ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Lukáš Vrábel, Jakub Křoustek
email: {krivka, ivrabel, ikroustek}@fit.vutbr.cz
18. září 2012

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ12.
Informace: diskusní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání: neděle 18. listopadu 2012, 23:59 (nepovinné).
Datum odevzdání: neděle 9. prosince 2012, 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, bude také reflektovat procentuální rozdělení bodů a bude zaokrouhlena na celé body.
- Body zapisované za programovou část včetně rozšíření budou také zaokrouhleny. Body nad 20 bodů budou zapsány do termínu "Projekt - Prémiové body".

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. Všechny termíny k projektu najdete v IS FIT nebo na stránkách předmětu[1].

- 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 IFJ12 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 - sémantická chyba v programu – nedefinovaná proměnná.
- 4 - sémantická chyba v programu – nedefinovaná funkce.
- 5 - ostatní sémantické chyby (např. nekorektní operace nad literály atd.).
- 10 - běhová chyba dělení nulou.
- 11 - běhová chyba nekompatibility typů (např. nekorektní operace nad proměnnými atd.).
- 12 - běhová chyba při přetypování na číslo (vestavěná funkce **`numeric`**).
- 13 - ostatní běhové chyby.
- 99 - interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti, chyba při otvírání vstupního souboru atd.).

Jméno souboru s řídicím programem v jazyce IFJ12 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! Znak konce řádku je na důležitých místech označován jako **`EOL`**.

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

3 Popis programovacího jazyka

Jazyk IFJ12 je podmnožinou jazyka Falcon[2], což je moderní skriptovací jazyk kombinující několik programovacích paradigmat.

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ12 je case-sensitive (tj. záleží na velikosti písmen) a je jazykem dynamicky typovaným, takže každá proměnná má svůj typ určen hodnotou do ní přiřazenou.

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ("_") začínající písmenem nebo podtržítkem. Jazyk IFJ12 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory:

  else, **end**, **false**,
  function, **if**, **nil**,
  return, **true**, **while**.

  Jazyk IFJ12 také obsahuje několik *rezervovaných slov*, která nemají specifický význam, ale nelze je použít jako identifikátor: **as**, **def**, **directive**, **export**, **from**, **import**, **launch**, **load** a **macro**[3].
- *Číselný literál* (rozsah C-double) vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou a desetinnou částí[4], 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**". Mezi jednotlivými částmi nesmí být jiný znak, celou a desetinnou část odděluje znak "**.**" (tečka)[5].
- *Řetězcový literál* je ohraničen uvozovkami (", ASCII hodnota 34) z obou stran. Tvoří ho znaky s ASCII hodnotou větší než 0. 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 "**\x***dd*", kde *dd* je dvoumístné hexadecimální číslo od **01** do **FF** (písmena **A**-**F** mohou být malá i velká). Délka řetězce není omezena (snad jen velikostí haldy interpretu). Například řetězec

  "\"Ahoj\nSvete\x21\""

  bude interpretován (např. vytisknut) jako
  "Ahoj
  Svete!"

[2]Online dokumentace k verzi 0.9.6.8 (Chimera) z 16. 9. 2012: `http://falconpl.org/index.ftd?page_id=manuals`

[3]Jazyk Falcon obsahuje ještě řadu dalších klíčových slov, která ale v IFJ12 mohou být použita v rámci rozšíření, nebo nebudou uvažována vůbec.

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

[5]Pozor, celočíselné literály jsou zakázány, protože jazyk IFJ12, na rozdíl od jazyka Falcon, neobsahuje celočíselný datový typ.

