ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Radim Kocman
email: {krivka, ikocman}@fit.vutbr.cz
22. září 2014

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ14.
Informace: diskusní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání: pátek 28. listopadu 2014, 23:59 (nepovinné).
Datum odevzdání: neděle 14. prosince 2014, 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 především 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 nebo na stránkách předmětu[1].

- Zadání obsahuje více variant. 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. 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 IFJ14 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 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.
- 5 - ostatní sémantické chyby.
- 6 - běhová chyba při načítání číselné hodnoty ze vstupu.
- 7 - běhová chyba při práci s neinicializovanou proměnnou.
- 8 - běhová chyba dělení nulou.
- 9 - ostatní běhové chyby.
- 99 - interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti, chyba při otvírání souboru s řídicím programem, špatné parametry příkazové řádky atd.).

Jméno souboru s řídicím programem v jazyce IFJ14 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

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

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 IFJ14 je až na marginální případy podmnožinou jazyka Pascal[2], což je staticky typovaný[3] především procedurální jazyk stále často využívaný pro výuku programování.

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ14 je case insensitive[4].

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ("_") začínající písmenem nebo podtržítkem. Jazyk IFJ14 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory:

  `begin`, **`boolean`**, **`do`**, **`else`**, **`end`**, **`false`**, **`find`**,
  `forward`, **`function`**, **`if`**, **`integer`**, **`readln`**, **`real`**,
  `sort`, **`string`**, **`then`**, **`true`**, **`var`**, **`while`**, **`write`**.

- *Celočíselný literál* (rozsah C-int) je tvořen neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě[5].
- *Desetinný literál* (rozsah C-double) také vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou a desetinnou částí, nebo celou částí a exponentem, nebo celou a desetinnou částí a exponentem. Celá i desetinná část je tvořena neprázdnou posloupností číslic. Celočíselný 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)[6].
- *Řetězcový literál* je ohraničen jednoduchým apostrofem (`'`, ASCII hodnota 39) z obou stran. Tvoří jej libovolný počet znaků zapsaných na jediném řádku programu. Možný je i prázdný řetězec (`''`). Znaky s ASCII hodnotou větší než 31 (mimo `'`) jsou zapisovány přímo. Ostatní znaky lze zapisovat pomocí escape sekvence: "`'#`*i*`'`", kde $i \in \langle 1, 255 \rangle$ je dekadické číslo (případné přebytečné počáteční nuly jsou ignorovány) vyjadřující ASCII hodnotu takto reprezentovaného znaku řetězce. Délka řetězce není omezena (snad jen velikostí haldy pro interpretaci daného programu). Apostrof v řetězci vytvoříme jako dva bezprostředně následující apostrofy. Například řetězcový literál

[2] Implementace Free Pascal jazyka Pascal: `http://www.freepascal.org/`; na serveru Merlin je pro studenty k dispozici překladač `fpc` verze 2.6.4 z 3. března 2014.

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

[4] Nezáleží tedy vůbec na velikosti písmen u identifikátorů i klíčových slov.

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

[6] Pro celou čast desetinného literálu i exponent platí, že přebytečné počáteční číslice `0` jsou ignorovány.

```
'Ahoj'#010'Sve''te'#33"
```

bude interpretován jako

```
Ahoj
Sve'te!
```

- *Booleovský literál* je tvořen konstantami **`true`**, resp. **`false`** a vyjadřuje odpovídající logickou hodnotu.
- *Datové typy* pro jednotlivé uvedené literály jsou označeny **`integer`**, **`real`**, **`string`**, resp. **`boolean`**. Typy se používají v definicích proměnných a funkcí a u sémantických kontrol.
- *Term* je libovolný literál (celočíselný, desetinný, řetězcový či booleovský) nebo identifikátor proměnné.
- Jazyk IFJ14 podporuje *blokové komentáře*. Tyto komentáře začínají symbolem "`{`" (levá složená závorka) a jsou ukončeny symbolem "`}`" (pravá složená závorka). Vnořené blokové komentáře neuvažujte.

4 Struktura jazyka

IFJ14 je strukturovaný programovací jazyk podporující definice proměnných a uživatelských funkcí včetně jejich rekurzivního volání.

4.1 Základní struktura jazyka

Program se skládá ze tří po sobě jdoucích sekcí: (1) definice globálních proměnných, (2) definice uživatelských funkcí a (3) blokem pro *hlavní tělo* programu. Na každém řádku se může nacházet libovolný (i nulový) počet příkazů jazyka IFJ14. Mezi každými dvěma lexémy může být libovolný počet bílých znaků (mezera, tabulátor, odřádkování, komentář atd.)[7].

