Assignment 7 – Implementation of Compiler Design Phases

Date: 09/05/2023

Aim:

To implement the various phases of a compiler.

Code:

//ex7.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror(char* s);
FILE* flex;
extern YYSTYPE yylval;
int line = 0;
%}

identifier [a-zA-Z_][a-zA-Z0-9_]*

intConst (\+|\-)?[0-9]+
doubConst (\+|\-)?[0-9]+\.[0-9]+
charConst \'.\'
strConst \"(.)*\"

arithop "+"|"-"|"*"|"/"
relop ">"|"<"|">="|"<="|"=="
boolop "!"|"&&"|"||"

%%

[ \t]   {}
\n      {}
```

```
"void"  {fprintf(flex, " %25s | %-25s \n", yytext, "type void"); return
VOID;}
"int"   {fprintf(flex, " %25s | %-25s \n", yytext, "type int"); return
INT;}

"("     {fprintf(flex, " %25s | %-25s \n", yytext, "sp char - left
parantheses"); return LEFTB;}
")"     {fprintf(flex, " %25s | %-25s \n", yytext, "sp char - right
parantheses"); return RIGHTB;}

"="     {fprintf(flex, " %25s | %-25s \n", yytext, "equal-to op"); return
ASSIGN_OP;}

"+"     {fprintf(flex, " %25s | %-25s \n", yytext, "plus op"); return
PLUS_OP;}
"-"     {fprintf(flex, " %25s | %-25s \n", yytext, "minus op"); return
MINUS_OP;}
"*"     {fprintf(flex, " %25s | %-25s \n", yytext, "mul op"); return
MUL_OP;}
"/"     {fprintf(flex, " %25s | %-25s \n", yytext, "div op"); return
DIV_OP;}

{intConst}|{doubConst} {fprintf(flex, " %25s | %-25s \n", yytext, "numeric
constant"); yylval.num = atoi(strdup(yytext)); return CONSTANT;}
{strConst} {fprintf(flex, " %25s | %-25s \n", yytext, "string constant");
yylval.string = strdup(yytext); return STRING;}

{identifier} {fprintf(flex, " %25s | %-25s \n", yytext, "identifier");
yylval.string = strdup(yytext); return ID;}

","     {fprintf(flex, " %25s | %-25s \n", yytext, "sp char - comma");
return COMMA;}
";"     {fprintf(flex, " %25s | %-25s \n", yytext, "sp char - semi-colon");
return EOS;}
.

%%
```

//ex7a.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror();
extern FILE* yyin;
extern FILE* flex;
FILE* fsyn;
FILE* ftac;
extern int line;
int error = 0;
int tempCount = 0;
char temp[10];
%}

%union {
    char* string;
    int num;
};

%type <string> expr

%token <string> VOID INT
%token <string> LEFTB RIGHTB COMMA EOS
%token <string> ASSIGN_OP PLUS_OP MINUS_OP MUL_OP DIV_OP
%token <string> ID STRING
%token <num> CONSTANT

%left PLUS_OP MINUS_OP
%left MUL_OP DIV_OP

%start program

%%
```

```
program:
statement program {
    return 0;
}
| '\n'
|
;

statement:
declnstatement EOS
| assignment EOS
;

declnstatement:
type var
;

type:
INT
| VOID
;

var:
var COMMA var
| ID
| assignment
;

assignment:
ID ASSIGN_OP expr {
    fprintf(ftac, "%s = %s\n", $1, $3);
}
;

expr:
ID {
    $$ = strdup($1);
}
| CONSTANT {
    char number[10];
```

```
    sprintf(number, "%d", $1);
    $$ = strdup(number);
}
| expr PLUS_OP expr {
    sprintf(temp, "temp%d", ++tempCount);
    fprintf(ftac, "%s = %s + %s\n", temp, $1, $3);
    $$ = strdup(temp);
}
| expr MINUS_OP expr {
    sprintf(temp, "temp%d", ++tempCount);
    fprintf(ftac, "%s = %s - %s\n", temp, $1, $3);
    $$ = strdup(temp);
}
| expr MUL_OP expr {
    sprintf(temp, "temp%d", ++tempCount);
    fprintf(ftac, "%s = %s * %s\n", temp, $1, $3);
    $$ = strdup(temp);
}
| expr DIV_OP expr {
    sprintf(temp, "temp%d", ++tempCount);
    fprintf(ftac, "%s = %s / %s\n", temp, $1, $3);
    $$ = strdup(temp);
}
;

%%

int yywrap() {
    return 1;
}

int yyerror() {
    fprintf(stderr, "\n\nInvalid syntax!\n");
    return 0;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Please enter file name as second argument!\n");
        exit(0);
    }
```

```
    yyin = fopen(argv[1], "r");
    if (!yyin) {
        printf("File not found!\n");
        exit(0);
    }

    flex = fopen("output_lex.txt", "w");
    if (!flex) {
        printf("Output file lex error!\n");
        exit(0);
    }

    fsyn = fopen("output_syntax.txt", "w");
    if (!fsyn) {
        printf("Output file syn error!\n");
        exit(0);
    }

    ftac = fopen("output_tac.txt", "w");
    if (!ftac) {
        printf("Output file TAC error!\n");
        exit(0);
    }

    yyparse();
    if (!error) {
        fprintf(fsyn, "Valid syntax!\n");
    }
}
```

