ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Radim Kocman, Marek Milkovič
email: {krivka, ikocman, imilkovic}@fit.vutbr.cz
22. září 2017

1 Obecné informace

Název projektu: Implementace překladače imperativního jazyka IFJ17.
Informace: diskuzní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání: pátek 24. listopadu 2017, 23:59 (nepovinné).
Datum odevzdání: středa 6. prosince 2017, 23:59.
Způsob odevzdání: prostřednictvím IS FIT do datového skladu předmětu IFJ.

Hodnocení:

- Do předmětu IFJ získá každý maximálně 25 bodů (15 celková funkčnost projektu, 5 dokumentace, 5 obhajoba).
- Do předmětu IAL získá každý maximálně 15 bodů (5 celková funkčnost projektu, 5 obhajoba, 5 dokumentace).
- Max. 30% bodů Vašeho individuálního hodnocení základní funkčnosti 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 těchto 20 bodů musíte získat nejméně 4 body za programovou část projektu.**
- Dokumentace bude hodnocena nejvýše polovinou bodů z hodnocení funkčnosti projektu, bude také reflektovat procentuální rozdělení bodů a bude zaokrouhlena na celé body.
- Body zapisované za programovou část včetně rozšíření budou také zaokrouhleny. Body nad 15 bodů budou zapsány v IFJ do termínu „Projekt - Prémiové body“.

Řešitelské týmy:

- Projekt budou řešit tří až čtyřčlenné 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 4). Vedoucí týmů budou mít plnou

pravomoc nad složením svého týmu. Rovněž vzájemná komunikace mezi vyučujícími a týmy bude probíhat nejlépe prostřednictvím vedoucích (ideálně v kopii dalším členům týmu). Ve výsledku bude u každého týmu prvně zaregistrovaný člen považován za vedoucího tohoto týmu. Všechny termíny k projektu najdete v IS FIT a další informace na stránkách předmětu[1].

- Zadání obsahuje dvě varianty, které se liší pouze ve způsobu implementace tabulky symbolů a jsou identifikované římskou číslicí I nebo II. Každý tým má své identifikační číslo, na které se váže vybraná varianta zadání. Výběr variant se provádí přihlášením do skupiny daného týmu v IS FIT.

2 Zadání

Vytvořte program v jazyce C, který načte zdrojový kód zapsaný ve zdrojovém jazyce IFJ17 a přeloží jej do cílového jazyka IFJcode17 (mezikód). Jestliže proběhne překlad 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 programu).
- 3 - sémantická chyba v programu – nedefinovaná funkce/proměnná, pokus o redefinici funkce/proměnné, atd.
- 4 - sémantická chyba typové kompatibility v aritmetických, řetězcových a relačních výrazech, příp. špatný počet či typ parametrů u volání funkce.
- 6 - ostatní sémantické chyby.
- 99 - interní chyba překladače tj. neovlivněná vstupním programem (např. chyba alokace paměti, atd.).

Překladač bude načítat řídicí program v jazyce IFJ17 ze standardního vstupu a generovat výsledný mezikód v jazyce IFJcode17 (viz kapitola 10) na standardní výstup. Všechna chybová hlášení, varování a ladicí výpisy provádějte na standardní chybový výstup; tj. bude se jednat o konzolovou aplikaci (tzv. filtr) bez grafického uživatelského rozhraní. Pro interpretaci výsledného programu v cílovém jazyce IFJcode17 bude na stránkách předmětu k dispozici interpret.

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

3 Popis programovacího jazyka

Jazyk IFJ17 je zjednodušenou podmnožinou jazyka FreeBASIC[2], jenž staví na legendárním jazyce BASIC, je staticky typovaný[3] imperativní jazyk.

[1] `http://www.fit.vutbr.cz/study/courses/IFJ/public/project`

[2] `https://www.freebasic.net/`; na serveru Merlin je pro studenty k dispozici překladač `fbc` verze 1.05.

[3] Jednotlivé proměnné mají předem určen datový typ svou deklarací/definicí.

3.1 Obecné vlastnosti a datové typy

V programovacím jazyce IFJ17 **nezáleží** na velikosti písmen u identifikátorů i klíčových slov (tzv. *case-insensitive*).

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ('_') začínající písmenem nebo podtržítkem.
- Jazyk IFJ17 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory[4]:

  `As`, `Asc`, `Declare`, `Dim`, `Do`, `Double`, `Else`, `End`, `Chr`,
  `Function`, `If`, `Input`, `Integer`, `Length`, `Loop`, `Print`, `Return`,
  `Scope`, `String`, `SubStr`, `Then`, `While`.

  Rezervovaná klíčová slova jsou:

  `And`, `Boolean`, `Continue`, `Elseif`, `Exit`, `False`, `For`, `Next`,
  `Not`, `Or`, `Shared`, `Static`, `True`.

- *Celočíselný literál* (rozsah C-int) je tvořen neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě

- *Datové typy* pro jednotlivé uvedené literály jsou označeny `Integer`, `Double` a `String`. Typy se používají v definicích proměnných a funkcí a u sémantických kontrol.
- *Term* je libovolný literál (celočíselný, desetinný či řetězcový) nebo identifikátor proměnné.
- Jazyk IFJ17 podporuje *řádkové* i *blokové komentáře* stejně jako FreeBASIC. Řádkový komentář začíná znakem apostrof (''', ASCII hodnota 39) a za komentář je považováno vše, co následuje až do konce řádku. Blokový komentář začíná dvojicí znaků '/''a je ukončen první následující dvojicí znaků ''/', takže hierarchické vnoření blokových komentářů není podporováno.

4 Struktura jazyka

IFJ17 je strukturovaný programovací jazyk podporující deklarace a definice proměnných a uživatelských funkcí, základní řídicí příkazy a příkaz přiřazení a volání funkce včetně rekurzivního. Vstupním bodem řídicího programu je hlavní tělo programu.

4.1 Základní struktura jazyka

Program se skládá ze sekvence deklarací a definic uživatelských funkcí a jako jeho poslední sekce následuje hlavní tělo programu. V těle definice funkce i v hlavním těle programu se pak může nacházet libovolný (i nulový) počet definic lokální proměnných a příkazů jazyka IFJ17.

Jednotlivé konstrukce jazyka IFJ17 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[8]. Ostatní bílé znaky (mezery, tabulátory, komentáře) se mohou vyskytovat v libovolném množství mezi všemi lexémy, i na začátku a na konci zdrojového textu.

4.2 Hlavní tělo programu

Hlavní tělo programu je uvozeno klíčovým slovem `Scope` následovaným sekvencí definic proměnných a dílčích příkazů ukončených koncem řádku (sekvence může být i prázdná). Definice proměnných a příkazy mohou být promíchány. Celá sekvence je ukončena pomocí `End Scope`[9]. Struktura definice proměnné a dílčího příkazu je uvedena v následujících sekcích.

