ZADÁNÍ PROJEKTU Z PŘEDMĚTŮ IFJ A IAL

Zbyněk Křivka, Radim Kocman
email: {krivka, ikocman}@fit.vutbr.cz
19. září 2016

1 Obecné informace

Název projektu: Implementace interpretu imperativního jazyka IFJ16.
Informace: diskusní fórum a wiki stránky předmětu IFJ v IS FIT.
Pokusné odevzdání: neděle 27. listopadu 2016, 23:59 (nepovinné).
Datum odevzdání: neděle 11. prosince 2016, 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 FIT 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 IFJ16 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 programu).
- 3 - sémantická chyba v programu – nedefinovaná třída/funkce/proměnná, pokus o redefinici třídy/funkce/proměnné, atd.
- 4 - sémantická chyba typové kompatibility v aritmetických, řetězcových a relačních výrazech, příp. špatný počet či typ parametrů u volání funkce.
- 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 IFJ16 bude předáno jako první a jediný parametr na příkazové řádce. Bude možné jej zadat s relativní i absolutní cestou. Program bude přijímat vstupy ze standardního vstupu, směrovat všechny své výstupy diktované řídicím programem na standardní výstup, všechna chybová hlášení na standardní chybový

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

výstup; tj. bude se jednat o konzolovou aplikaci, nikoliv o aplikaci s grafickým uživatelským rozhraním.

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

3 Popis programovacího jazyka

Jazyk IFJ16 je velmi zjednodušenou podmnožinou jazyka Java SE 8[2], což je staticky typovaný[3] objektově orientovaný jazyk.

3.1 Obecné vlastnosti a datové typy

V programovacím jazyce IFJ16 záleží na velikosti písmen u identifikátorů i klíčových slov[4].

- *Jednoduchý identifikátor* je definován jako neprázdná posloupnost číslic, písmen (malých i velkých) a znaků podtržítka ('_') a dolaru ('`$`') začínající písmenem, podtržítkem, nebo dolarem.
- *Plně kvalifikovaný identifikátor* je tvořen dvěma jednoduchými identifikátory oddělenými tečkou ('`.`') (bez jakýchkoli prázdných znaků), kde první označuje třídu a druhý proměnnou nebo funkci.
- Jazyk IFJ16 obsahuje navíc níže uvedená *klíčová slova*, která mají specifický význam, a proto se nesmějí vyskytovat jako identifikátory[5]:

  `boolean`, **`break`**, **`class`**, **`continue`**, **`do`**, **`double`**, **`else`**, **`false`**, **`for`**, **`if`**, **`int`**, **`return`**, **`String`**, **`static`**, **`true`**, **`void`**, **`while`**.

- *Celočíselný literál* (rozsah C-int) je tvořen neprázdnou posloupností číslic a vyjadřuje hodnotu celého nezáporného čísla v desítkové soustavě[6].
- *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)[7].

[2] `https://docs.oracle.com/javase/specs/`; na serveru Merlin je pro studenty k dispozici překladač `javac` verze 1.8.0_102

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

[4] tzv. case-sensitive jazyk

[5] Některá klíčová slova jsou využita až v rozšířeních, ale přesto se nesmí ani v základním zadání vyskytovat jako identifikátory a působí tedy jako tzv. rezervovaná slova.

[6] Přebytečné počáteční číslice `0` v nenulovém čísle vedou v rozšíření BASE na oktalovou reprezentaci celého čísla.

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

- *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: '**\"**', '**\n**', '**\t**', '****'. Znak v řetězci může být zadán také pomocí obecné oktalové escape sekvence '*****ddd*', kde *ddd* je třímístné oktalové číslo od **001** do **377**. Délka řetězce není omezena (resp. velikostí haldy pro interpretaci daného programu). Například řetězcový literál

  "Ahoj\nSve'te\\\042"

  bude interpretován jako
  Ahoj
  Sve'te\". Neuvažujte řetězce obsahující vícebajtové znaky kódování Unicode (např. UTF-8).
- *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 jednoduchý či plně kvalifikovaný identifikátor proměnné.
- Jazyk IFJ16 podporuje *řádkové* i *blokové komentáře* stejně jako jazyk Java. Řádkový komentář začíná dvojicí lomítek ('**//**', ASCII hodnoty 47) a za komentář je považováno vše, co následuje až do konce řádku. Blokový komentář začíná dvojicí znaků '**/***' a je ukončen první následující dvojicí znaků '***/**', takže hierarchické vnoření blokových komentářů není podporováno.