- sekce definice globálních proměnných je uvozena klíčovým slovem **`var`** a dále obsahuje neprázdnou sekvenci dílčích definic tvaru *id* `:` *typ* `;`, přičemž *typ* může být **`integer`**, **`real`**, **`string`** nebo **`boolean`**. Pokud nebudeme pracovat s žádnou globální proměnnou, lze tuto sekci zcela vynechat (včetně klíčového slova **`var`**).
- dopředné deklarace a definice uživatelských funkcí jsou popsány v podsekci 4.3.
- hlavní tělo programu je stejně jako v jazyce Pascal uvozeno klíčovým slovem **`begin`**, pak následuje sekvence (může být i prázdná) dílčích příkazů oddělených středníkem (`;`) a vše ukončuje klíčové slovo **`end.`** (včetně tečky na konci). Struktura jednotlivých dílčích příkazů je uvedena v sekci 4.4.1.

[7] Na začátku a konci zdrojového textu se též smí vyskytovat libovolný počet bílých znaků.

4.2 Proměnné

Dokud nemá proměnná přiřazenu hodnotu, je neinicializovaná. Je-li s takovou proměnnou manipulováno (předána jako parametr funkci, vrácena jako výsledek funkce, využita jako operand ve výrazu), nastane chyba 7.

Proměnné jazyka IFJ14 jsou globální nebo lokální podle místa jejich definice ve vyhrazených sekcích označených klíčovým slovem **`var`**. Nelze definovat proměnnou stejného jména jako jiná proměnná na stejné úrovni nebo některá deklarovaná/definovaná funkce (chyba 3).

4.3 Definice uživatelských funkcí

Každá uživatelská funkce je deklarována nejvýše jednou a definována právě jednou. Definice funkce se skládá z hlavičky a těla funkce a je zároveň její deklarací (tj. i opakovaná definice funkce stejného jména je chyba; nelze tedy například přetěžovat funkce, viz chyba 3). Definice funkce nemusí vždy lexikálně předcházet kód pro volání této funkce. Uvažujte například vzájemné rekurzivní volání funkcí (tj. funkce f volá funkci g, která opět může volat funkci f). Pokud potřebujeme uživatelskou funkci f volat ve funkci g dříve než je provedena definice f, použijeme tzv. dopřednou deklaraci před definicí funkce g a definici funkce f uvedeme v kódu později (za funkcí g).

- *Definice uživatelské funkce* je konstrukce ve tvaru:
  `function` *id* **`(`** *seznam_parametrů* **`)`** **`:`** *návratový_typ* **`;`**
  definice_lokálních_proměnných
  `begin`
  sekvence_příkazů
  `end` **`;`**
  - Seznam parametrů je tvořen posloupností definicí parametrů oddělených středníkem (**`;`**), přičemž za posledním z nich se středník neuvádí. Seznam může být i prázdný. Každá definice parametru obsahuje identifikátor parametru a jeho datový typ:
    identifikátor_parametru **`:`** *typ*
  - Definice lokálních proměnných je analogická sekci definice globálních proměnných (včetně možného vynechání této sekce). Definovaná lokální proměnná je alokována při zavolání této funkce a uvolněna po návratu z této funkce. Při shodnosti identifikátoru lokální proměnné s globální ji dočasně překryje.
  - Tělo funkce je sekvence (i prázdná) dílčích příkazů oddělených středníkem (**`;`**). Za klíčovým slovem **`end`** píšeme **`;`**. Syntaxe a sémantika dílčích příkazů je popsána v sekci 4.4.1.
  - Každá funkce vrací hodnotu, která je v těle funkce nastavena ve speciální proměnné (implicitně definované) se jménem shodným se jménem funkce a typem odpovídajícím návratovému typu funkce, jenž je uveden v hlavičce funkce.
  - Definice vnořených funkcí neuvažujte.
- *Dopředná deklarace uživatelské funkce* je konstrukce ve tvaru:
  `function` *id* **`(`** *seznam_parametrů* **`)`** **`:`** *návratový_typ* **`;`** **`forward`** **`;`**

– Pro každou funkcí může být v programu v jazyce IFJ14 nejvýše jedna dopředná deklarace. A naopak ke každé dopředné deklaraci musí existovat odpovídající definice funkce se stejnou hlavičkou[8], jinak bude hlášena chyba 3.

4.4 Hlavní tělo programu

Za definicemi uživatelských funkcí je uvedeno hlavní tělo programu obsahující sekvenci dílčích příkazů popsaných v podsekci 4.4.1.

