ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Ondřej Navrátil, Jakub Křoustek
email: {krivka, inavra, ikroustek}@fit.vutbr.cz
23. září 2013

1 Obecné informace

Název projektu:	Implementace interpretu imperativního jazyka IFJ13.
Informace:	diskusní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání:	pátek 29. listopadu 2013, 23:59 (nepovinné).
Datum odevzdání:	neděle 15. prosince 2013, 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 IFJ13 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, pokus o redefinici funkce.
- 4 - sémantická/běhová chyba v programu – chybějící parametr při volání funkce.
- 5 - sémantická/běhová chyba v programu – nedeklarovaná proměnná.
- 10 - sémantická/běhová chyba dělení nulou.
- 11 - sémantická/běhová chyba při přetypování na číslo (funkce **`doubleval`**).
- 12 - sémantická/běhová chyba typové kompatibility v aritmetických a relačních výrazech.
- 13 - ostatní sémantické/běhové chyby.
- 99 - interní chyba interpretu tj. neovlivněná vstupním programem (např. chyba alokace paměti atd.).

Jméno souboru s řídicím programem v jazyce IFJ13 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.

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

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 IFJ13 je až na marginální případy podmnožinou jazyka PHP[2], což je rozšířený skriptovací jazyk pro webové aplikace kombinující různá programovací paradigmata.

3.1 Obecné vlastnosti a datové typy

Programovací jazyk IFJ13 je case-sensitive[3] a je jazykem dynamicky typovaným, takže každá proměnná má svůj typ určen hodnotou do ní aktuálně přiřazenou.

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ("_") začínající písmenem nebo podtržítkem. Jazyk IFJ13 obsahuje navíc níže uvedená *klíčová slova* a *konstanty*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory funkcí:

Klíčová slova:	**`else`**, **`function`**, **`if`**, **`return`**, **`while`**
Konstanty:	**`false`**, **`null`**, **`true`**

- *Identifikátor proměnné* se skládá z předpony, což je znak $ (dolar), a identifikátoru[4].
- *Celočíselný literál* (rozsah C-int) je tvořen neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě.
- *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)[5].
- *Řetězcový literál* je ohraničen uvozovkami (**`"`**, ASCII hodnota 34) z obou stran. Tvoří ho znaky s ASCII hodnotou větší než 31 (mimo **`"`** a **`$`**). Řetězec může obsahovat escape sekvence: "**`\n`**", "**`\t`**", "**`\\`**", "**`\"`**", "**`\$`**". Jejich význam se shoduje s odpovídajícími znakovými literály jazyka C, vyjma sekvence "**`\$`**" reprezentující znak $. Narozdíl od jazyka C nelze escape sekvencí vytvořit chybu – pakliže znaky za zpětným lomítkem neodpovídají žádnému z uvedených vzorů, jsou (včetně lomítka) bez jakýchkoli náhrad součástí řetězce. Expanzi (interpolaci) proměnných v řetězcích

[2] Online dokumentace: `http://www.php.net/manual/en/index.php`; na serveru Merlin je pro studenty k dispozici verze 5.3.3 z 2. února. 2012.

[3] Záleží tudíž na velikosti písmen u funkcí, proměnných, konstant i klíčových slov. IFJ13 se v tomto ohledu záměrně liší od PHP, ve kterém jsou názvy funkcí a klíčová slova case-insensitive.

[4] Identifikátor proměnné může po $ obsahovat i klíčové slovo. Např. **`$return`**.