4 Struktura jazyka

IFJ16 je strukturovaný programovací jazyk podporující definice tříd, proměnných a statických uživatelských funkcí[8], základní řídicí příkazy a příkaz přiřazení a volání funkce včetně rekurzivního. Vstupním bodem interpretovaného programu je povinná třída *Main* a její hlavní bezparametrická statická funkce *run* bez návratové hodnoty.

4.1 Základní struktura jazyka

Program se skládá ze sekvence definic tříd, které obsahují definice statických proměnných a statických uživatelských funkcí. V těle definice funkce se pak může nacházet libovolný (i nulový) počet definic lokální proměnných a příkazů jazyka IFJ16. Příkazy a definice proměnných jsou ukončovány znakem '**;**' (středník, ASCII hodnota 59). Mezi každými dvěma lexémy může být libovolný počet bílých znaků (mezera, tabulátor, odřádkování a komentář)[9].

- definice uživatelských tříd je popsána v sekci 4.2, uživatelských statických funkcí v sekci 4.4 a statických a lokálních proměnných v sekci 4.3.

[8] V Javě jsou funkce zapouzdřeny do tříd a označovány jako tzv. statické metody.
[9] Na začátku a konci zdrojového textu se též smí vyskytovat libovolný počet bílých znaků.

- hlavní tělo programu je tvořeno statickou funkcí *run* s pevně danou hlavičkou (typovou signaturou[10]) a definovanou ve třídě *Main*.

  class *Main* { **static void** *run*() {...} }

  Chybějící definice statické funkce **run** ve třídě **Main** vede na chybu 3.

4.2 Třídy

Třída vytváří oddělený jmenných prostor pro jednoduché identifikátory v ní definované. Definice třídy se skládá z klíčového slova **class**, jednoduchého identifikátoru třídy a složených závorek, ve kterých se nachází všechny zahrnuté definice statických proměnných a statických funkcí. Všechny statické proměnné i funkce jsou vždy zapouzdřeny nějakou třídou. Statické proměnné a funkce lze odkazovat uvnitř zapouzdřující třídy pomocí jednoduchých i plně kvalifikovaných identifikátorů. Mimo tuto třídu je třeba pro odkazování použít plně kvalifikovaných identifikátorů. Ze třídy v IFJ16 nebudou vytvářeny instance (IFJ16 nezahrnuje objektové paradigma).

4.3 Proměnné

Proměnné jazyka IFJ16 jsou buď statické (globální), nebo lokální. Statické mají rozsah platnosti v celém programu. Lokální proměnné a parametry funkcí mají rozsah v dané funkci, kde byly definovány (od místa jejich definice po konec dané funkce). Definice statické proměnné je tvaru:

static *typ identifikátor* **=** *výraz* **;**

Definice proměnné obsahuje určení datového typu *typ* (viz sekce 3.1) následovaného jedním jednoduchým identifikátorem proměnné *identifikátor* s nepovinnou inicializací pomocí výrazu *výraz* (viz kapitola 5). Definice lokální proměnné neobsahuje klíčové slovo **static**.

Dokud nemá proměnná přiřazenu hodnotu, je neinicializovaná. Pokusíme-li se číst hodnotu neinicializované proměnné (předáním jako parametr funkci, vrácením jako výsledek funkce, využitím jako operand ve výrazu), nastane chyba 8.

Nelze definovat proměnnou stejného jména, jako má jiná proměnná na stejné úrovni nebo některá funkce definovaná v zapouzdřující třídě (chyba 3). Každá v programu použitá proměnná musí být definována, jinak se jedná o sémantickou chybu 3. Při definici lokální proměnné stejného jména, jako má již některá statická proměnná téže třídy, je pod daným jednoduchým identifikátorem viditelná ona lokální proměnná. K oné statické proměnné lze přistoupit pouze přes její plně kvalifikované jméno.