4.4.1 Syntaxe a sémantika příkazů

Dílčím příkazem se rozumí:

- *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) a nedochází k žádným implicitním konverzím.
- *Složený příkaz*:
  `begin` *příkaz*$_1$ `;` *příkaz*$_2$ `;` ... `;` *příkaz*$_n$ **`end`**
  Složený příkaz je sekvence (může být i prázdná) dílčích příkazů oddělených středníkem (`;`) umístěná mezi klíčová slova **`begin`** a **`end`**. Za posledním příkazem sekvence se středník nepíše.
  Sémantika složeného příkazu je následující: Proveď příkazy *příkaz*$_1$, *příkaz*$_2$, ..., *příkaz*$_n$ postupně v tomto pořadí.
- *Podmíněný příkaz*:
  `if` *výraz* **`then`** *složený_příkaz*$_1$ **`else`** *složený_příkaz*$_2$
  Sémantika příkazu je následující: Nejprve vyhodnotí daný výraz. Pokud je výsledná hodnota výrazu typu **`boolean`** a navíc pravdivá (**`true`**), vykonají se příkazy *složený_příkaz*$_1$, jinak se vykoná *složený_příkaz*$_2$. Pokud je *výraz* jiného datového typu, dojde k chybě 4.
- *Příkaz cyklu*:
  `while` *výraz* **`do`** *složený_příkaz*$_3$
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu v podmíněného příkazu. Opakuje provádění příkazů složeného příkazu *složený_příkaz*$_3$ tak dlouho, dokud je hodnota výrazu pravdivá.
- *Volání vestavěné nebo uživatelem definované funkce*:
  id `:=` *název_funkce* `(`*seznam_vstupních_parametrů*`)`
  Seznam_vstupních_parametrů je seznam termů (viz sekce 3.1) oddělených čárkami[9]. Seznam může být i prázdný. Sémantika vestavěných funkcí bude popsána v kapitole 6. Sémantika volání uživatelem definovaných funkcí je následující: Příkaz zajistí předání parametrů hodnotou (bez jakýchkoli implicitních konverzí) a předání řízení

[8] nejen typová signatura, ale i identifikátory parametrů

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

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), jedná se o chybu 4. Na konci těla funkce dojde ke kopii výsledku ze speciální lokální proměnné *název_funkce* do proměnné *id*, funkce je ukončena a běh programu pokračuje bezprostředně za příkazem volání funkce. Je-li proměnná *název_funkce* neinicializovaná, dojde k chybě 7.

- *Příkaz pro načtení hodnoty ze vstupu*:
  `readln(`*id***`)`**
  Sémantika příkazu je následující: Ze standardního vstupu načti jednu hodnotu typu odpovídajícího parametru *id* a hodnotu ulož do proměnné *id*. Pro typ **`string`** načti řetězec ukončený koncem řádku nebo koncem vstupu, kdy symbol konce řádku či konce vstupu již do načteného řetězce nepatří. Je-li *id* číselného typu, načti kladnou nebo zápornou číselnou hodnotu (**`integer`** nebo **`real`**). Prázdné znaky (mezera, tabulátor) před a za hodnotou ignoruj až po konec řádku či konec vstupu. Pokud bude zadáno na řádku více vstupních hodnot, budou ignorovány bez jakékoli jejich kontroly. Pozor na ošetření chyb nekorektnosti vstupní hodnoty (chyba 6)! Je-li *id* typu **`boolean`**, dojde k chybě 4.
- *Příkaz pro výpis hodnot*:
  `write(`*term*$_1$**`,` *term*$_2$**`, ...,` *term*$_n$**`)`**
  Sémantika příkazu je následující: Postupně vypisuj hodnoty jednotlivých termů na standardní výstup ihned za sebe bez žádných oddělovačů v patřičném formátu. Hodnota výrazu typu **`real`** bude vytištěna pomocí **`"%g"`**[10].

5 Výrazy

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

5.1 Aritmetické, řetězcové a relační operátory

Standardní binární operátory **`+`**, **`-`**, **`*`** značí sčítání, odčítání[11] a násobení. Jsou-li oba operandy typu **`integer`**, je i výsledek typu **`integer`**. Je-li jeden[12] či oba operandy typu **`real`**, výsledek je též typu **`real`**. Operátor **`/`** značí dělení, akceptuje operandy typu **`real`** či **`integer`** a výsledkem operace je hodnota typu **`real`**[13].

Operátor **`+`** navíc provádí se dvěma operandy typu **`string`** jejich konkatenaci. V případě, že je druhý operand jiného typu, dochází k sémantické chybě 4.

