COMPILER DESIGN

ASSIGNMENT PRESENTATION

Toy Compiler for

– Binary to Decimal Converter

Contents – Program Files

Team:

19Z304 – Aditya Sriram

19Z313 – Dharun Bharathi S

19Z351 – Surtik S

19Z357 – T S Swaminathan

Lex file - Tokencreation.l for Syntax Analyzer

```
/*Declaration part*/
%{
int line_count = 0;
%}

/*Translation rules*/
%%
#.* {
        /* Header */
        return INCLUDE;
        //printf("\nHeader files\t\t: %s",yytext);
}

\/\/.*|\/"*"(.*|\n)*"*"\/ {
        /* Both Multi line and single line Comments are detected */
        //printf("\nComment lines\t\t: %s",yytext);
        return COMMENTS;
}

"main"          { return MAIN; }
"for"           { return FOR; }
"while"         { return WHILE; }
"if"            { return IF; }
"else"          { return ELSE; }
"else if"       { return ELSEIF; }
"int"           { return INT; }
"float"         { return FLOAT; }
"char"          { return CHAR; }
"double"        { return DOUBLE; }
"long"          { return LONG; }
"return"        { return RETURN; }
"break"         { return BREAK; }
"continue"      { return CONTINUE; }
"do"            { return DO; }
"goto"          { return GOTO; }
"void"          { return VOID; }
"true"          { return TRUE; }
"false"         { return FALSE; }
"<="            { return LE; }
">="            { return GE; }
"<"             { return LT; }
">"             { return GT; }
"=="            { return EQUAL; }
"="             { return ASSIGN; }
"!="            { return NOTEQ; }
"++"            { return UNARY; }
"--"            { return UNARY; }
"+"             { return ADD; }
"-"             { return SUB; }
"*"             { return MUL; }
"/"             { return DIV; }
"%"             { return MOD; }
[\t]            { return TAB; }

[a-zA-Z][a-zA-Z0-9_]*\((.|\\n)*\); {
        return INBUILT;
}

["].*["]        { return STR; }
['].[']         { return CHARACTER; }

[+-]?[0-9]*"."[0-9]+    { return FLOAT_NUM; }
[+-]?[0-9]+             { return INTEGER; }

[a-zA-Z_][a-zA-Z0-9_]* {
        /* Variables or identifiers */
        return ID;
}
```

```
69
70 [(){}|, ;] {
71         /* Separators */
72         return SEPARATOR;
73 }
74
75 [\n] {
76         /* To count lines */
77         line_count++;
78 }
79
80 . {
81         return *yytext;
82         /* Any other are considered not to be tokens */
83         //printf("\nOther\t\t: %s",yytext);
84 }
85 %%
86
87 int yywrap(){}
88
```

Yacc file for Syntax Analyzer:

b2dy.y:

```
 1 /* Declaration Section */
 2
 3 %{
 4 #include<stdio.h>
 5 #include<stdlib.h>
 6 int flag = 0;
 7 %}
 8
 9 %token MAIN INCLUDE FOR WHILE IF ELSE ELSEIF INT CHAR FLOAT DOUBLE LONG RETURN BREAK CONTINUE DO GOTO VOID
10 TRUE FALSE LE GE LT GT EQUAL ASSIGN UNARY ADD SUB MUL DIV STR CHARACTER FLOAT_NUM INTEGER SEPARATOR NOTEQ
11 ID INBUILT MOD TAB COMMENTS
12
13 /* Translation Rules */
14 %%
15
16 program: headers datatype SEPARATOR MAIN SEPARATOR SEPARATOR SEPARATOR body SEPARATOR{
17         system("clear");
18         printf("Program is syntactically correct!\n\n");
19 }
20 ;
21
22 headers: headers headers
23 | INCLUDE
24 ;
25
26 datatype: INT
27 | FLOAT
28 | CHAR
29 | VOID
30 | DOUBLE
31 | LONG
32 ;
33
34 body: tabs FOR SEPARATOR statement SEPARATOR condition SEPARATOR statement SEPARATOR SEPARATOR body SEPARATOR
35 | tabs IF SEPARATOR condition SEPARATOR SEPARATOR body tabs SEPARATOR else
36 | statement SEPARATOR
37 | statement separator SEPARATOR separator statement SEPARATOR
38 | body tabs body tabs
39 | tabs INBUILT
40 | tabs RETURN SEPARATOR INTEGER SEPARATOR
41 | tabs COMMENTS
42 ;
43

44 else: tabs ELSE SEPARATOR body tabs SEPARATOR
45 |
46 ;
47
48 condition: value relop value
49 | TRUE
50 | FALSE
51 ;
52
```

```
statement: tabs datatype separator ID init
| tabs datatype separator ID
| tabs ID init
| tabs ID ASSIGN expression
| tabs ID relop expression
| tabs ID relop ID
| tabs ID UNARY
| tabs UNARY ID
;

init: ASSIGN separator value
|
;

expression: expression arithmetic expression
| expression MOD expression
| value
;

arithmetic: ADD
| SUB
| MUL
| DIV
;

relop: LT
| GT
| LE
| GE
| EQUAL
| NOTEQ
;

value: INTEGER
| FLOAT_NUM
| CHARACTER
| ID
;

separator: SEPARATOR
|
;

tabs: tabs tabs
| TAB
|
;

%%

#include "lex.yy.c"

//driver code
int main()
{
        while(yyparse());
        if(flag == 0){
                // Entered number is valid
        }
}

void yyerror(char *s)
{
        printf("\n\nERROR OCCURRED!\n\n");
        exit(0);
        flag = 1;
}
```

