ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Radim Kocman
email: {krivka, ikocman}@fit.vutbr.cz
21. září 2015

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ15.
Informace: diskusní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání: pátek 27. listopadu 2015, 23:59 (nepovinné).
Datum odevzdání: neděle 13. prosince 2015, 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 v jazyce C, který načte zdrojový soubor zapsaný v jazyce IFJ15 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 - chyba při odvozování datového typu proměnné.
- 6 - ostatní sémantické chyby.
- 7 - běhová chyba při načítání číselné hodnoty ze vstupu.
- 8 - běhová chyba při práci s neinicializovanou proměnnou.
- 9 - běhová chyba dělení nulou.
- 10 - 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 IFJ15 bude předáno jako první a jediný parametr na příkazové řádce. Bude možné jej zadat s relativní i absolutní cestou. Program

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

bude přijímat vstupy ze standardního vstupu, směrovat všechny své výstupy diktované řídicím programem na standardní výstup, všechna chybová hlášení na standardní chybový výstup; tj. bude se jednat o konzolovou aplikaci, nikoliv o aplikaci s grafickým uživatelským rozhraním.

Klíčová slova jsou sázena tučně a některé lexémy jsou pro zvýšení čitelnosti v apostrofech, přičemž znak apostrofu není v takovém případě součástí jazyka!

3 Popis programovacího jazyka

Jazyk IFJ15 je velmi zjednodušenou podmnožinou jazyka C++11[2], což je staticky typovaný[3] multiparadigmativní jazyk nabízející základní odvozování datových typů.

3.1 Obecné vlastnosti a datové typy

V programovacím jazyce IFJ15 záleží na velikosti písmen u identifikátorů i klíčových slov[4].

- *Identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaku podtržítka ('_') začínající písmenem nebo podtržítkem. Jazyk IFJ15 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory:

  `auto`, **`cin`**, **`cout`**, **`double`**, **`else`**,
  `for`, **`if`**, **`int`**, **`return`**, **`string`**.

- *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. Exponent je celočíselný, začíná znakem '**`e`**' nebo '**`E`**', následuje nepovinné znaménko '**`+`**' (plus) nebo '**`-`**' (mínus) a poslední částí je neprázdná posloupnost číslic. Mezi jednotlivými částmi nesmí být jiný znak, celou a desetinnou část odděluje znak '**`.`**' (tečka)[6].
- *Řetězcový literál* je ohraničen dvojitými uvozovkami (`"`, ASCII hodnota 34) 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 `"`) lze zapisovat přímo. Některé další znaky lze zapisovat pomocí escape sekvence: '**`\"`**',

[2] `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3337.pdf`; na serveru Merlin je pro studenty k dispozici překladač `g++/GCC` verze 4.9.3.

[3] Jednotlivé proměnné mají předem určen datový typ svou definicí nebo lze typ staticky odvodit během překladu.

[4] tzv. case-sensitive jazyk

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

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

'**\n**', '**\t**', '****'. Znak v řetězci může být zadán také pomocí obecné escape sekvence '**\x***dd*', kde *dd* je dvoumístné hexadecimální číslo od **01** do **FF** (písmena **A-F** mohou být malá nebo velká). Délka řetězce není omezena (snad jen velikostí haldy pro interpretaci daného programu). Například řetězcový literál

```
"Ahoj\nSve'te\\\x22"
```

bude interpretován jako

```
Ahoj
Sve'te\"
```

- *Datové typy* pro jednotlivé uvedené literály jsou označeny **int**, **double** a **string**. Typy se používají v definicích proměnných a funkcí a u sémantických kontrol.
- *Term* je libovolný literál (celočíselný, desetinný či řetězcový) nebo identifikátor proměnné.
- Jazyk IFJ15 podporuje *řádkové* i *blokové komentáře* stejně jako jazyk C++. Řádkový komentář začíná dvojicí lomítek ('**//**', ASCII hodnoty 47) a za komentář je považováno vše, co následuje až do konce řádku. Blokový komentář začíná dvojicí symbolů '**/***' a je ukončen první následující dvojicí symbolů '***/**', takže hierarchické vnoření blokových komentářů není podporováno.

