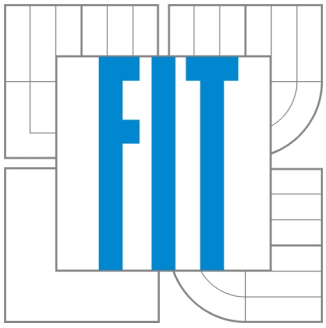
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



IFJ – Formální jazyky a překladače
Dokumentace k projektu
1. prosince 2015

Tým 055, varianta a/4/I
a – vyhledávání pomocí Knuth-Morris-Prattův algoritmu
4 – řazení pomocí algoritmu List-Merge sort
I – implementace tabulky symbolů pomocí binárního stromu

Řešitelé:
Manh Le Nguyen (xlengu00) – vedoucí – (25%)
Erik Macháček (xmacha63) – (25%)
Michal Klacik (xklaci00) – (0%)
Martin Malárik (xmalar02) – (25%)
Adam Bednárik (xbedna57) – (25%)

Rozšíření:
BASE
WHILE

Obsah

1 Úvod **2**

2 Konečný automat lexikálního analyzátoru **3**

3 LL gramatika a precedenční tabulka **4**

4 Popis řešení interpretu **5**
4.1 Návrh . . . 5
4.2 Vývojový cyklus . . . 5
4.3 Způsob práce v týmu . . . 5
4.4 Implementace a algoritmy . . . 5
4.4.1 Lexikální analyzátor . . . 5
4.4.2 Syntaxí řízený překlad . . . 5
4.4.3 Interpret a generátor kódu . . . 5
4.4.4 Testování . . . 5
4.4.5 Knuth-Morris-Prattův algoritmus . . . 6
4.4.6 List-Merge sort . . . 6
4.4.7 Binární strom . . . 6
4.4.8 Rozšíření . . . 6

5 Rozdělení práce **6**
5.1 main . . . 6
5.2 address_code . . . 6
5.3 deque . . . 6
5.4 errors . . . 6
5.5 ial . . . 6
5.6 interpret . . . 6
5.7 memory_manager . . . 6
5.8 parser . . . 7
5.9 scanner . . . 7
5.10 str . . . 7
5.11 symbol_table . . . 7
5.12 token . . . 7

