ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Jakub Křoustek, Zbyněk Křivka, Lukáš Vrábel
email: {ikroustek, krivka, ivrabel}@fit.vutbr.cz
26. září 2011

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ11.
Informace: diskusní fórum IFJ v IS FIT
Pokusné odevzdání: neděle 20. listopadu 2011, 23:59 (nepovinné)
Datum odevzdání: neděle 11. prosince 2011, 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 těchto 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 IFJ11 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 interpretace, kdy je struktura programu v pořádku, ale chyba nastala až za běhu programu při interpretaci konkrétních dat (dělení nulou, nekompatibilní typ hodnoty výrazu 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 IFJ11 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 IFJ11 je podmnožinou jazyka Lua[1], což je skriptovací jazyk často používaný například ve hrách.

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ11 je case-sensitive (při porovnání jmen záleží na velikosti písmen) a je jazykem netypovaným, takže každá proměnná má svůj typ určen hodnotou do ní přiřazenou.

[1] Online dokumentace ve verzi z 16.9.2011: `http://www.lua.org/manual/5.1/manual.html`

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ("_") začínající písmenem nebo podtržítkem. Jazyk IFJ11 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory:

do,	**else**,	**end**,	**false**,	**function**,
if,	**local**,	**nil**,	**read**,	**return**,
then,	**true**,	**while**,	**write**.	

  Jazyk IFJ11 také obsahuje několik *rezervovaných slov*, které nemají specifický význam, ale nelze je použít jako identifikátor: **and**, **break**, **elseif**, **for**, **in**, **not**, **or**, **repeat** a **until**.
- *Číselný literál* (rozsah C-double) vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou částí[2], nebo 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).
- *Řetězcový literál* je ohraničen uvozovkami ("**"**, ASCII hodnota 34) z obou stran. Tvoří ho znaky s ASCII hodnotou větší než 1. Řetězec může obsahovat escape sekvence: '**\n**', '**\t**', '****', '**\"**'. Jejich význam se shoduje s odpovídajícími znakovými literály jazyka C. Znak v řetězci může být zadán také pomocí escape sekvence '*****ddd*', kde *ddd* je číslo od 001 do 255. Délka řetězce není omezena (snad jen velikostí haldy interpretu). Například řetězec

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

  bude interpretován (např. vytisknut) jako

  ```
  "Ahoj
  Svete!"
  ```
- *Logický literál* nabývá hodnoty buď **true**, nebo **false**.
- *Datové typy* pro jednotlivé uvedené literály jsou označeny *number*, *string*, resp. *boolean*. Speciálním případem je typ *nil*, který nabývá pouze hodnoty **nil**. Typy se používají pouze interně a slouží například při provádění implicitních konverzí a sémantických kontrol.
- Jazyk IFJ11 podporuje *řádkové komentáře*. Komentář je označen pomocí dvojité pomlčky ("**--**", ASCII hodnota 45) a za komentář je považováno vše, co následuje až do konce řádku.
- Jazyk IFJ11 podporuje *blokové komentáře*. Tyto komentáře začínají posloupností symbolů "**--[[**" a jsou ukončeny dvojicí symbolů "**]]**". Vnořené blokové komentáře neuvažujte.

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

4 Struktura jazyka

IFJ11 je strukturovaný programovací jazyk podporující definice proměnných a funkcí včetně jejich rekurzivního volání. Vstupním bodem řídicího programu je hlavní tělo programu (funkce **`main`**).

4.1 Základní struktura jazyka

Program se skládá ze sekce definic funkcí následované hlavním tělem programu. Jednotlivé konstrukce jazyka IFJ11 lze zapisovat na jednom či více řádcích. Každý příkaz je vždy ukončen znakem "`;`" (středník, ASCII hodnota 59).

4.2 Sekce definice a deklarace funkcí

Sekce je tvořena definicemi (které jsou zároveň deklaracemi) funkcí. Každá funkce musí být definována právě jednou. Definice funkce musí být vždy k dispozici před prvním voláním funkce s výjimkou rekurze, kdy může funkce volat sama sebe (příkaz volání je definován níže; neuvažujte vzájemnou rekurzi). Definice funkcí mají následující tvar:

- *Definice funkce* je konstrukce ve tvaru:
  `function` *id* **`(`** *seznam_parametrů* **`)`**
  sekvence_deklarace_proměnných
  sekvence_příkazů
  `end`
  - Seznam parametrů je tvořen posloupností identifikátorů oddělených čárkou, přičemž za posledním z nich se čárka neuvádí. 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).

