ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Jakub Křoustek
email: {krivka, ikroustek}@fit.vutbr.cz
11. října 2009

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ09.
Informace: diskusní fórum IFJ ve WIS
Datum odevzdání: neděle 13. prosince 2009, 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 navíc za tvůrčí přístup (různá rozšíření apod.)
Udělení zápočtu je podmíněno získáním min. 20 bodů v průběhu semestru. Navíc v IFJ z těch 20 bodů musíte získat nejméně 5 bodů za programovou část projektu.

Řešitelské týmy:

- Projekt budou řešit čtyř až pětičlenné týmy. Týmy s jiným počtem členů jsou nepřípustné.
- Registrace do týmů se provádí zápisem na příslušnou variantu zadání v IS FIT.
- 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 IFJ09 a interpretuje jej. Jestliže proběhne činnost interpretu bez chyb, vrací se výstupní návratová hodnota `0` (nula). Jestliže došlo k nějaké chybě, vrací se výstupní 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, špatné přiřazení typů atd.)
- 4 = struktura programu je v pořádku, ale chyba nastala až v rámci interpretace konkrétních dat (dělení nulou, na vstupu chybný formát číselné hodnoty atd.)
- 5 = interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti, chyba při otvírání vstupního souboru atd.)

Jméno souboru s řídícím programem v jazyce IFJ09 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 obvyklé výstupy 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

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ09 je case-sensitive a je jazykem typovaným, tzn. jednotlivé proměnné mají předem určen datový typ svou deklarací.

- *Identifikátor* je definován jako neprázdná posloupnost číslic a písmen (malých i velkých) začínající písmenem nebo podtržítkem ("_"). Jazyk IFJ09 obsahuje navíc níže jmenovaná *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory: **`begin`**, **`div`**, **`do`**, **`double`**, **`else`**, **`end`**, **`find`**, **`if`**, **`integer`**, **`readln`**, **`sort`**, **`string`**, **`then`**, **`var`**, **`while`**, **`write`**. Dále není možné deklarovat více proměnných stejného názvu, protože každá proměnná má svůj jedinečný název.
- *Celočíselná konstanta* (rozsah C-int) je tvořena neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě[1].
- *Desetinná konstanta* (rozsah C-double) také vyjadřuje nezáporná čísla v desítkové soustavě, přičemž literál je tvořen celou částí a desetinnou částí, nebo celou částí a exponentem, nebo celou částí 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).

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

- *Řetězcová konstanta* je ohraničena jednoduchým apostrofem (**'**, ascii hodnota 39) z obou stran. Tvoří ji znaky s ascii hodnotou větší než 31. Řetězec může obsahovat escape sekvence: "**'#***i***'**", kde $i \in \langle 1, 31 \rangle$. Jejich význam se shoduje s odpovídajícími znakovými konstantami jazyka Pascal[2]. Délka této konstanty není omezena (snad jen velikostí haldy daného programu). Apostrof v řetězci vytvoříme jako dva bezprostředně následující apostrofy. Například řetězcová konstanta

  'Ahoj'#10'Sv''ete!'

  bude interpretován jako
  Ahoj
  Sve'te!

- Jazyk IFJ09 podporuje *blokové komentáře*. Tyto komentáře začínají symbolem "**{**" a jsou ukončeny symbolem "**}**". (stejně jako v jazyce Pascal)

4 Struktura jazyka

4.1 Základní struktura jazyka

Program se skládá ze sekce obsahující deklarace globálních proměnných a z hlavního těla programu.

- sekce deklarace globálních proměnných (může být i prázdná) je uvozena klíčovým slovem **var** a dále obsahuje sekvenci dílčích deklarací tvaru *id* **:** *typ* **;**, přičemž *typ* může být **integer**, **double**, **string**.

- hlavní tělo programu je stejně jako v jazyce Pascal uvozeno **begin** následovanou sekvencí (může být i prázdná) dílčích příkazů oddělených středníkem (**;**) a ukončená **end.** (včetně tečky na konci). Struktura jednotlivých dílčích příkazů je uvedena v následující sekci.

4.2 Syntaxe a sémantika příkazů, ze kterých se skládá tělo programu

Dílčím příkazem v těle programu se rozumí:

- Příkaz pro načtení řetězce: **readln(***id_typu_string***)**
  Sémantika příkazu je následující: Ze standardního vstupu načti řetězec ukončený koncem řádku (symbol konce řádku již do načítaného řetězce nepatří). Výsledek ulož do proměnné *id_typu_string*. Pozor na ošetření chyb nekorektnosti vstupní hodnoty!

