PROGRAMOVACÍ JAZYKY A PŘEKLADAČE

SEMESTRÁLNÍ PROJEKT

Obsah

O projektu

Cílem projektu je vytvořit funkční překladač s předem danou gramatikou, která určuje podobu programu. Překladač generuje mezikód v maximálním tvaru *A = B op C*, kde A je cílová adresa, reprezentující buď trvalou (programátorem definovanou) nebo dočasnou (temporary) proměnnou Ti. B a C jsou volitelné operandy a *op* je operátor.

Pro realizaci řídicích příkazů je třeba generovat ještě kódové instrukce skoku ve tvaru: *IF podmínka GOTO návěští* nebo *GOTO návěští.*

Kromě toho musí být překladač schopen ignorovat neviditelné mezery, jednořádkové komentáře uvozené dvojitým lomítkem a víceřádkové komentáře v klasické C notaci.

Výsledný překladač je realizován pomocí *flex* a *yacc* lexikálního a syntaktického analyzátoru. Pomocné deklarace jsou umístěny v pomocném souboru *header.h*.

Jednorázový překlad je řešen pomocí utility *make*.

Celý projekt překladače je realizován a testován v programu CYGWIN.

Návrh gramatiky

Návrh gramatiky proběhl na základě slovní specifikace.

```
PROGRAM:
    | PROGRAM FUNCTION
    | ε


FUNCTION:
    TYP IDENTIFIKATOR_TOK (ARGUMENTY) (DEFINICE)
    | ε

ARGUMENTY:
   TYP IDENTIFIKATOR_TOK ARGUMENTY
   |  ARGUMENTY
   | ε


DEFINICE:
   IFCOMMAND  DEFINICE
   | WHILECOMMAND DEFINICE
   | PRIRAZENI DEFINICE
   | RETURN
   | ε


RELACE:
    EXPRESSION
    |  EXPRESSION ASSIG_TOK EXPRESSION

IFCOMMAND:
    IF_TOK (RELACE) {DEFINICE}
    ELSE_TOK {DEFINICE}

WHILECOMMAND:
    WHILE_TOK (RELACE) {DEFINICE}

PRIRAZENI:
    IDENTIFIKATOR_TOK = VYRAZ

    | TYP PRIRAZENI

EXPRESSION:
    | ESPRESSION + EXPRESSION
    | ESPRESSION - EXPRESSION
    | ESPRESSION * EXPRESSION
    | ESPRESSION / EXPRESSION

VYRAZ:
   EXPRESSION ;

RETURN:
   RETURN_TOK VYRAZ
```

Funkčnost

Překladač je realizován pomocí generátorů.

Flex

V tomto souboru *project.l* jsou zahrnuty knihovny pro standartní vstupy i výstupy a také soubor, kde jsou definovány pomocné deklarace.

```
#include "project.tab.h"
#include "project.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Dále jsou zde definice pro tokeny, znaky, regulární výrazy pro proměnné a hodnoty, mezery a komentáře.

```
"while"   return WHILE_TOK;
"if"      return IF_TOK;
"else"    return ELSE_TOK;
"return"  return RETURN_TOK;

"int"     { yylval.type = 0; return TYP; }
"float"   { yylval.type = 1; return TYP; }


"<"    { yylval.assig = strdup(yytext); return ASSIG_TOK; }
">"    { yylval.assig = strdup(yytext); return ASSIG_TOK; }
"<="   { yylval.assig = strdup(yytext); return ASSIG_TOK; }
">="   { yylval.assig = strdup(yytext); return ASSIG_TOK; }
"=="   { yylval.assig = strdup(yytext); return ASSIG_TOK; }
"!="   { yylval.assig = strdup(yytext); return ASSIG_TOK; }

[a-zA-Z][a-zA-Z0-9]*  { yylval.type = symlook(yytext); return (IDENTIFIKATOR_TOK);}
[0-9]+                { yylval.type = symlook(yytext); return (NUMBER_TOK);      }


"//"[^\n]*          ;
"/*"                { BEGIN(C_COMMENT); }
<C_COMMENT>"*/"     { BEGIN(INITIAL); }
<C_COMMENT>\n       { ;}
<C_COMMENT>.        { ;}
```

Yacc

V tomto souboru jsou v první řadě také zavedeny knihovny. Dále datové typy, specifikace pro asociativitu, aby nedošlo k syntaktické chybě a gramatika. Soubor obsahuje i funkci, která slouží pro hledání dočasných proměnných pro mezivýpočet. Provádí to hledáním od největšího v tabulce symbolů a předá pozici v tabulce. Pokud je nenajde, vytvoří ji je.