4.3 Sekce hlavního těla programu

Hlavní tělo programu je uvozeno hlavičkou funkce **`main`** následovanou sekvencí deklarací proměnných a sekvencí dílčích příkazů (sekvence mohou být i prázdné). Celá sekvence je ukončena klíčovým slovem **`end`** a středníkem, tedy:

`function main ()`
sekvence_deklarace_proměnných
sekvence_příkazů
`end;`

Funkce **`main`** musí být poslední definovanou funkcí v programu. Struktura dílčích deklarací a příkazů je uvedena v následujících sekcích.

4.3.1 Deklarace proměnných

Proměnné jazyka IFJ11 jsou pouze lokální. Pro každou dílčí deklaraci proměnné s názvem *id* se používá následující zápis:

`local` *id* **`;`**

Proměnné jsou implicitně inicializovány na hodnotu **`nil`**. Deklarace proměnných může být spojena i s jejich uživatelskou inicializací, a to následujícím způsobem:

`local` *id* **`=`** *literál* **`;`**

V rámci jedné funkce nelze deklarovat více proměnných stejného jména. Dále nelze deklarovat proměnnou stejného jména jako některá již definovaná funkce.

4.3.2 Syntaxe a sémantika příkazů

Dílčím příkazem se rozumí:

- *Příkaz pro načtení hodnot*:
  id **`= read(`** *formát* **`);`**
  Sémantika příkazu je následující: Příkaz načte hodnotu ze standardního vstupu a uloží ji do proměnné *id*. Datový typ načtené hodnoty a chování funkce jsou určeny prvním parametrem ve formě formátovacího řetězce nebo čísla, a to následovně:

formát	návratový typ	funkce
`"*n"`	*number*	načte číslo
`"*l"`	*string*	načte všechny znaky až po konec řádku[3]
`"*a"`	*string*	načte všechny znaky až po *EOF*
kladné_číslo	*string*	načte daný počet (*kladné_číslo*) znaků

  Formát načítaného čísla je shodný s definicí číselných literálů v sekci 4.1. Pro formát načítaného řetězce pak platí definice řetězcového literálu s tím rozdílem, že není ohraničen uvozovkami a neuvažujeme escape sekvence. V případě nekorektních vstupních hodnot je vrácen **`nil`**.
- *Příkaz pro výpis hodnot*:
  `write(` $výraz_1$**`,`** $výraz_2$**`, ... ,`** $výraz_n$**`);`**
  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 parametrů nesmí být typu *nil* ani *boolean*[4]. Hodnota parametru typu *number* bude vytištěna pomocí `"%g"`[5].
- *Příkaz přiřazení*:
  id **`=`** *výraz* **`;`**
  Sémantika příkazu je následující: Příkaz provádí přiřazení hodnoty pravého operandu do levého operandu. Levý operand musí být vždy pouze proměnná (tzv. l-hodnota).

[3] Přečtený znak konce řádku není součástí načteného řetězce.
[4] Pro určení typu chyby se řiďte pokyny v kapitole 5.
[5] Formátovací řetězec standardní funkce **`printf`** jazyka C.

- *Podmíněný příkaz*:
  if *výraz* **then**
    sekvence_příkazů$_1$
  else
    sekvence_příkazů$_2$
  end;
  Sémantika příkazu je následující: Nejprve vyhodnotí daný výraz. Pokud výsledná hodnota výrazu je rovna hodnotě `false` nebo `nil`, výraz je považován za nepravdivý. Za pravdivý je výraz považován pro všechny ostatní hodnoty jeho výsledku (pozor, i hodnota `0` a prázdný řetězec jsou považovány za pravdivé).
  Pokud je hodnota výrazu pravdivá, vykoná se *sekvence_příkazů*$_1$, jinak se vykoná *sekvence_příkazů*$_2$. *Sekvence_příkazů*$_1$ a *sekvence_příkazů*$_2$ jsou opět sekvence dílčích příkazů (mohou být i prázdné) definované v této kapitole.
- *Příkaz cyklu*:
  while *výraz* **do**
    sekvence_příkazů
  end;
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako v případě podmíněného příkazu. Opakuje provádění sekvence dílčích příkazů (může být i prázdná) 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 literálů a proměnných oddělených čárkami[6]. Stejně jako v jazyce Lua není kontrolován počet zadaných parametrů, protože za chybějící parametry je dosazena hodnota **`nil`** a přebývající parametry jsou ignorovány. 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 proměnné *výstupní_id* a pokračování běhu programu bezprostředně za příkazem volání funkce.
- *Příkaz návratu z funkce*:
  return *výraz;*
  Příkaz může být použit v těle libovolné funkce. Jeho sémantika je následující: Dojde k vyhodnocení výrazu *výraz* (získání návratové hodnoty), okamžitému ukončení provádění těla funkce a návratu do místa volání, kam funkce vrátí vypočtenou návratovou hodnotu. Výjimkou je funkce **`main`**, která provede vyhodnocení výrazu a ukončení interpretace bez ovlivnění návratové hodnoty interpretu (tj. bezchybně interpretovaný validní program bude i bez provedení příkazu **`return`** vždy vracet `0`). Ukončení uživatelem definované funkce bez provedení **`return`** vrací **`nil`**.

