ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Jakub Křoustek, Zbyněk Křivka
email: {ikroustek, krivka}@fit.vutbr.cz
21. září 2010

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ10.
Informace: diskusní fórum IFJ v IS FIT
Datum odevzdání: neděle 12. prosince 2010, 23:59
Způsob odevzdání: prostřednictvím IS FIT do datového skladu předmětu IFJ

Hodnocení:

- Do předmětu IFJ získá každý maximálně 25 bodů (20 funkčnost projektu, 5 dokumentace).
- Do předmětu IAL získá každý maximálně 15 bodů (10 prezentace, 5 dokumentace).
- Max. 25% bodů Vašeho individuálního základního hodnocení do předmětu IFJ navíc za tvůrčí přístup (různá rozšíření apod.).
- **Udělení zápočtu z IFJ i IAL je podmíněno získáním min. 20 bodů v průběhu semestru. Navíc v IFJ z těch 20 bodů musíte získat nejméně 5 bodů za programovou část projektu.**
- Dokumentace bude hodnocena nejvýše polovinou bodů z hodnocení funkčnosti projektu a bude také reflektovat procentuální rozdělení bodů.

Řešitelské týmy:

- Projekt budou řešit čtyř až pětičlenné týmy. Týmy s jiným počtem členů jsou nepřípustné.
- Registrace do týmů se provádí přihlášením na příslušnou variantu zadání v IS FIT. Registrace je dvoufázová. V první fázi se na jednotlivé varianty projektu přihlašují **pouze** vedoucí týmů (kapacita je omezena na 1). Ve druhé fázi se pak sami doregistrují ostatní členové (kapacita bude zvýšena na 5). Vedoucí týmů budou mít plnou pravomoc nad složením a velikostí svého týmu. Rovněž vzájemná komunikace mezi vyučujícími a týmy bude probíhat výhradně prostřednictvím vedoucích. Ve výsledku bude u každého týmu prvně zaregistrovaný člen považován za vedoucího tohoto týmu. Začátek obou fází registrace bude předem oznámen.

- Zadání obsahuje více variant. Každý tým řeší pouze jednu vybranou variantu. Výběr variant se provádí přihlášením do skupiny řešitelů dané varianty v IS FIT. Převážná část zadání je pro všechny skupiny shodná a jednotlivé varianty se liší pouze ve způsobu implementace tabulky symbolů a vestavěných funkcí pro vyhledávání podřetězce v řetězci a pro řazení. V IS je pro označení variant použit zápis `písmeno/arabská číslice/římská číslice`, kde písmeno udává variantu implementace metody pro vyhledávání podřetězce v řetězci, arabská číslice udává variantu řadicí metody a římská číslice způsob implementace tabulky symbolů.

2 Zadání

Vytvořte program, který načte zdrojový soubor zapsaný v jazyce IFJ10 a interpretuje jej. Jestliže proběhne činnost interpretu bez chyb, vrací se návratová hodnota `0` (nula). Jestliže došlo k nějaké chybě, vrací se návratová hodnota následovně:

- 1 = chyba v programu v rámci lexikální analýzy (chybná struktura aktuálního lexému).
- 2 = chyba v programu v rámci syntaktické analýzy (chybná syntaxe struktury programu).
- 3 = chyba v programu v rámci sémantických kontrol (nedeklarovaná proměnná, nekompatibilita typů atd.).
- 4 = chyba za běhu programu, kdy je struktura programu v pořádku, ale chyba nastala až v rámci interpretace konkrétních dat (dělení nulou, na vstupu chybný formát číselné hodnoty atd.).
- 5 = interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti, chyba při otvírání vstupního souboru atd.).

Jméno souboru s řídicím programem v jazyce IFJ10 bude předáno jako první a jediný parametr na příkazové řádce. Bude možné jej zadat s relativní i absolutní cestou. Program bude přijímat vstupy ze standardního vstupu, směrovat všechny své výstupy diktované řídicím programem na standardní výstup, všechna chybová hlášení na standardní chybový výstup; tj. bude se jednat o konzolovou aplikaci, nikoliv o aplikaci s grafickým uživatelským rozhraním.

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

3 Popis programovacího jazyka

Jazyk IFJ10 je podmnožinou jazyka FreeBASIC, jenž staví na legendárním jazyce BASIC.

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ10 je case-insensitive (při porovnání jmen nezáleží na velikosti písmen) a je jazykem typovaným, takže každá proměnná má předem určen datový typ svou deklarací.

