Jazyk IFJ12

Dokumentácia z predmetu IFJ a IAL

27. listopadu 2012

Varianta - Tým 086, varianta a/3/II

Hodnotenie:
Viktor Malík 20% xmalik00,
Martin Maga 20% xmagam00,
Vojtěch Meca 20% xmevac00 ,
Vít Mojžíš 20% xmojzi00(vedúci projektu),
Jiří Macků 20% xmacku00

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Táto dokumentácia sa zaoberá vývojom, implementáciou a testovaním interpreta pre jazyk IFJ12. Dokumentácia je logicky rozdelená do celkou, ktoré popisujú jednotlivé fázy, ktorými musel projekt postupne prejsť. Jednotlivé kapitoly a podkapitoly popisujú podstatné problémy, algoritmy a zpôsoby, ktoré sme použili pri implementácii interpreta. Dokumentácia obsahuje taktiež grafický návrh lexikálneho analyzátora a LL-gramatiku.

2 Práca v týme

2.1 Príprava na projekt

Pred samotným začatím projektu prebiehala základné naštudovanie zadania projektu. V počiatočným fázoch bolo zadanie nejasné z dôvodu neznalosti problematiky implementácie interpretu a jeho súčasti. Preto sme sa snažili naštudovať informácie popri konkretnej implementácií, čo nám niekedy prácu trochu spomaľovalo. Rozhodli sme sa používať verzovací program *Git*, ktorý je vzhľadom na jeho ľahké a intuitívne používanie celkom príjemný a tak isto existuje množstvo návodov v prípade, že sme museli riešiť nejakú chybu za chodu s verziami programu.

2.2 Komunikácia v týme

Na prvom stretnutí sme sa dohodli, že budeme organizovať každý týždeň teamové stretnutie, kde budeme vždy prejednávať aktuálne rozpravacované časti projektu a taktiež problémy, ktoré sme mali pri vývoji jednotlvých častí interpretu. Vzhľadom na časovú náročnosť komunikácie online sme sa rozhodli použiť narýchlejšií spôsob a tak sme využili voľne dostupný instant messenger[1].

2.3 Použité metodiky

Pri počiatočnom vývoji sme nemali celkom jasný smer akým sa budeme uberať. Na začiatku sme zvolili trend, ktorý vychádza z metódy *Extreme programming*. Rozdelili sme si celý interpret na fázy, z ktorých sa skladá: lexikálna analýza, syntaktická analýza, sémantická analýza, generátor vnútorné kódy, interpreter. Jednotlivé fázy boli takmer vždy pridelované 2 členom vývojové týmu, kvôli snahe o najoptimálnejší výsledok kódu a taktiež maximálnu časovú efektívnosť. Následne po implementácií boli jednotlivé časti dôsledne otestované a to z dôvodu vysokej nadväznosti medzi jednotlivými časťami interpretra. Pri takom pridelený práce slúžili ostatní členovia tímu ako pomocná sila a vytvárali parciálne časti, ktoré boli väčsinou reprezentované funkciami.

[1] Program umožňujúci posielať správy a čítať správy v reálnom čase

3 Implementácia

3.1 Merge-sort

Merge-Sort je radiaci algoritmus, ktorý je v podstate metóda, ktorá využíva priamy prístup k prvkom poľa. tento algoritmus využíva postup, pri ktorom sa strieda pozícia zdrojového a cieľového poľa a aj krok proti sebe postupujúcim neklesajúcim postupnostiam[1]. Merge sort je algoritmus, ktorý je nestabilný a nie je možné ho realizovať priamo len s danou štruktúrou, ktorá obsahuje položky k zoradenie „in situ“[2]. Asymtotická zložitosť je semilogaritmická, preto môžme považovať tento algoritmus za veľmi rychly, ale nie je jednoduchý postup jeho implementácie. Existuje aj niekoľko jeho vylepšení, ktoré zlepšujú chovanie pôvodného algoritmu. Jedným z vylepšením je algoritmus *List Merge-Sort*, ktorý využíva princíp zlučovania ale bez presunu položiek.