- *Logický literál* nabývá hodnoty buď **`true`**, nebo **`false`**.
- *Term* je hodnota (číslo, řetězec, logický literál, **`nil`**) nebo identifikátor proměnné (nesmí se jednat o klíčové nebo rezervované slovo, ani identifikátor vestavěné funkce).
- *Datové typy* pro jednotlivé uvedené literály jsou označeny *numeric*, *string*, resp. *boolean*. Speciálním případem je typ *nil*, který připouští pouze hodnotu **`nil`**. Typy se používají pouze interně a mají význam například při provádění implicitních konverzí a sémantických kontrol (statických i dynamických).
- Jazyk IFJ12 podporuje *řádkové* a *blokové komentáře* podobně jako v jazyce C/C++. Řádkový komentář začíná dvojicí lomítek ("**`//`**", ASCII hodnota 47) a za komentář je považováno vše, co následuje až do konce řádku (vyjma konce řádku).
- Blokový komentář začíná posloupností symbolů "**`/*`**" a je ukončen dvojicí symbolů "**`*/`**". Vnořené blokové komentáře neuvažujte.

4 Struktura jazyka

IFJ12 je strukturovaný programovací jazyk podporující definice proměnných a funkcí včetně jejich rekurzivního volání. Vstupním bodem interpretovaného programu je neoznačená nesouvislá sekvence příkazů mezi definicemi uživatelských funkcí, tzv. *hlavní tělo* programu.

4.1 Základní struktura jazyka

Program se skládá z definic funkcí prolínající se s hlavním tělem programu. Jednotlivé konstrukce jazyka IFJ12 lze zapisovat na jednom či více řádcích dle níže uvedených pravidel. Na každém řádku se nachází nejvýše jeden příkaz jazyka IFJ12. Mezi libovolnými dvěma lexémy může být libovolný počet bílých znaků (mezera, tabulátor, komentář atd.). Výjimkou jsou znaky konců řádků, které se mohou vyskytovat pouze mezi příkazy a definicemi funkcí (viz příklady v kapitole 10).

4.2 Definice funkcí

Definice funkce je zároveň její deklarací. Každá funkce musí být definována právě jednou (tj. i opakovaná definice funkce je chyba; nelze tedy například přetěžovat funkce). Definice funkce nemusí být vždy k dispozici před prvním voláním této funkce. Uvažujte také možnost vzájemné rekurze (tj. funkce f volá funkci g, která opět může volat funkci f). Definice funkcí mají následující tvar:

- *Definice funkce* je konstrukce ve tvaru:
  `function` *id* **`(`** *seznam_parametrů* **`) EOL`**
  sekvence_příkazů
  `end EOL`
  - Seznam parametrů je tvořen posloupností identifikátorů oddělených čárkou, přičemž za posledním z nich se čárka neuvádí.
  - Tělo funkce je sekvence příkazů zakončená klíčovým slovem **`end`**. Syntaxe a sémantika těchto příkazů je shodná s příkazy v hlavním těle programu (viz sekce 4.3.2 níže).

4.3 Hlavní tělo programu

Hlavní tělo programu je neoznačená sekce příkazů jazyka IFJ12, které se prolínají s definicemi funkcí. Hlavní tělo programu může být tvořeno i prázdnou sekvencí příkazů, kdy je pouze vrácena návratová hodnota programu (možné hodnoty viz kapitola 2). Celá sekvence je ukončena koncem zdrojového souboru. Struktura jednotlivých příkazů je popsána v následujících sekcích.

4.3.1 Proměnné

Proměnné jazyka IFJ12 jsou pouze lokální (i v případě definice v hlavním těle programu). Jazyk IFJ12 neobsahuje specifický příkaz pro deklaraci proměnné, ale deklarace proběhne v rámci první definice dané proměnné tj. přiřazením hodnoty výrazu do proměnné nebo příkazem volání funkce (viz popis příkazů níže). Proměnné do doby jejich první definice nejsou inicializovány žádnou implicitní hodnotou (ani hodnotou `nil`). Tím pádem proměnnou nelze použít před její definicí, jinak interpret skončí s chybou 3. Dále nelze definovat proměnnou stejného jména jako některá definovaná funkce (chyba 5).