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ("_") začínající písmenem nebo podtržítkem. Jazyk IFJ10 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory: **`As`**, **`Declare`**, **`Dim`**, **`Do`**, **`Double`**, **`Else`**, **`End`**, **`find`**, **`Function`**, **`If`**, **`Input`**, **`Integer`**, **`Loop`**, **`Print`**, **`Return`**, **`Scope`**, **`sort`**, **`String`**, **`Then`**, **`While`**. Dále není možné deklarovat více proměnných stejného jména.
- *Celočíselná konstanta* (rozsah C-int) je tvořena neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě[1].
- *Desetinná konstanta* (rozsah C-double) také vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou a desetinnou částí, nebo celou částí a exponentem, nebo celou a desetinnou částí a exponentem. Celá i desetinná část je tvořena neprázdnou posloupností číslic. Exponent má před touto posloupností nepovinné znaménko "**`+`**" (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 uvozovkami (**`"`**, ASCII hodnota 34) z obou stran. Tvoří ji znaky s ASCII hodnotou větší než 31. Řetězec může obsahovat escape sekvence: '**`\n`**', '**`\t`**', '**`\\`**', '**`\"`**'. Jejich význam se shoduje s odpovídajícími znakovými konstantami jazyka FreeBASIC[2]. Délka této konstanty není omezena (snad jen velikostí haldy daného programu). Například řetězcová konstanta

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

  bude interpretována (např. vytisknuta) jako
  `"Ahoj`
  `Svete!"`
- *Datové typy* pro jednotlivé uvedené literály jsou označeny **`Integer`**, **`Double`** a **`String`**. Typy se dále používají ve spojitosti s proměnnými, s typy výsledků funkcí a s jejich parametry.
- Jazyk IFJ10 podporuje *řádkové komentáře*. Komentář je označen pomocí apostrofu ("**`'`**", ASCII hodnota 39) a za komentář je považováno vše, co následuje až do konce řádku.
- Jazyk IFJ10 podporuje *blokové komentáře*. Tyto komentáře začínají dvojicí symbolů "**`/'`**" a jsou ukončeny dvojicí symbolů "**`'/`**". Vnořené blokové komentáře neuvažujte.

4 Struktura jazyka

IFJ10 je strukturovaný programovací jazyk podporující deklarace a definice proměnných i funkcí včetně jejich rekurzivního volání. Vstupním bodem řídicího programu je hlavní tělo programu.

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

[2] Přesněji implementace FreeBASIC jazyka BASIC:
`http://www.freebasic.net/wiki/wikka.php?wakka=TblEscapeSequences`

4.1 Základní struktura jazyka

Program se skládá ze sekce deklarací a definic funkcí následované hlavním tělem programu. Jednotlivé konstrukce jazyka IFJ10 jsou (až na výjimky) jednořádkové a jako takové jsou vždy ukončeny znakem konce řádku (dále jen **EOL**). U definic víceřádkových konstrukcí jsou dodatečné znaky **EOL** explicitně uvedeny[3]. Ostatní bílé znaky (mezery, tabulátory, ...) se mohou vyskytovat v libovolném množství mezi všemi lexémy.

4.2 Sekce deklarace a definice funkcí

Sekce je tvořena deklaracemi a definicemi funkcí. Každá funkce musí být definována právě jednou, 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:
  Declare Function *id* **(** *seznam_parametrů* **) As** *návratový_typ_funkce*
- *Definice funkce* je víceřádková konstrukce ve tvaru:
  Function *id* **(** *seznam_parametrů* **) As** *návratový_typ_funkce* **EOL**
  sekvence_deklarace_proměnných
  sekvence_příkazů
  End Function
  - Deklarace jednotlivých formálních parametrů jsou odděleny čárkou, za poslední z nich se čárka neuvádí. Deklarace parametru má tvar:
    id **As** *datový_typ_parametru*
    Parametry jsou vždy předávány hodnotou.
  - Tělo funkce je sekvence deklarací proměnných a příkazů. Jejich syntaxe a sémantika je shodná s deklaracemi a příkazy v hlavním těle programu (viz dále).

Z hlediska sémantiky je potřeba kontrolovat, zda souhlasí počet parametrů a pořadí typů v deklaraci funkce a v hlavičce definice funkce. Případná deklarace funkce musí předcházet její definici. Každá deklarovaná funkce musí být definována. Přetěžování funkcí není povoleno.

4.3 Sekce hlavního těla programu

Hlavní tělo programu je uvozeno klíčovým slovem **Scope** následovaným sekvencí deklarací proměnných a sekvencí dílčích příkazů ukončených koncem řádku (sekvence mohou být i prázdné). Nejprve jsou uvedeny všechny deklarace proměnných a až poté příkazy. Celá sekvence je ukončena pomocí **End Scope**. Struktura deklarací proměnných a dílčích příkazů je uvedena v následujících sekcích.