[5] Pro celočíselný literál a celou čast desetinného literálu platí, že přebytečné počáteční číslice `0` jsou ignorovány.

neuvažujte, nicméně znak dolar lze sázet jen pomocí escape sekvence "**\$**". Znak v řetězci může být zadán také pomocí escape sekvence "**\x***dd*", kde *dd* je dvoumístné hexadecimální číslo od **00** do **FF** (písmena **A**-**F** mohou být malá i velká). Délka řetězce není omezena (snad jen velikostí haldy interpretu). Například řetězec

```
"\"Ahoj\n\$vete\x21'"
```

bude interpretován (např. vytisknut) jako

```
"Ahoj
$vete!'
```

- *Logický literál* je tvořen konstantami **true**, resp. **false** a vyjadřuje odpovídající logickou hodnotu.
- *Datové typy* pro jednotlivé uvedené literály jsou označeny *integer*, *double*, *string*, resp. *boolean*. Speciálním případem je typ *null*, který připouští pouze hodnotu **null**. Typy se používají pouze interně a mají význam například při provádění implicitních konverzí a sémantických kontrol (statických i dynamických).
- *Term* je libovolný literál (celočíselný, desetinný, řetězcový či logický), identifikátor proměnné nebo **null**.
- Jazyk IFJ13 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.
- Blokový komentář začíná posloupností symbolů "**/*****" a je ukončen dvojicí symbolů "*****/**". Vnořené blokové komentáře neuvažujte.

4 Struktura jazyka

IFJ13 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 prolínajících se s hlavním tělem programu. Na každém řádku se může nacházet libovolný (i nulový) počet příkazů jazyka IFJ13. Mezi každými dvěma lexémy může být libovolný počet bílých znaků (mezera, tabulátor, odřádkování, komentář atd.).

4.2 Definice funkcí

Definice funkce je zároveň její deklarací. Každá funkce musí být definována právě jednou (tj. i opakovaná definice funkce stejného jména je chyba; nelze tedy například přetěžovat funkce, viz chyba 3). Definice funkce nemusí vždy lexikálně předcházet kódu pro volání 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* **`)`**
  `{`
  sekvence_příkazů
  `}`
  - Seznam parametrů je tvořen posloupností identifikátorů parametrů oddělených čárkou, přičemž za posledním z nich se čárka neuvádí. Seznam může být i prázdný.
  - Tělo funkce je sekvence příkazů (i prázdná) uzavřená do složených závorek. Syntaxe a sémantika těchto příkazů je shodná s příkazy v hlavním těle programu (viz kapitola 4.3.2 níže).
  - Definice vnořených funkcí neuvažujte.

4.3 Hlavní tělo programu

Zdrojový kód vždy začíná úvodní značkou "**`<?php`**" a alespoň jedním bílým znakem. Před touto značkou se nesmí nacházet žádný text či znaky (ani komentáře a bílé znaky). Hlavní tělo programu je neoznačená sekvence příkazů jazyka IFJ13, která se prolíná s definicemi funkcí (na nejvyšší úrovni příkazů, definice funkce tudíž nemůže narušit integritu příkazu, a to ani složeného). 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[6]. Struktura jednotlivých příkazů je popsána v následujících kapitolách.

4.3.1 Proměnné

Proměnné jsou značeny pomocí identifikátorů proměnné (viz kapitola 3.1). Proměnné jazyka IFJ13 jsou pouze lokální (i v případě definice v hlavním těle programu). Jazyk IFJ13 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 **`null`**). Tím pádem proměnnou nelze použít před její definicí, jinak interpret skončí[7] s chybou 5.

4.3.2 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).

[6] Na rozdíl od PHP jazyk IFJ13 nepodporuje uzavírací značku "**`?>`**" a celý zdrojový soubor je tudíž interpretován jako PHP kód. Nelze tedy kombinovat PHP kód s XML či HTML.

[7] PHP v tomto případě vypíše varování a pokračuje dále v interpretaci. V IFJ13 se však jedná o chybu.