4.3.2 Syntaxe a sémantika příkazů

Dílčím příkazem se rozumí:

- *Příkaz přiřazení*:
  id **=** *výraz* **EOL**
  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).
- *Příkaz výběru podřetězce*:
  id **=** *řetězec* **[** *od* **:** *do* **]** **EOL**
  Sémantika příkazu je následující: Příkaz provádí přiřazení podřetězce z termu *řetězec* (typu *string*; jinak chyba 5) do proměnné *id*. Termy *od* a *do* jsou nezáporná čísla a mohou být vynechány. Desetinná část čísel je případně oříznuta. Term *od* indexuje první znak (indexováno od nuly) a *do* určuje znak za posledním znakem vybíraného podřetězce řetězce *řetězec*. Je-li hodnota *od* nebo *do* mimo interval $\langle$`0.0`, len(*řetězec*)$\rangle$ nebo $do < od$, je vybraný podřetězec prázdný. Je-li vynechána hodnota *od*, začíná vybíraný podřetězec na začátku řetězce *řetězec* (tj. *od* = `0.0`). Je-li vynechána hodnota *do*, končí vybíraný podřetězec na konci řetězce *řetězec* (tj. *do* = len(*řetězec*)).

```
"abc"[0.0:0.0] == ""
"abc"[1.0:]  == "bc"
"abc"[:1.0]  == "a"
"abc"[1.2:2.9]  == "b"
"abc"[2.0:5.0] == ""
"abc"[5.0:5.0] == ""
"abc"[2.0:1.0] == ""
```

- *Podmíněný příkaz*:
  if *výraz* **EOL**
  sekvence_příkazů$_1$
  else EOL
  sekvence_příkazů$_2$
  end EOL
  Sémantika příkazu je následující: Nejprve vyhodnotí daný výraz. Pokud výsledná hodnota výrazu je rovna hodnotám **false**, **0.0**, **nil** nebo **""**, tak je výraz považován za nepravdivý. Za pravdivý je výraz považován pro všechny ostatní hodnoty jeho výsledku.
  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* **EOL**
  sekvence_příkazů
  end EOL
  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 nebo uživatelem definované funkce*:
  id **=** *název_funkce* **(***seznam_vstupních_parametrů***) EOL**
  Seznam_vstupních_parametrů je seznam termů oddělených čárkami[6]. Stejně jako v jazyce Falcon 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 *id* a pokračování běhu programu bezprostředně za příkazem volání funkce.
- *Příkaz návratu z funkce*:
  return *výraz* **EOL**
  Příkaz může být použit v těle libovolné funkce nebo v hlavním těle programu. Jeho sémantika je následující: Dojde k vyhodnocení výrazu *výraz* (tj. získání návratové hodnoty), okamžitému ukončení provádění těla funkce (či celého programu) a návratu do místa volání, kam funkce vrátí vypočtenou návratovou hodnotu. Výjimkou je hlavní tělo programu, kde se provede vyhodnocení výrazu a ukončení interpretace bez ovlivnění návratové hodnoty interpretu (tj. bezchybně interpretovaný validní program bude i s provedením příkazu například **return 10.0** vždy vracet 0). Hlavní tělo programu ani uživatelem definovaná funkce nemusí příkaz **return** vůbec obsahovat. Ukončení uživatelem definované funkce bez provedení **return** vrací jako výsledek funkce hodnotu **nil**.

[6]Poznámka: Parametrem funkce není výraz. Jedná se o součást nepovinného bodovaného rozšíření projektu FUNEXP.

5 Výrazy

Výrazy jsou tvořeny termy, závorkami nebo výrazy tvořenými binárními aritmetickými a relačními operátory. Na rozdíl od jazyka Falcon nejsou implicitní konverze povoleny až na specifikované výjimky. Je-li některý operand dané operace nekompatibilního typu, jedná se o sémantickou chybu (tj. chyba 5, jde-li o literál), jinak jde o chybu interpretace (tj. chyby 10 až 13).

5.1 Aritmetické a relační operátory

Operátory `+`, `-`, `*`, `/` a `**` značí sčítání (případně konkatenaci), odčítání, násobení (případně mocninu řetězce), 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. V případě, že je druhý operátor jiného typu, tak je převeden implicitní konverzí na typ *string* se sémantikou konverze popsané u vestavěné funkce **`print`**.

Operátor `*` vytvoří ze dvou operandů typu *string* a *number* *n*-tou mocninu řetězce daného prvním operandem (kde *n* je celočíselná část hodnoty druhého operandu, který musí být nezáporný). V případě, že je druhý operátor jiného typu, je výraz považován za chybný.