Yacc file for Symbol Table Construction:

b2dy_st.y:

```
/* Declaration Section */

%{
#include<stdio.h>
#include<stdlib.h>

void add(char);
void insert_type();
int search(char *);
void insert_type();

struct symbolTable{
        char *id_name;
        char *data_type;
        char *type;
        int lno;
}symbol_table[50];

int count = 0;
int available;
char type[10];
extern int line_count;

%}

%token MAIN INCLUDE FOR WHILE IF ELSE ELSEIF INT CHAR FLOAT DOUBLE LONG RETURN BREAK
CONTINUE DO GOTO VOID TRUE FALSE LE GE LT GT EQUAL ASSIGN UNARY ADD SUB MUL DIV STR
CHARACTER FLOAT_NUM INTEGER SEPARATOR NOTEQ ID INBUILT MOD TAB COMMENTS

/* Translation Rules */
%%

program: headers datatype SEPARATOR MAIN SEPARATOR SEPARATOR SEPARATOR body SEPARATOR{
        system("clear");
        add('M');
        printf("The program inputted is syntactically correct!\n");
        printf("Continuing with symbol table creation...\n");
}
;

headers: headers headers
| INCLUDE { add('H'); }
;

datatype: INT { insert_type(); }
| FLOAT { insert_type(); }
| CHAR { insert_type(); }
| VOID { insert_type(); }
| DOUBLE { insert_type(); }
| LONG { insert_type(); }
;

body: tabs FOR { add('K'); } SEPARATOR statement SEPARATOR condition SEPARATOR statement SEPARATOR SEPARATOR body SEPARATOR
| tabs IF { add('K'); } SEPARATOR condition SEPARATOR SEPARATOR body tabs SEPARATOR else
| statement SEPARATOR
| statement separator SEPARATOR separator statement SEPARATOR
| body tabs body tabs
| tabs INBUILT { add('F'); }
| tabs RETURN { add('K'); } SEPARATOR INTEGER SEPARATOR
| tabs COMMENTS { add('X'); }
;

else: tabs ELSE { add('K'); } SEPARATOR body tabs SEPARATOR
|
;

condition: value relop value
| TRUE { add('K'); }
| FALSE { add('K'); }
;
```

```
statement: tabs datatype separator ID { add('V'); } init
| tabs datatype separator ID
| tabs ID init
| tabs ID ASSIGN expression
| tabs ID relop expression
| tabs ID relop ID
| tabs ID UNARY
| tabs UNARY ID
;

init: ASSIGN separator value
|
;

expression: expression arithmetic expression
| expression MOD expression
| value
;

arithmetic: ADD
| SUB
| MUL
| DIV
;

relop: LT
| GT
| LE
| GE
| EQUAL
| NOTEQ
;

value: INTEGER { add('C'); }
| FLOAT_NUM { add('C'); }
| CHARACTER
| ID
;

separator: SEPARATOR
|
;

tabs: tabs tabs
| TAB
|
;

%%

#include "lex.yy.c"

//driver code
int main()
{
        yyparse();
        printf("\n");
        printf("\t\t+----------------------------\n");
        printf("\t\t| SYMBOL TABLE CONSTRUCTION |\n");
        printf("\t\t----------------------------+\n\n");
        printf("_____________________________________________________________\n");
        printf("\nSYMBOL\t\tDATATYPE\t  TYPE\t\t  LINE NUMBER \n");
        printf("_____________________________________________________________\n\n");
        int i=0;
        for(i=0; i<count; i++) {
                printf("%s\t\t   %s\t\t%s\t\t%d\t\t\t\n", symbol_table[i].id_name, symbol_table[i].data_type, symbol_table[i].type, symbol_table[i].lno);
        }
        printf("\n\n");
}

int search(char *id) {
        int i;
        for(i=count-1; i>=0; i--) {
                if(strcmp(symbol_table[i].id_name, id)==0) {
                        return -1;
                        break;
                }
        }
        return 0;
}
```

```
153 void add(char c) {
154   available =search(yytext);
155   if(!available) {
156     if(c == 'H') {
157                 symbol_table[count].id_name=strdup("#inc");
158                 symbol_table[count].data_type=strdup("-");
159                 symbol_table[count].lno=line_count;
160                 symbol_table[count].type=strdup("Header\t");
161                 count++;
162         }
163         else if(c == 'K') {
164                 symbol_table[count].id_name=strdup(yytext);
165                 symbol_table[count].data_type=strdup("-");
166                 symbol_table[count].lno=line_count;
167                 symbol_table[count].type=strdup("Keyword\t");
168                 count++;
169         }
170         else if(c == 'V') {
171                 symbol_table[count].id_name=strdup(yytext);
172                 symbol_table[count].data_type=strdup(type);
173                 symbol_table[count].lno=line_count;
174                 symbol_table[count].type=strdup("Variable");
175                 count++;
176         }
177         else if(c == 'C') {
178                 symbol_table[count].id_name=strdup(yytext);
179                 symbol_table[count].data_type=strdup("-");
180                 symbol_table[count].lno=line_count;
181                 symbol_table[count].type=strdup("Constant");
182                 count++;
183         }
184         else if(c == 'F') {
185                 symbol_table[count].id_name=strdup("fun()");
186                 symbol_table[count].data_type=strdup(type);
187                 symbol_table[count].lno=line_count;
188                 symbol_table[count].type=strdup("Function");
189                 count++;
190         }

198          else if(c == 'X') {
199                   symbol_table[count].id_name=strdup("/* */");
200                   symbol_table[count].data_type=strdup("-");
201                   symbol_table[count].lno=line_count;
202                   symbol_table[count].type=strdup("Comments");
203                   count++;
204          }
205    }
206 }
207
208 void insert_type() {
209         strcpy(type, yytext);
210 }
211
212 void yyerror(char *s)
213 {
214         printf("\n\nERROR OCCURRED!\n\n");
215         exit(0);
216 }
217
```