- *Podmíněný příkaz*:
  if (*výraz* **)**
  {
  sekvence_příkazů$_1$
  }
  else
  {
  sekvence_příkazů$_2$
  }
  Sémantika příkazu je následující: Nejprve vyhodnotí daný výraz. Pokud je výsledná hodnota výrazu po implicitním přetypování na typ *boolean* (viz funkce **boolval** v kapitole 6) 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* **)**
  {
  sekvence_příkazů
  }
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu v podmíněného příkazu. Opakuje provádění sekvence dílčích příkazů (může být i prázdná) tak dlouho, dokud je hodnota výrazu pravdivá.
- *Volání vestavěné nebo uživatelem definované funkce*:
  $id **=** *název_funkce* **(***seznam_vstupních_parametrů***)** ;
  Seznam_vstupních_parametrů je seznam termů oddělených čárkami[8]. Seznam může být i prázdný. Sémantika vestavěných funkcí bude popsána v kapitole 6. Sémantika volání uživatelem definovaných funkcí je následující: Příkaz zajistí předání parametrů hodnotou a předání řízení do těla funkce. V případě, že příkaz volání funkce obsahuje méně parametrů, než funkce očekává (tedy než je uvedeno v její definici, a to i u vestavěných funkcí), jedná se o chybu 4. Přebývající parametry jsou interpretem ignorovány a nejsou považovány za chybu. Po návratu z těla funkce (viz dále) dojde k uložení výsledku do proměnné *$id* a pokračování běhu programu bezprostředně za příkazem volání funkce.
- *Příkaz návratu z funkce*:
  return *výraz* ;
  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 9;** vždy vracet 0). Hlavní

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

tělo programu ani uživatelem definovaná funkce nemusí příkaz **`return`** vůbec obsahovat. Ukončení uživatelem definované funkce bez provedení **`return`** vrací jako výsledek funkce hodnotu **`null`**.

5 Výrazy

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

5.1 Aritmetické, řetězcové a relační operátory

Standardní binární operátory `+`, `-`, `*` značí sčítání, odčítání[9] a násobení. Jsou-li oba operandy typu *integer*, je i výsledek typu *integer*. Je-li jeden či oba operandy typu *double*, výsledek je typu *double*. Operátor `/` značí dělění, akceptuje operandy typu *double* či *integer* a výsledkem operace je hodnota typu *double*[10].

Řetězcový operátor `.` provádí se dvěma operandy typu *string* jejich konkatenaci. V případě, že je druhý operand jiného typu, je implicitně převeden na typ *string* se sémantikou konverze popsané u vestavěné funkce **`strval`**.

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 negací operátoru `===`.

Pro operátory `<`, `>`, `<=`, `>=` platí: Pokud je první operand stejného typu jako druhý operand, a to *null*, *boolean*, *integer*, *double*, nebo *string*, výsledek je typu *boolean*. Na rozdíl od PHP se u řetězců porovnání vždy provádí lexikograficky[11]. Jiné než uvedené kombinace typů ve výrazech pro dané operátory jsou považovány za chybu 12.

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

[9] Číselné literály jsou sice nezáporné, ale výsledek výrazu přiřazený do proměnné již záporný být může.
[10] Celočíselné dělení tudíž neuvažujte.
[11] Neuvažujte tedy vlastnost PHP související s tzv. numerical strings.

z/na	*bool*	*int*	*double*	*string*
null	**false**	0	0.0	""
bool	beze změny	**true** na 1, **false** na 0	**true** na 1.0, **false** 0.0	**true** na "1", **false** na ""
int	0 na **false**, jinak **true**	beze změny	pouze změna reprezentace	popsáno u **strval**
double	0.0 na **false**, jinak **true**	oříznutí desetinného rozvoje	beze změny	popsáno u **strval**
string	"" na **false**, jinak **true**	popsáno u **intval**	popsáno u **doubleval**	beze změny

Tabulka 1: Konverze datových typů

6 Vestavěné funkce

Interpret bude poskytovat některé základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ13 bez jejich definice). Pro všechny tyto funkce platí, že v případě, že některý z předaných parametrů není naznačeného datového typu, je na něj implicitně přetypován[12]. Vestavěné funkce jazyka IFJ13 jsou:

- **boolval(***term***)** – Vrátí hodnotu termu *term* konvertovanou na pravdivostní hodnotu (viz tabulka 1).
- **doubleval(***term***)** – Vrátí hodnotu termu *term* konvertovanou na desetinné číslo (viz tabulka 1). Při konverzi z řetězce na desetinné číslo jsou nejprve oříznuty všechny uvozující bílé znaky. Dále je načteno číslo dle definice *celočíselného* či *desetinného* literálu v kapitole 3.1 až po první nevyhovující znak (zbytek řetězce včetně tohoto znaku je ignorován) nebo konec řetězce. Není-li tímto způsobem načten ani jeden znak, vrátí funkce hodnotu 0.0. Je-li takto načtená posloupnost znaků nevyhovující definici číselného literálu, dojde k chybě 11[13].
- **intval(***term***)** – Vrátí hodnotu termu *term* konvertovanou na celé číslo (viz tabulka 1). Při konverzi z řetězce na celé číslo jsou nejprve oříznuty všechny uvozující bílé znaky. Pak je načteno číslo dle definice *celočíselného* literálu v kapitole 3.1 až po první nevyhovující znak (zbytek řetězce včetně tohoto znaku je ignorován) nebo konec řetězce. Není-li tímto způsobem načten ani jeden znak, vrátí funkce hodnotu 0.
- **strval(***term***)** – Vrátí hodnotu termu *term* konvertovanou na řetězec (viz tabulka 1). Konverzi z celého, resp. desetinného čísla na řetězec proveďte pomocí příslušných standardních funkcí s formátem `"%d"`, resp. `"%g"`[14].

[12] Přetypování lze teoreticky provést vždy, neb v případě souhlasného typu tato operace nijak nemění hodnotu.

[13] Narozdíl od PHP skončí např. **doubleval("3.1e")** v IFJ13 chybou.

[14] Formátovací řetězec standardní funkce **printf** jazyka C.

- **`get_string()`** – Načte jeden řádek ze standardního vstupu a vrátí jej jako hodnotu typu *string*. Načítání je ukončeno znakem odřádkování nebo konec souboru. Přečtený znak konce řádku, stejně jako znak konce souboru, nejsou součástí načteného řetězce. Pokud již na vstupu nejsou žádná data (tedy první načtený znak je EOF), vrátí funkce prázdný řetězec (`""`).
- **`put_string(`***string*$_1$**`,`** *string*$_2$**`,`** ...**`)`** – Vypíše hodnoty parametrů *string*$_1$, *string*$_2$, ... na standardní výstup ihned za sebe, bez jakýchkoliv oddělovačů. Vrací počet obdržených/vypsaných parametrů jako celé číslo.
- **`strlen(`***string***`)`** – Vrátí délku (počet znaků) řetězce zadaného prvním parametrem. Např. **`strlen(true) === 1`**.
- **`get_substring(`***string***`,`** *integer*$_1$**`,`** *integer*$_2$**`)`** – Vrátí podřetězec zadaného řetězce *string*. Druhým parametrem je dán začátek požadovaného podřetězce (počítáno od nuly) a třetí parametr určuje index za posledním znakem podřetězce.
  Funkce dále vyvolá chybu 13, nastane-li některý z těchto případů:
  - *integer*$_1 < 0$
  - *integer*$_2 < 0$
  - *integer*$_1 >$ *integer*$_2$
  - *integer*$_1 \geq$ **`strlen(`***string***`)`**
  - *integer*$_2 >$ **`strlen(`***string***`)`**
- **`find_string(`***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`. Pokud podřetězec není nalezen, je vrácena hodnota `-1`. Pro vyhledání podřetězce v řetězci použijte metodu, která odpovídá vašemu zadání. Výčet metod korespondující k variantám zadání je uveden v kapitole 7.
- **`sort_string(`***string***`)`** – Seřadí znaky v daném řetězci 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_string`**, 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_string`**, 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 IFJ13.