Příklad použití operátoru `*` nad řetězcem a číslem:

```
"abc" * 3.9 == "abcabcabc"
```

Pro operátor `==` platí: Pokud je první operand jiného typu než druhý operand, výsledek je **`false`** (takže `0.0 == "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	`**`	levá
2	`*` `/`	levá
3	`+` `-`	levá
4	`<` `>` `<=` `>=` `!=` `==`	levá

6 Vestavěné funkce

Interpret bude poskytovat tyto základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ12 bez jejich definice):

- **`input()`** – Načte hodnotu ze standardního vstupu a vrátí ji jako řetězec. Přečtený znak konce řádku není součástí načteného řetězce.
- **`numeric(`***id*`)`** – Vrátí číslo vzniklé převodem hodnoty termu *id* typu řetězec nebo číslo. Je-li parametrem logická hodnota nebo **`nil`**, dojde k ukončení interpretace s chybou 12. Při konverzi z řetězce jsou nejprve ořízнuty všechny uvozující bílé znaky a pak je načteno číslo dle definice číselného literálu v sekci 3.1 až po první nevyhovující znak (zbytek řetězce včetně tohoto znaku je ignorován) nebo konec řetězce. Je-li takto načtené číslo nevyhovující definici číselného literálu, dojde k chybě 12.
- **`print (`** $term_1$`,` $term_2$`, ...)` – Vypíše hodnoty termů $term_1$, $term_2$, ... (viz kapitola 3.1) na standardní výstup ihned za sebe, bez jakýchkoliv oddělovačů a v patřičném formátu. Pro parametry hodnoty **`nil`**, **`true`**, nebo **`false`** bude vytištěn řetězec `Nil`[7], `true`, nebo `false`. Hodnota parametru typu *numeric* bude vytištěna pomocí `"%g"`[8]. Vestavěná funkce **`print`** vrací hodnotu **`nil`**.
- **`typeOf(`***id*`)`** – Vrátí číselný identifikátor datového typu proměnné, funkce nebo literálu *id* (viz následující tabulka).

Typ	Číselný identifikátor typu
nil	`0.0`
pravdivostní literál	`1.0`
číslo	`3.0`
funkce	`6.0`
řetězec	`8.0`

- **`len(`***id*`)`** – Vrátí délku (počet znaků) řetězce zadaného proměnnou nebo literálem *id*. Pro ostatní základní typy vrací `0.0`.
- **`find(`***string*`,` *string*`)`** – Vyhledá první výskyt zadaného podřetězce v řetězci a vrátí jeho pozici (počítáno od nuly). První parametr je řetězec, ve kterém bude daný podřetězec vyhledáván. Druhým parametrem je vyhledávaný podřetězec. Prázdný řetězec se vyskytuje v libovolném řetězci na pozici `0.0`. Pokud podřetězec není nalezen, je vrácena hodnota `-1.0`. Pokud jsou parametry jiného datového typu než *string*, nastane chyba 11. Pro vyhledání podřetězce v řetězci použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k variantám zadání je uveden dále.
- **`sort(`***string*`)`** – Seřadí znaky v daném řetězci tak, aby znak s nižší ordinální hodnotou vždy předcházel znak s vyšší ordinální hodnotou. Vrácen je řetězec obsahující seřazené znaky. Pokud je parametr jiného datového typu než *string*, nastane chyba 11. Pro řazení použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k variantám zadání je uveden dále.

[7] TOTO NENÍ PŘEKLEP!

[8] Formátovací řetězec standardní funkce **`printf`** jazyka C.

7 Implementace vyhledávání podřetězce v řetězci

Metoda vyhledávání, kterou bude využívat vestavěná funkce **`find`**, je ve variantě 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 ve variantě 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 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 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 IFJ12.