Statická proměnná je v domovské třídě dostupná jak přes plnou kvalifikaci, tak i přes její jednoduchý identifikátor, pokud tedy není překryta stejnojmennou lokální proměnnou či parametrem funkce. Mimo domovskou třídu je nutné statickou proměnnou odkazovat vždy s využitím plně kvalifikovaného identifikátoru.

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

4.4 Definice uživatelských funkcí

Definice funkce (vždy tzv. statická) se skládá z hlavičky a těla funkce. Každá uživatelská funkce s daným identifikátorem je v téže třídě definována nejvýše jednou[11]. 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). Situace, kdy volání funkce lexikálně předchází její definici, je typicky řešena dvouprůchodovou analýzou, nebo zpětným doplněním vazeb na použitý symbol až při analýze jeho definice (analogicky i pro případ použití statické proměnné v nějak statické funkci lexikálně předcházející definici této statické proměnné).

Definice uživatelské funkce je konstrukce ve tvaru:
`static` *návrat_typ id* **`(`** *seznam_parametrů* **`)`**
`{` *tělo_funkce* **`}`**

- V případě, že funkce nebude vracet žádnou hodnotu, je návratový datový typ *návrat_typ* uvedený za klíčovým slovem **`static`** nahrazen klíčovým slovem **`void`** a tuto funkci označujeme jako tzv. **`void`**-funkci.
- 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 jednoduchý identifikátor parametru:

  typ identifikátor_parametru

- Tělo funkce je tvořeno libovolným počtem definic lokálních proměnných a dílčích příkazů (viz sekce 4.5). V těle funkce jsou její parametry chápány jako předdefinované lokální proměnné.
- Každá funkce vrací hodnotu danou vyhodnocením výrazu v příkazu **`return`**. V případě chybějící návratové hodnoty kvůli neprovedení příkazu **`return`** dojde k chybě 8.

Platnost jména statické funkce a pravidla pro využívání plně kvalifikovaného identifikátoru funkce jsou analogické s pravidly pro statické proměnné.

4.5 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.

[11] Tzv. přetěžování funkcí není v IFJ16 bez rozšíření podporováno.

- *Složený příkaz*:
  { *příkaz*$_1$ *příkaz*$_2$ ... *příkaz*$_n$ **}**
  Složený příkaz je posloupnost (může být i prázdná) dílčích příkazů umístěná ve složených závorkách. Ve složeném příkazu není povoleno definovat lokální proměnné. Sémantika složeného příkazu je následující: Proveď 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 pravdivostní *výraz* (typicky využívající některý relační operátor). Pokud je vyhodnocený výraz pravdivý, vykoná se *složený_příkaz*$_1$, jinak se vykoná *složený_příkaz*$_2$. Pokud výsledná hodnota výrazu není pravdivostní (tj. pravda či nepravda), nastává chyba 4.
- *Příkaz cyklu*:
  while (*výraz* **)** *složený_příkaz*$_3$
  Sémantika příkazu cyklu je následující: Pravidla pro určení pravdivosti výrazu jsou stejná jako u výrazu v podmíněného příkazu. Opakuje provádění příkazů složeného příkazu *složený_příkaz*$_3$ tak dlouho, dokud je hodnota výrazu pravdivá.
- *Volání vestavěné nebo uživatelem definované funkce*:
  id **=** *název_funkce* **(***seznam_vstupních_parametrů***) ;**
  název_funkce **(***seznam_vstupních_parametrů***) ;**
  Identifikátor *název_funkce* může být jednoduchý nebo plně kvalifikovaný. *Seznam_vstupních_parametrů* je seznam termů (viz kapitola 3.1) oddělených čárkami[12]. 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 může být návratová hodnota přiřazena do proměnné *id* a běh programu pokračuje bezprostředně za příkazem volání právě provedené funkce. Pokus o přiřazení návratové hodnoty z **void**-funkce vede na chybu 8. Nedošlo-li k vykonání žádného příkazu **return** a nejedná se o **void**-funkci, nastává běhová chyba 8.
- *Příkaz návratu z funkce*:
  return *výraz* **;**
  Příkaz může být použit v těle libovolné funkce (včetně **Main.run**). 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. U **void**-funkce musí být *výraz* vynechán a provedení příkazu **return** během vykonání celé funkce není nutné.