4.2.1 Definice proměnných

Proměnné jazyka IFJ17 jsou pouze lokální. Lokální proměnné a parametry funkcí mají rozsah v daném bloku/funkci, kde byly definovány (od místa jejich definice po konec daného bloku/funkce). Pro každou dílčí definici proměnné s názvem *id* se používá následující

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

[9] V rámci základního zadání není třeba podporovat zanořování bloků.

zápis:
`Dim` *id* `As` *datový_typ*
`Dim` *id* `As` *datový_typ* `=` *výraz*
Definice proměnné obsahuje identifikátor této proměnné *id*, určení typu *datový_typ* (viz sekce 3.1) a případně nepovinnou inicializaci pomocí výrazu *výraz* (viz kapitola 5). Číselné proměnné jsou implicitně inicializovány na hodnotu 0 (resp. 0.0), řetězcové na `!""` (prázdný řetězec).

Nelze definovat proměnnou stejného jména, jako má jiná proměnná v tomtéž bloku nebo některá již deklarovaná/definovaná funkce. Každá v programu použitá proměnná musí být definována, jinak se jedná o sémantickou chybu 3.

4.3 Deklarace a definice uživatelských funkcí

Každá funkce musí být definována právě jednou, deklarována maximálně jednou. Deklarace funkcí slouží pro vzájemné rekurzivní volání dvou či více funkcí. Definice nebo alespoň deklarace funkce musí být vždy k dispozici před prvním voláním této funkce[10] (příkaz volání je definován níže). Deklarace a definice funkce je následujícího tvaru:

- *Deklarace funkce* je ve tvaru:
  `Declare Function` *id* `(` *seznam_parametrů* `) As` *návratový_typ_funkce*
- *Definice funkce* je víceřádková konstrukce (hlavička a tělo) ve tvaru:
  `Function` *id* `(` *seznam_parametrů* `) As` *návratový_typ_funkce* `EOL`
  sekvence_definic_proměnných_a_příkazů
  `End Function`
  - Deklarace/definice jednotlivých formálních parametrů jsou odděleny čárkou (`,`), za poslední z nich se čárka neuvádí. Seznam může být i prázdný. Deklarace parametru má tvar:
    id `As` *datový_typ*
    Identifikátor formálního parametru v deklaraci funkce a v definici funkce se může lišit, ale počty a typy parametrů musí souhlasit. Parametry jsou vždy předávány hodnotou.
  - Tělo funkce je sekvence definic proměnných a dílčích příkazů (viz sekce 4.4). Jejich syntaxe a sémantika je shodná s definicemi a příkazy v hlavním těle programu (viz sekce 4.2). V těle funkce jsou její parametry chápány jako předdefinované lokální proměnné.

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. Redefinice uživatelských funkcí a přetěžování funkcí není povoleno.

4.4 Syntaxe a sémantika příkazů

Dílčím příkazem se rozumí:

[10] Uvažujte například vzájemné rekurzivní volání funkcí (tj. funkce f volá funkci g, která opět může volat funkci f).

- *Příkaz přiřazení*:
  id = *výraz*
  Sémantika příkazu je následující: Příkaz provádí přiřazení hodnoty pravého operandu *výraz* (viz kapitola 5) do levého operandu *id*. Levý operand musí být vždy pouze proměnná (tzv. l-hodnota). Oba operandy musí být stejného nebo kompatibilního typu dle implicitních konverzí (viz sekce 5).
- *Příkaz pro načtení hodnot*:
  `Input` *id*
  Sémantika příkazu je následující: Příkaz nejprve vypíše na standardní výstup řetězec `!"? "`. Poté ze standardního vstupu načte jednu hodnotu dle typu proměnné *id*. Úvodní bílé znaky jsou ignorovány. Vstup je ukončen odřádkováním, které není součástí načtené hodnoty a je ignorováno stejně jako znaky vyskytující se za prvním nevhodným znakem (včetně) pro daný typ načítané hodnoty. Načtená hodnota je po případných implicitních konverzích uložena do proměnné *id*[11]. Hodnota typu `String` může být ohraničena uvozovkami, přičemž ohraničující uvozovky do řetězce nepatří a znaky za ukončující uvozovkou jsou ignorovány. Načítaný řetězec nepodporuje escape sekvence. V případě chybějící hodnoty na vstupu nebo jejího špatného formátu bude proměnná *id* dle jejího typu nastavena na 0, 0.0, nebo `!""`.
- *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'`[12], hodnota výrazu typu `Double` pak pomocí `'% g'`[12]. Všimněte si, že každý výraz je ukončen středníkem. Příkaz musí obsahovat alespoň jeden výraz.
- *Podmíněný příkaz*:
  `If` *výraz* `Then EOL`
  sekvence_příkazů$_1$
  `Else EOL`
  sekvence_příkazů$_2$
  `End If`
  Sémantika příkazu je následující: Nejprve se vyhodnotí daný *výraz* (typicky využívající některý relační operátor). Pokud je vyhodnocený výraz pravdivý, vykoná se *sekvence_příkazů*$_1$, jinak se vykoná *sekvence_příkazů*$_2$. Pokud výsledná hodnota výrazu není pravdivostní (tj. pravda či nepravda - v základním zadání pouze jako výsledek aplikace relačního operátoru dle sekce 5.1), nastává chyba 4. *Sekvence_příkazů*$_1$ a *sekvence_příkazů*$_2$ (mohou být i prázdné) jsou opět sekvence dílčích příkazů definované v této sekci (rekurzivní definice).
- *Příkaz cyklu*:
  `Do While` *výraz* `EOL`

[11] Je-li *id* typu `Integer`, ale načtená hodnota je desetinné číslo, tak je načtená hodnota implicitně zkonvertována místo načtení pouze celočíselné části vstupu.

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

sekvence_příkazů
`Loop`

Sémantika příkazu cyklu je následující: Opakuje provádění sekvence příkazů *sekvence_příkazů* (viz příkazy v této sekci) tak dlouho, dokud je hodnota výrazu pravdivá. Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu v podmíněného příkazu.

