

# **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

```
1 /*Declaration part*/
2 %{
3 int line_count = 0;
4 %}
5
6 /*Translation rules*/
7 %%
8 #. * {
9     /* Header */
10     return INCLUDE;
11     //printf("\nHeader files\t\t: %s",yytext);
12 }
13
14 \\/. *|\\/"*(.*)\\n)"*"\\ {
15     /* Both Multi line and single line Comments are detected */
16     //printf("\nComment lines\t\t: %s",yytext);
17     return COMMENTS;
18 }
19
20 "main"           { return MAIN; }
21 "for"            { return FOR; }
22 "while"          { return WHILE; }
23 "if"             { return IF; }
24 "else"           { return ELSE; }
25 "else if"        { return ELSEIF; }
26 "int"            { return INT; }
27 "float"          { return FLOAT; }
28 "char"           { return CHAR; }
29 "double"         { return DOUBLE; }
30 "long"           { return LONG; }
31 "return"         { return RETURN; }
32 "break"          { return BREAK; }
33 "continue"       { return CONTINUE; }
34 "do"             { return DO; }
35 "goto"           { return GOTO; }
36 "void"           { return VOID; }
37 "true"           { return TRUE; }
38 "false"          { return FALSE; }
39 "<="             { return LE; }
40 ">="             { return GE; }
41 "<"             { return LT; }
42 ">"             { return GT; }
43 "=="             { return EQUAL; }
44 "="              { return ASSIGN; }
45 "!="             { return NOTEQ; }
46 "++"             { return UNARY; }
47 "--"             { return UNARY; }
48 "+"              { return ADD; }
49 "-"              { return SUB; }
50 "*"              { return MUL; }
51 "/"              { return DIV; }
52 "%"              { return MOD; }
53 "\\t"            { return TAB; }
54
55 [a-zA-Z][a-zA-Z0-9_]*\\((\\.|\\n)*\\); {
56     return INBUILT;
57 }
58
59 ["].*["]         { return STR; }
60 ['].[']          { return CHARACTER; }
61
62 [+]?[0-9]*"."[0-9]+ { return FLOAT_NUM; }
63 [+]?[0-9]+        { return INTEGER; }
64
65 [a-zA-Z][a-zA-Z0-9_]* {
66     /* Variables or identifiers */
67     return ID;
68 }
```

```

69
70 [(O){}], ;] {
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

```

```

53 statement: tabs datatype separator ID init
54 | tabs datatype separator ID
55 | tabs ID init
56 | tabs ID ASSIGN expression
57 | tabs ID relop expression
58 | tabs ID relop ID
59 | tabs ID UNARY
60 | tabs UNARY ID
61 ;
62
63 init: ASSIGN separator value
64 |
65 ;
66
67 expression: expression arithmetic expression
68 | expression MOD expression
69 | value
70 ;
71
72 arithmetic: ADD
73 | SUB
74 | MUL
75 | DIV
76 ;
77
78 relop: LT
79 | GT
80 | LE
81 | GE
82 | EQUAL
83 | NOTEQ
84 ;
85
86 value: INTEGER
87 | FLOAT_NUM
88 | CHARACTER
89 | ID
90 ;
91
92 separator: SEPARATOR
93 |
94 ;

96 tabs: tabs tabs
97 | TAB
98 |
99 ;
100
101 %%
102
103 #include "lex.yy.c"
104
105 //driver code
106 int main()
107 {
108     while(yyparse());
109     if(flag == 0){
110         // Entered number is valid
111     }
112 }
113
114 void yyerror(char *s)
115 {
116     printf("\n\nERROR OCCURRED!\n\n");
117     exit(0);
118     flag = 1;
119 }
120