Všimněte si, že středníkem jsou ukončeny jen některé příkazy (tzv. jednoduché),

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

abychom zachovali zvyklost z Javy.

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[13] a násobení. Jsou-li oba operandy typu `int`, je i výsledek typu `int`. Je-li jeden[14] či oba operandy typu `double`, výsledek je též typu `double`. Operátor `/` značí dělení, akceptuje operandy typu `int` či `double`. 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`. Je-li jeden z operandů typu `String`, má operátor `+` význam konkatenace řetězců (druhý operand je případně převeden též na řetězec dle popisu u vestavěné funkce `ifj16.print` níže).

Pro operátory `<`, `>`, `<=`, `>=`, `==`, `!=` platí, že výsledek porovnání je pravdivostní hodnota. Je-li jeden operand `int` a druhý `double`, je operand typu `int` konvertován na `double`. Bez rozšíření BOOLOP není s výsledkem porovnání možné dále pracovat a lze jej využít pouze u příkazů `if` a `while`. Relační operátory nepodporují porovnání řetězců (viz vestavěná funkce `ifj16.compare`).

Je-li to nutné, bude interpret provádět implicitní konverze operandů i výsledků výrazů z `int` na `double`.

Jiné než uvedené kombinace typů (včetně případných povolených implicitních konverzí) ve výrazech pro popsané 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
4	`+` `-`	levá
6	`<` `>` `<=` `>=`	žádná
7	`==` `!=`	žádná

6 Vestavěné funkce

Interpret bude poskytovat některé základní vestavěné funkce sdružené ve vestavěné třídě *ifj16*, které bude možné využít v programech jazyka IFJ16. Pokus o vytvoření uživatelské třídy *ifj16* je chyba 3. Ve třídě *ifj16* budou k dispozici tyto statické vestavěné funkce:

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

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

Vestavěné funkce pro načítání literálů a výpis termů:

- **int readInt ( ) ;**
  double readDouble ( ) ;
  String readString ( ) ;
  Ze standardního vstupu se načte řetězec ukončený koncem řádku nebo koncem vstupu, kdy symbol konce řádku či konce vstupu již do načteného řetězce nepatří (tj. lze načíst i prázdný řetězec). **readString** načtený řetězec vrátí jako výsledek. Funkce **readInt** a **readDouble** načtený řetězec převedou na právě jedno celé nebo desetinné číslo a vratí jej jako výsledek. Při převodu na číslo nesmí být vypuštěn žádný bílý znak (mezera, tabulátor), takže formát vstupního řetězce musí přesně odpovídat lexikálním pravidlům pro daný literál (viz sekce 3.1), jinak nastane chyba 7.
- **void print (** *term_nebo_konkatenace* **) ;**
  Je-li *term_nebo_konkatenace* term, tak se jeho hodnota vypíše na standardní výstup bez jakýchkoli oddělovačů v patřičném formátu. Term typu **int** nebo **double** bude automaticky převeden na řetězec a pak teprve vytištěn. Hodnota typu **double** bude vytištěna pomocí '**%g**'[15].
  Druhou možností je, že bude parametrem jednoduchý výraz obsahující neprázdnou posloupnost termů oddělených operátorem **+**. Tento zjednodušený výraz bude vyhodnocen a potom vypsán dle pravidel pro výpis termu. V základním zadání bude výsledek zjednodušeného výrazu vždy typu **String** a není třeba jej zpracovávat metodou závaznou pro práci s výrazy.