[3] Vně zmíněných konstrukcí je znak **EOL** brán jako tzv. *bílý znak*. Je tedy například možné pro zpřehlednění kódu vložit mezi dva příkazy prázdný řádek.

4.3.1 Deklarace proměnných

Proměnné jazyka IFJ10 jsou pouze lokální. Pro každou dílčí deklaraci proměnné s názvem *id* se používá následující zápis:

`Dim` *id* **`As`** *datový_typ*

Číselné proměnné jsou inicializovány na hodnotu 0 (resp. 0.0), řetězcové na `""`.

4.3.2 Syntaxe a sémantika příkazů

Dílčím příkazem se rozumí:

- *Příkaz pro načtení hodnot*:
  `Input` `;` id_1`,` id_2`, ...,` id_n
  Sémantika příkazu je následující: Příkaz nejprve vypíše na standardní výstup řetězec `"? "`. Poté postupně ze standardního vstupu načítá jednotlivé hodnoty oddělené libovolným nenulovým počtem bílých znaků (mezera, tabulátor) a ukládá je do proměnných $id_1, id_2, \ldots, id_n$ v daném pořadí. Hodnoty typu **`String`** budou ohraničeny uvozovkami, přičemž tyto znaky do řetězce nepatří. Poslední hodnota musí být potvrzena koncem řádku. Pokud bude zadáno více vstupních hodnot než počet parametrů (n), budou ignorovány. Při nedostatečném počtu hodnot budou zbývající parametry (proměnné) inicializovány na 0, resp. 0.0, resp. `""`. Příkaz musí obsahovat alespoň jeden parametr. Pozor na ošetření chyb nekorektnosti vstupních hodnot!
- *Příkaz pro výpis hodnot*:
  `Print` $výraz_1$`;` $výraz_2$`; ...;` $výraz_n$`;`
  Sémantika příkazu je následující: Postupně vyhodnocuje jednotlivé výrazy (podrobněji popsány v kapitole 5) a vypisuje jejich hodnoty na standardní výstup ihned za sebe, bez jakýchkoliv oddělovačů a v patřičném formátu. Hodnota výrazu typu **`Integer`** bude vytištěna pomocí `"% d"`[4], hodnota výrazu typu **`Double`** pak pomocí `"% g"`. Všimněte si, že každý výraz je ukončen středníkem, což je rozdíl oproti zápisu parametrů příkazu **`Input`**. Příkaz musí obsahovat alespoň jeden výraz.
- *Příkaz přiřazení*:
  id **`=`** *výraz*
  Sémantika příkazu je následující: Příkaz provádí přiřazení hodnoty pravého operandu do levého operandu. Platí pro něj: levý operand musí být stejného typu jako pravý operand, a to **`Integer`**, **`Double`** nebo **`String`**. Potom výsledek výrazu je opět tohoto typu. Druhou možností je, že levý operand je typu **`Double`** a pravý operand typu **`Integer`**. Potom je provedena implicitní typová konverze a výsledek výrazu bude typu **`Double`**. Levý operand musí být vždy pouze proměnná (tzv. l-hodnota).
- *Podmíněný příkaz*:
  `If` *výraz* **`Then EOL`**
  $sekvence\_příkazů_1$
  `Else EOL`
  $sekvence\_příkazů_2$
  `End If`

[4] Formátovací řetězec standardní funkce **`printf`** jazyka C. Nepřehlédněte mezeru za znakem procenta.

Sémantika příkazu je následující: Nejprve vyhodnotí daný výraz. Pokud výsledek výrazu je jiného typu než **Integer**, jedná se o sémantickou chybu. Jinak pokud výsledná hodnota výrazu je rovna číslu `0` (nula), výraz je považován za nepravdivý. Za pravdivý je výraz považován pro všechny nenulové hodnoty jeho výsledku.
Sekvence_příkazů$_1$ a *sekvence_příkazů*$_2$ jsou opět sekvence dílčích příkazů (mohou být i prázdné) definované v této kapitole (rekurzivní definice). Pokud je hodnota výrazu pravdivá, vykoná se *sekvence_příkazů*$_1$, jinak se vykoná *sekvence_příkazů*$_2$.