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

5 Výrazy

Výrazy jsou tvořeny literály, již definovanými proměnnými, závorkami nebo výrazy tvořenými binárními aritmetickými a relačními operátory. Na rozdíl od jazyka Lua nejsou implicitní konverze povoleny až na specifikované výjimky. Je-li některý operand dané operace nekompatibilního typu, jedná se o sémantickou chybu (jde-li o literál), jinak jde o chybu interpretace.

5.1 Aritmetické a relační operátory

Operátory `+`, `-`, `*`, `/` a `^` značí sčítání, odčítání, násobení, dělení a umocňování. Platí pro ně: Pokud jsou oba operandy typu *number*, výsledek je typu *number*.

Řetězcový operátor `..` provádí se dvěma operandy typu *string* jejich konkatenaci.

Pro operátor `==` platí: Pokud je první operand jiného typu než druhý operand, výsledek je `false` (takže `0 == "0"` bude vyhodnoceno jako `false`). Pokud jsou operandy stejného typu, tak se porovnají hodnoty daných operandů. Operátor `~=` je negace operátoru `==`.

Pro operátory `<`, `>`, `<=`, `>=` platí: Pokud je první operand stejného typu jako druhý operand, a to *number* nebo *string*, výsledek je typu *boolean*. U řetězců se porovnání provádí lexikograficky. 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):

Priorita	Operátory	Asociativita
1	`^`	← (pravá)
2	`*` `/`	→ (levá)
3	`+` `-`	→
4	`..`	→
5	`<` `>` `<=` `>=` `~=` `==`	→

6 Vestavěné funkce

Interpret bude poskytovat tyto základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ11 bez jejich definice či deklarace):

- **`type(`***id***`)`** – Vrátí řetězec obsahující datový typ proměnné nebo literálu *id* (viz sekce 3.1).
- **`substr(`***string*`,` *number*`,` *number*`)`** – Vrátí podřetězec v zadaném řetězci. První parametr je vstupní řetězec, druhým parametrem je dán začátek požadovaného podřetězce (počítáno od jedničky) a třetí parametr určuje poslední znak podřetězce. Druhý a třetí parametr může být záporný. Desetinná část čísel je případně ořízнuta.

Je-li druhý nebo třetí parametr záporný, **-1** označuje poslední znak řetězce, **-2** předposlední znak řetězce atd. Je-li některý z indexů příliš vysoký nebo nízký, chová se funkce konzistentně s knihovní funkcí `string.sub` jazyka Lua. Například:

```
substr("abc", -4, -4) == ""
substr("abc", -4, -3) == "a"
substr("abc",  3,  6) == "c"
substr("abc",  5,  5) == ""
```

Pokud jsou parametry jiného typu, je vrácen **nil**.

- **find(***string***,** *string***)** – Vyhledá první výskyt zadaného podřetězce v řetězci a vrátí jeho pozici (počítáno od jedničky). První parametr je řetězec, ve kterém bude daný podřetězec vyhledáván. Druhým parametrem je vyhledávaný podřetězec. Prázdný řetězec se vyskytuje v libovolném řetězci na pozici `0`. Pokud podřetězec není nalezen, je vrácena hodnota **false**. Pokud jsou parametry jiného datového typu než *string*, je vrácen **nil**. Pro vyhledání podřetězce v řetězci použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k zadáním je uveden dále.
- **sort(***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. Pokud je parametr jiného datového typu než *string*, je vrácen **nil**. Pro řazení použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k zadáním je uveden dále.