4 Struktura jazyka

IFJ15 je strukturovaný programovací jazyk podporující definice proměnných a uživatelských funkcí včetně jejich rekurzivního volání. Vstupním bodem interpretovaného programu je povinná hlavní funkce **main**.

4.1 Základní struktura jazyka

Program se skládá ze sekvence deklarací a definic uživatelských funkcí včetně definice hlavní funkce **main**. Na každém řádku těla definice funkce se může nacházet libovolný (i nulový) počet příkazů jazyka IFJ15. Mezi každými dvěma lexémy může být libovolný počet bílých znaků (mezera, tabulátor, odřádkování a komentář)[7]. Jednoduché příkazy a definice lokálních proměnných ve složeném příkazu jsou ukončovány znakem '**;**' (středník, ASCII hodnota 59).

- deklarace a definice uživatelských funkcí jsou popsány v podsekci 4.3.
- hlavní tělo programu je stejně jako v jazyce C/C++ tvořeno povinnou funkcí **main** s pevně danou hlavičkou (typovou signaturou)

```
int main()
```

následovanou složeným příkazem (viz sekce 4.3). Struktura jednotlivých dílčích příkazů je uvedena v sekci 4.4. Pro jednoduchost nebude v testovaných programech IFJ15 funkce **main** explicitně volána (tedy ani rekurzivně). Chybějící definice funkce **main** vede na chybu 3.

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

4.2 Proměnné

Proměnné jazyka IFJ15 jsou pouze lokální a mají rozsah platnosti v bloku, ve kterém byly definovány, od místa jejich definice až po konec tohoto bloku. Blokem je libovolný složený příkaz (tedy i tělo funkce). Definice proměnné je tvaru:

typ identifikátor **=** *výraz* **;**

Definice lokální proměnné obsahuje určení datového typu *typ* (viz sekce 3.1) nebo modifikátor **`auto`** následovaný jedním identifikátorem proměnné *identifikátor* s nepovinnou inicializací pomocí výrazu *výraz* (viz kapitola 5). Je-li využit modifikátor **`auto`**, je inicializace povinná a podle datového typu hodnoty inicializačního výrazu je odvozen i typ inicializované proměnné. Chybí-li při použití modifikátoru **`auto`** inicializace nebo typ nelze odvodit, jde o chybu 5.

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 8.

Nelze definovat proměnnou stejného jména jako má jiná proměnná na stejné úrovni nebo některá deklarovaná/definovaná funkce (chyba 3). Každá proměnná musí být definována před jejím použitím, jinak se jedná o sémantickou chybu 3. Při definici proměnné stejného jména jako některá proměnná z nadřazené úrovně je viditelná pouze později definovaná proměnná, která je na daném místě platná. Ostatní proměnné stejného jména mohou být platné, i když nejsou v dané části programu viditelné.

4.3 Definice uživatelských funkcí

Definice funkce se skládá z hlavičky a těla funkce. Každá uživatelská funkce s daným identifikátorem je definována nejvýše jednou[8]. Definice funkce nemusí vždy lexikálně předcházet kódu 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, zapíšeme alespoň tzv. deklaraci funkce f před definicí funkce g a definici funkce f uvedeme v kódu později (za funkcí g). Funkci je povoleno deklarovat vícekrát, ovšem pouze se stejnou typovou signaturou[9].

- *Definice uživatelské funkce* je konstrukce ve tvaru:
  typ id **(** *seznam_parametrů* **)**
  složený_příkaz

  - Seznam parametrů je tvořen posloupností definic parametrů oddělených čárkou (**`,`**), přičemž za poslední z nich se čárka neuvádí. Seznam může být i prázdný. Každá definice parametru obsahuje datový typ a identifikátor parametru:

    typ identifikátor_parametru

[8] tzv. přetěžování funkcí není v IFJ15 bez rozšíření podporováno.

[9] Typová signatura zahrnuje typy všech parametrů funkce a typ návratové hodnoty funkce.

  - Tělo funkce je tvořeno složeným příkazem (viz sekce 4.4), kde jsou parametry funkce chápány jako předdefinované lokální proměnné.
  - Každá funkce vrací hodnotu danou vyhodnocením výrazu v příkazu **`return`**. V případě chybějící návratové hodnoty kvůli neprovedení žádného příkazu **`return`** dojde k chybě 8.
  - Definice vnořených funkcí je zakázána.

