unsigned` je konkrétní rozsah přípustných hodnot daný implementací – na mnoha systémech jsou tyto typy 32bitové.

² Starší standardy jazyka C neurčují, jaké kódování se použije pro znaménkové typy, novější již požadují dvojkový doplňkový kód (viz také předchozí kapitola).

4 Vyhodnocení výrazu Nyní víme, jak výrazy vypadají (jakou mají **syntaxi**), můžeme tedy přistoupit k otázce, co takové výrazy **znamena**jí (jakou mají **sémantiku**). Všechny zde uvedené výrazy⁶ popisují nějakou **hodnotu** a výraz samotný je návodem, jak tuto hodnotu získat.

Vyhodnocení výrazu (provedení výpočtu tímto výrazem popsaného) budeme samozřejmě realizovat pomocí již zavedeného výpočetního stroje **tiny**. Abychom mohli výpočet skutečně provést, musíme určit registr, do kterého má být výsledek zapsán – budeme mluvit o **vyhodnocení výrazu E do registru R**.⁷

1. Výraz **lit** se vyhodnotí přímo na číselnou hodnotu zapsanou ve zdrojovém kódu. Například vyhodnocení výrazu **7** do registru **rv** se realizuje instrukcí **put 7 → rv**.
2. Výraz **var** se vyhodnotí na hodnotu, která je v momentě vyhodnocení tohoto výrazu uložena v objektu svázaném s proměnnou **var**. Prozatím uvažujeme pouze situace, kdy je objekt svázaný s **var** uložen přímo v registru. Je-li např. **var** uloženo v **l1**, vyhodnocení výrazu **var** do registru **rv** realizujeme instrukcí **copy l1 → rv**.
3. Uvažme nyní výraz tvaru **e₁ + e₂**. Víme, že **e₁** a **e₂** popisují nějaké hodnoty. Abychom mohli vyčíslit hodnotu **e₁ + e₂**, budeme nejprve potřebovat tyto hodnoty. Na to použijeme **dočasné registry** – vyhodnocení **e₁ + e₂** do **rv** bude vypadat takto:
 - a. vyhodnoť **e₁** do registru **t₁**,
 - b. vyhodnoť **e₂** do registru **t₂**,
 - c. proveď **add t₁, t₂ → rv**.

Musíme samozřejmě zabezpečit, že výpočet **e₂** nepřepíše registr **t₁** – jak přesně se toho dosáhne budeme zkoumat později.⁸

Analogicky se vypočtou ostatní aritmetické, bitové, atd. operátory (k hodnotám s/bez znaménka a operacím dělení se ještě vrátíme).

4. Výrazy tvaru **var = e₁** mají krom hodnoty také **vedlejší efekt** – zápis do objektu svázaného s proměnnou **var**. Jejich realizace vypadá takto – vyhodnocujeme do registru **rv**, objekt svázaný s **var** nechť žije v **l1**:
 - a. vyhodnoť **e₁** do registru **rv**
 - b. proveď **copy rv → l1**.

Všimněte si, že hodnota **e₁** je zároveň hodnotou celého výrazu, a zůstává uložená v registru **rv**, jak bylo požadováno.

⁶ Toto tvrzení v jazyce C neplatí obecně pro všechny výrazy – existují i takové, které hodnotu nemají.

⁷ Tato konstrukce skutečně tvoří základ překladu výrazů v překladači jazyka C. Rozdíl je, že překladač pracuje s dočasnými registry mnohem hospodárněji, než naivní překladač zde popsaný – tím šetří nejen volné registry, ale i instrukce, které by hodnoty mezi registry zbytečně přesouvaly. Toto platí i pro velmi jednoduché překladače (např. také **tinycc**).

⁸ Prozatím si vystačíme s představou, že při překladu udržujeme množinu volných dočasných registrů (takových, které jsme zatím nepoužili, nebo kterých hodnotu už jsme nepotřebili, a nebudeme ji v dalším výpočtu potřebovat). Je asi jasné, že ať začneme s jakkoliv velkou konečnou množinou dočasných registrů, při výpočtu dostatečně složitěho výrazu nám musí dojít – jak se s tímto problémem vypořádat si ukážeme v příští kapitole.

5. Složené přiřazení **var += e₁** je analogické, pouze je operaci **copy** předřazena příslušná aritmetická nebo logická operace:⁹
 - a. vyhodnoť **e₁** do registru **t₁**
 - b. proveď **add t1, l1 → rv**,
 - c. proveď **copy rv → l1**.

Výrazy zvýšení a snížení o jedničku jsou analogické, liší se pouze ve výsledné hodnotě. Prefixové verze, **++var**, **--var** jsou pouze syntaktické zkratky pro **var += 1** resp. **var -= 1**, ale postfixové se liší – vyhodnocení **var++** do **rv** proběhne takto (**var** je svázané s **l1**):

- a. proveď **copy l1 → rv**,
- b. proveď **add l1, 1 → l1**.

Hodnota výrazu **var++** je tedy **původní** hodnota **var**, předtím, než bylo provedeno zvýšení proměnné o jedničku.

6. Výraz **e₁, e₂** představuje „zapomenutí hodnoty“ výrazu **e₁** – výraz **e₁** je proveden pouze pro svoje vedlejší efekty (např. výše uvedené přiřazení). Vyhodnocení **e₁, e₂** do registru **rv** lze realizovat např. takto:
 - a. vyhodnoť **e₁** do **rv**,
 - b. vyhodnoť **e₂** do **rv**.
7. Zbývá zatím nejsložitější typ výrazů, a to jsou booleovské logické operace. **XXX**

Uvažme nyní několik konkrétních příkladů:

1. **var + 1** se vypočte **XXX**

5 Příkazy

- výraz + středník
- složený příkaz
- **if, else**
- **for**
- **while**
- **break**
- **continue**

⁹ Pro výrazy tvarů, které jsme zatím zavedli, je **var += e₁** ekvivalentní výrazu **var = var + e₁**. V obecném případě, kdy je na levé straně složeného přiřazení složitější výraz (a nikoliv pouze název proměnné), to už ale neplatí!