```

## Yacc file for Symbol Table Construction:

### b2dy\_st.y:

```
1 /* Declaration Section */
2
3 %{
4 #include<stdio.h>
5 #include<stdlib.h>
6
7 void add(char);
8 void insert_type();
9 int search(char *);
10 void insert_type();
11
12 struct symbolTable{
13     char *id_name;
14     char *data_type;
15     char *type;
16     int lno;
17 }symbol_table[50];
18
19 int count = 0;
20 int available;
21 char type[10];
22 extern int line_count;
23
24 %}
25
26 %token MAIN INCLUDE FOR WHILE IF ELSE ELSEIF INT CHAR FLOAT DOUBLE LONG RETURN BREAK
27 CONTINUE DO GOTO VOID TRUE FALSE LE GE LT GT EQUAL ASSIGN UNARY ADD SUB MUL DIV STR
28 CHARACTER FLOAT_NUM INTEGER SEPARATOR NOTEQ ID INBUILT MOD TAB COMMENTS
29
30 /* Translation Rules */
31 %%
32
33 program: headers datatype SEPARATOR MAIN SEPARATOR SEPARATOR SEPARATOR body SEPARATOR{
34     system("clear");
35     add('M');
36     printf("The program inputted is syntactically correct!\n");
37     printf("Continuing with symbol table creation...\n");
38 }
39 ;
40
41 headers: headers headers
42 | INCLUDE { add('H'); }
43 ;
44
45 datatype: INT { insert_type(); }
46 | FLOAT { insert_type(); }
47 | CHAR { insert_type(); }
48 | VOID { insert_type(); }
49 | DOUBLE { insert_type(); }
50 | LONG { insert_type(); }
51 ;
52
53 body: tabs FOR { add('K'); } SEPARATOR statement SEPARATOR condition SEPARATOR statement SEPARATOR SEPARATOR body SEPARATOR
54 | tabs IF { add('K'); } SEPARATOR condition SEPARATOR SEPARATOR body tabs SEPARATOR else
55 | statement SEPARATOR
56 | statement separator SEPARATOR separator statement SEPARATOR
57 | body tabs body tabs
58 | tabs INBUILT { add('F'); }
59 | tabs RETURN { add('K'); } SEPARATOR INTEGER SEPARATOR
60 | tabs COMMENTS { add('X'); }
61 ;
62
63 else: tabs ELSE { add('K'); } SEPARATOR body tabs SEPARATOR
64 |
65 ;
66
67 condition: value relop value
68 | TRUE { add('K'); }
69 | FALSE { add('K'); }
70 ;
71
```

```

72 statement: tabs datatype separator ID { add('V'); } init
73 | tabs datatype separator ID
74 | tabs ID init
75 | tabs ID ASSIGN expression
76 | tabs ID relop expression
77 | tabs ID relop ID
78 | tabs ID UNARY
79 | tabs UNARY ID
80 ;
81
82 init: ASSIGN separator value
83 |
84 ;
85
86 expression: expression arithmetic expression
87 | expression MOD expression
88 | value
89 ;
90
91 arithmetic: ADD
92 | SUB
93 | MUL
94 | DIV
95 ;
96
97 relop: LT
98 | GT
99 | LE
100 | GE
101 | EQUAL
102 | NOTEQ
103 ;
104
105 value: INTEGER { add('C'); }
106 | FLOAT_NUM { add('C'); }
107 | CHARACTER
108 | ID
109 ;
110
111 separator: SEPARATOR
112 |
113 ;
114
115 tabs: tabs tabs
116 | TAB
117 |
118 ;
119
120 %%
121
122 #include "lex.yy.c"
123
124 //driver code
125 int main()
126 {
127     yyparse();
128     printf("\n");
129     printf("\t\t+-----\n");
130     printf("\t\t| SYMBOL TABLE CONSTRUCTION |\n");
131     printf("\t\t+-----+\n\n");
132     printf("_____\n");
133     printf("\nSYMBOL\tDATATYPE\t TYPE\t\t LINE NUMBER \n");
134     printf("_____\n\n");
135     int i=0;
136     for(i=0; i<count; i++) {
137         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);
138     }
139     printf("\n\n");
140 }
141
142 int search(char *id) {
143     int i;
144     for(i=count-1; i>=0; i--) {
145         if(strcmp(symbol_table[i].id_name, id)==0) {
146             return -1;
147             break;
148         }
149     }
150     return 0;
151 }

```

```

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

```
1 /* Declaration Part */
2
3 %{
4 #include<stdio.h>
5 #include<stdlib.h>
6 #include"y.tab.h"
7 extern int yylval;
8 %}
9
10
11 /* Translation rules */
12 %%
13 0 {
14     yylval=0;
15     return ZERO;
16 }
17
18 1 {
19     yylval=1;
20     return ONE;
21 }
22
23 \n return 0;
24
25 . return 0;
26
27 %%
28
29 int yywrap()
30 {
31     return 1;
32 }
33
```

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

```
1 /* Declaration Section */
2
3 %{
4 #include<stdio.h>
5 #include<stdlib.h>
6 int flag = 0;
7 %}
8
9 %token ZERO
10 %token ONE
11
12 /* Translation Rules */
13 %%
14
15 N      : L {
16         printf("\nDecimal Equivalent : %d\n\n", $$);
17     }
18 L      : L B {
19         $$=$1*$2+$2;
20     }
21       | B {
22         $$=$1;
23     }
24
```



```

25 B      : ZERO {
26          $$=$1;
27          }
28      | ONE {
29          $$=$1;
30          };
31
32 %%
33
34 //driver code
35 int main()
36 {
37     system("clear");
38     printf("\n\n");
39     printf("\t\t\t+-----\n");
40     printf("\t\t\t| Binary to Decimal Conversion |\n");
41     printf("\t\t\t+-----\n");
42     printf("\n\nEnter a binary number to find the decimal equivalent: \n");
43     while(yyparse());
44     if(flag == 0){
45         // Entered number is valid
46     }
47 }
48
49 void yyerror(char *s)
50 {
51     printf("The entered input is not a binary number! \n\n\n");
52     exit(0);
53     flag = 1;
54 }
55

```

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

```

13 /* Translation Rules */
14 %%
15
16 N      : L {
17          printf("\nResult = %d\n\n", $$);
18          }
19
20 L      : L B {
21          p++;
22          printf("\nt%d = t%d*2 + t%d",p,$1,$2);
23          $$ = p;
24          }
25
26      | B {
27          p++;
28          printf("\nt%d = t%d",p,$1);
29          $$ = p;
30          }
31
32 B      : ZERO {
33          p++;
34          printf("\nt%d = 0",p);
35          $$=p;
36          }
37
38      | ONE {
39          p++;
40          printf("\nt%d = 1",p);
41          $$=p;
42          };
43 %%

```