10.1 Výpočet faktoriálu (iterativně)

```
<?php
// Program 1: Vypocet faktorialu (iterativne)

// Hlavni telo programu
$x = put_string("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n");
$a = get_string();
$a = intval($a);

if ($a < 0)
{
    $x = put_string("Faktorial␣nelze␣spocitat\n");
}
else
{
    $vysl = 1;
    while ($a > 0)
    {
```

```
        $vysl = $vysl * $a;
        $a = $a - 1;
    }
    $x = put_string("Vysledek␣je:␣", $vysl, "\n");
}
```

10.2 Výpočet faktoriálu (rekurzivně)

```
<?php
// Program 2: Vypocet faktorialu (rekurzivne)

// Hlavni telo programu
$z = put_string("Zadejte␣cislo␣pro␣vypocet␣faktorialu\n");
$a = get_string();
$a = intval($a);

// Definice funkce pro vypocet hodnoty faktorialu
function factorial($n)
{
    if ($n < 2)
    {
        $result = 1;
    }
    else
    {
        $decremented_n = $n - 1;
        $temp_result = factorial($decremented_n);
        $result = $n * $temp_result;
    }
    return $result;
}

if ($a < 0) // Pokracovani hlavniho tela programu
{
    $message = "Faktorial␣nelze␣spocitat\n";
}
else
{
    $vysl = factorial($a);
    $message = "Vysledek␣je:␣" . $vysl . "\n";
}
$zzz = put_string($message);
```

10.3 Práce s řetězci a vestavěnými funkcemi

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

function MyMain($str)
{
    $str1 = $str;
    $str2 = $str1 . ",␣ktery␣jeste␣trochu␣obohatime";
    $x = put_string($str1, "\n", $str2, "\n");
    $p = find_string($str2, "text");
    $x = put_string("Pozice␣retezce␣\"text\"␣v␣\$str2:␣", $p, "\n");
```

```
    $x = put_string("Zadejte␣posloupnost␣vsech␣malych␣pismen␣a-h,␣");
    $x = put_string("aby␣se␣pismena␣v␣posloupnosti␣neopakovala:\n");
    $str1 = get_string();
    $str2 = sort_string($str1);
    while ($str2 !== "abcdefgh") {
        $x = put_string("Spatne␣zadana␣posloupnost,␣zkuste␣znovu:\n");
        $str1 = get_string();
        $str2 = sort_string($str1);
    }
    return 0;
}

$foo = MyMain("Toto␣je␣nejaky␣text␣v␣programu␣jazyka␣IFJ13");
```

11 Doporučení k testování

Programovací jazyk je schválně navržen tak, aby byl podmnožinou jazyka PHP. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ13, může si to ověřit následovně. Z IS FIT si stáhněte ze *Souborů* k předmětu IFJ ze složky *Projekt* soubor `ifj13.php` obsahující kód, který doplňuje kompatibilitu s interpretem jazyka PHP na serveru `merlin` (obsahuje např. definice vestavěných funkcí, které jsou součástí jazyka IFJ13, ale chybí v jazyce PHP nebo tam mají mírně odlišnou sémantiku). Váš program v jazyce IFJ13 uložený například v souboru `./tests/testovanySkript.php` pak lze interpretovat na serveru `merlin` pomocí příkazu:

```
php -d open_basedir="" ifj13.php ./tests/testovanySkript.php
```

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že PHP je nadmnožinou jazyka IFJ13 a tudíž může zpracovat i konstrukce, které nejsou v IFJ13 povolené (např. mírně odlišné implicitní konverze a volnost datových typů). 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:

```
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 `/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 než předepsané formáty dokumentace 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[15] 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, ...).

[15] Vyjma obyčejného loga fakulty na úvodní straně.

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 IFJ13:

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka PHP. Podrobnější informace lze získat z referenční příručky[16].

- 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).

[16] `http://www.php.net/manual/en/langref.php`

- EXPAND: Interpret bude podporovat expanzi proměnných v řetězcových literálech. Bližší informace viz kapitola *Strings*, sekce *Variable parsing* v příručce[16] (+1 bod). Příklad:

```
$num = 123;
$str = "num␣is␣$num";
$z = put_string($str); // "num is 123"
```

- ARRAY: Interpret bude podporovat práci s poli. Prázdné pole je definováno přiřazením:

```
$var = array();
```

