

College of Engineering Trivandrum

## Compiler Design Lab



Abhishek Manoharan

S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

November 27, 2020



## Exp 3

### 1 YACC

#### 1.1 Aim

Generate YACC specification for a few syntactic categories.

#### 1.2 Theory

**YACC.**

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR(1) (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

#### 1.3 Algorithm

---

**Algorithm 1:** General Algorithm / Structure of YACC

---

```
1 /* definitions */
2 ....
3
4 %%
5 /* rules */
6 ....
7 %%
8
9 /* auxiliary routines */
10 ....
```

---

#### 1.4 Arithmetic Expression

##### 1.4.1 Question

Program to recognize a valid arithmetic expression that uses operator +, -, \*, and /.

##### 1.4.2 Algorithm

---

**Algorithm 2:** Algorithm to check valid arithmetic expression.

---

**Step1:** Start the program

**Step2:** Reading an expression

**Step3:** Checking the validating of the given expression according to the rule using yacc.

**Step4:** Using expression rule print the result of the given values

**Step5:** Stop the program

---

##### 1.4.3 Code

```
1 %{
2
3     #include<stdio.h>
4
5     int valid=1;
6
7 %}
8
```

```

9 %token num id op
10
11 %%
12
13 start : id '=' s ','
14
15 s :      id x
16      | num x
17      | '-' num x
18      | '(' s ')' x
19      ;
20
21 x :      op s
22      | '-' s
23      |
24      ;
25
26 %%
27
28 int yyerror()
29 {
30     valid=0;
31     printf("\nInvalid expression!\n");
32     return 0;
33 }
34
35 int main()
36 {
37     printf("\nEnter the expression:\n");
38     yyparse();
39     if(valid)
40     {
41         printf("\nValid expression!\n");
42     }
43 }

```

```

1 %{
2
3     #include "y.tab.h"
4
5 %}
6
7 %%
8
9 [a-zA-Z_][a-zA-Z_0-9]* return id;
10
11 [0-9]+(\.[0-9]*)?      return num;
12
13 [+/*]                  return op;
14
15 .                      return yytext[0];
16
17 \n                     return 0;
18
19 %%
20
21 int yywrap()

```

```
22
23 {
24
25 return 1;
26
27 }
```

#### 1.4.4 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a=b+c+d;

Valid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c+d=c/b

Invalid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c=d/e+h-i;

Invalid expression!
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a=b+c+d;

Valid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c+d=c/b

Invalid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c=d/e+h-i;

Invalid expression!
```

## 1.5 Identifier

### 1.5.1 Question

Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.

### 1.5.2 Algorithm

---

**Algorithm 3:** Algorithm to identify identifier

---

**Step1:** Start the program

**Step2:** Reading an expression

**Step3:** Checking the validating of the given expression according to the rule using yacc.

**Step4:** Using expression rule print the result of the given values

**Step5:** Stop the program

---

### 1.5.3 Code

```
1 %{
2
3     #include<stdio.h>
4
5     int valid=1;
6
7 %}
8
9 %token digit letter
10
11 %%
12 start : letter s
13 s :      letter s
14       | digit s
15       |
16       ;
17
18 %%
19
20 int yyerror()
21 {
22     printf("\nIts not a identifier!\n");
23     valid=0;
24     return 0;
25 }
26
27 int main()
28 {
29     printf("\nEnter a name to tested for identifier ");
30     yyparse();
31     if(valid)
32     {
33         printf("\nIt is a identifier!\n");
34     }
35 }
36
37 }
```

```
1 %{
2
```

```

3      #include "y.tab.h"
4
5  %}
6
7  %%
8
9  [a-zA-Z_][a-zA-Z_0-9]* return letter;
10
11 [0-9]                return digit;
12
13 .                    return yytext[0];
14
15 \n                    return 0;
16
17 %%
18
19 int yywrap()
20 {
21
22
23 return 1;
24
25 }

```

#### 1.5.4 Output

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ gcc lex.yy.c y.tab.c -w
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhishek

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi9

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier 9as

Its not a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi$

Its not a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ lex id.l
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ yacc -d id.y
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ gcc lex.yy.c y.tab.c -w
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

```

Enter a name to tested for identifier abhishek

It is a identifier!