3.2 Knuth-Moorris-Prattov algoritmus

Tento algoritmus bol použitý pri implementácia vstavanej funkcie *find*, ktorá je súčasťou jazyka *IFJ12*. Tento algoritmus slúži na vyhľadávanie podreťazca v rámci reťazca. Pri implementácií sme sa inšpirovali informáciami, ktoré boli podané na predmete IAL. Tento algoritmus využíva pri svojej práci konečný automat. Tento automat zjednodušene využíva 3 typy uzlov: 1. START, 2.STOP, 3. READ.vytvoríme si pred začatím vyhľadávania tabuľku, v ktorej budeme mať pre každú pozíciu vo vzore napísané číslo, ktoré nám bude určovať, koľký prvok vzoru máme porovnávať s aktuálnym znakom v reťazci, ak bolo porovnanie na tejto pozícii neúspešné a my sa nechceme vrátiť v reťazci späť k pozícii začatia porovnávania (presnejšie k nasledujúcej pozícii od pozície začatia porovnávania). Algoritmus má zložisť O(n+m) teda lineárnu, preto môžme tento algoritmus považovať za veľmi efektívny.

3.3 Tabulka symbolov

Spôsob, ktorým sme implementovali tabulku symbolov odpovedá hashovacej tabulke. Je to vlastne štruktúra, ktorá sa skláda z položiek, ktoré obsahujú nejaké dáta. Prístup k jednotlivým položkám je realizovaný pomocou hashovacej funkcia, ktorá pre danú hodnotu kľúča vypočíta index do tabuľky. Jednotlivé položky môžu byť zreťazané. Tento algoritmus prináša značnú efektivitu práce a to z dôvodu konštantnej časovej zložitosti, ktorá daná pristúp k najdlhšej postupnosti zreťazených položiek v tabulke.

3.4 Lexikálny analyzátor

Základný princíp implementácia lexikálneho analyzátora spočíval v korektnom návrhu konečného automatu, ktorý tento analyzátor vytvára, ten musel vychádzať zo špecifikácie jazyka *IFJ12*. Lexikálny analyzátor slúži na rozpoznávanie lexém zdrojové kódu. Jednotlivé lexémy boli rozpoznávané na základe jednotlivých stavov konečného automatu. V prípade, že lexikálny analyzátor narazil na lexikálnu chybu vrátil príslušný chybný token s informáciou o chybe v opačnom prípade vrátil typ tokenu a hodnotu tokenu, ktorú analýzator zistil. Konečný automat rozpoznával taktiež klúčové a rezervované slová jazyka *IFJ12* a to pomocou prechodu poľa reťazcov, ktoré bolo týmito hodnotami naplnené.

[2] Priamo s danou štruktúrou bez použitia nejakej pomocnej štruktúry

3.5 Syntaxou riadený preklad

Pri implementácií syntaktického analyzátoru sme využili LL-gramatiku a postupovali sme prostredníctvom zásobniku a postfix notácie. Podstatou syntaktickej analýzy je tabulka na základe, ktorej je riadená redukcia a vyhodnocovanie výrazov.

3.6 Interpret

3.7 Testovanie

Testovanie našeho projektu prebiehalo na architektúrach Windows a Linux. Bolo založené na vopred napísaných testoch, ktorá porovnávali jednotlivé výsledky testovanej časti s referenčnými vysledkami. Testovanie prebiehol po častiach postupne ako boli implementované jednotlivé časti intepretra jazyka. Pričom sa venoval značný dôraz na testy lexikálneho analýzatora. V konečnej fáze bolivykonané komplexné testy, ktoré overili funkčnost jazyka *IFJ12* podľa špecifikácie uvedenej v zadaní. V pripáde, že bola objavená chyba bola ihneď odstránená a interpreter bol opäť otestovaný.