- *Deklarace uživatelské funkce* je konstrukce ve tvaru:
  typ id **`(`** *seznam_parametrů* **`);`**
  - Pro každou funkci může být v programu v jazyce IFJ15 uvedena nula, jedna či více deklarací. Ke každé deklaraci funkce musí existovat odpovídající definice funkce se stejnou hlavičkou (shodovat se musí nejen typová signatura, ale i identifikátory parametrů), jinak bude hlášena chyba 3.

4.4 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). Možné implicitní konverze jsou popsány v kapitole 5.
- *Složený příkaz*:
  `{` *definice_či_příkaz*$_1$ *definice_či_příkaz*$_2$ ... *definice_či_příkaz*$_n$ **`}`**
  Složený příkaz je posloupnost (může být i prázdná) definic lokálních proměnných a dílčích příkazů umístěná ve složených závorkách.
  Sémantika složeného příkazu je následující: Proveď definice lokálních proměnných s platností v tomto bloku a dílčí příkazy postupně v zadaném pořadí.
- *Podmíněný příkaz*:
  `if (` *výraz* **`)`** *složený_příkaz*$_1$ **`else`** *složený_příkaz*$_2$
  Sémantika příkazu je následující: Nejprve se vyhodnotí daný *výraz*. Pokud je výsledná hodnota výrazu implicitně konvertovatelná na **`int`** a navíc je pravdivá (hodnota je různá od **`0`**), vykoná se *složený_příkaz*$_1$, jinak se vykoná *složený_příkaz*$_2$. Pokud výsledná hodnota výrazu ani po konverzi není typu **`int`**, nastává chyba 4.
- *Příkaz* **`for`** *cyklu*:
  `for (` *definice* **`;`** *výraz* **`;`** *příkaz_přiřazení* **`)`** *složený_příkaz*
  Příkaz **`for`** se skládá z hlavičky uzavřené v kulatých závorkách a z těla tvořeného složeným příkazem *složený_příkaz*.
  Sémantika příkazu **`for`** cyklu je následující: Před provedením první iterace je definována lokální proměnná (tzv. iterační) podle *definice* (včetně případné inicializace výrazem). Před provedením těla cyklu je vždy vyhodnocena podmínka *výraz* a v případě pravdivosti je provedeno tělo cyklu. Na konci prováděného těla cyklu je vykonáno přiřazení *příkaz_přiřazení*, které typicky modifikuje iterační proměnnou nebo jinak ovlivňuje pravdivost podmínky pro vykonání další iterace. Pak následuje opět

vyhodnocení a kontrola pravdivosti podmínky pro případné opětovné provedení těla cyklu. Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu podmíněného příkazu.

Nově definovaná iterační proměnná je platná v hlavičce i následném těle příkazu **for**, nicméně její výskyt například v podmínce *výraz* není povinný. Proměnné definované až v těle cyklu nejsou viditelné v hlavičce cyklu.

- *Volání vestavěné nebo uživatelem definované funkce*:
  id **=** *název_funkce* **(***seznam_vstupních_parametrů***) ;**

  Seznam_vstupních_parametrů je seznam termů (viz sekce 3.1) oddělených čárkami[10]. 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 (včetně případných implicitních konverzí) a předání řízení 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) včetně případné aplikace implicitních konverzí, jedná se o chybu 4. Po dokončení provádění zavolané funkce je přiřazena návratová hodnota do proměnné *id* a běh programu pokračuje bezprostředně za příkazem volání právě provedené funkce. Nedošlo-li k vykonání žádného příkazu **return**, nastává běhová chyba 8.
- *Příkaz návratu z funkce*:
  return *výraz* **;**

  Příkaz může být použit v těle libovolné funkce (včetně **main**). Jeho sémantika je následující: Dojde k vyhodnocení výrazu *výraz* (tj. získání návratové hodnoty), okamžitému ukončení provádění těla funkce a návratu do místa volání, kam funkce vrátí vypočtenou návratovou hodnotu. Výjimkou je implicitně volaná funkce **main**, kdy 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).
- *Příkaz pro načtení hodnoty ze vstupu*:
  cin >> id_1 **>>** id_1 **>>** $\ldots$ **>>** id_n **;**

