Popis jazyka FLP-2014-C

Peter Matula
email: imatula@fit.vutbr.cz

20. února 2014

1 Úvod

Tento dokument popisuje syntaxi a sémantiku jazyka FLP-2014-C. Jazyk FLP-2014-C je inspirován jazykem C.

Klíčová slova jsou sázena tučně a některé lexémy jsou pro zvýšení čitelnosti v uvozovkách, přičemž znak uvozovek není v takovém případě součástí jazyka!

2 Obecné vlastnosti a datové typy

Programovací jazyk FLP-2014-C je case-sensitive a je jazykem typovaným, jednotlivé proměnné mají předem určen datový typ svou deklarací.

Identifikátor je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a podtržítek začínající písmenem nebo podtržítkem ("_"). Jazyk FLP-2014-C obsahuje navíc níže jmenovaná *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory: `double`, `else`, `if`, `int`, `print`, `scan`, `string`, `while`. Dále není možné deklarovat více proměnných stejného názvu, protože každá proměnná má svůj jedinečný název.

Celočíselná konstanta je tvořena neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě[1].

Desetinná konstanta je tvořena celou částí, desetinnou částí a volitelně exponentem. Celá i desetinná část je tvořena neprázdnou posloupností číslic. Exponent má před touto posloupností nepovinné znaménko "`+`"

[1] Přebytečné počáteční číslice `0` jsou ignorovány.

(plus) nebo "`-`" (mínus) a začíná znakem "`e`" nebo "`E`". Mezi jednotlivými částmi nesmí být jiný znak, celou a desetinnou část odděluje znak "`.`" (tečka).

Řetězcová konstanta je ohraničena dvojitými uvozovkami(", ASCII hodnota 34) z obou stran. Tvoří ji znaky s ASCII hodnotou větší než 31. Řetězec může obsahovat escape sekvence "`\n`" (konec řádku), "`\\`" (zpětné lomítko) a "`\"`" (dvojité uvozovky). Délka této konstanty není omezena. Například řetězcová konstanta

```
"Ahoj\nSvete!"
```

bude interpretován jako

```
Ahoj
Svete!
```

Jazyk FLP-2014-C podporuje *blokové komentáře*. Tyto komentáře začínají symboly "`/*`" a jsou ukončeny symboly "`*/`". Rovněž podporuje řádkové komentáře, které jsou uvozeny znaky "`//`" (stejně jako v jazyce C).

3 Struktura jazyka

FLP-2014-C je strukturovaný programovací jazyk podporující deklarace a definice proměnných i funkcí včetně jejich rekurzivního volání.

3.1 Základní struktura jazyka

Program se skládá ze sekce deklarace globálních proměnných a ze sekce deklarací a definicí funkcí.

3.2 Sekce deklarace globálních proměnných

Sekce deklarace globálních proměnných (může být i prázdná) obsahuje sekvenci deklarací tvaru *typ id* `;`, přičemž *typ* může být `int`, `double`či `string` a *id* je identifikátor.

3.3 Sekce deklarace a definice funkcí

Tato sekce je tvořena deklaracemi a definicemi funkcí. Každá funkce musí být definována právě jednou a deklarována maximálně jednou. Deklarace funkcí slouží pro křížové rekurzivní volání dvou či více funkcí navzájem. Definice

nebo alespoň deklarace funkce musí být vždy k dispozici před prvním voláním funkce (příkaz volání je definován níže). Deklarace a definice funkcí mají následující tvar:

Deklarace funkce je ve tvaru:
návratový_typ id `(` *seznam_parametrů* `)` `;`

Definice funkce je ve tvaru:
návratový_typ id `(` *seznam_parametrů* `)`
{
sekvence_deklarace_proměnných
sekvence_příkazů
}

Deklarace jednotlivých formálních parametrů jsou odděleny čárkou, za posledním z nich se čárka neuvádí. Deklarace parametru má tvar:
datový_typ id
Parametry jsou vždy předávány hodnotou.

Deklarace lokální proměnné má tvar:
datový_typ id `;`

3.4 Vstupní bod programu

Vstupním bodem programu je podobně jako v jazyce C funkce *main* bez parametrů vracející `int` . Její hlavička tedy vypadá takto:

```
int main();
```

Struktura jednotlivých příkazů je uvedena v následující části.

4 Syntaxe a sémantika příkazů

Příkazem se rozumí:

Příkaz pro načtení řetězce `scan(`*id_typu_string*`);`
Sémantika příkazu je následující: Ze standardního vstupu načti řetězec ukončený koncem řádku (symbol konce řádku již do načítaného řetězce nepatří). Výsledek ulož do proměnné *id_typu_string*. Pozor na ošetření chyb nekorektnosti vstupní hodnoty!

Příkaz pro načtení číselné hodnoty `scan(`*id*`);`

Sémantika příkazu je následující: Příkaz ze standardního vstupu načte řádek (ukončený znakem *EOL*), který obsahuje libovolný počet mezer a tabulátorů, následně hodnotu zapsanout podle pravidel pro konstanty, a poté další libovolně dlouhou sekvenci mezer a tabulátorů. Hodnotu uloží do proměnné *id*.