- *Volání vestavěné či uživatelem definované funkce*:
  id = *název_funkce*`(`*seznam_vstupních_parametrů*`)`

  Seznam_vstupních_parametrů je seznam termů (viz sekce 3.1) oddělených čárkami[13]. Seznam může být i prázdný. Sémantika vestavěných funkcí bude popsána v kapitole 6. Sémantika volání funkce je následující: Příkaz zajistí předání parametrů hodnotou (včetně případných implicitních konverzí, viz kapitola 5) a předání řízení do těla funkce. V případě, že příkaz volání funkce obsahuje jiný počet nebo typy parametrů, než funkce očekává (tedy než je uvedeno v její hlavičce, a to i u vestavěných funkcí) včetně případné aplikace implicitních konverzí (viz kapitola 5), jedná se o chybu 4. Po návratu z těla funkce (viz dále) dojde k uložení výsledku do *id* a pokračování běhu programu bezprostředně za příkazem volání právě provedené 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 vyhodnocení výrazu *výraz* (tj. získání návratové hodnoty), okamžitému ukončení provádění těla funkce a návratu do místa volání, kam funkce vrátí vypočtenou návratovou hodnotu. Pokud typ výsledku výrazu neodpovídá návratovému typu funkce, dojde k implicitní typové konverzi dle popisu v kapitole 5. V případě nekompatibility typů se jedná o sémantickou chybu. Není-li proveden příkaz `Return` před dokončením provádění funkce, vrací se implicitní hodnota dle návratového typu funkce (0, 0.0, nebo `!""`).

5 Výrazy

Výrazy jsou tvořeny termy, závorkami a binárními aritmetickými, řetězcovým a relačními operátory.

Je-li to nutné, jsou prováděny implicitní konverze operandů, parametrů funkcí i výsledků výrazů či funkcí. Možné implicitní konverze datových typů jsou: (a) `Integer` na `Double`, (b) `Double` na `Integer` (zaokrouhlením s preferencí sudé číslice[14]).

Nesprávné kombinace datových typů (včetně případných povolených implicitních konverzí) jsou považovány za chybu 4.

[13] Parametrem volání funkce není výraz. Jedná se o součást nepovinného bodovaného rozšíření projektu FUNEXP.

[14] Viz `https://cs.wikipedia.org/wiki/Zaokrouhlen%C3%AD`.

5.1 Aritmetické, řetězcové a relační operátory

Standardní binární operátory `+`, `-`, `*` značí sčítání, odčítání[15] a násobení. Jsou-li oba operandy typu `Integer`, je i výsledek typu `Integer`. Je-li jeden[16] či oba operandy typu `Double`, výsledek je též typu `Double`. Operátor `+` navíc provádí se dvěma operandy typu `String` jejich konkatenaci.

Operátor `/` značí dělení dvou číselných operandů, jehož výsledek je bez ohledu na případnou celočíselnost operandů vždy `Double`. U operátoru `\` se jedná o celočíselné dělení stejně jako v jazyce FreeBASIC. Operátor `\` pracuje pouze s operandy typu `Integer`, výsledek je stejného typu.

Pro operátory `<`, `>`, `<=`, `>=`, `=`, `<>` platí, že výsledkem porovnání je pravdivostní hodnota. Tyto operátory pracují s operandy stejného typu, a to `Integer`, `Double` nebo `String`. Je-li jeden operand `Integer` a druhý `Double`, je operand typu `Integer` konvertován na `Double`. Operátory mají stejnou sémantiku jako v jazyce FreeBASIC[17]. U řetězců se porovnání provádí lexikograficky. Bez rozšíření BOOLOP není s výsledkem porovnání možné dále pracovat a lze jej využít pouze u příkazů `If` a `Do While`.

5.2 Priorita operátorů

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

Priorita	Operátory	Asociativita
1	`*` `/`	levá
2	`\`	levá
3	`+` `-`	levá
4	`=` `<>` `<` `<=` `>` `>=`	žádná

6 Vestavěné funkce

Překladač bude poskytovat některé základní vestavěné funkce, které bude možné využít v programech jazyka IFJ17.

Pro generování kódu vestavěných funkcí lze výhodně využít specializovaných instrukcí jazyka IFJcode17.

- `Length(`*s* ***`As String`***`)` `As Integer` – Vrátí délku (počet znaků) řetězce zadaného jediným parametrem *s*. Např. platí, že `Length(!"x\nz") = 3`.
- `SubStr(`*s* `As String,` *i* `As Integer,` *n* `As Integer)` `As String` – Vrátí podřetězec zadaného řetězce *s*. Druhým parametrem *i* je dán začátek požadovaného podřetězce (počítáno od jedničky) a třetí parametr *n* určuje délku podřetězce. Je-li *s* prázdné nebo $i \leq 0$, vrací prázdný řetězec. Je-li $n < 0$ nebo $n > \texttt{Length}(s) - i$, jsou jako řetězec vráceny od *i*-tého znaku všechny zbývající znaky řetězce *s*.

[15] Číselné literály jsou sice nezáporné, ale výsledek výrazu přiřazený do proměnné již záporný být může.
[16] Pak samozřejmě proběhne implicitní konverze druhého operandu též na `Double`.
[17] Povšimněte si přetížení operátoru `=`, který u příkazů slouží pro přiřazení a u výrazů pro porovnání.

- `Asc(`*s* `As String,` *i* `As Integer) As Integer` – Vrátí ordinální hodnotu (ASCII) znaku na pozici *i* v řetězci *s*. Je-li pozice mimo meze řetězce, vrací `0`.
- `Chr(`*i* `As Integer) As String` – Vrátí jednoznakový řetězec se znakem, jehož ASCII kód je zadán parametrem *i*. Případ, kdy je *i* mimo interval $[0;255]$, má nedefinované chování.

7 Implementace tabulky symbolů

Tabulka symbolů bude implementována pomocí abstraktní datové struktury, která je ve variantě zadání pro daný tým označena římskými číslicemi I-II, a to následovně:

I) Tabulku symbolů implementujte pomocí binárního vyhledávacího stromu.

II) Tabulku symbolů implementujte pomocí tabulky s rozptýlenými položkami.

Implementace tabulky symbolů bude uložena v souboru `symtable.c` (případně `symtable.h`). Více viz sekce 12.2.

8 Příklady

Tato kapitola uvádí tři jednoduché příklady řídicích programů v jazyce IFJ17.

8.1 Výpočet faktoriálu (iterativně)

```
/' Program 1: Vypocet faktorialu (iterativne) '/
/' Vcetne ukazky case-insensitive vlastnosti jazyka IFJ17 '/

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
```

8.2 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
```

8.3 Práce s řetězci a vestavěnými funkcemi

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

Scope 'Hlavni telo programu
  Dim s1 As String
  Dim s2 As String
  Dim s1len As Integer

  s1 = !"Toto␣je␣nejaky␣text"
  s2 = s1 + !",␣ktery␣jeste␣trochu␣obohatime"
  Print s1; !"\n"; s2; !"\n";
  s1len = Length(s1)
  s1len = s1len - 4 + 1
  s1 = SubStr(s2, s1len, 4)
  Print !"4␣znaky␣od␣"; s1len; !".␣znaku␣v␣\""; s2; !"\":"; s1; !"\n";
  Print !"Zadejte␣serazenou␣posloupnost␣vsech␣malych␣pismen␣a-h,␣";
  Print !"pricemz␣se␣pismena␣nesmeji␣v␣posloupnosti␣opakovat";
  Input s1
  Do While (s1 <> !"abcdefgh")
    Print !"\nSpatne␣zadana␣posloupnost,␣zkuste␣znovu";
    Input s1
  Loop
End Scope
```