Vestavěné funkce pro práci s řetězci:

- **int length(*String* *s*)** – Vrátí délku (počet znaků) řetězce zadaného jediným parametrem *s*. Např. **ifj16.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 metodu **substring** třídy **String** z jazyka Java a vracejte chybu 10.
- **int compare(*String* *s1*, *String* *s2*)** – Lexikograficky porovná dva zadané řetězce *s*1 a *s*2 a vrátí celočíselnou hodnotu dle toho, zda je *s*1 před, roven, nebo za *s*2. Jsou-li řetězce *s*1 a *s*2 stejné, vrací **0**; je-li *s*1 větší než *s*2, vrací **1**; jinak vrací **-1**. Z hlediska funkčnosti simulujte metodu **compareTo** třídy **String** z jazyka Java.
- **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.

[15] formátovací řetězec standardní funkce **printf** jazyka C

7 Implementace vyhledávání podřetězce v řetězci

Metoda vyhledávání, kterou bude využívat vestavěná funkce **`ifj16.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 **`ifj16.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 uvádí tři jednoduché příklady řídicích programů v jazyce IFJ16.

10.1 Výpočet faktoriálu (iterativně)

```
/* Program 1: Vypocet faktorialu (iterativne) */
class Main
{
    static void run()
    {
        int a;
        ifj16.print("Zadejte␣cislo␣pro␣vypocet␣faktorialu:␣");
        a = ifj16.readInt();
        int vysl;
        if (a < 0) { // nacitani zaporneho cisla nemusite podporovat
            ifj16.print("Faktorial␣nelze␣spocitat!\n");
        }
        else {
            vysl = 1;
            while (a > 0) {
                vysl = vysl * a;
                a = a - 1;
            }
            ifj16.print("Vysledek␣je:␣" + vysl + "\n");
        }
    }
}
```

10.2 Výpočet faktoriálu (rekurzivně)

```
/* Program 2: Vypocet faktorialu (rekurzivne) */
class Main
{
    static void run()
    {
        int a; int vysl; int neg;
        ifj16.print("Zadejte␣cislo␣pro␣vypocet␣faktorialu:␣");
        a = ifj16.readInt();
        if (a < 0) {
            ifj16.print("Faktorial␣nelze␣spocitat!\n");
        }
        else {
            vysl = factorial(a);
            neg = 0 - vysl;
            ifj16.print("Vysledek:␣" + vysl);
            ifj16.print("␣(zaporny:␣" + neg+ ")\n");
        }
    }

    static int factorial(int n) // Definice funkce pro vypocet faktorialu
    {
        int temp_result;
        int decremented_n = n - 1;
        if (n < 2) {
            return 1;
        }
        else {
```

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

10.3 Práce s řetězci a vestavěnými funkcemi

```
/* Program 3: Prace s retezci a vestavenymi funkcemi */
class Main
{
    static int x;
    static void run()
    {
        String str1;
        str1 = "Toto je nejaky text";
        String str2;
        str2 = str1 + ", ktery jeste trochu obohatime";
        Main.x = ifj16.find(str2, "text");
        ifj16.print("Pozice retezce \"text\" v retezci str2: " + x + "\n");
        Game.play(str1);
    }   // end of static void run()
}   // end of class Main
class Game
{
    static void play(String str)
    {
        ifj16.print("Zadejte nejakou posloupnost vsech malych pismen a-h, ");
        ifj16.print("pricemz se pismena nesmeji v posloupnosti opakovat:");
        str = ifj16.readString();
        str = ifj16.sort(str);
        int cmp = ifj16.compare(str, "abcdefgh");
        if (cmp != 0) {
            while (cmp != 0) {
                ifj16.print("Spatne zadana posloupnost, zkuste znovu:");
                str = ifj16.readString();
                str = ifj16.sort(str);
                cmp = ifj16.compare(str, "abcdefgh");
            }
        }
        else {
        }
        return;
    }   // end of static void play(String)
}   // end of class Game
```