7 Implementace vyhledávání podřetězce v řetězci

Metoda vyhledávání, kterou bude využívat vestavěná funkce **find**, je v zadání pro daný tým označena písmeny 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` (případně `ial.h`). 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 arabskými číslicemi 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` (případně `ial.h`). 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 římskými číslicemi I-II, a to následovně:

I) Tabulku symbolů implementujte pomocí binárního vyhledávacího stromu.

II) Tabulku symbolů implementujte pomocí hashovací tabulky.

Implementace tabulky symbolů bude uložena taktéž v souboru `ial.c` (případně `ial.h`). 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 IFJ11.

10.1 Výpočet faktoriálu (iterativně)

```
-- Program 1: Vypocet faktorialu (iterativne)

function main() --Hlavni telo programu
  local a;
  local vysl;

  write("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n");
  a = read("*n");
  if a < 0 then
    write("Faktorial␣nelze␣spocitat\n");
  else
    vysl = 1;
    while a > 0 do
      vysl = vysl * a;
      a = a - 1;
    end;
    write("Vysledek␣je:␣", vysl, "\n");
  end;
end;
```

10.2 Výpočet faktoriálu (rekurzivně)

```
-- Program 2: Vypocet faktorialu (rekurzivne)

function factorial(n)
  local temp_result;
  local decremented_n;
  local result;

  if n < 2 then
    result = 1;
  else
    decremented_n = n - 1;
    temp_result = factorial(decremented_n);
```

```
    result = n * temp_result;
  end;
  return result;
end

function main() --Hlavni telo programu
  local a;
  local vysl;

  write("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n");
  a = read("*n");
  if a < 0 then
    write("Faktorial␣nelze␣spocitat\n");
  else
    vysl = factorial(a);
    write("Vysledek␣je:␣", vysl, "\n");
  end;
end;
```

10.3 Práce s řetězci a vestavěnými funkcemi

```
-- Program 3: Prace s retezci a vestavenymi funkcemi

function main() --Hlavni telo programu
  local str1 = "Toto␣je␣nejaky␣text␣v␣programu␣jazyka␣IFJ11";
  local str2;
  local str3;
  local p;

  str1 = substr(str1, 1, -25);
  str2 = str1 .. ",␣ktery␣je␣ulozeny␣jako␣";
  str3 = type(str2);
  str2 = str2 .. str3;
  write(str1, "\n", str2, "\n");
  p = find(str2, "text");
  write("Pozice␣retezce␣\"text\"␣v␣retezci␣str2:␣", p, "\n");
  write("Zadejte␣nejakou␣posloupnost␣vsech␣malych␣pismen␣a-h,␣");
  write("pricemz␣se␣pismena␣nesmeji␣v␣posloupnosti␣opakovat\n");
  str1 = read("*l");
  str2 = sort(str1);
  while (str2 ~= "abcdefgh") do
    write("Spatne␣zadana␣posloupnost,␣zkuste␣znovu\n");
    str1 = read("*l");
    str2 = sort(str1);
  end;
  return 0;
end;
```

11 Doporučení k testování

Programovací jazyk je schválně navržen tak, aby byl podmnožinou jazyka Lua. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ11, může si to ověřit následovně. Před program jazyka IFJ11 přidá následující část kódu (obsahuje

definice vestavěných funkcí, které jsou součástí jazyka IFJ11, ale chybí v základních knihovnách jazyka Lua):

```
-- Zajisteni zakladni kompatibility IFJ11->Lua
function read(n)
  return io.read(n);
end

function write(...)
  io.write(unpack(arg));
end

function substr(str, i, j)
  if (type(str) ~= "string" or
      type(i) ~= "number" or type(j) ~= "number") then
    return nil;
  end;
  return string.sub(str, i, j);
end

function find (haystack, needle)
  if type(haystack) ~= "string" or type(needle) ~= "string" then
    return nil;
  elseif needle == "" then
    return 0;
  end;
  local found = string.find(haystack, needle);
  if found == nil then
    return false;
  end;
  return found;
end

function sort (s)
  -- Bubble Sort
  local i = 1;
  local j;
  local n;
  local finished;
  local result = s;

  if (type(s) ~= "string") then
    return nil;
  end;
  n = #s;
  repeat
    finished = true;
    for j = n, i+1, -1 do
      if (string.byte(result, j-1) > string.byte(result, j)) then
        -- Retezce jsou v Lua nemenitelne (immutable) hodnoty
        result = string.sub(result,1,j-2) .. string.sub(result,j,j) ..
                 string.sub(result,j-1,j-1) .. string.sub(result,j+1, n);
        finished = false;
      end;
```

```
    end;
    i = i + 1;
  until (finished or (i == n + 1));
  return result;
end
-- Zde bude nasledovat program jazyka IFJ11
```