9 Doporučení k testování

Programovací jazyk IFJ17 je schválně navržen tak, aby byl téměř kompatibilní s podmnožinou jazyka FreeBASIC[18]. Pokud si student není jistý, co by měl cílový kód přesně vykonat pro nějaký zdrojový kód jazyka IFJ17, může si to ověřit následovně. Z IS FIT si stáhne ze *Souborů* k předmětu IFJ ze složky *Projekt* soubor `ifj17.bas` obsahující kód, který doplňuje kompatibilitu IFJ17 s překladačem `fbc` jazyka FreeBASIC na serveru `merlin`. Soubor `ifj17.bas` obsahuje definice vestavěných funkcí, které jsou součástí jazyka IFJ17, ale chybí v základních knihovnách jazyka FreeBASIC:

```
' Zajisteni zakladni kompatibility IFJ17->FreeBASIC '

Function Length(s As String) As Integer
  Return Len(s)
End Function

Function SubStr(s as String, i As Integer, n As Integer) As String
  Return Mid(s, i, n)
End Function

' Zde bude nasledovat program jazyka IFJ17 '
```

Váš program v jazyce IFJ17 uložený například v souboru `testPrg.ifj` pak lze provést na serveru `merlin` například pomocí trojice příkazů:

```
cat ifj17.bas testPrg.ifj > tmp.bas
fbc tmp.bas
./tmp < test.in > test.out
```

Tím lze jednoduše zkontrolovat, co by měl provést zadaný zdrojový kód resp. vygenerovaný cílový kód. Je ale potřeba si uvědomit, že jazyk FreeBASIC je nadmnožinou jazyka IFJ17, a tudíž může zpracovat i konstrukce, které nejsou v IFJ17 povolené (např. bohatší syntaxe a sémantika příkazů `Print` a `Input`). Výčet těchto odlišností bude uveden na wiki stránkách a můžete jej diskutovat na fóru předmětu IFJ.

10 Cílový jazyk IFJcode17

Cílový jazyk IFJcode17 je mezikódem, který zahrnuje instrukce tříadresné (typicky se třemi argumenty) a zásobníkové (typicky bez parametrů a pracující s hodnotami na datovém zásobníku). Každá instrukce se skládá z operačního kódu (klíčové slovo s názvem instrukce), u kterého nezáleží na velikosti písmen (tj. case insensitive). Zbytek instrukcí tvoří operandy, u kterých na velikosti písmen záleží (tzv. case sensitive). Operandy oddělujeme libovolným nenulovým počtem mezer či tabulátorů. Odřádkování slouží pro oddělení jednotlivých instrukcí, takže na každém řádku je maximálně jedna instrukce a není povoleno jednu instrukci zapisovat na více řádků. Každý operand je tvořen proměnnou, konstantou nebo návěštím. V IFJcode17 jsou podporovány jednořádkové komentáře začínající mřížkou (#). Kód v jazyce IFJcode17 začíná úvodním řádkem s tečkou následovanou jménem jazyka:

[18] Online dokumentace ze 7. 11. 2016: `http://www.freebasic.net/wiki/wikka.php?wakka=DocToc`

10.1 Hodnotící interpret `ic17int`

Pro hodnocení a testování mezikódu v IFJcode17 je k dispozici interpret pro příkazovou řádku (`ic17int`):

```
ic17int prg.code < prg.in > prg.out
```

Chování interpretu lze upravovat pomocí přepínačů/parametrů příkazové řádky. Nápovědu k nim získáte pomocí přepínače `--help`.

Proběhne-li interpretace bez chyb, vrací se návratová hodnota `0` (nula). Chybovým případům odpovídají následující návratové hodnoty:

- 50 - chybně zadané vstupní parametry na příkazovém řádku při spouštění interpretu.
- 51 - chyba při analýze (lexikální, syntaktická) vstupního kódu v IFJcode17.
- 52 - chyba při sémantických kontrolách vstupního kódu v IFJcode17.
- 53 - běhová chyba interpretace – špatné typy operandů.
- 54 - běhová chyba interpretace – přístup k neexistující proměnné (rámec existuje).
- 55 - běhová chyba interpretace – rámec neexistuje (např. čtení z prázdného zásobníku rámců).
- 56 - běhová chyba interpretace – chybějící hodnota (v proměnné nebo na datovém zásobníku).
- 57 - běhová chyba interpretace – dělení nulou.
- 58 - běhová chyba interpretace – chybná práce s řetězcem.
- 60 - interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti, chyba při otvírání souboru s řídicím programem atd.).

10.2 Paměťový model

Hodnoty během interpretace nejčastěji ukládáme do pojmenovaných proměnných, které jsou sdružovány do tzv. rámců, což jsou v podstatě slovníky proměnných s jejich hodnotami. IFJcode17 nabízí tři druhy rámců:

- globální, značíme GF (Global Frame), který je na začátku interpretace automaticky inicializován jako prázdný; slouží pro ukládání globálních proměnných;
- lokální, značíme LF (Local Frame), který je na začátku nedefinován a odkazuje na vrcholový/aktuální rámec na zásobníku rámců; slouží pro ukládání lokálních proměnných funkcí (Zásobník rámců lze s výhodou využít při zanořeném či rekurzivním volání funkcí.);
- dočasný, značíme TF (Temporary Frame), který slouží pro chystání nového nebo úklid starého rámce (např. při volání nebo dokončování funkce), jenž může být přesunut na zásobník rámců a stát se aktuálním lokálním rámcem. Na začátku interpretace je dočasný rámec nedefinovaný.

K překrytým (dříve vloženým) lokálním rámcům v zásobníku rámců nelze přistoupit dříve, než vyjmeme později přidané rámce.

Další možností pro ukládání nepojmenovaných hodnot je datový zásobník využívaný zásobníkovými instrukcemi.

10.3 Datové typy

Interpret IFJcode17 pracuje s typy operandů dynamicky, takže je typ proměnné (resp. paměťového místa) dán obsaženou hodnotou. Není-li řečeno jinak, jsou implicitní konverze zakázány. Interpret podporuje čtyři základní datové typy (int, bool, float a string), jejichž rozsahy i přesnosti jsou kompatibilní s jazykem IFJ17.

Zápis každé konstanty v IFJcode17 se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení typu konstanty (int, bool, float, string) a samotné konstanty (číslo, literál). Např. `float@1.15`, `bool@true` nebo `int@-5`.