- Příkaz pro načtení číselných hodnot: **readln(***id_1***, ***id_2***, ..., ***id_n***)**
  Sémantika příkazu je následující: Postupně ze standardního vstupu načítej jednotlivé číselné hodnoty (**integer** nebo **double**) oddělené libovolným nenulovým počtem prázdných znaků (mezera, konec řádku, tabulátor) a ukládej je do parametrů id_1, id_2, ..., id_n v daném pořadí. Poslední číslo musí být potvrzeno koncem řádku. Pokud bude zadáno více vstupních hodnot než počet parametrů (tj. n), budou ignorovány. Pozor na ošetření chyb nekorektnosti vstupní hodnoty!

[2] přesněji implementace FreePascal jazyka Pascal: `http://www.freepascal.org/`

- Příkaz pro výpis hodnot: **`write(`***vyraz*$_1$**`,` *vyraz*$_2$**`, ..., ` *vyraz*$_n$**`)`**
  Sémantika příkazu je následující: Postupně vyhodnocuj jednotlivé výrazy (*vyraz* je podrobněji popsán v následující kapitole) a vypisuj jejich hodnoty na standardní výstup ihned za sebe bez žádných oddělovačů v patřičném formátu. Hodnota výrazu typu **`double`** bude vytištěna pomocí `"%g"`[3].

- Příkaz přiřazení: *id* **`:=`** *vyraz*
  Tento příkaz začíná identifikátorem, za kterým následuje dvojznak "**`:=`**". Přesná struktura výrazu je popsána v následující kapitole.

- Podmíněný příkaz: **`if`** *vyraz* **`then`** *prikaz*$_1$ **`else`** *prikaz*$_2$
  Sémantika příkazu je následující: Vyhodnoť daný výraz. Pokud výsledek výrazu je jiného typu než **`integer`**, jedná se o sémantickou chybu. Jinak pokud výsledná hodnota výrazu je rovna číslu `0` (nula) a výraz je považován za nepravdivý. Za pravdivý je výraz *vyraz* považován pro všechny ostatní celočíselné nenulové hodnoty. *prikaz*$_1$ a *prikaz*$_2$ jsou příkazy uvedené v této sekci (rekurzivní definice). Pokud je hodnota výrazu *vyraz* pravdivá, vykonej *prikaz*$_1$, jinak vykonej *prikaz*$_2$.

- Příkaz cyklu: **`while`** *vyraz* **`do`** *prikaz*
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako v případě příkazu podmínky. Opakuj provádění příkazu *prikaz* tak dlouho, dokud je hodnota výrazu *vyraz* pravdivá.

- Složený příkaz: **`begin`** *prikaz*$_1$**`;`** *prikaz*$_2$**`; ...;`** *prikaz*$_n$ **`end`**
  Sémantika složeného příkazu je následující: Proveď příkazy *prikaz*$_1$, *prikaz*$_2$, ..., *prikaz*$_n$ postupně v tomto pořadí. Za posledním příkazem posloupnosti příkazů se středník nepíše. Seznam příkazů může být také prázdný.

- Volání vestavěné funkce: *vystupní_id* **`:=`** *název_funkce* **`(`***seznam_vstupních_parametrů***`)`**
  kde *seznam_vstupních_parametrů* je seznam řetězcových konstant a proměnných typu **`string`**. Parametry předáváme hodnotou. Jednotlivé věstavěné funkce budou popsány v sekci 6.

5 Výrazy

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

5.1 Binární a relační operátory

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

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

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

U operátoru **div** se jedná o celočíselné dělení stejně jako v jazyce Pascal. Operátor **div** neakceptuje operandy typu **double**.

Pro operátory **<**, **>**, **<=**, **>=**, **=**, **<>** platí: Pokud je první operand stejného typu jako druhý operand, a to **integer**, **double** nebo **string**, výsledek je typu **integer**. U řetězců se porovnání provádí lexikograficky. Výsledkem porovnání je celočíselná hodnota **1** v případě kladného výsledku, jinak **0**. Operátory mají stejnou sémantiku jako v jazyce Pascal.