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out
```

Enter a name to tested for identifier abhi9

It is a identifier!

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out
```

Enter a name to tested for identifier 9as

Its not a identifier!

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier\$ ./a.out

Enter a name to tested for identifier abhi\$

Its not a identifier!

## 1.6 Calculator

### 1.6.1 Question

Implementation of Calculator using LEX and YACC.

### 1.6.2 Algorithm

---

**Algorithm 4:** Algorithm for calculator

---

**Step1:** A Yacc source program has three parts as follows:

Declarations

**Step2:** Declarations Section: This section contains entries that:

i. Include standard I/O header file.

ii. Define global variables.

iii. Define the list rule as the place to start processing.

iv. Define the tokens used by the parser. v. Define the operators and their precedence.

**Step3:** Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

**Step4:** Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

**Step5:** Main- The required main program that calls the yyparse subroutine to start the program.

**Step6:** yyerror(s) -This error-handling subroutine only prints a syntax error message.

**Step7:** yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as program file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

**Step8:** calc.lex contains the rules to generate these tokens from the input stream.

---

### 1.6.3 Code

```
1 %{
2
3     #include<stdio.h>
4
5     int flag=0;
6
7
8
9 %}
10
11 %token NUMBER
12
13
14
15 %left '+' '-'
16
17 %left '*' '/' '%'
18
19 %left '(' ')'
20
21 %%
22
23 ArithmeticExpression: E{
24
25     printf("\nResult=%d\n",$$);
26
27     return 0;
```

```

28     };
29
30
31 E:E '+' E {$$=$1+$3;}
32
33 |E '-' E {$$=$1-$3;}
34
35 |E '*' E {$$=$1*$3;}
36
37 |E '/' E {$$=$1/$3;}
38
39 |E '%' E {$$=$1%$3;}
40
41 | '(' E ')' {$$=$2;}
42
43 | NUMBER {$$=$1;}
44
45 ;
46
47 %%
48
49
50
51 void main()
52 {
53
54     printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
55           Multiplication, Division, Modulus and Round brackets:\n");
56
57     yyparse();
58
59     if(flag==0)
60
61         printf("\nEntered arithmetic expression is Valid\n\n");
62
63
64
65 }
66
67 void yyerror()
68 {
69
70
71     printf("\nEntered arithmetic expression is Invalid\n\n");
72
73     flag=1;
74
75 }

1  %{
2
3  #include <stdio.h>
4
5  #include "y.tab.h"
6
7  extern int yylval;
8
9  %}
10
11
12
13 %%
14
15 [0-9]+ {
16
17     yylval=atoi(yytext);
18
19     return NUMBER;
20
21 }
22
23 [\t] ;
24
25 [\n] return 0;
26
27 . return yytext[0];

```



```

28
29 %%
30
31 int yywrap()
32 {
33
34
35 return 1;
36
37 }

```

#### 1.6.4 Output

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out
Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
))
Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out
Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(3)-)
Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out
Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(3+2)*(1+1)/(1+4)
Result=2
Entered arithmetic expression is Valid

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

```

```

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
))

```

```

Entered arithmetic expression is Invalid

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

```

```

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
(3)-)

```

```

Entered arithmetic expression is Invalid

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

```

```

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
(3+2)*(1+1)/(1+4)

```

```

Result=2

```

```

Entered arithmetic expression is Valid

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$

```

## 1.7 BNF to YACC

### 1.7.1 Question

Convert the BNF rules into YACC form and write code to generate abstract.

### 1.7.2 Algorithm

---

**Algorithm 5:** Algorithm for BNF to YACC

---

**Step1:** Reading an expression.

**Step2:** Calculate the value of given expression

**Step3:** Display the value of the nodes based on the precedence.

**Step4:** Using expression rule print the result of the given values

---

### 1.7.3 Code