Příkaz pro výpis hodnoty `print(`*vyraz*`);`

Sémantika příkazu je následující: Vyhodnoť zadaný výraz (podrobněji popsány v následující kapitole) a vypiš jeho hodnotu následovanou koncem řádku na standardní výstup.

Příkaz přiřazení *id* `=` *vyraz* `;`

Tento příkaz začíná identifikátorem, za kterým následuje znak "`=`". Přesná struktura výrazu je popsána v následující kapitole.

Podmíněný příkaz

`if (` *vyraz* `) {`
  sekvence_prikazu_1
`} else {`
  sekvence_prikazu_2
`}`

Sémantika příkazu je následující: Vyhodnoť daný výraz. Pokud výsledek výrazu je jiného typu než `int`, jedná se o sémantickou chybu. Jinak pokud výsledná hodnota výrazu je rovna číslu `0` (nula) a výraz je považován za nepravdivý. Za pravdivý je výraz *vyraz* považován pro všechny ostatní celočíselné nenulové hodnoty. Pokud je hodnota výrazu *vyraz* pravdivá, vykonej příkazy *sekvence_prikazu_1*, jinak vykonej příkazy *sekvence_prikazu_2*.

Příkaz cyklu

`while (` *vyraz* `) {`
  sekvence_prikazu
`}`

Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako v případě příkazu podmínky. Opakuj provádění příkazů *sekvence_prikazu* tak dlouho, dokud je hodnota výrazu *vyraz* pravdivá.

Příkaz pro ukončení funkce `return` *vyraz* `;`

Sémantika příkazu je následující: Příkaz ukončí funkci a vrátí hodnotu určenou zadaným výrazem. V případě funkce `main` ukončí celý program (hodnota výrazu v tomto případě nemá vliv.

5 Výrazy

Výrazy jsou tvořeny konstantami, již definovanými proměnnými, voláními funkcí, závorkami nebo výrazy tvořenými binárními a relačními operátory.

Volání funkce má tvar

```
id_funkce(vyraz1, vyraz2, ..., vyrazN)
```

a rámci předání parametrů může dojít k implicitní typové konverzi z datového typu `int` na `double`.

5.1 Binární a relační operátory

Pro operátory `+`, `-`, `*`, `/` platí: Pokud jsou oba operandy typu `int`, výsledek je typu `int`. Pokud je jeden z operandů typu `double` a druhý typu `int` nebo `double`, výsledek je typu `double`.

Operátor `+` navíc provádí se dvěma operandy typu `string` jejich konkatenaci.

Pro operátory `<`, `>`, `<=`, `>=`, `==`, `!=` (menší než, větší než, menší rovno, větší rovno, rovno, nerovno) platí: Pokud je první operand stejného typu jako druhý operand, a to `int`, `double` nebo `string`, výsledek je typu `int`. U řetězců se porovnání provádí lexikograficky. Výsledkem porovnání je celočíselná hodnota `1` v případě kladného výsledku, jinak `0`.

Jiné než uvedené kombinace typů ve výrazech pro dané operátory jsou považovány za chybu.

5.2 Priorita operátorů

Prioritu operátorů lze explicitně upravit závorkováním podvýrazů. Následující tabulka udává priority operátorů (nahoře nejvyšší):

pr.	Operátory	Asociativita
1	`*` `/`	→
2	`+` `-`	→
3	`<` `<=` `>` `>=` `==` `!=`	→

6 Příklady programů v jazyce FLP-2014-C

Tato část popisuje dva jednoduché programy v jazyce FLP-2014-C.

6.1 Výpočet faktoriálu (iterativně)

```
// Program 1: Vypocet faktorialu

// globalni promenne
int vysl;

int main()
{
   int a;
   print("Zadejte cislo pro vypocet faktorialu: ");
   scan(a);
   if (a < 0) {
      print("Faktorial nelze spocitat\n");
   } else {
      vysl = 1;
      while (a > 0) {
        vysl = vysl * a;
        a = a - 1;
      }
      print("Vysledek je: ");
      print(vysl);
      print("\n");
   }
}
```

6.2 Výpočet faktoriálu (rekurzivně)

```
/* Program 2: Vypocet faktorialu (rekurzivne) */

int factorial(int n);

int factorial(int n)
{
    int result;
    if (n < 2) {
      return 1;
    } else {
      return n * factorial(n-1);
    }
}

int main()
{
  int a;
  int vysl;

  print("Zadejte cislo pro vypocet faktorialu: ");
  scan(a);
```

```
  if (a < 0) {
    print("Faktorial nelze spocitat.\n");
  } else {
    vysl = factorial(a);
    print("Vysledek je: ");
    print(vysl);
    print("\n");
  }
}
```

7 Možná rozšíření

- Deklarace proměnných na místě jakéhokoliv příkazu, překrytí viditelnosti (až 5%).
- Přetěžování funkcí (až 10%).
- ...