Takto vytvořený program lze interpretovat například na serveru `merlin` pomocí příkazů:

```
export LUA_INIT=@nazev_programu.lua
lua -e "main()"
```

Na Windows pak pomocí příkazů:

```
set LUA_INIT=@nazev_programu.lua
lua -e "main()"
```

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že Lua je nadmnožinou jazyka IFJ11 a tudíž může zpracovat i konstrukce, které nejsou v IFJ11 povolené (např. mírně odlišný výpis desetinných čísel nebo sémantika příkazů **`write`** a **`read`**). 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%).

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 WIS a případně rozdělení bodů reklamovat ještě před obhajobou projektu.

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

Projekt bude hodnocen pouze jako funkční celek a nikoli jako soubor separátních dohromady nekooperujících modulů. 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 a vytvářet nové či modifikovat existující soubory (ani v adresáři `/temp`). 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 (označení vedoucího) + údaje o rozdělení bodů, identifikaci vašeho variantního zadání ve tvaru "Tým *číslo*, varianta $\alpha/n/X$" a výčet zvolených rozšíření včetně identifikátorů.
- 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 jak se podílel na jednotlivých částech projektu; příp. zdůvodněte odchylky od rovnoměrného rozdělení bodů).

Dokumentace nesmí:

- Obsahovat kopii zadání či text, obrázky[7] 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.

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!

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 vypisovaných na standardní chybový

[7] Vyjma loga fakulty na úvodní straně.

výstup 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 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 v první 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 Pokusné odevzdání

Pro zvýšení motivace studentů pro včasné vypracování projektu zavádíme nový 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 finál-

ního termínu. Využití pokusného termínu není povinné, ale jeho nevyužití může být vzato v úvahu 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". Není nutné zahrnout dokumentaci, která spolu s rozšířeními pokusně vyhodnocena nebude. Pokusně odevzdává nejvýše jeden člen týmu (nejlépe vedoucí).

13.6 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 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[8], takže stále platí získatelné maximum 25 bodů.

13.6.1 Ceník prémiových bodů za některá rozšíření (začínají identifikátorem):

- BYTECODE: Generování standardního Lua bajt-kódu[9] a jeho následná interpretace. Generování se bude zapínat parametrem `-g`, interpretace bajt-kódu parametrem `-b` (+3 body).
- MINUS: Interpret bude pracovat i s operátorem unární mínus[10]. Do dokumentace je potřeba uvést, jak je tento problém řešen (+1 bod).
- LENGTH: Podpora unárního operátoru "**`#`**", který realizuje výpočet délky řetězce[10] (+0,5 bodu).
- MODULO: Podpora binárního operátoru "**`%`**", který realizuje funkci modulo[10] (+0,5 bodu).
- REPEAT: Interpret bude podporovat i cyklus **`repeat`** ... **`until`** (+1 bod).
- FORDO: Interpret bude podporovat i cyklus **`for`** ... **`do`** ... **`end`** (+1,5 bodu).
- FUNEXP: Funkce mohou být součástí výrazu, zároveň mohou být výrazy v parametrech funkcí (+1 bod).
- IFTHEN: Podpora příkazu **`if`**-**`then`**-**`elseif`**-**`else`**-**`end`** přesně dle manuálu[1]. Části **`elseif`** a **`else`** jsou tedy volitelné. Do dokumentace je potřeba uvést, jak je tento problém řešen[11] (+2 body).

[8] Body za rozšíření se však nepočítají do minima na zápočet!

[9] `http://www.lua.org/source/5.1/lopcodes.h.html`

[10] Uvažujte precedenci operátorů dle manuálu `http://www.lua.org/manual/5.1/manual.html#2.5.6`

[11] 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`**.

- TABLE: Projekt bude pracovat i s proměnnými typu pole (kompatibilní s jazykem Lua[1]). Tj. bude možné deklarovat a definovat tyto proměnné a indexovat jejich hodnoty (+2 body).
- LOGOP: Podpora logických operátorů **`and`**, **`or`** a **`not`** ve výrazech[12] (+1 bod).
- MUTREC: Podpora vzájemné rekurze funkcí (+2 body).
- ...

14 Opravy zadání

- 9. 11. 2011 – Opraveno zobrazování poznámky pod čarou číslo 3.

[12] `http://www.lua.org/manual/5.1/manual.html#2.5.3`