Typ int reprezentuje 32-bitové celé číslo (rozsah C-int). Typ bool reprezentuje pravdivostní hodnotu (true nebo false). Typ float popisuje desetinné číslo (rozsah C-double) a v případě zápisu konstant používejte v jazyce C formátovací řetězec `'%g'`. Literál pro typ string je v případě konstanty zapsán jako sekvence tisknutelných ASCII znaků (vyjma bílých znaků, mřížky (#) a zpětného lomítka (\)) a escape sekvencí, takže není ohraničen uvozovkami. Escape sekvence, která je nezbytná pro znaky s ASCII kódem 000-032, 035 a 092, je tvaru *xyz*, kde *xyz* je dekadické číslo v rozmezí 000-255 složené právě ze tří číslic; např. konstanta

string@retezec\032s\032lomitkem\032\092\032a\010novym\035radkem

reprezentuje řetězec

```
retezec s lomitkem \ a
novym#radkem
```

Pokus o práci s neexistující proměnnou (čtení nebo zápis) vede na chybu 54. Pokus o čtení hodnoty neinicializované proměnné vede na chybu 56. Pokus o interpretaci instrukce s operandy nevhodných typů dle popisu dané instrukce vede na chybu 53.

10.4 Instrukční sada

U popisu instrukcí sázíme operační kód tučně a operandy zapisujeme pomocí neterminálních symbolů (případně číslovaných) v úhlových závorkách. Neterminál ⟨*var*⟩ značí proměnnou, ⟨*symb*⟩ konstantu nebo proměnnou, ⟨*label*⟩ značí návěští. Identifikátor proměnné se skládá ze dvou částí oddělených zavináčem (znak `@`; bez bílých znaků), označení rámce LF, TF nebo GF a samotného jména proměnné (sekvence libovolných alfanumerický a speciálních znaků bez bílých znaků začínající písmenem nebo speciálním znakem, kde speciální znaky jsou: _, -, $, &, %, *). Např. `GF@_x` značí proměnnou `_x` uloženou v globálním rámci.

Na zápis návěští se vztahují stejná pravidla jako na jméno proměnné (tj. část identifikátoru za zavináčem).

Instrukční sada nabízí instrukce pro práci s proměnnými v rámcích, různé skoky, operace s datovým zásobníkem, aritmetické, logické a relační operace, dále také konverzní, vstupně/výstupní a ladicí instrukce.

10.4.1 Práce s rámci, volání funkcí

MOVE ⟨*var*⟩ ⟨*symb*⟩ — Přiřazení hodnoty do proměnné

Zkopíruje hodnotu ⟨*symb*⟩ do ⟨*var*⟩. Např. `MOVE LF@par GF@var` provede zkopírování hodnoty proměnné `var` v globálním rámci do proměnné `par` v lokálním rámci.

CREATEFRAME — Vytvoř nový dočasný rámec

Vytvoří nový dočasný rámec a zahodí případný obsah původního dočasného rámce.

PUSHFRAME — Přesun dočasného rámce na zásobník rámců

Přesuň TF na zásobník rámců. Rámec bude k dispozici přes LF a překryje původní rámce na zásobníku rámců. TF bude po provedení instrukce nedefinován a je třeba jej před dalším použitím vytvořit pomocí CREATEFRAME. Pokus o přístup k nedefinovanému rámci vede na chybu 55.

POPFRAME — Přesun aktuálního rámce do dočasného

Přesuň vrcholový rámec LF ze zásobníku rámců do TF. Pokud žádný rámec v LF není k dispozici, dojde k chybě 55.

DEFVAR ⟨*var*⟩ — Definuj novou proměnnou v rámci

Definuje proměnnou v určeném rámci dle ⟨*var*⟩. Tato proměnná je zatím neinicializovaná a bez určení typu, který bude určen až přiřazení nějaké hodnoty.

CALL ⟨*label*⟩ — Skok na návěští s podporou návratu

Uloží inkrementovanou aktuální pozici z interního čítače instrukcí do zásobníku volání a provede skok na zadané návěští (případnou přípravu rámce musí zajistit další instrukce).

RETURN — Návrat na pozici uloženou instrukcí CALL

Vyjme pozici ze zásobníku volání a skočí na tuto pozici nastavením interního čítače instrukcí (úklid lokálních rámců musí zajistit další instrukce).

10.4.2 Práce s datovým zásobníkem

Operační kód zásobníkových instrukcí je zakončen písmenem „S“. Zásobníkové instrukce načítají chybějící operandy z datového zásobníku a výslednou hodnotu operace ukládají zpět na datový zásobník.

PUSHS ⟨*symb*⟩ — Vlož hodnotu na vrchol datového zásobníku

Uloží hodnotu ⟨*symb*⟩ na datový zásobník.

POPS ⟨*var*⟩ — Vyjmy hodnotu z vrcholu datového zásobníku

Není-li zásobník prázdný, vyjme z něj hodnotu a uloží ji do proměnné ⟨*var*⟩, jinak dojde k chybě 56.

CLEARS — Vymazání obsahu celého datového zásobníku

Pomocná instrukce, která smaže celý obsah datového zásobníku, aby neobsahoval zapomenuté hodnoty z předchozích výpočtů.

10.4.3 Aritmetické, relační, booleovské a konverzní instrukce

V této sekci jsou popsány tříadresné i zásobníkové verze instrukcí pro klasické operace pro výpočet výrazu. Zásobníkové verze instrukcí z datového zásobníku vybírají operandy se vstupními hodnotami dle popisu tříadresné instrukce od konce (tj. typicky nejprve ⟨$symb_2$⟩ a poté ⟨$symb_1$⟩).

ADD ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Součet dvou číselných hodnot

Sečte ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

SUB ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Odečítání dvou číselných hodnot

Odečte ⟨*symb*$_2$⟩ od ⟨*symb*$_1$⟩ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

MUL ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Násobení dvou číselných hodnot

Vynásobí ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ (musí být stejného číselného typu int nebo float) a výslednou hodnotu téhož typu uloží do proměnné ⟨*var*⟩.

DIV ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Dělení dvou desetinných číselných hodnot

Podělí desetinnou hodnotu ze ⟨*symb*$_1$⟩ druhou desetinnou hodnotou ze ⟨*symb*$_2$⟩ a výsledek typu float přiřadí do proměnné ⟨*var*⟩. Použití celočíselného operandu vede na chybu 53. Dělení nulou způsobí chybu 57.

ADDS/SUBS/MULS/DIVS Zásobníkové verze instrukcí ADD, SUB, MUL a DIV

LT/GT/EQ ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Relační operátory menší, větší, rovno

Instrukce vyhodnotí relační operátor mezi ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ (stejného typu; int, bool, float nebo string) a do booleovské proměnné ⟨*var*⟩ zapíše false při neplatnost nebo true v případě platnosti odpovídající relace. Řetězce jsou porovnávány lexikograficky a false je menší než true. Pro výpočet neostrých nerovností lze použít AND/OR/NOT.