11 Doporučení k testování

Programovací jazyk IFJ16 je schválně navržen tak, aby byl téměř kompatibilní s podmnožinou jazyka Java SE 8. Pokud si student není jistý, co by měl interpret přesně vykonat pro nějaký kód jazyka IFJ16, může si to ověřit následovně. Z IS FIT si stáhněte ze *Souborů* k předmětu IFJ ze složky *Projekt* soubor `ifj16.java` obsahující kód, který

doplňuje kompatibilitu IFJ16 s překladačem `javac` jazyka Java na serveru `merlin` (obsahuje např. definici třídy se statickými vestavěnými funkcemi, které jsou součástí jazyka IFJ16, ale v jazyce Java mají odlišný způsob volání apod.). Váš program v jazyce IFJ16 uložený například v souboru `testovanyProgram.ifj` pak lze interpretovat na serveru `merlin` například pomocí trojice příkazů:

```
cat ifj16.java testovanyProgram.ifj > tmp.java
javac tmp.java
java ifj16 # nebo java ifj16 < test.in > test.out
```

Tím lze jednoduše zkontrolovat, co by váš interpret měl provést. Je ale potřeba si uvědomit, že jazyk Java je nadmnožinou jazyka IFJ16, a tudíž může zpracovat i konstrukce, které nejsou v IFJ16 povolené (např. bohatší množina typů, možnost vytvářet objekty, 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 (Ⓛ zastupuje unixové odřádkování):

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

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 registrované 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; povinně 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[16] 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, ...).

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.

[16] Vyjma obyčejného loga fakulty na úvodní straně.

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 skript `is_it_ok.sh` 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 ter-

mínu (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 jako přitěžující okolnost 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 IFJ16:

Popis rozšíření vždy začíná jeho identifikátorem. Většina těchto rozšíření je založena na dalších vlastnostech jazyka Java. Podrobnější informace lze získat ze specifikace jazyka[2] Java.

- 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 Java. Do dokumentace je potřeba uvést, jak je tento problém řešen (+1,0 bodu).
- BASE: Celočíselné konstanty (nikoliv však escape sekvence) je možné zadávat i ve dvojkové (číslo začíná **`0b`**), osmičkové (nenulové číslo začíná znakem **`0`**) a šestnáctkové (číslo začíná **`0x`**) soustavě. Desetinná čísla je možné kromě desítkové soustavy zadávat i v šestnáctkové soustavě, kdy je povinný celočíselný exponent uvozený písmenem **`p`** nebo **`P`** (např. **`0xFF.FFp-1`**) (+0,5 bodu). Číselné konstanty či celočíselné části budou obsahovat alespoň jednu platnou číslici. Ve všech soustavách lze použít speciální symbol _ (podtržítko), který slouží jako oddělovač skupin číslic v čísle pro zajištění lepší čitelnosti. Více viz popis číselných literálů v normě[2] jazyka Java.

- CYCLES: Interpret bude podporovat i cykly typu **`for`** a **`do-while`** včetně příkazů **`break`** a **`continue`** (+1,5 bodu). Příkaz cyklu **`do-while`** je převrácenou obdobou příkazu cyklu **`while`** a jeho sémantika, stejně tak jako sémantika příkazů **`break`** a **`continue`**, odpovídá zvyklostem jazyka Java. Příkaz cyklu **`for`** má tvar:

  `for (` *definice_proměnné*`;` *výraz*`;` *příkaz_přiřazení* **`)`** *složený_příkaz*

  Skládá se z hlavičky uzavřené v kulatých závorkách (definice proměnné ani příkaz přiřazení zde nejsou ukončeny standardním středníkem) a z těla tvořeného složeným příkazem. Sémantika příkazu cyklu **`for`** je následující: Před provedením první iterace je definována lokální proměnná (tzv. iterační) podle *definice_proměnné* (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. Nově definovaná iterační proměnná je platná v hlavičce a následném těle příkazu.
- FUNEXP: Volání statické 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 podmíněného 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 **`boolean`**, booleovských hodnot **`true`** a **`false`**, booleovských výrazů včetně kulatých závorek a základních booleovských operátorů (**`!`**, **`&&`**, **`||`**), jejichž priorita a asociativita odpovídá jazyku Java. Dále podpora pro výpis a definice proměnných typu **`boolean`** (+1,0 bodu).
- ...