10.1 Výpočet faktoriálu (iterativně)

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

// Hlavni telo programu
x = print("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n")
a = input()
a = numeric(a)

if a < 0.0
    x = print("Faktorial␣nelze␣spocitat\n")
else
    vysl = 1.0
    while a > 0.0
        vysl = vysl * a
        a = a - 1.0
    end
    x = print("Vysledek␣je:␣", vysl, "\n")
end
```

10.2 Výpočet faktoriálu (rekurzivně)

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

// Hlavni telo programu
z = print("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n")
a = input()
a = numeric(a)

// Definice funkce pro vypocet hodnoty faktorialu
function factorial(n)
    if n < 2.0
        result = 1.0
    else
        decremented_n = n - 1.0
        temp_result = factorial(decremented_n)
        result = n * temp_result
    end
    return result
end

if a < 0.0  // Pokracovani hlavniho tela programu
    zz = print("Faktorial␣nelze␣spocitat\n")
else
    vysl = factorial(a)
    zzz = print("Vysledek␣je:␣", vysl, "\n")
end
```

10.3 Práce s řetězci a vestavěnými funkcemi

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

function MyMain(str) // Hlavni telo programu
    str1 = str[:19.0]
    str2 = str1 + ",␣ktery␣je␣ulozeny␣jako␣"
```

```
    t = typeOf(str2)
    str2 = str2 + t
    x = print(str1, "\n", str2, "\n")
    p = find(str2, "text")
    x = print("Pozice retezce \"text\" v retezci str2: ", p, "\n")
    x = print("Zadejte nejakou posloupnost vsech malych pismen a-h, ")
    x = print("pricemz se pismena nesmeji v posloupnosti opakovat\n")
    str1 = input()
    str2 = sort(str1)
    while str2 != "abcdefgh"
        x = print("Spatne zadana posloupnost, zkuste znovu\n")
        str1 = input()
        str2 = sort(str1)
    end
    return 0.0
end

foo = MyMain("Toto je nejaky text v programu jazyka IFJ12")
```

11 Doporučení k testování

Programovací jazyk je schválně navržen tak, aby byl podmnožinou jazyka Falcon. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ12, může si to ověřit následovně. Vytvoří soubor `ifj12.fal` obsahující následující kód (obsahuje definice vestavěných funkcí, které jsou součástí jazyka IFJ12, ale chybí v jazyce Falcon):

```
// Modul pro zajisteni zakladni kompatibility IFJ12->Falcon
export find, sort

function find(str, needle)
    if needle == ""
        return 0
    end

    return strFind(str, needle)
end

function sort(str)
    // BubbleSort
    if len(str) <= 1
        return str
    end
    data = strSplit(str)
    newn = n = len(data) - 1
    while(newn > 0)
        newn = 0
        for i = 1 to n
            if data[i-1] > data[i]
                tmp = data[i]
                data[i] = data[i-1]
                data[i-1] = tmp
                newn = i
            end
```

```
        end
        n = newn
    end
    return strMerge(data)
end
```

Program v jazyku IFJ12 pak lze interpretovat například na serveru `merlin` pomocí příkazu[9]:

```
falcon -p ifj12 nazev_programu.fal
```

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že Falcon je nadmnožinou jazyka IFJ12 a tudíž může zpracovat i konstrukce, které nejsou v IFJ12 povolené (např. mírně odlišný výpis desetinných čísel nebo sémantika funkcí **`print`** a **`input`**). Výčet těchto odlišností bude uveden na wiki stránkách a můžete jej diskutovat na 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 a ohodnotit, což může vést až k úplnému 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 `xlogin00.tgz`, či `xlogin00.zip`, kde `xlogin00` je Vaše školní přihlašovací jméno. 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:

[9] Stejný příkaz je po nainstalování interpretu jazyka Falcon použitelný i na OS MS Windows.

```
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 IS FIT 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, společně 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 vaší varianty zadání ve tvaru "Tým *číslo*, varianta $\alpha/n/X$" a výčet identifikátorů implementovaných 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 jak se podílel na jednotlivých částech projektu; příp. zdůvodněte odchylky od rovnoměrného rozdělení bodů).
- Literatura, reference na čerpané zdroje včetně správné citace převzatých částí (obrázky, magické konstanty, vzorce).

Dokumentace nesmí:

- obsahovat kopii zadání či text, obrázky[10] 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` 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 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 skutečně 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 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.