LTS/GTS/EQS Zásobníková verze instrukcí LT/GT/EQ

AND/OR/NOT ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Základní booleovské operátory

Aplikuje konjunkci (logické A)/disjunkci (logické NEBO) na ⟨*symb*$_1$⟩ a ⟨*symb*$_2$⟩ nebo negaci na ⟨*symb*$_1$⟩ (NOT má pouze 2 operandy) a výsledek zapíše do ⟨*var*⟩ (všechny operandy jsou typu bool).

ANDS/ORS/NOTS Zásobníková verze instrukcí AND, OR a NOT

INT2FLOAT ⟨*var*⟩ ⟨*symb*⟩ Převod celočíselné hodnoty na desetinnou

Převede celočíselnou hodnotu ⟨*symb*⟩ na desetinné číslo a uloží je do ⟨*var*⟩.

FLOAT2INT ⟨*var*⟩ ⟨*symb*⟩ Převod desetinné hodnoty na celočíselnou (oseknutí)

Převede desetinnou hodnotu ⟨*symb*⟩ na celočíselnou oseknutím desetinné části a uloží ji do ⟨*var*⟩.

FLOAT2R2EINT ⟨*var*⟩ ⟨*symb*⟩ Zaokrouhlení na celé číslo (na sudou)

Převede desetinnou hodnotu ⟨*symb*⟩ na celočíselnou tzv. zaokrouhlením s preferencí sudé číslice[14] a uloží ji do ⟨*var*⟩; např. 1.5 i 2.5 zaokrouhlí na 2.

FLOAT2R2OINT ⟨*var*⟩ ⟨*symb*⟩ Zaokrouhlení na celé číslo (na lichou)

Převede desetinnou hodnotu ⟨*symb*⟩ na celočíselnou tzv. zaokrouhlením s preferencí liché číslice[14] a uloží ji do ⟨*var*⟩; např. 1.5 zaokrouhlí na 1 a 2.5 zaokrouhlí na 3.

INT2CHAR ⟨*var*⟩ ⟨*symb*⟩ Převod celého čísla na znak

Číselná hodnota ⟨*symb*⟩ je dle ASCII převedena na znak, který tvoří jednoznakový řetězec přiřazený do ⟨*var*⟩. Je-li ⟨*symb*⟩ mimo interval $[0;255]$, dojde k chybě 58.

STRI2INT ⟨*var*⟩ ⟨*symb*$_1$⟩ ⟨*symb*$_2$⟩ Ordinální hodnota znaku

Do ⟨*var*⟩ uloží ordinální hodnotu znaku (dle ASCII) v řetězci ⟨*symb*$_1$⟩ na pozici ⟨*symb*$_2$⟩ (indexováno od nuly). Indexace mimo daný řetězec vede na chybu 58.

INT2FLOATS/FLOAT2INTS/ FLOAT2R2EINTS/FLOAT2R2OINTS/ INT2CHARS/STRI2INTS Zásobníkové verze konverzních instrukcí

10.4.4 Vstupně-výstupní instrukce

READ ⟨*var*⟩ ⟨*type*⟩ Načtení hodnoty ze standardního vstupu

Načte jednu hodnotu dle zadaného typu ⟨*type*⟩∈ {int, float, string, bool} (včetně případné konverze vstupní hodnoty float do proměnné typu int) a uloží tuto hodnotu do proměnné ⟨*var*⟩. Formát hodnot je kompatibilní s chováním příkazu `Input` jazyka IFJ17. V případě chybného vstupu bude do proměnné ⟨*var*⟩ uložena implicitní hodnota (dle typu 0, 0.0, false, nebo prázdný řetězec).

WRITE ⟨*symb*⟩ Výpis hodnoty na standardní výstup

Vypíše hodnotu ⟨*symb*⟩ na standardní výstup. Formát výpisu je kompatibilní s příkazem `Print` jazyka IFJ17.

10.4.5 Práce s řetězci

CONCAT ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Konkatenace dvou řetězců

Do proměnné ⟨*var*⟩ uloží řetězec vzniklý konkatenací dvou řetězcových operandů ⟨$symb_1$⟩ a ⟨$symb_2$⟩ (jiné typy nejsou povoleny).

STRLEN ⟨*var*⟩ ⟨*symb*⟩ Zjisti délku řetězce

Zjistí délku řetězce v ⟨*symb*⟩ a délka je uložena jako celé číslo do ⟨*var*⟩.

GETCHAR ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Vrať znak řetězce

Do ⟨*var*⟩ uloží řetězec z jednoho znaku v řetězci ⟨$symb_1$⟩ na pozici ⟨$symb_2$⟩ (indexováno celým číslem od nuly). Indexace mimo daný řetězec vede na chybu 58.

SETCHAR ⟨*var*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Změň znak řetězce

Zmodifikuje znak řetězce uloženého v proměnné ⟨*var*⟩ na pozici ⟨$symb_1$⟩ (indexováno celočíselně od nuly) na znak v řetězci ⟨$symb_2$⟩ (první znak, pokud obsahuje ⟨$symb_2$⟩ více znaků). Výsledný řetězec je opět uložen do ⟨*var*⟩. Při indexaci mimo řetězec ⟨*var*⟩ nebo v případě prázdného řetězec v ⟨$symb_2$⟩ dojde k chybě 58.

10.4.6 Práce s typy

TYPE ⟨*var*⟩ ⟨*symb*⟩ Zjisti typ daného symbolu

Dynamicky zjistí typ symbolu a do ⟨*var*⟩ zapíše řetězec značící tento typ (int, bool, float nebo string). Je-li ⟨*symb*⟩ neinicializovaná proměnná, označí její typ prázdným řetězcem.

10.4.7 Instrukce pro řízení toku programu

Neterminál ⟨*label*⟩ označuje návěští, které slouží pro označení pozice v kódu IFJcode17. V případě skoku na neexistující návěští dojde k chybě 52.

LABEL ⟨*label*⟩ Definice návěští

Speciální instrukce označující pomocí návěští ⟨*label*⟩ důležitou pozici v kódu jako potenciální cíl libovolné skokové instrukce. Pokus o redefinici existujícího návěští je chybou 52.

JUMP ⟨*label*⟩ Nepodmíněný skok na návěští

Provede nepodmíněný skok na zadané návěští ⟨*label*⟩.

JUMPIFEQ ⟨*label*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Podmíněný skok na návěští při rovnosti

Pokud jsou ⟨$symb_1$⟩ a ⟨$symb_2$⟩ stejného typu (jinak chyba 53) a zároveň se jejich hodnoty rovnají, tak provede skok na návěští ⟨*label*⟩.