Pro operátory **`<`**, **`>`**, **`<=`**, **`>=`**, **`=`**, **`<>`** platí: Pokud je první operand stejného typu jako druhý operand, a to **`integer`**, **`real`**, **`string`** nebo **`boolean`**, výsledek je typu **`boolean`**. Výsledkem porovnání je hodnota typu **`boolean`** dle pravdivosti (sémantika operátorů je stejná jako v jazyce Pascal). Porovnání řetězců se provádí lexikograficky.

[10] formátovací řetězec standardní funkce **`printf`** jazyka C

[11] Číselné literály jsou sice nezáporné, ale výsledek výrazu přiřazený do proměnné již záporný být může.

[12] pak proběhne implicitní konverze druhého operandu též na **`real`**

[13] Celočíselné dělení tudíž neuvažujte.

Jiné než uvedené kombinace typů ve výrazech pro dané operátory jsou považovány za chybu 4.

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

6 Vestavěné funkce

Interpret bude poskytovat některé základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ14 bez jejich definice či deklarace). Vestavěné funkce jazyka IFJ14 jsou:

- **`length(`***s* **`: string) : integer`** – Vrátí délku (počet znaků) řetězce zadaného jediným parametrem *s*. Např. **`length('x'#10'z') = 3`**.
- **`copy(`***s* **`: string;`** *i* **`: integer;`** *n* **`: integer) : string`** – Vrátí podřetězec zadaného řetězce *s*. Druhým parametrem *i* je dán začátek požadovaného podřetězce (počítáno od jedničky) a třetí parametr *n* určuje délku podřetězce. V okrajových případech simulujte stejnojmennou funkci z jazyka Pascal.
- **`find(`***s* **`: string;`** *search* **`: string) : integer`** – Vyhledá první výskyt zadaného podřetězce *search* v řetězci *s* a vrátí jeho pozici (počítáno od jedničky). Prázdný řetězec se vyskytuje v libovolném řetězci na pozici `1`. Pokud podřetězec není nalezen, je vrácena hodnota `0`. Pro vyhledání podřetězce v řetězci použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k variantám zadání je uveden v kapitole 7.
- **`sort(`***s* **`: string) : string`** – Seřadí znaky v daném řetězci *s* tak, aby znak s nižší ordinální hodnotou vždy předcházel znaku s vyšší ordinální hodnotou. Vrácen je řetězec obsahující seřazené znaky. Pro řazení použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k variantám zadání je uveden v kapitole 8.

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`). Metodu implementujte dle varianty algoritmu popsané 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`). Danou metodu implementujte dle varianty algoritmu popsané 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 IFJ14.

10.1 Výpočet faktoriálu (iterativně)

```
{ Program 1: Vypocet faktorialu (iterativne) }

var
    a : integer;
    vysl : integer;

begin
    write('Zadejte␣cislo␣pro␣vypocet␣faktorialu:␣');
    readln(a);
    if a < 0 then
    begin
        write('Faktorial␣nelze␣spocitat'#10'')
    end
    else
    begin
        vysl := 1;
        while a > 0 do
```

```
        begin
            vysl := vysl * a;
            a := a - 1
        end;
        write('Vysledek je: ', vysl, ''#10'')
    end
end.
```

10.2 Výpočet faktoriálu (rekurzivně)

```
{ Program 2: Vypocet faktorialu (rekurzivne) }
var
    a : integer;
    vysl : integer;

{ Definice funkce pro vypocet hodnoty faktorialu }
function factorial(n : integer) : integer;
var
    temp_result : integer;
    decremented_n : integer;
begin
    if n < 2 then
    begin
        factorial := 1
    end
    else
    begin
        decremented_n := n - 1;
        temp_result := factorial(decremented_n);
        factorial := n * temp_result
    end
end;

{ Hlavni telo programu }
begin
    write('Zadejte cislo pro vypocet faktorialu: ');
    readln(a);

    if a < 0 then   { Pokracovani hlavniho tela programu }
    begin
        write('Faktorial nelze spocitat'#10'')
    end
    else
    begin
        vysl := factorial(a);
        write('Vysledek je: ', vysl, ''#10'')
    end
end.
```