```
int findOrCreateT(int idx1, int idx3){

	int index = -1;
	int pomtemp = tempcount;
	while(pomtemp > 1){
		pomtemp--;
		char str[6];
		snprintf(str, 5, "T%i", pomtemp);
		str[5] = '\0';

		if(strcmp(str, symtable[idx1]) == 0) {
			index = idx1;
		}
		if(strcmp(str, symtable[idx3]) == 0) {
			index = idx3;
		}
	}

	if((-1 != index))

		return index;
	else
		return gettemp();
}
```

Zachycuje chyby v symbolu na konkrétním řádku. Pokud vše proběhne v pořádku, přeloží soubor a vypíše zprávu.

Makefile

Soubor *Makefile* slouží pro kompilaci souboru do finální podoby.

```
CC = gcc
LIBS = -lfl
LEX = flex
CFLAGS = -g -std=gnu99
YACC = byacc -d -v -t


all: project.exe

#-------------------------------------------------



project.exe: project.tab.o project.yy.o
	${CC} -o project.exe project.tab.o project.yy.o ${LIBS} $(CFLAGS)

project.tab.c project.tab.h: project.y
	${YACC} -o project.tab.c project.y

project.yy.c: project.l
	${LEX} -o project.yy.c project.l


#-------------------------------------------------


test: test0 test1 test2

test0:
	./project.exe < ./test_files/test1.txt

test1:
	./project.exe < ./test_files/test2.txt

test2:
	./project.exe < ./test_files/test3.txt


 #-------------------------------------------------

clear: clean
clean:
	rm -f *.yy.c *.tab.c
	rm -f *.tab.h
	rm -f *.o
	rm -f *.output *.stackdump
#	rm -f project.exe
```

Ukázka

Kompilace *.exe* souboru pomocí *make*

```
voracek@NTB-VORACEK-2 ~/PJP/sample_project
$ make
byacc -d -v -t   -o project.tab.c project.y
byacc: 2 rules never reduced
byacc: 24 shift/reduce conflicts, 28 reduce/reduce conflicts.
gcc -g -std=gnu99     -c -o project.tab.o project.tab.c
flex -o project.yy.c project.l
gcc -g -std=gnu99     -c -o project.yy.o project.yy.c
gcc -o project.exe project.tab.o project.yy.o -lfl -g -std=gnu99
```

Spuštění *souboru.exe* na testovacím souboru *.txt*

Vstup: *test3.txt*

```
int prvni(){}
int druha(int a, int x){}
int treti(int a, int b){ int c = a+b; }
int ctvrta(int a, int b){ int c = a+b-c*d; }
int pata(){int c = a+(b-c)*d;}
int sesta(){
 if(3*5 <= x){
  if(a < 10){
   int b = 20;
  }
  else {
   int b = 40;
  }
 }
 else {
  int b = x;
  while (a < 10) {
  int a = 100;
 }
 return b;
 }
}
```

Spuštění souboru *.exe*

```
voracek@NTB-VORACEK-2 ~/PJP/sample_project
$ ./project < test3.txt
T1 = a + b ;
c = T1 ;
T1 = a + b ;
T2 = c * d ;
T1 = T1 - T2 ;
c = T1 ;
T1 = b - c ;
T1 = T1 * d ;
T1 = a + T1 ;
c = T1 ;
T1 = 3 * 5 ;
T1 = T1 ;
T1 = T1 <= x ;
if not T1 goto L1
T1 = a ;
T1 = T1 < 10 ;
if not T1 goto L2
b = 20 ;
goto L3
L2:
b = 40 ;
L3:
goto L4
L1:
b = x ;
T1 = a ;
T1 = T1 < 10 ;
L5 :
if not T1 goto L6
a = 100 ;
GOTO L5 :
L6:
L4:
Gramatika je v poradku !!!
```

Závěr

Překladač byl realizován pomocí generátorů, které byly zkompilovány souborem typu *make*, součásti kompilace je také vygenerování spustitelného souboru *.exe* Ten lze spustit na základě vstupního zdrojového kódu v testovacím textovém souboru. Soubor je přeložen na základě syntaktické analýzy. Výstupem je tříadresový kód.