  Sémantika příkazu je následující: Ze standardního vstupu načti pro každou proměnnou postupně jednu hodnotu typu odpovídajícího proměnné id_i, kde $1 \leq i \leq n$, $n \geq 1$, a hodnotu ulož do proměnné id_i. Formát testovacích vstupů pro číselné typy bude odpovídat lexikálním pravidlům, jinak nastane chyba 7. Pro typ **string** načti řetězec začínající prvním nebílým znakem a poté ukončený prvním bílým znakem (tj. načítáme jedno slovo). Případně může být vstup ukončen koncem vstupu. Následný bílý znak nebo symbol konce vstupu již do načteného řetězce nepatří (tj. lze načíst i prázdný řetězec). Bílé znaky (mezera, tabulátor, konec řádku) před a za hodnotou ignoruj. Na jednom řádku může být zadáno i více vstupních hodnot.
- *Příkaz pro výpis hodnot*:
  cout << $term_1$ **<<** $term_2$ **<<** $\ldots$ **<<** $term_n$ **;**

  Sémantika příkazu je následující: Postupně vypisuj hodnoty jednotlivých termů na

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

standardní výstup ihned za sebe bez žádných oddělovačů v patřičném formátu. Příkaz musí obsahovat alespoň jeden term k výpisu. Hodnota výrazu typu **`double`** bude vytištěna pomocí '**`%g`**'[11].

Všimněte si, že středníkem jsou ukončeny jen některé příkazy (tzv. jednoduché), abychom zachovali zvyklost z C++.

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[12] a násobení. Jsou-li oba operandy typu **`int`**, je i výsledek typu **`int`**. Je-li jeden[13] či oba operandy typu **`double`**, výsledek je též typu **`double`**. Operátor **`/`** značí dělení, akceptuje operandy typu **`double`** či **`int`**. Jsou-li oba operandy typu **`int`**, jedná se o celočíselné dělení s výsledkem typu **`int`**. Jinak se jedná o běžné dělení a výsledkem operace je hodnota typu **`double`**.

Pro operátory **`<`**, **`>`**, **`<=`**, **`>=`**, **`==`**, **`!=`** platí: Pokud je první operand stejného typu jako druhý operand, a to **`int`**, **`double`** nebo **`string`**, výsledek je typu **`int`**. Je-li jeden operand **`int`** a druhý **`double`**, je operand typu **`int`** konvertován na **`double`**. Výsledkem porovnání je hodnota typu **`int`** dle pravdivosti (sémantika operátorů je stejná jako v jazyce C++; pravdivá hodnota je **`1`** a nepravdivá **`0`**). Porovnání řetězců se provádí lexikograficky.

Je-li to nutné, bude interpret provádět implicitní konverze operandů i výsledků výrazů. Možné implicitní konverze datových typů jsou: (a) **`int`** na **`double`**, (b) **`double`** na **`int`** (oříznutím desetinné části).

Jiné než uvedené kombinace typů (včetně případných povolených implicitních konverzí) 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
3	**`*`** **`/`**	levá
6	**`+`** **`-`**	levá
9	**`<`** **`>`** **`<=`** **`>=`** **`==`** **`!=`**	levá

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

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

[13] pak proběhne implicitní konverze druhého operandu též na **`double`**

6 Vestavěné funkce

Interpret bude poskytovat některé základní vestavěné funkce (tj. funkce, které jsou přímo implementovány v interpretu a využitelné v programovacím jazyce IFJ15; znovudefinice těchto funkcí je chybou). Vestavěné funkce jazyka IFJ15 jsou:

- **`int length(string s)`** – Vrátí délku (počet znaků) řetězce zadaného jediným parametrem *s*. Např. **`length("x\nz") == 3`**.
- **`string substr(string s, int i, int n)`** – Vrátí podřetězec zadaného řetězce *s*. Druhým parametrem *i* je dán začátek požadovaného podřetězce (počítáno od nuly) a třetí parametr *n* určuje délku podřetězce. V okrajových případech simulujte stejnojmennou metodu třídy **`basic_string`** z jazyka C++11.
- **`string concat(string s1, string s2)`** – Vrátí řetězec, který je konkatenací řetězce **`s1`** a řetězce **`s2`**.
- **`int find(string s, string search)`** – Vyhledá první výskyt zadaného podřetězce *search* v řetězci *s* a vrátí jeho pozici (počítáno od nuly). 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.
- **`string sort(string s)`** – 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-Morris-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`). Více viz sekce 13.2.