[10] Vyjma obyčejného loga fakulty na úvodní straně.

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. V Souborech předmětu v IS FIT je k dispozici i skript 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 interpretu, 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 se vzájemně (nespoléhejte pouze na vedoucího), 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** 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, intermediální reprezentace, interpret, vestavěné funkce, tabulka symbolů, 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.

13.5 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 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), který následně obdrží jeho vyhodnocení a informuje zbytek týmu.

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 wiki stránky a 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.

13.6.1 Bodové hodnocení některých rozšíření jazyka IFJ12:

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka Falcon. Podrobnější informace lze získat z manuálu[11].

- MINUS: Interpret bude pracovat i s operátorem unární mínus. Operátor má prioritu 0 (nejvyšší) a je pravě asociativní. Do dokumentace je potřeba uvést, jak je tento problém řešen (+1 bod).
- ZAVINAC: Interpret bude pracovat i s unárním operátorem "`@`" (ASCII hodnota 64), jehož operandem je řetězcový literál. Operátor provede "rozbalení" hodnot všech proměnných uvozených znakem "`$`" (ASCII hodnota 36) v rámci tohoto řetězce. Operace bude součástí výrazů používaných v ostatních konstrukcích jazyka IFJ12. Bližší informace viz kapitola "String expansion operator" v manuálu[11] (+1 bod). Příklad:

```
num = 123.0
str = @"num␣is␣$num"
z = print(str) // "num is 123"
```

- LOOP: Interpret bude podporovat i cyklus **`loop`** ... **`end`** (+0,5 bodu).
- ARRAY: Mezi termy bude patřit i datový typ pole. Definice proměnné typu pole bude mít tvar:
  id **`= [`** *sekvence_výrazů* **`]`** **`EOL`**
  Sémantika příkazu je následující: Příkaz provádí inicializaci proměnné typu pole sekvencí výrazů oddělených čárkami. Hranaté závorky jsou povinné. Neuvažujte zanořená pole. K jednotlivým prvkům pole se přistupuje pomocí indexace pole (indexuje se od nuly). Detaily viz manuál[11] (+1 bod). Příklad práce s polem:

[11] `http://falconpl.org/docs/pdf/current/FalconSurvivalGuide.pdf`

```
hit_list = [ "Krivka", "Vrabel", "Kroustek" ]
target = hit_list[2.0]
z = print(target) // "Kroustek"
```

- ARRAY2: Podpora zanořených polí. Toto rozšíření vyžaduje implementaci rozšíření ARRAY (+0,5 bodu).
- FORDO: Interpret bude podporovat i cyklus **for**:
  for *id* **in** *array* **EOL**
  sekvence_příkazů
  end EOL
  kde *id* je proměnná (re)definovaná příkazem **for** a *array* je literál, nebo proměnná typu pole. Sémantika příkazu viz manuál[11]. Toto rozšíření vyžaduje implementaci rozšíření ARRAY (+1 bod).

```
for target in hit_list
    z = print(target, "␣") //"Krivka Vrabel Kroustek "
end
```

- FUNEXP: Funkce mohou být součástí výrazu, zároveň mohou být výrazy v parametrech funkcí (+1 bod).
- LOGOP: Podpora úplných logických výrazů včetně kulatých závorek, logických operátorů **and**, **or** a **not** a relačních operátorů **in** a **notin**. Sémantiky operátorů viz manuál[11] (+0,5 bodu).
- ...

14 Opravy zadání

- 18. 9. 2012 – odstraněn středník v příkaze **return**, opraven odkaz u vestavěné funkce **numeric**, upřesnění popisu funkce **len** (délka = počet znaků).
- 21. 9. 2012 – oprava funkce **sort** v kapitole 11.
- 26. 9. 2012 – za příkazem **return** *výraz* doplněn chybějící **EOL**.
- 2. 10. 2012 – oprava chybějící desetinné části v rozšíření ARRAY; oprava komentáře s výpisem pro rozšíření FORDO; v příkladu 10.3 doplněna desetinná část u příkazu **return 0**.
- 9. 10. 2012 – hodnota **nil** je také term.