- *Příkaz cyklu*:
  Do While *výraz* **EOL**
  sekvence_příkazů
  Loop
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako v případě podmíněného příkazu. Opakuje provádění sekvence příkazů tak dlouho, dokud je hodnota výrazu pravdivá.
- *Volání vestavěné a uživatelem definované funkce*:
  výstupní_id **=** *název_funkce* **(***seznam_vstupních_parametrů***)**
  Seznam_vstupních_parametrů je seznam konstant a proměnných oddělených čárkami[5]. Pokud je některý parametr typu **Integer** a funkce očekává na jeho pozici hodnotu typu **Double**, dojde k implicitní typové konverzi na typ **Double**. Sémantika vestavěných funkcí bude popsána v kapitole 6. Sémantika volání uživatelem definovaných funkcí je následující: příkaz zajistí předání argumentů hodnotou a předání řízení do těla funkce. Po návratu z těla funkce (viz dále) dojde k uložení výsledku do *výstupní_id* a pokračování běhu programu bezprostředně za příkazem volání funkce.
- *Příkaz návratu z funkce*:
  Return *výraz*
  Příkaz může být použit pouze v tělech funkcí (tj. ne v těle hlavního programu). Jeho sémantika je následující: Dojde k okamžitému ukončení provádění těla funkce a návratu do místa volání. Návratová hodnota bude získaná vyhodnocením výrazu. Pokud výsledek výrazu je typu **Integer** a návratový typ funkce je **Double**, dojde k implicitní typové konverzi na typ **Double**. V ostatních případech nejednotnosti typů se jedná o sémantickou chybu. Ukončení uživatelem definované funkce bez volání **Return** způsobí chybu při interpretaci.

5 Výrazy

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

5.1 Aritmetické a relační operátory

Pro operátory **+**, **-** a ***** platí: Pokud jsou oba operandy typu **Integer**, výsledek je typu **Integer**. Pokud je jeden z operandů typu **Double** a druhý typu **Integer** nebo **Double**, výsledek je typu **Double**.

[5] Pozn. parametrem funkce není výraz. Jedná se o nepovinné (bodované) rozšíření projektu.

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

Operátor **/** značí dělení dvou číselných operandů, jehož výsledek je bez ohledu na typy operandů vždy **`Double`**.

U operátoru **** se jedná o celočíselné dělení stejně jako v jazyce FreeBASIC. Operátor **** akceptuje pouze operandy typu **`Integer`**, výsledek je stejného typu.

Pro operátory **<**, **>**, **<=**, **>=**, **=**, **<>** platí: Pokud je první operand stejného typu jako druhý operand, a to **`Integer`**, **`Double`** nebo **`String`**, výsledek je typu **`Integer`**. U řetězců se porovnání provádí lexikograficky. Výsledkem porovnání je celočíselná hodnota `-1` (mínus jedna) v případě pravdivosti relace, jinak `0`. Operátory mají stejnou sémantiku jako v jazyce FreeBASIC[6].

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	**`*`** **`/`**	→ (levá)
2	**`\`**	→
3	**`+`** **`-`**	→
4	**`=`** **`<>`** **`<`** **`<=`** **`>`** **`>=`**	→

6 Vestavěné funkce

Interpret bude poskytovat tyto základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ10 bez jejich definice či deklarace):

- **`find(String, String) : Integer`** – Vyhledá zadaný podřetězec v řetězci a vrátí jeho pozici (počítáno od jedničky). První parametr je řetězec, ve kterém bude daný podřetězec vyhledáván. Druhým parametrem je vyhledávaný podřetězec. Pokud podřetězec není nalezen, je vrácena hodnota **`0`**. Pro vyhledání podřetězce v řetězci použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k zadáním je uveden dále.
- **`sort(String) : String`** – Seřadí znaky v daném řetězci tak, aby znak s nižší ordinální hodnotou vždy předcházel před znakem s vyšší ordinální hodnotou. Vrácen je řetězec obsahující seřazené znaky. Pro řazení použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k zadáním je uveden dále.

[6] Povšimněte si přetížení operátoru **=**, který u příkazů slouží pro přiřazení a u výrazů pro porovnání.

7 Implementace vyhledávání podřetězce v řetězci

Metoda vyhledávání, kterou bude využívat vestavěná funkce **`find`**, je v zadání pro daný tým označena písmenem a-b, a to následovně:

a) Pro vyhledávání použijte Knuth-Moris-Prattův algoritmus.
b) Pro vyhledávání použijte Boyer-Mooreův algoritmus (libovolný typ heuristiky).

Metoda vyhledávání podřetězce v řetězci musí být implementována v souboru se jménem `ial.c`. Oba algoritmy budou probírány v rámci předmětu IAL.