6 Závěr **7**

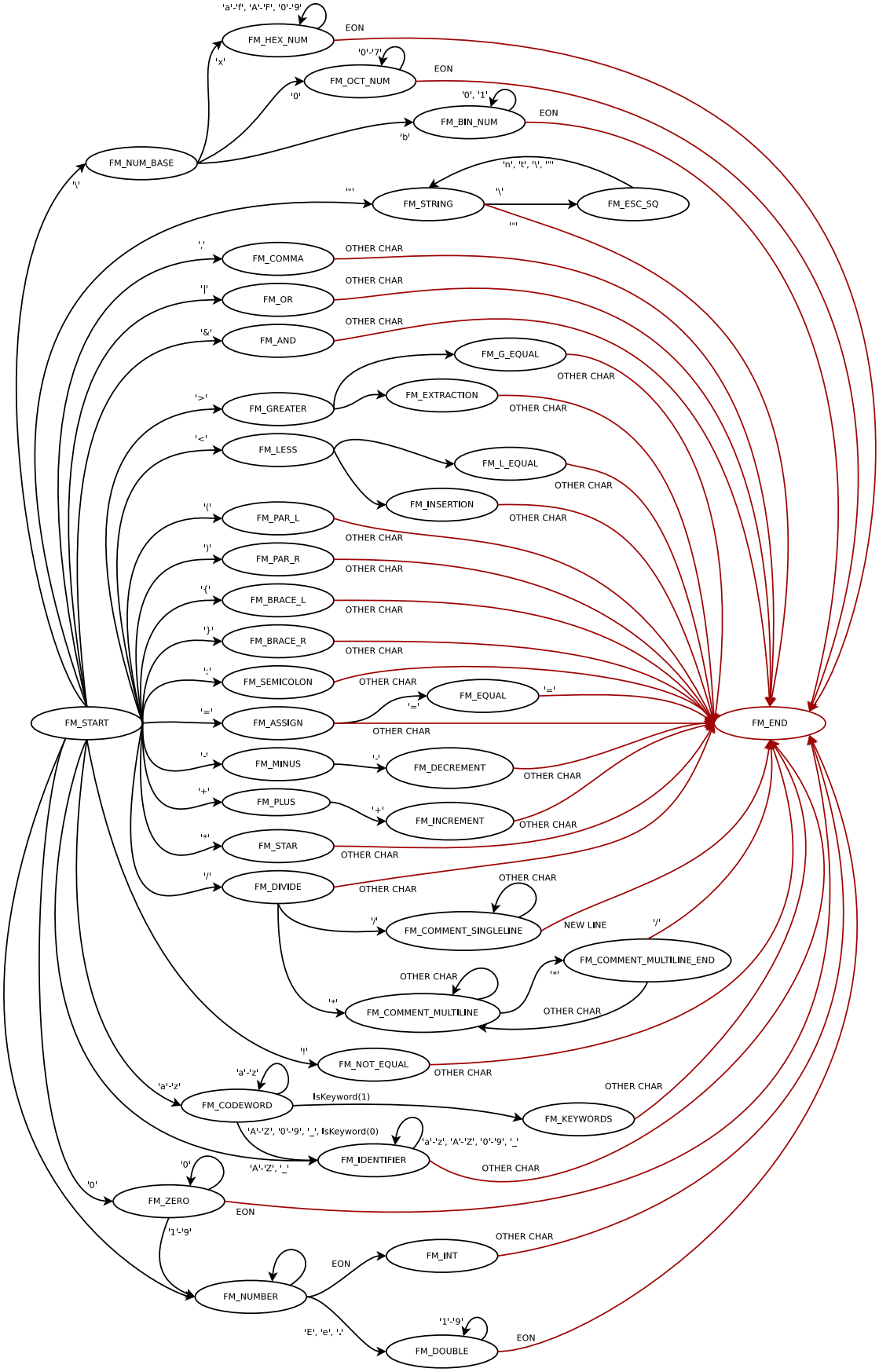
7 Refenrence a zdroje **8**

1 Úvod

Dokumentace se zabývá vývojem a implementací interpretu jazyka IFJ15. Popisuje jednotlivé algoritmy využité při implementaci, včetně implementace modulů. Obsahuje také konečný automat lexikálního analyzátoru a LL gramatiku, včetně precedenční tabulky. Dále popisuje řešení imterpretu od návrhu, pokračuje vývojovým cyklem, způsobem práce v týmu až po implementací a algoritmy.

2 Konečný automat lexikálního analyzátoru

Klíčová slova: auto, cin, cout, double, else, for, if, int, return, string, bool, do, while, true, false



3 LL gramatika a precedenční tabulka

```
                EOF,if, else,    for,return, cin,cout,    {,  },  ),  ',',;,  <<, >>, id, typ,exp,=,  do, while
$               {1, 0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  0,  0,  0,  0,  2,  0,  0,  0,  0},
FUNKCE_LS       {29,0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  0,  0,  0,  0,  3,  0,  0,  0,  0},
FUNKCE_HEAD     {0, 0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  0,  0,  0,  0,  4,  0,  0,  0,  0},
FUNKCE_P        {0, 0,  0,       0,  0,      0,  0,       0,  0,  6,  9,  0,  0,  0,  0,  5,  0,  0,  0,  0},
FUNKCE_H_END    {0, 0,  0,       0,  0,      0,  0,       8,  0,  0,  0,  7,  0,  0,  0,  0,  0,  0,  0,  0},
STAT_LS         {0, 10, 0,       10, 10,     10, 10,      10, 22, 0,  0,  0,  0,  0,  10, 10, 0,  0,  10, 10},
STAT            {0, 23, 0,       25, 21,     15, 18,      24, 0,  0,  0,  0,  0,  0,  14, 11, 0,  0,  27, 28},
VAR_END         {0, 0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  12, 0,  0,  0,  0,  0,  13, 0,  0},
CIN_LS          {0, 0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  17, 0,  16, 0,  0,  0,  0,  0,  0},
COUT_LS         {0, 0,  0,       0,  0,      0,  0,       0,  0,  0,  0,  19, 20, 0,  0,  0,  0,  0,  0,  0},
```