//ex7b.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror();
```

```
extern FILE* yyin;
extern FILE* flex;
FILE* fsyn;
FILE* fopttac;
extern int line;
int error = 0;
int tempCount = 0;
char temp[10];
%}

%union {
    char* string;
    int num;
};

%type <string> expr
%type <num> numexpr

%token <string> VOID INT
%token <string> LEFTB RIGHTB COMMA EOS
%token <string> ASSIGN_OP PLUS_OP MINUS_OP MUL_OP DIV_OP
%token <string> ID STRING
%token <num> CONSTANT

%left PLUS_OP MINUS_OP
%left MUL_OP DIV_OP

%start program

%%

program:
statement program {
    return 0;
}
| '\n'
|
;

statement:
declnstatement EOS
```

```
| assignment EOS
;

declnstatement:
type var
;

type:
INT
| VOID
;

var:
var COMMA var
| ID
| assignment
;

assignment:
ID ASSIGN_OP expr {
    fprintf(fopttac, "%s = %s\n", $1, $3);
}
| ID ASSIGN_OP numexpr {
    fprintf(fopttac, "%s = %d\n", $1, $3);
}
;

expr:
ID {
    $$ = strdup($1);
}
| expr PLUS_OP expr {
    if (strcmp($3, "0") && strcmp($1, "0")) {
        sprintf(temp, "temp%d", ++tempCount);
        fprintf(fopttac, "%s = %s + %s\n", temp, $1, $3);
        $$ = strdup(temp);
    }
    else {
        if (strcmp($1, "0"))
            $$ = strdup($1);
```

```
        else
            $$ = strdup($3);
    }
}
| expr MINUS_OP expr {
    if (strcmp($3, "0") && strcmp($1, "0")) {
        sprintf(temp, "temp%d", ++tempCount);
        fprintf(fopttac, "%s = %s - %s\n", temp, $1, $3);
        $$ = strdup(temp);
    }
    else {
        if (strcmp($1, "0"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
}
| expr MUL_OP expr {
    if (strcmp($3, "1") && strcmp($1, "1")) {
        sprintf(temp, "temp%d", ++tempCount);
        fprintf(fopttac, "%s = %s * %s\n", temp, $1, $3);
        $$ = strdup(temp);
    }
    else {
        if (strcmp($1, "1"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
}
| expr DIV_OP expr {
    if (strcmp($3, "1") && strcmp($1, "1")) {
        sprintf(temp, "temp%d", ++tempCount);
        fprintf(fopttac, "%s = %s / %s\n", temp, $1, $3);
        $$ = strdup(temp);
    }
    else {
        if (strcmp($1, "1"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
```

```
    }
}
| numexpr {
    char number[10];
    sprintf(number, "%d", $1);
    $$ = strdup(number);
}
;

numexpr:
CONSTANT {
    $$ = $1;
}
| numexpr PLUS_OP numexpr {
    $$ = $1 + $3;
}
| numexpr MINUS_OP numexpr {
    $$ = $1 - $3;
}
| numexpr MUL_OP numexpr {
    $$ = $1 * $3;
}
| numexpr DIV_OP numexpr {
    $$ = $1 / $3;
}
;