JUMPIFNEQ ⟨*label*⟩ ⟨$symb_1$⟩ ⟨$symb_2$⟩ Podmíněný skok na návěští při nerovnosti

Jsou-li ⟨$symb_1$⟩ a ⟨$symb_2$⟩ stejného typu (jinak chyba 53), ale různé hodnoty, tak provede skok na návěští ⟨*label*⟩.

JUMPIFEQS/JUMPIFNEQS ⟨*label*⟩ Zásobníková verze JUMPIFEQ a JUMPIFNEQ

Zásobníkové skokové instrukce mají i jeden operand mimo datový zásobník, a to návěští ⟨*label*⟩, na které se případně provede skok.

10.4.8 Ladící instrukce

BREAK Výpis stavu interpretu na `stderr`

Na standardní chybový výstup (stderr) vypíše stav interpretu v danou chvíli (tj. během vykonávání této instrukce). Stav se mimo jiné skládá z pozice v kódu, výpisu globálního, aktuálního lokálního a dočasného rámce a počtu již vykonaných instrukcí.

DPRINT ⟨*symb*⟩ Výpis hodnoty na `stderr`

Vypíše zadanou hodnotu ⟨*symb*⟩ na standardní chybový výstup (stderr). Výpisy touto instrukcí bude možné vypnout pomocí volby interpretu (viz nápověda interpretu).

11 Pokyny 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, zpracovat a ohodnotit, což může vést až ke ztrátě všech bodů z projektu!

11.1 Obecné informace

Za celý tým odevzdá projekt jediný student. Všechny odevzdané soubory budou zkomprimovány programem TAR+GZIP, TAR+BZIP či ZIP do jediného archivu, který se bude jmenovat `xlogin00.tgz`, `xlogin00.tbz` nebo `xlogin00.zip`, kde místo zástupného řetězce `xlogin00` použijte školní přihlašovací jméno **vedoucího** týmu. 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.

Vždy platí, že je třeba při řešení problémů aktivně a konstruktivně komunikovat nejen uvnitř týmu, ale občas i se cvičícím. Při komunikaci uvádějte číslo týmu i login vedoucího.

11.2 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 (␊ zastupuje unixové odřádkování):

```
xnovak01:30␊
xnovak02:40␊
xnovak03:30␊
xnovak04:00␊
```

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 registrované členy týmu (i ty hodnocené 0 %).

Vedoucí týmu je před odevzdáním projektu povinen celý tým informovat o rozdělení bodů. Každý člen týmu je navíc povinen rozdělení bodů zkontrolovat po odevzdání do IS FIT a případně rozdělení bodů reklamovat ještě před obhajobou projektu.

12 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.

12.1 Závazné metody pro implementaci překladače

Projekt bude hodnocen pouze jako funkční celek, a nikoli jako soubor separátních, společně nekooperujících modulů. Při tvorbě lexikální analýzy využijete znalosti konečných automatů. Při konstrukci syntaktické analýzy založené na LL-gramatice (vše kromě výrazů) **povinně** využijte buď **metodu rekurzivního sestupu** (doporučeno), nebo prediktivní analýzu řízenou LL-tabulkou. Výrazy zpracujte pouze 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 překladače je zcela v režii řešitelských týmů. Není dovoleno spouštět další procesy a vytvářet nové či modifikovat existující soubory (ani v adresáři `/tmp`). Nedodržení těchto metod bude penalizováno značnou ztrátou bodů!

12.2 Implementace tabulky symbolů v souboru `symtable.c`

Implementaci tabulky symbolů (dle varianty zadání) proveďte dle přístupů probíraných v předmětu IAL a umístěte ji do souboru `symtable.c`. Pokud se rozhodnete o odlišný způsob implementace, vysvětlete v dokumentaci důvody, které vás k tomu vedly, a uveďte zdroje, ze kterých jste čerpali.

12.3 Textová část řešení

Součástí řešení bude dokumentace vypracovaná ve formátu PDF a uložená v jediném souboru `dokumentace.pdf`. Jakýkoliv jiný než předepsaný formát dokumentace bude ignorován, což povede ke ztrátě bodů za dokumentaci. Dokumentace bude vypracována v českém, slovenském nebo anglickém jazyce v rozsahu cca. 3-5 stran A4.

V dokumentaci popisujte návrh (části překladače a předávání informací mezi nimi), implementaci (použité datové struktury, tabulku symbolů, generování kódu), vývojový cyklus, způsob práce v týmu, speciální použité techniky a algoritmy a různé odchylky od přednášené látky či tradičních přístupů. Nezapomínejte také citovat literaturu a uvádět reference na čerpané zdroje včetně správné citace převzatých částí (obrázky, magické konstanty, vzorce). Nepopisujte záležitosti obecně známé či přednášené na naší fakultě.

Dokumentace musí povinně obsahovat (povinné tabulky a diagramy se nezapočítávají do doporučeného rozsahu):

- 1. strana: jména, příjmení a přihlašovací jména řešitelů (označení vedoucího) + údaje o rozdělení bodů, identifikaci vaší varianty zadání ve tvaru "Tým *číslo*, varianta *X*" a výčet identifikátorů implementovaných rozšíření.
- Rozdělení práce mezi členy týmu (uveďte kdo a jak se podílel na jednotlivých částech projektu; povinně zdůvodněte odchylky od rovnoměrného rozdělení bodů).
- Diagram konečného automatu, který specifikuje lexikální analyzátor.
- LL-gramatiku, LL-tabulku a precedenční tabulku, které jsou jádrem vašeho syntaktického analyzátoru.

Dokumentace nesmí:

- obsahovat kopii zadání či text, obrázky[19] nebo diagramy, které nejsou vaše původní (kopie z přednášek, sítě, WWW, ...).
- být založena pouze na výčtu a obecném popisu jednotlivých použitých metod (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.

12.4 Programová část řešení

Programová část řešení bude vypracována v jazyce C bez použití generátorů lex/flex, yacc/bison či jiných podobného ražení a musí být přeložitelná 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

[19] Vyjma obyčejného loga fakulty na úvodní straně.

vždy za platný považován výsledek překladu na serveru `merlin` bez použití jakýchkoliv dodatečných nastavení (proměnné prostředí, ...).

Součástí řešení bude soubor `Makefile` sloužící pro překlad projektu pomocí příkazu `make`. 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 automaticky.

Binární soubor (přeložený překladač) v žádném případě do archívu nepřikládejte!

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

Veškerá chybová hlášení vzniklá v průběhu činnosti překladače budou vždy vypisována na standardní chybový výstup. Veškeré texty tištěné řídicím programem budou vypisovány na standardní výstup, pokud není explicitně řečeno jinak. Kromě chybových/ladicích hlášení vypisovaných na standardní chybový výstup nebude generovaný mezikód přikazovat výpis žádných znaků či dokonce celých textů, které nejsou přímo předepsány řídicím programem. Základní testování bude probíhat pomocí automatu, který bude postupně vaším překladačem kompilovat sadu testovacích příkladů, kompilát interpretovat naším interpretem jazyka IFJcode17 a porovnávat produkované výstupy na standardní výstup s výstupy očekávanými. Pro porovnání výstupů bude použit program `diff` (viz `info diff`). Proto jediný neočekávaný znak, který bude při hodnotící interpretaci vámi vygenerovaného kódu svévolně vytisknut, povede k nevyhovujícímu hodnocení aktuálního výstupu, a tím snížení bodového hodnocení celého projektu.