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 List-Merge sort.

Metoda řazení bude součástí souboru `ial.c` (případně `ial.h`). Více viz sekce 13.2.

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 tabulky s rozptýlenými položkami.

Implementace tabulky symbolů bude uložena taktéž v souboru `ial.c` (případně `ial.h`). Více viz sekce 13.2.

10 Příklady

Tato kapitola popisuje tři jednoduché příklady řídicích programů v jazyce IFJ15.

10.1 Výpočet faktoriálu (iterativně)

```
/* Program 1: Vypocet faktorialu (iterativne) */
int main()
{
    int a;
    int vysl;
    cout << "Zadejte␣cislo␣pro␣vypocet␣faktorialu:␣";
    cin >> a;
    if (a < 0) // cin nacitani zaporneho cisla nemusi podporovat
    {
        cout << "Faktorial␣nelze␣spocitat!\n";
    }
    else
    {
        vysl = 1;
        for (int foo; a > 0; a = a - 1)
        {
            vysl = vysl * a;
        }
        cout << "Vysledek␣je:␣" << vysl << "\n";
    }
    return 0;
}
```

10.2 Výpočet faktoriálu (rekurzivně)

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

int factorial(int n) // Definice funkce pro vypocet hodnoty faktorialu
{
    int temp_result;
    auto decremented_n = n - 1;

    if (n < 2) {
        return 1;
```

```
    } else {
        temp_result = factorial(decremented_n);
        return n * temp_result;
    }
}

int main()
{
    int a; int vysl;

    cout << "Zadejte␣cislo␣pro␣vypocet␣faktorialu:␣";
    cin >> a;

    if (a < 0) {
        cout << "Faktorial␣nelze␣spocitat!\n";
    }   else {
        vysl = factorial(a);
        auto neg = 0 - vysl;
        cout << "Vysledek:␣" << vysl << "␣(zaporny:␣" << neg << ")\n";
    }

    return 0;
}
```

10.3 Práce s řetězci a vestavěnými funkcemi

```
/* Program 3: Prace s retezci a vestavenymi funkcemi */
int main()
{
    string str1;
    { // vnoreny blok s lokalni promennou str2 a pristupem k str1
        int x;
        str1 = "Toto␣je␣nejaky␣text";
        string str2;
        str2 = concat(str1, ",␣ktery␣jeste␣trochu␣obohatime");
        x = find(str2, "text");
        cout << "Pozice␣retezce␣\"text\"␣v␣retezci␣str2:␣"
             << x << "\n";
        cout << "Zadejte␣nejakou␣posloupnost␣vsech␣malych␣pismen␣a-h,␣"
                 << "pricemz␣se␣pismena␣nesmeji␣v␣posloupnosti␣opakovat:";
    }
    cin >> str1;
    str1 = sort(str1);
    if (str1 != "abcdefgh")
    {
        for (auto s = str1; s != "abcdefgh"; s = s)
        {
            cout << "Spatne␣zadana␣posloupnost,␣zkuste␣znovu:";
            cin >> str1;
            s = sort(str1);
        }
    }
    else {}
    return 0;
}
```

11 Doporučení k testování

Programovací jazyk IFJ15 je schválně navržen tak, aby byl téměř kompatibilní s podmnožinou jazyka C++11. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ15, může si to ověřit následovně. Z IS FIT si stáhněte ze *Souborů* k předmětu IFJ ze složky *Projekt* soubor `ifj15.cc` obsahující kód, který doplňuje kompatibilitu IFJ15 s překladačem jazyka C++11 na serveru `merlin` (obsahuje např. definice vestavěných funkcí, které jsou součástí jazyka IFJ15, ale chybí v jazyce C++ nebo tam mají mírně odlišnou syntaxi). Váš program v jazyce IFJ15 uložený například v souboru `testovanyProgram.ifj` pak lze interpretovat na serveru `merlin` například pomocí dvojice příkazů:

```
cat ifj15.cc testovanyProgram.ifj > testovanyProgram.cc
g++ -std=c++11 -o testovanyProgram testovanyProgram.cc
```

Tím lze tedy velice jednoduše zkontrolovat, co by daný interpret měl provést. Je ale potřeba si uvědomit, že jazyk C++11 je nadmnožinou jazyka IFJ15 a tudíž může zpracovat i konstrukce, které nejsou v IFJ15 povolené (např. odlišné implicitní konverze, volnější syntaxe nebo složitější zpracování standardního vstupu). 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):

```
xnovak01:30LF
xnovak02:40LF
xnovak03:30LF
xnovak04:00LF
```

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 Implementace metod v souboru `ial.c`

Metody pro vyhledávání podřetězce v řetězci, pro řazení a pro tabulku symbolů implementujte podle algoritmů probíraných v předmětu IAL. Pokud se rozhodnete některou z výše uvedených metod implementovat odlišným způsobem, vysvětlete v dokumentaci důvody, které vás k tomu vedly, a uveďte zdroj či zdroje, ze kterých jste čerpali.

13.3 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.

13.4 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, ...).

[14] 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.

13.5 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.

13.6 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.

13.7 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.

13.7.1 Bodové hodnocení některých rozšíření jazyka IFJ15:

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka C++11. Podrobnější informace lze získat z referenční příručky[2].

- UNARY: Interpret bude pracovat i s unárními operátory $-$ (unární mínus), $--$ (prefixová i postfixová dekrementace), $++$ (prefixová i postfixová inkrementace). Priorita a asociativita nových operátorů odpovídá jazyku C++11. Do dokumentace je potřeba uvést, jak je tento problém řešen (+1,0 bodu).

- BASE: celočíselné konstanty a escape sekvence je možné zadávat i ve dvojkové (číslo začíná znakem \b), osmičkové (číslo začíná znakem \0) a v šestnáctkové (číslo začíná znakem \x) soustavě (+0,5 bodu). Escape sekvence jsou pak omezeny na právě 8, 3, respektive 2 platné číslice a musí být v dekadickém rozsahu 1-255. Celočíselné konstanty budou obsahovat alespoň jednu platnou číslici. Všimněte se, že toto rozšíření přináší nekompatibilitu s C++11.
- WHILE: Interpret bude podporovat i cykly typu **`while`** a **`do`** - **`while`**. (+0,5 bodu)
- FUNEXP: Volání funkce může být součástí výrazu, zároveň mohou být výrazy v parametrech volání funkce (+1,0 bodu).
- SIMPLE: Interpret bude zpracovávat i zjednodušený syntaktický zápis, kdy u podmíněné příkazu lze použít **`if`** bez části **`else`** a u podmíněného příkazu a cyklu lze místo složeného příkazu použít i právě jeden libovolný příkaz (+1,0 bodu).
- BOOLOP: Podpora typu **`bool`**, booleovských hodnot **`true`** a **`false`**, booleovských výrazů včetně kulatých závorek a základních booleovských operátorů (**`!`**, **`&&`**, **`||`**), jejichž priorita a asociativita odpovídá C++11. Dále podpora implicitního přetypování z/na další datové typy (dle C++11) (+1,0 bodu).
- COMMA: (1) Podpora definice více proměnných stejného typu najednou (pro daný datový typ jsou proměnné a případně inicializace odděleny čárkou) včetně možnosti použití i v hlavičce cyklu **`for`**. (2) Podpora operátoru čárka ve výrazech (vyhodnotí se všechny podvýrazy oddělené čárkami, ale výsledkem je výsledek posledního výrazu). (3) Ve třetí části hlavičky cyklu **`for`** je možno mít čárkami odděleno více příkazů přiřazení. (4) Všechny části hlavičky cyklu **`for`** mohou být prázdné (se zachováním oddělujícího středníku). (5) Podporujte dva nové příkazy **`break`** a **`continue`** (ukončené středníkem). Všechny části tohoto kumulativního rozšíření jsou syntakticky i sémanticky inspirovány C++11 (+1,5 bodu).
- ...

14 Opravy zadání

- 5. 10. 2015 – upřesnění požadavků na rozšíření BOOLOP a navýšení jeho ceny z 0,5 na 1,0 bodu.
- 6. 10. 2015 – doplnění požadavků na rozšíření BASE
- 8. 10. 2015 – přidáno rozšíření COMMA