RULE 0: Odstraní jeden token z deque tokenů a jeden token ze zásobníku parseru
RULE 1: Odstraní poslední token z deque tokenů a poslední token ze zásobníku parseru – parsování dokončeno
RULE 2: $ → FUNKCE_LS $
RULE 3: FUNKCE_LS → FUNCKE_HEAD FUNKCE_HEAD_END FUNKCE_LS
RULE 4: FUNKCE_HEAD → TYP_UNIVERSAL IDENTIFIER, (FUNKCE_P)
RULE 5: FUNCKE_P → TYP_UNIVERSAL IDENTIFIER FUNKCE_P
RULE 6: FUNCKE_P → _NOTHING
RULE 7: FUNKCE_HEAD_END → ;
RULE 8: FUNKCE_HEAD_END → { STAT_LS }
RULE 9: FUNKCE_P → COMMA FUNKCE_P
RULE 10: STAT_LS → STAT STAT_LS
RULE 11: STAT → TYP_UNIVERSAL IDENTIFIER VAR_END
RULE 12: VAR_END → ;
RULE 13: VAR_END → ASSIGN EXP ;
RULE 14: STAT → IDENTIFIER ASSIGN EXP ;
RULE 15: STAT → CIN ≫ IDENTIFIER CIN_LS
RULE 16: CIN_LS → ≫ IDENTIFIER, CIN_LS
RULE 17: CIN_LS → ;
RULE 18: STAT → COUT ≪ ID_OR_TYP COUT_LS
RULE 19: COUT_LS → ;
RULE 20: COUT_LS → ≪ ID_OR_TYP COUT_LS
RULE 21: STAT → RETURN EXP ;
RULE 22: STAT_LS → NOTHING
RULE 23: STAT → if (EXP) { STAT_LS } else { STAT_LS }
RULE 24: STAT → { STAT_LS }
RULE 25: STAT → for (TYP_UNIVERSAL IDENTIFIER ASSIGN EXP ; EXP ; IDENTIFIER ASSIGN EXP) { STAT_LS }
RULE 27: STAT → do { STAT_LS } while (EXP) ;
RULE 28: STAT → while (EXP) { STAT_LS }
RULE 29: STAT_LS → _NOTHING

_NOTHING představuje pouze odstranění za zásobníku parseru

4 Popis řešení interpretu

4.1 Návrh

Úkolem našeho projektu bylo naimplementovat interpret v jazyce C, který načte zdrojový soubor v jazyce IFJ15. Předtím než jsme začali psát kódy, tak jsme nejprve načrtli na papír diagram celého projektu. Určili jsme moduly, které musí být naimplementovány první, a všechno, co do nich patří. Program byl námi rozdělen na 3 hlavní moduly SCANNER, PARSER a INTERPRET. Dále jsme vytvořili několik podpůrných modulů. Moduly jsou úzce propojeny a bez jakékoli z částí, program nemusel fungovat správně. K jednotlivým modulům jsme také vytvořili souborové hlavičky.

4.2 Vývojový cyklus

Vypracované soubory jsme průběžně ukládali na privátní účet na Githubu. Přístup na tento účet měl každý z členů týmu. Moduly byly aktualizovány pravidelně, a proto nezbytnou součástí bylo komentování jednotlivých pasáží v kódech a dodržení pravidel při úpravě nebo tvorbě nových modulů. Tím usnadnili práci nejen sobě, ale i ostatním. Samozřejmě ne vždy se nám to povedlo. Pokud byl upravený modul špatně naimplementován a byl nahrán na Github. Okamžitě jsme ho zahodili a nahráli jsme zpátky jeho starou funkční verzi, aby ostatní mohli nadále pracovat. Občas se nám také stávalo to, že moduly mezi sebou špatně komunikovaly, jelikož každý člen své moduly vyvíjel jinak, než druhý očekával.