12.5 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 `gcc` nepodporuje. Před použitím nějaké vyspělé konstrukce je dobré si ověřit, že jí disponuje i překladač `gcc` na serveru `merlin`. Po vypracování je též vhodné vše ověřit na serveru Merlin, aby při překladu a hodnocení projektu vše proběhlo bez problémů. V *Souborech* předmětu v IS FIT je k dispozici skript `is_it_ok.sh` na kontrolu většiny formálních požadavků odevzdávaného archivu, který doporučujeme využít.

Teoretické znalosti, potřebné pro vytvoření projektu, získáte v průběhu semestru na přednáškách, wiki stránkách a diskuzním fóru IFJ. Postupuje-li Vaše realizace projektu rychleji než probírání témat na přednášce, doporučujeme využít samostudium (viz zveřejněné záznamy z minulých let a detailnější pokyny na wiki stránkách IFJ). Je nezbytné, aby na řešení projektu spolupracoval celý tým. Návrh překladače, základních rozhraní a rozdělení práce lze vytvořit již v první čtvrtině 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` a extrémní případy řešte přímo se cvičícími. Je ale nutné, abyste si vzájemně (nespoléhejte pouze na vedoucího), nejlépe na pravidelných schůzkách týmu, ověřovali skutečný pokrok v práci na projektu a případně včas přerozdělili práci.

Maximální počet bodů získatelný na jednu osobu za programovou implementaci je **20** včetně bonusových bodů za rozšíření projektu.

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

12.6 Pokusné odevzdání

Pro zvýšení motivace studentů pro včasné vypracování projektu nabízíme koncept nepovinného pokusného odevzdání. Výměnou za pokusné odevzdání do uvedeného termínu (několik týdnů před finálním termínem) dostanete zpětnou vazbu v podobě procentuálního hodnocení aktuální kvality vašeho projektu.

Pokusné odevzdání bude relativně rychle vyhodnoceno automatickými testy a studentům zaslána informace o procentuální správnosti stěžejních částí pokusně odevzdaného projektu z hlediska části automatických testů (tj. nebude se jednat o finální hodnocení; proto nebudou sdělovány ani body). Výsledky nejsou nijak bodovány, a proto nebudou individuálně sdělovány žádné detaily k chybám v zaslaných projektech, jako je tomu u finálního termínu. Využití pokusného termínu není povinné, ale jeho nevyužití může být vzato v úvahu jako přitěžující okolnost v případě různých reklamací.

Formální požadavky na pokusné odevzdání jsou totožné s požadavky na finální termín a odevzdání se bude provádět do speciálního termínu „Projekt - Pokusné odevzdání“ předmětu IFJ. Není nutné zahrnout dokumentaci, která spolu s rozšířeními pokusně vyhodnocena nebude. Pokusně odevzdává nejvýše jeden člen týmu (nejlépe vedoucí), který následně obdrží jeho vyhodnocení a informuje zbytek týmu.

12.7 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 (do stanoveného termínu) bude postupně (případně i na váš popud) aktualizován ceník rozšíření a identifikátory rozšíření projektu (viz wiki stránky a diskuzní 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 20 bodů.

12.7.1 Bodové hodnocení některých rozšíření jazyka IFJ17

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka FreeBASIC. Podrobnější informace lze získat ze specifikace jazyka[2] FreeBASIC. Do dokumentace je potřeba kromě zkratky na úvodní stranu také uvést, jak jsou implementovaná rozšíření řešena.

- SCOPE: Podporujte libovolné zanořování bloků (sekvence definic proměnných a příkazů uvozených klíčovým slovem `Scope` a ukončených dvojicí `End Scope`) včetně správné práce s viditelností lokálních proměnných dle jazyka FreeBASIC. Při definici lokální proměnné stejného jména, jako má již některá proměnná na vyšší úrovni, je pod daným identifikátorem viditelná ona lokální proměnná. Umožněte také definice proměnných přímo ve větvích podmíněného příkazu a v sekvenci příkazů příkazu cyklu (+1,0 bodu).
- GLOBAL: Podporujte statické proměnné (`Static`) a globální proměnné (`Shared`). Kolize jmen proměnných řešte analogicky jako překladač jazyka FreeBASIC (+1,0 bodu).
- UNARY: Rozšíření zavádí podporu pro unárním operátor − (unární mínus) a přiřazovací příkazy `+=`, `-=`, `*=`, `\=`, `/=`. Priorita a asociativita unárního mínus odpovídá jazyku FreeBASIC (+0,5 bodu).
- 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 (+0,5 bodu).
- CYCLES: Překladač bude podporovat i cykly tvaru `For ... Next` a kompletní podpora `Do ... Loop`[20] včetně příkazů `Continue` a `Exit`. Při definici iterační proměnné stejného jména v příkazu `For`, jako má již některá proměnná na vyšší úrovni, je pod daným identifikátorem v rámci těla cyklu viditelná ona iterační proměnná (+1,5 bodu).
- FUNEXP: Volání funkce může být součástí výrazu, zároveň mohou být výrazy v parametrech volání funkce (+1,5 bodu).
- IFTHEN: Podporujte zjednodušený podmíněný příkaz `If-Then` bez části `Else` a rozšířený podmíněný příkaz s volitelným vícenásobným výskytem `Elseif-Then` části (+0,5 bodu).
- BOOLOP: Podpora typu `Boolean`, booleovských hodnot `True` a `False`, booleovských výrazů včetně kulatých závorek a základních booleovských operátorů (`NOT`, `AND`, `OR`), jejichž priorita a asociativita odpovídá jazyku FreeBASIC. Pravdivostní hodnoty lze porovnávat jen operátory `=` a `<>`. Dále podporujte výpisy a definice proměnných typu `Boolean`. Implicitní hodnotou booleovské proměnné je `False` (+1,0 bodu).
- ...

13 Opravy zadání

- 22. 9. 2017 – doplnění komentáře v IFJcode17, # je nepovolený znak v řetězci IFJcode17
- 3. 10. 2017 – oprava překlepu v instrukci JUMIFNEQS na JUMPIFNEQS
- 9. 10. 2017 – oprava chyby v nadpisu sekce 12.1 („interpretu“ → „překladače“)

[20] Viz `https://www.freebasic.net/wiki/wikka.php?wakka=KeyPgDoloop`.