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

5.2 Priorita operátorů

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

pr.	Operátory	Asociativita
1	*** div**	→
2	**+ -**	→
3	**< <= > >=**	→
4	**= <>**	→

6 Vestavěné funkce

Interpret bude poskytovat tyto základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ09):

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

a) Pro vyhledávání použijte Knuth-Moris-Prattův algoritmus

b) Pro vyhledávání použijte Boyer-Mooreův algoritmus

Metoda vyhledávání podřetězce v řetězci bude součástí souboru obsahujícího implementaci funkcí pro práci s řetězci. Obě metody budou probírány v rámci předmětu IAL.

8 Implementace řazení

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

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

Metoda řazení bude součástí souboru obsahujícího implementaci funkcí pro práci s řetězci. Všechny tyto metody budou probírány v rámci předmětu IAL.

9 Implementace tabulky symbolů

Tabulka symbolů bude implementována metodou, která je v zadání pro daný tým označena římskou číslicí:

I) pomocí binárního vyhledávacího stromu

II) pomocí hashovací tabulky

Implementace tabulky symbolů bude uložena ve speciálním souboru s názvem `bvs.c` nebo `hash.c` podle zvolené varianty. Oba druhy vyhledávání budou probírány v rámci předmětu IAL.

10 Příklad programu v jazyce IFJ09

10.1 Výpočet faktoriálu

```
{ Program 1: Vypocet faktorialu }

var
  a : integer;
  vysl : integer;

begin
   write('Zadejte cislo pro vypocet faktorialu: ');
   readln(a);
   if a < 0 then
      write('Faktorial nelze spocitat'#10'')
   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 Práce s řetězci a věstavěnými funkcemi

```
{ Program 2: Prace s retezci a vestavenymi funkcemi }

var
  str1 : string;
  str2 : string;
  p : integer;

begin
  str1 := 'Toto je nejaky text';
  str2 := str1 + ', ktery jeste trochu obohatime';
  p := find(str2, 'text');
  write('Pozice retezce "text" v retezci str2: ',
        p, ''#10'');
  write('Zadejte nejakou posloupnost vsech malych pismen a-h, ',
        'pricemz se pismena nesmeji v posloupnosti opakovat:');
  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 je schválně navržen tak, aby byl podmnožinou jazyka FreePascal[4]. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ09, může si to ověřit následovně. Před program jazyka IFJ09 přidá následující část kódu (obsahuje nastavení fpc kompilátoru a definice chybějících vestavěných funkcí, které jsou součástí jazyka IFJ09):

```
{ Zajisteni zakladni kompatibility IFJ09-FreePascal }
{$H+}                    {Aktivuje neomezene retezce}
type integer = longint;      {Podpora 32-bit Integer}

function sort(str : string) : string;
{Bubble Sort}
var i, j, n : integer;
    finished : boolean;
    swap : char;
begin
  sort := str;
  n := Length(sort);
  i := 2;
  repeat
    finished := true;
    for j := n downto i do
```

[4]Referenční příručka: `ftp://ftp.freepascal.org/pub/fpc/docs-pdf/ref.pdf`

```
      if sort[j-1] > sort[j] then
      begin
        swap := sort[j-1];
        sort[j-1] := sort[j];
        sort[j] := swap;
        finished := false
      end;
    i := i + 1
  until finished or (i = (n + 1))
end;

function find(haystack : string; needle : string) : integer;
begin
  find := Pos(needle, haystack)
end;

{ Zde bude nasledovat program jazyka IFJ09 }
```

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

`fpc` `nazev_programu.pas`

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že FreePascal je nadmnožinou jazyka IFJ09 a tudíž může zpracovat i konstrukce, které jsou v IFJ09 nepovolené (např. volnější syntaxe escape sekvencí, odlišný výpis desetinných čísel, nebo sémantika příkazu **`readln`**).

12 Instrukce ke způsobu vypracování a odevzdání

12.1 Obecné informace

Tyto obecné informace nepodceňujte, neboť projekty bude částečně opravovat automat a nedodržení těchto pokynů povede k tomu, že automat daný projekt nebude schopen zkompilovat!

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!).

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!) Jeden řádek tohoto souboru bude vypadat například takto:

```
xnovak99:25
```

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.