4 LL gramatika

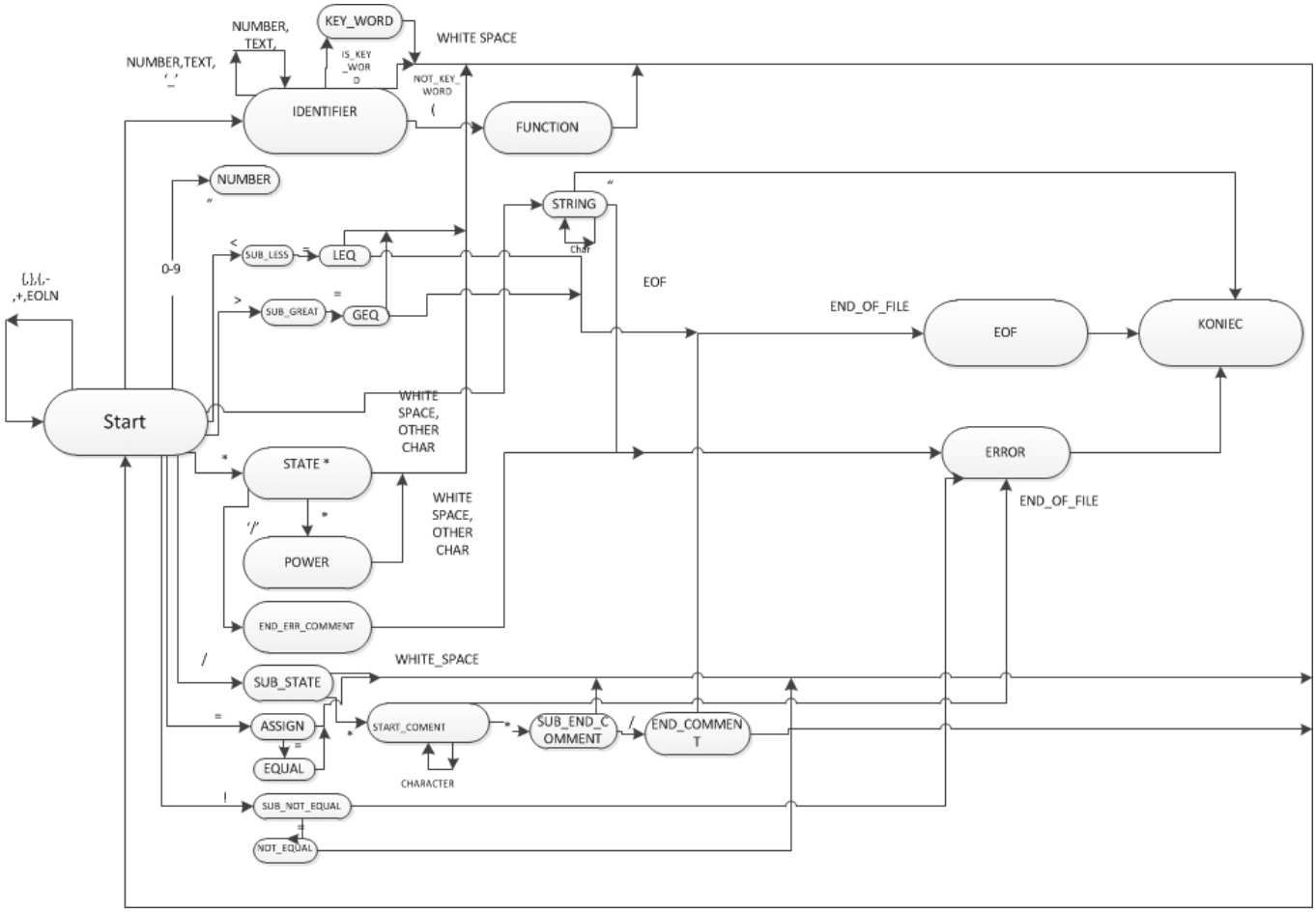
REAL, ADDING, REAL → REAL
REAL, SUBTRACKING, REAL→ REAL
REAL, MULTIPLY, REAL→ REAL
REAL, DIVISION, REAL→ REAL
REAL, POWER, REAL→ REAL
STRING, ADDING, STRING→ STRING
BOOLEAN, ADDING, STRING→ STRING
REAL, ADDING, STRING→ STRING
REAL, POWER, STRING→ STRING
STRING, LOG_OP, REAL→ BOOLEAN
REAL, LOG_OP, STRING→BOOLEAN
BOOLEAN, LOG_OP, STRING→ BOOLEAN
BOOLEAN, LOG_OP, REAL→ BOOLEAN
BOOLEAN, LOG_OP, BOOLEAN→ BOOLEAN
REAL, LOG_OP, BOOLEAN→ BOOLEAN
STRING, LOG_OP, BOOLEAN→ BOOLEAN
NIL, LOG_OP, NIL→ BOOLEAN
BOOLEAN, LOG_OP, NIL→ BOOLEAN
REAL, LOG_OP, NIL→ BOOLEAN
STRING, LOG_OP, NIL→ BOOLEAN
NIL, LOG_OP, STRING→ BOOLEAN
NIL, LOG_OP, REAL→ BOOLEAN
NIL, LOG_OP, BOOLEAN→ BOOLEAN
STRING, LOG_OP, STRING→ BOOLEAN
REAL, LOG_OP, REAL, BOOLEAN
RIGHT_BRACKET, REAL, LEFT_BRACKET→ REAL
RIGHT_BRACKET, STRING, LEFT_BRACKET→ STRING
RIGHT_BRACKET, BOOLEAN, LEFT_BRACKET→ BOOLEAN
NOTDEF, ADDING, REAL→ NOTDEF
NOTDEF, SUBTRACKING, REAL→ REAL
NOTDEF, MULTIPLY, REAL→ REAL
NOTDEF, DIVISION, REAL→REAL
NOTDEF, POWER, REAL→ REAL
NOTDEF, ADDING, STRING→ STRING
NOTDEF, POWER, STRING→ STRING
NOTDEF, LOG_OP, BOOLEAN→ BOOLEAN
NOTDEF, LOG_OP, STRING→ BOOLEAN
NOTDEF, LOG_OP, REAL→ BOOLEAN
NOTDEF, LOG_OP, NIL→ BOOLEAN
NIL, LOG_OP, NOTDEF→ BOOLEAN
RIGHT_BRACKET, NOTDEF→ LEFT_BRACKET, NOTDEF
NOTDEF, ADDING, NOTDEF→ NOTDEF
NOTDEF, SUBTRACKING, NOTDEF→ REAL
NOTDEF, MULTIPLY, NOTDEF→ REAL
NOTDEF, DIVISION, NOTDEF→ REAL