%%

int yywrap() {
    return 1;
}

int yyerror() {
    fprintf(stderr, "\n\nInvalid syntax!\n");
    return 0;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Please enter file name as second argument!\n");
```

```
        exit(0);
    }

    yyin = fopen(argv[1], "r");
    if (!yyin) {
        printf("File not found!\n");
        exit(0);
    }

    flex = fopen("output_lex.txt", "w");
    if (!flex) {
        printf("Output file lex error!\n");
        exit(0);
    }

    fsyn = fopen("output_syntax.txt", "w");
    if (!fsyn) {
        printf("Output file syn error!\n");
        exit(0);
    }

    fopttac = fopen("output_opttac.txt", "w");
    if (!fopttac) {
        printf("Output file TAC error!\n");
        exit(0);
    }

    yyparse();
    if (!error) {
        fprintf(fsyn, "Valid syntax!\n");
    }
}
```

//ex7c.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "string.h"
```

```
int yylex(void);
int yyerror();
extern FILE* yyin;
extern FILE* flex;
FILE* fsyn;
FILE* ftar;
extern int line;
int error = 0;
int regCount = 0;
char reg[10];
%}

%union {
    char* string;
    int num;
};

%type <string> expr
%type <num> numexpr

%token <string> VOID INT
%token <string> LEFTB RIGHTB COMMA EOS
%token <string> ASSIGN_OP PLUS_OP MINUS_OP MUL_OP DIV_OP
%token <string> ID STRING
%token <num> CONSTANT

%left PLUS_OP MINUS_OP
%left MUL_OP DIV_OP

%start program

%%

program:
statement program {
    return 0;
}
| '\n'
|
;
```

```
statement:
declnstatement EOS
| assignment EOS
;

declnstatement:
type var
;

type:
INT
| VOID
;

var:
var COMMA var
| ID
| assignment
;

assignment:
ID ASSIGN_OP expr {
    fprintf(ftar, "MOV %s, %s\n", $1, $3);
}
| ID ASSIGN_OP numexpr {
    fprintf(ftar, "MOV %s, %d\n", $1, $3);
}
;

expr:
ID {
    sprintf(reg, "R%d", ++regCount);
    fprintf(ftar, "MOV %s, %s\n", reg, $1);
    $$ = strdup(reg);
}
| expr PLUS_OP expr {
    if (strcmp($3, "0") && strcmp($1, "0")) {
        fprintf(ftar, "ADD %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
```

```
        if (strcmp($1, "0"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
}
| expr MINUS_OP expr {
    if (strcmp($3, "0") && strcmp($1, "0")) {
        fprintf(ftar, "SUB %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        if (strcmp($1, "0"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
}
| expr MUL_OP expr {
    if (strcmp($3, "1") && strcmp($1, "1")) {
        fprintf(ftar, "MUL %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        if (strcmp($1, "1"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
}
| expr DIV_OP expr {
    if (strcmp($3, "1") && strcmp($1, "1")) {
        fprintf(ftar, "DIV %s, %s\n", $1, $3);
        $$ = strdup($1);
    }
    else {
        if (strcmp($1, "1"))
            $$ = strdup($1);
        else
            $$ = strdup($3);
    }
```

```
}
| numexpr {
    char number[10];
    sprintf(number, "%d", $1);
    $$ = strdup(number);
}
;

numexpr:
CONSTANT {
    $$ = $1;
}
| numexpr PLUS_OP numexpr {
    $$ = $1 + $3;
}
| numexpr MINUS_OP numexpr {
    $$ = $1 - $3;
}
| numexpr MUL_OP numexpr {
    $$ = $1 * $3;
}
| numexpr DIV_OP numexpr {
    $$ = $1 / $3;
}
;

%%

int yywrap() {
    return 1;
}

int yyerror() {
    fprintf(stderr, "\n\nInvalid syntax!\n");
    return 0;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Enter file name as second argument\n");
        exit(0);
```

```
    }

    yyin = fopen(argv[1], "r");
    if (!yyin) {
        printf("File not found!\n");
        exit(0);
    }

    flex = fopen("output_lex.txt", "w");
    if (!flex) {
        printf("Output file lex error!\n");
        exit(0);
    }

    fsyn = fopen("output_syntax.txt", "w");
    if (!fsyn) {
        printf("Output file syn error!\n");
        exit(0);
    }

    ftar = fopen("output_target.txt", "w");
    if (!ftar) {
        printf("Output file target error!\n");
        exit(0);
    }

    yyparse();
    if (!error) {
        fprintf(fsyn, "\nValid syntax!\n");
    }

    return 0;
}
```