4.3 Způsob práce v týmu

Projekt jsme rozdělili na jednotlivé části a ke každé části jsme určili počet členů potřebný pro konkrétní úlohu. Většinou jsme moduly vypracovávali ve více počtech. Prioritní ideou našeho týmu byla skupinová sešlost v plném zastoupení týmu, avšak občas z různých důvodů se někdo z týmu nemohl dostavit. Museli jsme pro to použít jiné komunikační prostředky (např. Teamspeak, Skype, Facebook. . .).

4.4 Implementace a algoritmy

Program jsme implementovali v podobě jazyka C dle normy C99. Překlad modulů probíhal pomocí Makefilu. Také jsme naimplementovali rozšíření BASE a WHILE.

4.4.1 Lexikální analyzátor

Lexikální analyzátor je realizován pomocí konečného automatu (viz. sekce 2), který načítá v rámci jednoho cyklu jeden znak. Na základě přečteného znaku se automat rozhoduje, do jakého stavu se přepne. Výchozí stav při čtení nového tokenu je FM_START a stav pro ukončení tokenu je FM_END. Pro ukončení čísel se používá množina znaků End of Number (EON), která se skládá z: ”;”, ” ”, ”)”, ”+”, ”-”, ”/”, ”*”. Skupina znaků OTHER CHAR představuje množinu znaků, které nevyhovují tokenu v daném stavu konečného automatu. Pro rozpoznávání klíčových slov slouží implementovaná funkce IsKeyword(). Samotné tokeny jsou realizovány pomocí datové struktury, která uchovává typ tokenu a obsah tokenu. Modul následně vrací tokeny na požádání.

4.4.2 Syntaxí řízený překlad

Pro syntaxí řízený překlad jsme se rozhodli použít prediktivní analýzu řízenou LL-tabulkou. Syntaxí řízený překlad je realizován pomocí zásobníkového automatu. Kde v každému cyklu dojde k zjištění pravidla z LL_Tabulky podle posledního získaného tokenu a tokenu z vrchu zásobníku automatu. Jejich zpracování dle pravidel uvedených v sekci 3.

4.4.3 Interpret a generátor kódu

Interpret si vezme obousměrnou frontu z generátoru kódu a spouští akce, podle toho v jakém pořadí přichádzejí. Na začátku si najde funkci main. Následně začne cyklicky spouštět instrukce, které vykonává, dokud nenarazí na konec fronty, anebo je program ukončen. Interpret vkládá na předpřipravená místa data, která spracovává. Taky kontroluje typy proměnných, aby nenastal problém při zápisu hodnot či jejich znovupoužití.

4.4.4 Testování

Testování probíhalo v několika fázích. První fází bylo testování jednotlivých funkcí v modulech. Toto testování prováděl každý člen týmu sám. Druhou fází bylo testovní celého modulu, na tomto testování se podíleli všichni členové pracující na tomto modulu. Poslední fází testování bylo testování celého pojektu, které probíhalo za spolupráce vedoucích modulů. K testování byly použity jak tři doporučené testy v zadání, tak i testy vlastní.

Níže algoritmy jsme implementovali podle opory předmětu IAL (viz. 7).

4.4.5 Knuth-Morris-Prattův algoritmus

K hledání umístění podřetězce v řetězci nám byl vybrán Knutt-Morris-Prattův algoritmus, který pracuje následovně. Pro hledaný string si vytvoříme tabulku, ve které bude zapsáno, s kterým znakem se bude dál porovnávat při neúspěšném porovnávání, abychom se vyhnuli předávání všech prvků. Tuto tabulku vrátíme do původního rodičovského algoritmu, kde porovnáme vzor a hledání, dokud nenajdeme shodu. Pokud budeme úspěšní, vrátíme pozici na které se nachází. V případě, že počet znaků už nevyhovuje vzoru, vrátíme -1 jako chybu.