  Pole bude možné indexovat termem pomocí hranatých závorek **`$var[`***term*`]`**. Tento operátor má nejvyšší prioritu a lze jej použít pouze na proměnné. Tuto konstrukci lze používat ve výrazech, případně jako l-hodnoty u příkazu přiřazení (nikoli u parametrů funkcí, není-li implementováno rozšíření FUNEXP). Pole podporují celočíselné a řetězcové indexy[17]. Při indexaci desetinným číslem či pravdivostní hodnotou se provede implicitní konverze na typ *integer* (viz kapitola 6). Indexace hodnotou **`null`** je shodná s indexací prázdným řetězcem. Indexace jiným polem vyvolá chybu 5. Vícerozměrná pole neuvažujte (nebude testováno).
  U polí se stejně jako u proměnných bude kontrolovat, zda-li byl daný index před použitím deklarován. V opačném případě dojde k chybe 5. Stejnou chybu vyvolá i pokus o indexaci proměnné, která není polem (+1,5 bodu). Příklad:

```
$array = array();
$array[""] = "Ahoj,";
$array[1] = $array[null];
$array[false] = "␣svete!";
$message = $array[1.5] . $array[0];
put_string($message);
```

- FOR: Interpret bude podporovat i cyklus **`for`**:
  `for (` id_1 **`=`** *výraz*$_1$ **`;`** *výraz*$_2$ **`;`** id_3 **`=`** *výraz*$_3$ **`)`**
  `{`
  sekvence_příkazů
  `}`
  kde id_1 je proměnná (re)definovaná příkazem **`for`**. Libovolná ze tří částí hlavičky příkazu **`for`** může být i prázdná. Sémantika příkazu viz manuál[16]. Podporujte i příkazy **`break`** a **`continue`** (+1 bod). Příklad:

```
for ($i = 0; $i < 5; $i = $i + 1)
{
    if ($i === 2)
    {
        continue;
    }
    put_string($i, "\n");
}
```

[17] PHP indexaci pomocí numerical strings opět neuvažujte.

- FUNEXP: Volání funkce může být součástí výrazu, zároveň mohou být výrazy v parametrech funkcí (+1 bod).
- ELSEIF: Podpora příkazu **if-elseif-else** dle manuálu. Části **elseif** a **else** jsou tedy volitelné. Do dokumentace je potřeba uvést, jak je tento problém řešen[18] (+1 bod).
- LOGOP: Podpora úplných logických výrazů včetně kulatých závorek a všech logických operátorů (**and**, **or**, **!**, **&&** a **||**). Sémantika operátorů viz manuál[16] (+0,5 bodu).
- DEFINE: Implementace další vestavěné funkce **define** se syntaxí:

  define(*string***,** *term* **);**

  Tato vestavěná funkce dynamicky definuje globální konstanty platné až do konce provádění programu. Jméno konstanty určuje první parametr a hodnotu druhý. Názvy globálních konstant vytvořené pomocí **define** mohou být stejné s názvy uživatelských i vestavěných funkcí a proměnných (+1 bod). Např.

  ```
  define("PI", 3.1415);
  $x = 2 * PI * 10;
  put_string($x);
  ```

- DEFARG: Interpret umožní definovat seznam parametrů funkce s výchozími hodnotami. Parametr s výchozí hodnotou má tvar **$id =** *literál* (nebo **null**). Vyskytuje-li se v definici funkce parametr s výchozí hodnotou, všechny následující parametry v této definici musejí mít též uvedenu výchozí hodnotu, jinak dojde k chybě 4.
  Při volání funkce jsou chybějící parametry nahrazeny svými výchozími hodnotami. Pokud však při volání funkce není předán dostatek argumentů pro pokrytí parametrů bez výchozí hodnoty, dojde k chybě 4 (+0,5 bodu). Příklad:

  ```
  function make($what = "coffee")
  {
      put_string("Making␣a␣cup␣of␣", $what);
  }
  make("tea"); //Making a cup of tea
  make(null); //Making a cup of
  make(); //Making a cup of coffee
  ```

- ...

14 Opravy zadání

- 26. 9. 2013 – přidán typ *null* jako validní pro operandy relačních operátorů.

[18] Například, jak řešit u zanoření příkazu **if** a **if-else** párování **if** a **else**? U moderních programovacích jazyků platí konvence párování **else** s nejbližším **if**.