10.3 Práce s řetězci a vestavěnými funkcemi

```
{ Program 3: Prace s retezci a vestavenymi funkcemi }
var
    str1 : string;
    str2 : string;
```

```
    n : integer;
begin
    str1 := 'Toto je nejaky text';
    str2 := copy(str1, 9, 11);
    write(str1, ''#10'', str2, ''#10'');
    n := find(str2, 'text');
    write('Pozice retezce "text" v str2: ', n, ''#10'');
    write('Zadejte posloupnost vsech malych pismen a-h, ',
          'aby se pismena v posloupnosti neopakovala:');
    readln(str1);
    str2 := sort(str1);
    while str2 <> 'abcdefgh' do
    begin
        write('Spatne zadana posloupnost, zkuste znovu:'#10'');
        readln(str1);
        str2 := sort(str1)
    end
end.
```

11 Doporučení k testování

Programovací jazyk IFJ14 je schválně navržen tak, aby byl téměř kompatibilní s podmnožinou jazyka Pascal. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ14, může si to ověřit následovně. Z IS FIT si stáhněte ze *Souborů* k předmětu IFJ ze složky *Projekt* soubor `ifj14.pas` obsahující kód, který doplňuje kompatibilitu IFJ14 s překladačem jazyka Pascal na serveru `merlin` (obsahuje např. definice vestavěných funkcí, které jsou součástí jazyka IFJ14, ale chybí v jazyce Pascal nebo tam mají mírně odlišnou sémantiku). Váš program v jazyce IFJ14 uložený například v souboru `testovanyProgram.ifj` pak lze interpretovat na serveru `merlin` například pomocí dvojice příkazů:

```
cat ifj14.pas testovanyProgram.ifj > testovanyProgram.pas
fpc testovanyProgram.pas
```

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že jazyk Pascal je nadmnožinou jazyka IFJ14 a tudíž může zpracovat i konstrukce, které nejsou v IFJ14 povolené (např. odlišné implicitní konverze a volnější syntaxe). 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 ke ztrátě všech bodů z projektu!

12.1 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 `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.

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 (`<LF>` zastupuje unixové odřádkování):

```
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 pro kontext jazyka založeného

na LL-gramatice (vše kromě výrazů) 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 interpretu 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!

13.2 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. 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.
- Diagram konečného automatu, který specifikuje lexikální analyzátor.
- LL-gramatiku a precedenční tabulku, které jsou 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[14] 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.

[14] Vyjma obyčejného loga fakulty na úvodní straně.

13.3 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 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 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 ve zkompilovaném 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 `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 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 IFJ14:

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka Pascal. Podrobnější informace lze získat z referenční příručky[2].

- 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 (+0,5 bodu).
- BASE: celočíselné konstanty a escape sekvence[15] je možné zadávat i ve dvojkové (číslo začíná znakem %), osmičkové (číslo začíná znakem &) a v šestnáctkové (číslo začíná znakem $) soustavě (+0,5 bodu).
- ARRAY: Interpret bude podporovat práci s poli (kompatibilními s jazykem Pascal) (+1,5 bodu). Pole je definováno v sekci **var** následovně:
  id **:** **array[** *dolní* **..** *horní* **]** **of** *typ* **;**
  kde **array** a **of** jsou klíčová slova, *dolní* a *horní* jsou nezáporná celá čísla typu **integer** udávající konstantní rozsah pole.
  Pole **a** bude možné indexovat termem pomocí hranatých závorek **a[***term***]**. Tuto konstrukci lze používat ve výrazech, případně jako l-hodnoty u příkazu přiřazení. Vícerozměrná pole a předávání pole jako parametr funkce neuvažujte (nebude testováno).

```
var a : array[0..10] of integer;
    b : array[1..2] of string;
    message : string;
    i : integer;
begin
  a[0] := 10 - 5;
  a[7] := 0 - 10;
  b[1] := 'Ahoj';
  b[2] := '␣svete!';
  message := b[1] + b[2];
  i := a[0] + a[7];
  write(message, ''#10'', i, ''#10'')
end.
```

- REPEAT: Interpret bude podporovat i cyklus **repeat** - **until**. Tělo tohoto cyklu musí obsahovat alespoň jeden příkaz (+1 bod).
- FUNEXP: Volání funkce může být součástí výrazu, zároveň mohou být výrazy v parametrech funkcí (+1 bod).
- ELSEIF: Interpret bude zpracovávat i zjednodušený podmíněný příkaz **if-then** bez části **else** (+0,5 bodu).
- BOOLOP: Podpora booleovských výrazů včetně kulatých závorek a všech booleovských operátorů (**not**, **and**, **or**, **xor**) (+0,5 bodu).
- ...

[15] Implementace Free Pascal verze 2.6.4 nepodporuje správně escape sekvence zadané dvojkovým číslem.

14 Opravy zadání

- 23. 9. 2014 – odstraněno přebytečné klíčové slovo **`div`**, doplněna poznámka [7], upřesnění rozšíření ARRAY.