12.3 Závazné metody pro implementaci interpretu

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

13 Požadavky na řešení

13.1 Textová část řešení

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

- 1. strana: jména, příjmení a přihlašovací jména řešitelů + údaje o rozdělení bodů, identifikaci vašeho variantního zadání, výčet zvolených rozšíření
- strukturu konečného automatu, který specifikuje lexikální analyzátor
- LL-gramatiku, která je jádrem vašeho syntaktického analyzátoru
- popis vašeho způsobu řešení interpretu (z pohledu IFJ) - návrh, implementace, vývojový cyklus, způsob práce v týmu, speciální použité techniky, algoritmy
- popis vašeho způsobu řešení řadicího algoritmu, vyhledávání podřetězce v řetězci a tabulky symbolů (z pohledu předmětu IAL)

Dokumentace nesmí:

- obsahovat kopii zadání či text, který není váš původní (kopie z přednášek, sítě, WWW, . . .)

- být založena pouze na výčtu a popisu jednotlivých metod analýzy (jde o váš vlastní přístup k řešení; 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.

13.2 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!

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

Úvod **všech** zdrojových textů musí obsahovat název projektu, přihlašovací jména a jména všech řešitelů.

Veškerá chybová hlášení vzniklá v průběhu činnosti interpretu budou vždy vypisována na standardní chybový výstup. Veškeré texty požadované řídícím programem budou vypisovány na standardní výstup. Kromě chybových hlášení nebude interpret v průběhu interpretace na žádný výstup vypisovat žádné znaky či dokonce celé texty, které nejsou přímo předepsány řídícím 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 textu a tím snížení bodového hodnocení celého projektu.

13.3 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 demonstračních cvičeních. Je nezbytné, aby na řešení projektu spolupracoval celý tým. Návrh interpretu, základních rozhraní a rozdělení práce lze vytvořit již zhruba v polovině semestru. Je dobré, když se celý tým domluví na pravidelných schůzkách a komunikačních kanálech, které bude během řešení projektu využívat (instant messaging, konference, verzovací systém, štábní kulturu atd.).

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

Nenechávejte řešení projektu až na poslední týden. Projekt je tvořen z několika částí (např. lexikální analýza, syntaktická analýza, sémantická analýza, intermediální reprezentace, interpret, tabulka symbolů, knihovna pro práci s potencionálně nekonečně dlouhými řetězci, dokumentace, testování!) a dimenzován tak, aby některé části bylo možno navrhnout a implementovat již v průběhu roku na základě znalostí získaných na přednáškách předmětů IFJ a IAL a na demonstračních cvičeních předmětu IFJ.

13.4 Registrovaná rozšíření

V průběhu řešení bude postupně aktualizován ceník rozšíření projektu. V něm budou uvedena hodnocená rozšíření projektu, za která lze získat prémiové body. Můžete cvičícím během semestru zasílat návrhy na dosud neuvedená rozšíření, které byste chtěli navíc implementovat. Cvičící rozhodnou o přijetí/nepřijetí rozšíření a hodnocení rozšíření dle jeho náročnosti. Body za implementovaná rozšíření se počítají do bodů za programovou implementaci, takže stále platí získatelné maximum 25 bodů.

13.4.1 Některá rozšíření, která budou oceněna prémiovými body:

- program bude obsahovat i tzv. řádkové komentáře jazyka FreePascal, tj. komentáře, které začínají dvojicí znaků **`//`** a jsou ukončeny koncem řádku (+0,5 bodů)
- konstanty je možné zadávat i ve dvojkové (číslo začíná znakem %), osmičkové (číslo začíná znakem &) a v šestnáctkové (číslo začíná znakem $) soustavě (+1 bod)
- interpret bude pracovat i s operátorem unárním mínus – je potřeba do dokumentace uvést, jak je tento problém řešen (+1 bod)
- interpret bude zpracovávat i zjednodušený podmíněný příkaz **`if`**-**`then`** bez části **`else`** – je potřeba do dokumentace uvést, jak je tento problém řešen[5] (+1,5 bodu)
- interpret bude podporovat i cyklus **`repeat`** - **`until`** (+1 bod)
- součástí výrazu mohou být i logické operátory **`and`**, **`or`**, **`not`** (+1 bod)

[5] 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`**. Dále Pascal vyžaduje nepsaní středníku za příkaz před **`else`**.

- projekt bude pracovat i s proměnnými typu pole **`array`** (kompatibilní s jazykem Pascal) (+2 body)
- ...

14 Opravy zadání

- 24. 10. 2009 – V kapitole 11 je opraven přepínač `{$H on}` na `{$H+}`.
- 26. 10. 2009 – Upřesnění složeného příkazu a parametrů příkazu s vestavěnými funkcemi. Opravena chyba v programu 2.