8 Implementace řazení

Metoda řazení, kterou bude využívat vestavěná funkce **`sort`**, je v zadání pro daný tým označena arabskou číslicí 1-4, a to následovně:

1) Pro řazení použijte algoritmus Quick sort.
2) Pro řazení použijte algoritmus Heap sort.
3) Pro řazení použijte algoritmus Shell sort.
4) Pro řazení použijte algoritmus Merge sort.

Metoda řazení bude součástí souboru `ial.c`. Všechny tyto algoritmy budou probírány v rámci předmětu IAL.

9 Implementace tabulky symbolů

Tabulka symbolů bude implementována pomocí abstraktní datové struktury, která je v zadání pro daný tým označena římskou číslicí I-II, a to následovně:

I) binární vyhledávací strom.

II) hashovací tabulka.

Implementace tabulky symbolů bude uložena taktéž v souboru `ial.c`. Oba druhy struktur a vyhledávání v nich budou probírány v rámci předmětu IAL.

10 Příklady

Tato kapitola popisuje tři jednoduché příklady řídicích programů v jazyce IFJ10.

10.1 Výpočet faktoriálu (iterativně)

```
/' Program 1: Vypocet faktorialu (iterativne) '/
/' Vcetne ukazky case-insensitive vlastnosti jazyka IFJ10 '/

scope 'Hlavni telo programu
  Dim a As Integer
  DIM vysl AS INTEGER

  PrinT "Zadejte cislo pro vypocet faktorialu";
  InpuT ; A
  If a < 0 THEN
    print "\nFaktorial nelze spocitat\n";
  ELSE
    Vysl = 1
    Do WHile A > 0
      VYSL = vysl * a
      a = A - 1
    LooP
    Print "\nVysledek je:" ; vYsl ; "\n";
  end IF
END SCOPE
```

10.2 Výpočet faktoriálu (rekurzivně)

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

Declare Function factorial (n As Integer) As Integer
Function factorial (n As Integer) As Integer
  Dim temp_result As Integer
  Dim decremented_n As Integer
  Dim result As Integer
  If n < 2 Then
    result = 1
  Else
    decremented_n = n - 1
    temp_result = factorial(decremented_n)
    result = n * temp_result
  End If
  Return result
End Function

Scope 'Hlavni telo programu
  Dim a As Integer
  Dim vysl As Integer

  Print "Zadejte cislo pro vypocet faktorialu";
  Input ; a
  If a < 0 Then
    Print "\nFaktorial nelze spocitat\n";
  Else
    vysl = factorial(a)
    Print "\nVysledek je:" ; vysl ; "\n";
  End If
End Scope
```

10.3 Práce s řetězci a vestavěnými funkcemi

```
/' Program 3: Prace s retezci a vestavenymi funkcemi '/

Scope 'Hlavni telo programu
  Dim str1 As String
  Dim str2 As String
  Dim p As Integer

  str1 = "Toto je nejaky text"
  str2 = str1 + ", ktery jeste trochu obohatime"
  Print str1; "\n"; str2; "\n";
  p = find(str2, "text")
  Print "Pozice retezce \"text\" v retezci str2:"; p; "\n";
  Print "Zadejte nejakou posloupnost vsech malych pismen a-h, ";
  Print "pricemz se pismena nesmeji v posloupnosti opakovat";
  Input ; str1
  str2 = sort(str1)
  Do While (str2 <> "abcdefgh")
    Print "\nSpatne zadana posloupnost, zkuste znovu";
    Input ; str1
    str2 = sort(str1)
  Loop
  Print "\n";
End Scope
```

11 Doporučení k testování

Programovací jazyk je schválně navržen tak, aby byl podmnožinou jazyka FreeBASIC[7]. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ10, může si to ověřit následovně. Před program jazyka IFJ10 přidá následující část kódu (obsahuje nastavení překladače `fbc` a definice vestavěných funkcí, které jsou součástí jazyka IFJ10, ale chybí v základních knihovnách jazyka FreeBASIC):

```
/' Zajisteni zakladni kompatibility IFJ10->FreeBASIC '/
#Lang "deprecated"      /' Starsi dialekt BASICu '/
Option Escape           /' Aktivuje escape sekvence '/

Function sort (s As String) As String
/' Bubble Sort '/
Dim i, j, n, finished As Integer
Dim _swap As Byte
Dim result As String
  result = s
  n = Len(result)
  i = 1
  Do
    finished = 1
    For j = n - 1 To i Step -1
      If result[j-1] > result[j] Then
        _swap = result[j-1]
```

[7] Online dokumentace ve verzi z 1.9.2010: `http://www.freebasic.net/wiki/wikka.php?wakka=DocToc`