4.4.6 List-Merge sort

K sortování stringů jsme použili algoritmus List-Merge sort (dále jen LMS), který má podobný mechanismus řazení jako Merge sort (dále jen MS). Narozdíl od MS, LMS pracuje se seznamem listů. Abych tento řadicí algoritmus lépe vysvětlil. Popíši podrobněji naši implemetaci. Nejdříve jsme si vyzvedli původní neseřazený string. Potom jsme porovnávali jednotlivé indexy, abychom zřetězili neklesající posloupnosti znaků. Následně jsme vytvářeli seznam začátků a do něj vložili začátky všech neklesajících posloupností. Začátek se zjistil tak, že daný index byl menší anebo se rovnal předchozímu indexu ve stringu. Cyklus skončil tehdy, až jsme došli nakonec stringu. V dalším cyklu jsme si vyzvedli ze seznamu začátky dvou posloupností. Jejich seřízením jsme dostali novou zřetězenou neklesající posloupnost a její začátek jsme vložili do seznamu. Tím vznikl nový list. Cyklus skončil tehdy, kdy v seznamu zbyla jen pouze jedna posloupnost.

4.4.7 Binární strom

Posledním použitým algoritmem byla implementace tabulky symbolů pomocí binárního vyhledávacího stromu (dále jen BVS). Jednotlivé funkce BVS jsme implementovali rekurzivně pro jednoduchost a přehlehlednější zápis. Při inicializaci jsme ukazatel na kořen stromu nastavili na NULL. Vyhledávání ve stromu jsme rozdělili na vyhledávání podle symbolu a vyhledávání podle klíče. Pokud při některých situacích vyhledávání podle klíče nebylo postačující, použili jsme vyhledávání podle symbolu. Při vkládání uzlu do stromu a mazání stromu jsme použili námi implementované funkce na alokování a uvolňování paměti.

4.4.8 Rozšíření

BASE – Toto rozšíření umožňuje čtení čísel a znaků zadaných pomocí escape sekvencí.
WHILE – Rozšíření dovoluje využívat konstrukce cyklů while a do-while.

5 Rozdělení práce

Každý modul měl jednoho vedoucího a několik pomocníků.

5.1 main

vedoucí: xlengu00 pomocníci: xbedna57, xmacha63, xmalar02

5.2 address_code

vedoucí: xbedna57 pomocníci: xlengu00, xmalar02, xmacha63

5.3 deque

vedoucí: xmacha63 pomocníci: xbedna57, xmalar02, xlengu00

5.4 errors

vedoucí: xmalar02 pomocníci: xbedna57, xmacha63, xlengu00

5.5 ial

vedoucí: xbedna57 pomocníci: xlengu00, xmalar02, xmacha63

5.6 interpret

vedoucí: xmalar02 pomocníci: xbedna57, xmacha63, xlengu00

5.7 memory_manager

vedoucí: xlengu00 pomocníci: xbedna57, xmacha63, xmalar02

5.8 parser

vedoucí: xmacha63 pomocníci: xbedna57, xmalar02, xlengu00

5.9 scanner

vedoucí: xmacha63 pomocníci: xbedna57, xmalar02, xlengu00

5.10 str

vedoucí: xmalar02 pomocníci: xbedna57, xmacha63, xlengu00

5.11 symbol_table

vedoucí: xbedna57 pomocníci: xlengu00, xmalar02, xmacha63

5.12 token

vedoucí: xlengu00 pomocníci: xbedna57, xmacha63, xmalar02

6 Závěr

Tento projekt byl pro nás velkou výzvou. Díky tomuto projektu jsme si prohloubili znalosti v programovacím jazyce C. Získali jsme nové zkušenosti v rámci logického uvažování a téže z hlediska práce a komunikace v týmu. Vyzkoušeli jsme si vývoj komplexnější aplikace v týmu a zároveň jsme se naučili používat Github.

7 Refenrence a zdroje

https://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm
https://www.ics.uci.edu/~eppstein/161/960227.html
Skripta k předmětu IAL