//compiler.sh

```
#!/bin/bash
echo "__COMPILER__"

lex ex7.l
```

```
yacc -d ex7a.y
gcc -o tac.out y.tab.c lex.yy.c
./tac.out $1

yacc -d ex7b.y
gcc -o opttac.out y.tab.c lex.yy.c
./opttac.out $1

yacc -d ex7c.y
gcc -o tar.out y.tab.c lex.yy.c
./tar.out $1

echo -e "\n\nLexical Analyser: "
cat output_lex.txt

echo -e "\n\nSyntax Analyser: "
cat output_syntax.txt

echo -e "\n\nTAC: "
cat output_tac.txt

echo -e "\n\nOptimised TAC: "
cat output_opttac.txt

echo -e "\n\nTarget code: "
cat output_target.txt

echo -e "\n\nSuccessful compilation!"
```

//sample.txt

```
int a = b + 2, b = 6 * 8;
c = a * 3 * 4 + 2;
d = a + b * c - d + 0;
```

Output:

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex7$ bash compiler.sh sample.txt
__COMPILER__
ex7a.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
ex7b.y: warning: 9 shift/reduce conflicts [-Wconflicts-sr]
ex7b.y: warning: 2 reduce/reduce conflicts [-Wconflicts-rr]
ex7c.y: warning: 9 shift/reduce conflicts [-Wconflicts-sr]
ex7c.y: warning: 2 reduce/reduce conflicts [-Wconflicts-rr]


Lexical Analyser:
                      int | type int
                        a | identifier
                        = | equal-to op
                        b | identifier
                        + | plus op
                        2 | numeric constant
                        , | sp char - comma
                        b | identifier
                        = | equal-to op
                        6 | numeric constant
                        * | mul op
                        8 | numeric constant
                        ; | sp char - semi-colon
                        c | identifier
                        = | equal-to op
                        a | identifier
                        * | mul op
                        3 | numeric constant
                        * | mul op
                        4 | numeric constant
                        + | plus op
                        2 | numeric constant
                        ; | sp char - semi-colon
                        d | identifier
                        = | equal-to op
                        a | identifier
                        + | plus op
                        b | identifier
                        * | mul op
                        c | identifier
                        - | minus op
                        d | identifier
                        + | plus op
                        0 | numeric constant
                        ; | sp char - semi-colon


Syntax Analyser:

Valid syntax!
```

```
TAC:
temp1 = b + 2
a = temp1
temp2 = 6 * 8
b = temp2
temp3 = a * 3
temp4 = temp3 * 4
temp5 = temp4 + 2
c = temp5
temp6 = b * c
temp7 = a + temp6
temp8 = temp7 - d
temp9 = temp8 + 0
d = temp9


Optimised TAC:
temp1 = b + 2
a = temp1
b = 48
temp2 = a * 14
c = temp2
temp3 = b * c
temp4 = a + temp3
temp5 = temp4 - d
d = temp5
```

```
Target code:
MOV R1, b
ADD R1, 2
MOV a, R1
MOV b, 48
MOV R2, a
MUL R2, 14
MOV c, R2
MOV R3, a
MOV R4, b
MOV R5, c
MUL R4, R5
ADD R3, R4
MOV R6, d
SUB R3, R6
MOV d, R3


Successful compilation!
kri@Krithika-PC-Win11:/mnt
```

UCS1602 - Compiler Design Lab **Name:** *Krithika Swaminathan*
AY: *2022-23* **Roll No.:** *205001057*

Learning outcomes:

- The internal working of a compiler was analysed and understood.
- The concept of tokens and parsing for tokens in Java was understood and implemented.
- A syntax analyser was implemented for a Java program using the lex and yacc tools.
- Intermediate code was generated for the given sample code using the lex and yacc tools.
- Intermediate code was optimised for the given sample code.
- Target was generated for the given sample code.
- A compiler was implemented for a given sample code to generate its corresponding target code.