```
        result[j-1] = result[j]
        result[j] = _swap
        finished = 0
      End If
    Next j
    i += 1
  Loop Until (finished or (i = n + 1))
  Return result
End Function

Function find (haystack As String, needle As String) As Integer
  Return InStr(haystack, needle)
End Function

/' Zde bude nasledovat program jazyka IFJ10 '/
```

Takto vytvořený program lze zkompilovat například na serveru `merlin` pomocí příkazu:

`fbc` `nazev_programu.bas`

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že FreeBASIC je nadmnožinou jazyka IFJ10 a tudíž může zpracovat i konstrukce, které jsou v IFJ10 nepovolené (např. odlišný výpis desetinných čísel nebo sémantika příkazů **`Print`** a **`Input`**). Výčet těchto odlišností bude uveden na diskuzním fóru předmětu IFJ.

12 Instrukce ke způsobu vypracování a odevzdání

Tyto důležité informace nepodceňujte, neboť projekty bude částečně opravovat automat a nedodržení těchto pokynů povede k tomu, že automat daný projekt nebude schopen zkompilovat, což může vést až k nehodnocení vašeho projektu!

12.1 Obecné informace

Za celý tým odevzdá projekt jediný student. Všechny odevzdané soubory budou zkomprimovány programem TAR+GZIP či ZIP do jediného archivu, který se bude jmenovat `přihlašovací_jméno.tgz`, či `přihlašovací_jméno.zip`. Archiv nesmí obsahovat adresářovou strukturu ani speciální či spustitelné soubory. Názvy všech souborů budou obsahovat pouze malá písmena, číslice, tečku a podtržítko (ne velká písmena ani mezery – krom souboru `Makefile`!).

Celý projekt je třeba odevzdat v daném termínu (viz výše). Pokud tomu tak nebude, je projekt považován za neodevzdaný. Stejně tak, pokud se bude jednat o plagiátorství jakéhokoliv druhu, je projekt hodnocený nula body, navíc v IFJ ani v IAL nebude udělen zápočet a bude zváženo zahájení disciplinárního řízení.

12.2 Dělení bodů

Odevzdaný archív bude povinně obsahovat soubor **`rozdeleni`**, ve kterém zohledníte dělení bodů mezi jednotlivé členy týmu (i při požadavku na rovnoměrné dělení). Na každém řádku je uveden login jednoho člena týmu, bez mezery je následován dvojtečkou a po ní je bez mezery uveden požadovaný celočíselný počet procent bodů bez uvedení znaku %. Každý řádek (i poslední) je poté ihned ukončen jedním znakem ⟨LF⟩ (ASCII hodnota 10, tj. unixové ukončení řádku, ne windowsovské!). Obsah souboru bude vypadat například takto:

```
xnovak01:30<LF>
xnovak02:40<LF>
xnovak03:30<LF>
xnovak04:00<LF>
```

Součet všech procent musí být roven 100. V případě chybného celkového součtu všech procent bude použito rovnoměrné rozdělení. Formát odevzdaného souboru musí být správný a obsahovat všechny členy týmu (i ty hodnocené 0%).

13 Požadavky na řešení

Kromě požadavků na implementaci a dokumentaci obsahuje tato kapitola i výčet rozšíření za prémiové body a několik rad pro zdárné řešení tohoto projektu.

13.1 Závazné metody pro implementaci interpretu

Při tvorbě lexikální analýzy využijete znalosti konečných automatů. Při konstrukci syntaktické analýzy využijte **metodu rekurzivního sestupu** pro kontext jazyka založeného na LL-gramatice (vše kromě výrazů). Výrazy zpracujte pomocí **precedenční syntaktické analýzy**. Vše bude probíráno na přednáškách v rámci předmětu IFJ. Implementace bude provedena **v jazyce C**, čímž úmyslně omezujeme možnosti použití objektově orientovaného návrhu a implementace. Návrh implementace interpretu je zcela v režii řešitelských týmů. Není dovoleno spouštět další procesy. Nedodržení těchto metod bude penalizováno značnou ztrátou bodů!

13.2 Textová část řešení

Součástí řešení bude dokumentace, vypracovaná ve formátu PDF a uložená v jediném souboru **`dokumentace.pdf`**. Jakékoliv jiné formáty dokumentace, než předepsané, budou ignorovány, což povede ke ztrátě bodů za dokumentaci. Dokumentace bude vypracována v českém, slovenském nebo anglickém jazyce v rozsahu cca. 4-7 stran A4. **Dokumentace musí** povinně obsahovat:

- 1. strana: jména, příjmení a přihlašovací jména řešitelů + údaje o rozdělení bodů, identifikaci vašeho variantního zadání ve tvaru "Tým *číslo*, varianta $\alpha/n/X$", výčet zvolených rozšíření.
- Strukturu konečného automatu, který specifikuje lexikální analyzátor.
- LL-gramatiku, která je jádrem vašeho syntaktického analyzátoru.

- Popis vašeho způsobu řešení interpretu (z pohledu IFJ) - návrh, implementace, vývojový cyklus, způsob práce v týmu, speciální použité techniky, algoritmy.
- Popis vašeho způsobu řešení řadicího algoritmu, vyhledávání podřetězce v řetězci a tabulky symbolů (z pohledu předmětu IAL).
- Rozdělení práce mezi členy týmu (uveďte kdo a jakým způsobem se podílel na jednotlivých částech projektu).

Dokumentace nesmí:

- Obsahovat kopii zadání či text, obrázky[8] nebo diagramy, které nejsou vaše původní (kopie z přednášek, sítě, WWW, ...).
- Být založena pouze na výčtu a obecném popisu jednotlivých metod analýzy (jde o váš vlastní přístup k řešení; a proto dokumentujte postup, kterým jste se při řešení ubírali; překážkách, se kterými jste se při řešení setkali; problémech, které jste řešili a jak jste je řešili; atd.).

V rámci dokumentace bude rovněž vzat v úvahu stav kódu jako jeho čitelnost, srozumitelnost a dostatečné, ale nikoli přehnané komentáře.

13.3 Programová část řešení

Programová část řešení bude vypracována v jazyce C bez použití generátorů lex/flex, yacc/bison či jiných podobného ražení. Programy musí být přeložitelné překladačem `gcc`. Při hodnocení budou projekty překládány na školním serveru `merlin`. Počítejte tedy s touto skutečností (především, pokud budete projekt psát pod jiným OS). Pokud projekt nepůjde přeložit či nebude správně pracovat kvůli použití funkce nebo nějaké nestandardní implementační techniky závislé na OS, ztrácíte právo na reklamaci výsledků. Ve sporných případech bude vždy za platný považován výsledek překladu na serveru `merlin` bez použití jakýchkoliv dodatečných nastavení (proměnné prostředí, ...).

Součástí řešení bude soubor `Makefile` (projekty budou překládány pomocí `gmake`), ve kterém lze nastavit případné parametry překladu (viz `info gmake`). Pokud soubor pro sestavení cílového programu nebude obsažen nebo se na jeho základě nepodaří sestavit cílový program, nebude projekt hodnocený! Jméno cílového programu není rozhodující, bude přejmenován automatem.

Binární soubor (přeložený interpret) v žádném případě do archívu nepřikládejte! Stejně tak nevytvářejte žádné soubory s definicemi či jakýmkoli pomocným obsahem, které by přeložený binární soubor vašeho projektu navíc vyžadoval, ani v `/tmp` adresáři.

Úvod **všech** zdrojových textů musí obsahovat zakomentovaný název projektu, přihlašovací jména a jména studentů, kteří se na něm autorsky podíleli.

Veškerá chybová hlášení vzniklá v průběhu činnosti interpretu budou vždy vypisována na standardní chybový výstup. Veškeré texty tištěné řídicím programem budou vypisovány na standardní výstup. Kromě chybových hlášení nebude interpret v průběhu interpretace na žádný výstup vypisovat žádné znaky či dokonce celé texty, které nejsou přímo předepsány řídicím programem. Základní testování bude probíhat pomocí automatu, který