NOTDEF, POWER, NOTDEF→ NOTDEF
NOTDEF, LOG_OP, NOTDEF→ BOOLEAN
REAL, ADDING, NOTDEF→ NOTDEF
REAL, SUBTRACKING, NOTDEF→ REAL
REAL, MULTIPLY, NOTDEF→ REAL
REAL, DIVISION, NOTDEF→ REAL
STRING, ADDING, NOTDEF→ STRING
BOOLEAN, ADDING, NOTDEF→ STRING
REAL, POWER, NOTDEF→ NOTDEF
STRING, LOG_OP, NOTDEF→ BOOLEAN
REAL, LOG_OP, NOTDEF→BOOLEAN
BOOLEAN, LOG_OP, NOTDEF→ BOOLEAN

5 Záver

Práca na projekte, ktorého cieľom bola implementácia jazyka IFJ12 nám priniesla bohaté skúsenosti s rozsiahlymi projektami. Naučili sme sa ako si správne rozdeliť prácu v týme a tak isto aké efektívne spôsoby komunikácie treba zvoliť pri riešení problématiky. Tak isto sme sa naučili používať pokročilé nástroje pri správe verzií programu. Tento projekt nám priniesol tak isto nové skúsenosti pri vývoji interpretov jazyka. Náš projekt bol testovaný na platforme GNU Linux a výsledky bolo porovnané s nami vytvorenými referenčnými vysledkami.

6 Konečný automat lexikálneho analyzátoru



7 Referencie

Reference

[1] PROKOP, J.: *Algoritmyv jazyku C a C++0 :praktický průvodce.* Brno:Grada Publishing, 2009, ISBN 978-80-247-2751-6.

Skriptá z IAL-u