Program files for Binary to Decimal Evaluation:

a) Lex file – bin.l

```
/* Declaration Part */

%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
%}


/* Translation rules */
%%
0 {
        yylval=0;
        return ZERO;
}

1 {
        yylval=1;
        return ONE;
}

\n return 0;

. return 0;

%%

int yywrap()
{
        return 1;
}

```

b) Yacc file – bintodec.y (for evaluation)

```
/* Declaration Section */

%{
#include<stdio.h>
#include<stdlib.h>
int flag = 0;
%}

%token ZERO
%token ONE

/* Translation Rules */
%%

N       : L {
                printf("\nDecimal Equivalent : %d\n\n", $$);
                }
L       : L B {
                $$=$1*2+$2;
                }
        | B {
                $$=$1;
                }

```

```
B       : ZERO {
                $$=$1;
                }
        | ONE {
                $$=$1;
                };

%%

//driver code
int main()
{
        system("clear");
        printf("\n\n");
        printf("\t\t\t+-------------------------------\n");
        printf("\t\t\t| Binary to Decimal Conversion |\n");
        printf("\t\t\t+-------------------------------+\n");
        printf("\n\nEnter a binary number to find the decimal equivalent: \n");
        while(yyparse());
        if(flag == 0){
                // Entered number is valid
        }
}

void yyerror(char *s)
{
        printf("The entered input is not a binary number! \n\n\n");
        exit(0);
        flag = 1;
}
```

c) Yacc file – tac.y (for 3 address code generation) (Only translation rules need to be changed)

```
/* Translation Rules */
%%

N       : L {
                printf("\nResult = t%d\n\n", $$);
                }

L       : L B {
                p++;
                printf("\nt%d = t%d*2 + t%d",p,$1,$2);
                $$ = p;
                }

        | B {
                p++;
                printf("\nt%d = t%d",p,$1);
                $$ = p;
                }

B       : ZERO {
                p++;
                printf("\nt%d = 0",p);
                $$=p;
                }
        | ONE {
                p++;
                printf("\nt%d = 1",p);
                $$=p;
                };

%%
```