[8] Vyjma loga fakulty na úvodní straně.

bude postupně spouštět sadu testovacích příkladů v odevzdaném interpretu a porovnávat produkované výstupy s výstupy očekávanými. Pro porovnání výstupů bude použit program `diff` (viz `info diff`). Proto jediný neočekávaný znak, který váš interpret svévolně vytiskne, povede k nevyhovujícímu hodnocení aktuálního výstupu a tím snížení bodového hodnocení celého projektu.

13.4 Jak postupovat při řešení projektu

Při řešení je pochopitelně možné využít vlastní výpočetní techniku. Instalace překladače `gcc` není nezbytně nutná, pokud máte jiný překladač jazyka C již instalován a nehodláte využívat vlastností, které právě tento a žádné jiné překladače nemají. Před použitím nějaké vyspělé konstrukce je dobré si ověřit, že jí disponuje i překladač, který nakonec bude použit při opravování. Po vypracování je též vhodné vše ověřit na cílovém překladači, aby při bodování projektu vše proběhlo bez problémů.

Teoretické znalosti, potřebné pro vytvoření projektu, získáte v průběhu semestru na přednáškách a diskuzním fóru IFJ. Je nezbytné, aby na řešení projektu spolupracoval celý tým. Návrh interpretu, základních rozhraní a rozdělení práce lze vytvořit již zhruba v polovině semestru. Je dobré, když se celý tým domluví na pravidelných schůzkách a komunikačních kanálech, které bude během řešení projektu využívat (instant messaging, konference, verzovací systém, štábní kulturu atd.).

Situaci, kdy je projekt ignorován částí týmu, lze řešit prostřednictvím souboru `rozdeleni`. Je ale nutné, abyste se vzájemně, nejlépe na pravidelných schůzkách týmu, ujišťovali o skutečném pokroku na jednotlivých částech projektu a případně včas přerozdělili práci. **Maximální počet bodů** získatelný na jednu osobu za programovou implementaci je **25**.

Nenechávejte řešení projektu až na poslední týden. Projekt je tvořen z několika částí (např. lexikální analýza, syntaktická analýza, sémantická analýza, intermediální reprezentace, interpret, tabulka symbolů, dokumentace, testování!) a dimenzován tak, aby některé části bylo možno navrhnout a implementovat již v průběhu semestru na základě znalostí získaných na přednáškách předmětů IFJ a IAL a na diskuzním fóru IFJ.

13.5 Registrovaná rozšíření

V případě implementace některých registrovaných rozšíření bude odevzdaný archív obsahovat soubor **`rozsireni`**, ve kterém uvedete na každém řádku identifikátor jednoho implementovaného rozšíření (řádky jsou opět ukončeny znakem ⟨LF⟩.

V průběhu řešení bude postupně aktualizován ceník rozšíření a identifikátory rozšíření projektu (viz fórum k předmětu IFJ). V něm budou uvedena hodnocená rozšíření projektu, za která lze získat prémiové body. Cvičícím můžete během semestru zasílat návrhy na dosud neuvedená rozšíření, která byste chtěli navíc implementovat. Cvičící rozhodnou o přijetí/nepřijetí rozšíření a hodnocení rozšíření dle jeho náročnosti včetně přiřazení unikátního identifikátoru. Body za implementovaná rozšíření se počítají do bodů za programovou implementaci, takže stále platí získatelné maximum 25 bodů.

13.5.1 Některá rozšíření, která budou oceněna prémiovými body (začínají identifikátorem):

- BASE: Celočíselné konstanty je možné zadávat i ve dvojkové (číslo začíná znaky "**&B**"), osmičkové (číslo začíná znaky "**&O**") a v šestnáctkové (číslo začíná znaky "**&H**") soustavě (+1 bod).
- MINUS: Interpret bude pracovat i s operátorem unární mínus. Do dokumentace je potřeba uvést, jak je tento problém řešen (+1 bod).
- FORNEXT: Interpret bude podporovat i cyklus **For** ... **Next** (+1,5 bodu).
- LOGOP: Podpora logických operátorů **AndAlso** a **OrElse** ve výrazech (+0,5 bodu)[9].
- IFTHEN: Zjednodušený podmíněný příkaz **If**-**Then** bez části **Else**. Do dokumentace je potřeba uvést, jak je tento problém řešen[10] (+1,5 bodu).
- FUNEXP: Funkce mohou být součástí výrazu, zároveň mohou být výrazy v parametrech funkcí (+1 bod).
- ARRAY: Projekt bude pracovat i s proměnnými typu pole (kompatibilní s jazykem FreeBASIC) (+2 body)
- SUB: Podpora procedur **Sub** (+1 bod).
- INITVAR: Inicializace proměnných v rámci jejich deklarace (+0,5 bodu).

  Dim *id* **As** *typ* **=** *hodnota*

- OVERLOAD: Přetěžování funkcí (+2 body).
- ...

14 Opravy zadání

- 25. 9. 2010 – Opravena chyba v programu 2.
- 11. 11. 2010 – Opraven překlep **Int** na **Integer** v kapitole 3.1.

[9] Viz `http://www.freebasic.net/wiki/wikka.php?wakka=DocToc`

[10] Například, jak řešit u zanoření **If**-**Then** a **If**-**Then**-**Else** párování **If** a **Else**? U moderních programovacích jazyků platí konvence párování **Else** s nejbližším **If**.