```
1  %{
2
3  #include<string.h>
4
5  #include<stdio.h>
6
7  struct quad
8
9  {
10
11  char op[5];
12
13  char arg1[10];
14
15  char arg2[10];
16
17  char result[10];
18
19  }QUAD[30];
20
21  struct stack
22
23  {
24
25
26
27
28
29  int items[100];
30
31  int top;
32
33  }stk;
34
35  int Index=0,tIndex=0,StNo,Ind,tInd;
36
37  extern int LineNo;
38
39  %}
40
41  %union
42
43  {
44
45  char var[10];
46
47  }
48
49  %token <var> NUM VAR RELOP
50
51  %token MAIN IF ELSE WHILE TYPE
52
53  %type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
54
55  %left '-' '+'
56
57  %left '*' '/'
```

```

58
59 %%
60
61 PROGRAM : MAIN BLOCK
62
63 ;
64
65 BLOCK: '{' CODE '}'
66
67 ;
68
69 CODE: BLOCK
70
71 | STATEMENT CODE
72
73 | STATEMENT
74
75 ;
76
77 STATEMENT: DESCT ';'
78
79 | ASSIGNMENT ';'
80
81 | CONDST
82
83 | WHILEST
84
85 ;
86
87 DESCT: TYPE VARLIST
88
89 ;
90
91 VARLIST: VAR ',' VARLIST
92
93 | VAR
94
95 ;
96
97 ASSIGNMENT: VAR '=' EXPR{
98
99 strcpy(QUAD[Index].op, "=");
100
101 strcpy(QUAD[Index].arg1, $3);
102
103 strcpy(QUAD[Index].arg2, "");
104
105 strcpy(QUAD[Index].result, $1);
106
107 strcpy($$, QUAD[Index++].result);
108
109 }
110
111 ;
112
113 EXPR: EXPR '+' EXPR {AddQuadruple("+", $1, $3, $$);}
114
115 | EXPR '-' EXPR {AddQuadruple("-", $1, $3, $$);}
116
117 | EXPR '*' EXPR {AddQuadruple("*", $1, $3, $$);}
118
119 | EXPR '/' EXPR {AddQuadruple("/", $1, $3, $$);}
120
121 | '-' EXPR {AddQuadruple("UMIN", $2, "", $$);}
122
123 | '(' EXPR ')' {strcpy($$, $2);}
124
125 | VAR
126
127 | NUM
128
129
130
131
132
133 ;

```

```

134
135 CONST: IFST{
136
137 Ind=pop();
138
139 sprintf(QUAD[Ind].result,"%d",Index);
140
141 Ind=pop();
142
143 sprintf(QUAD[Ind].result,"%d",Index);
144
145 }
146
147 | IFST ELSEST
148
149 ;
150
151 IFST: IF '(' CONDITION ')' {
152
153 strcpy(QUAD[Index].op,"=");
154
155 strcpy(QUAD[Index].arg1,$3);
156
157 strcpy(QUAD[Index].arg2,"FALSE");
158
159 strcpy(QUAD[Index].result,"-1");
160
161 push(Index);
162
163 Index++;
164
165 }
166
167 BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");
168
169 strcpy(QUAD[Index].arg2,"");
170
171 strcpy(QUAD[Index].result,"-1");
172
173 push(Index);
174
175 Index++;
176
177 };
178
179 ELSEST: ELSE{
180
181 tInd=pop();
182
183 Ind=pop();
184
185 push(tInd);
186
187 sprintf(QUAD[Ind].result,"%d",Index);
188
189 }
190
191 BLOCK{
192
193 Ind=pop();
194
195 sprintf(QUAD[Ind].result,"%d",Index);
196
197 };
198
199 CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
200
201 StNo=Index-1;
202
203 }
204
205 | VAR
206
207 | NUM
208
209 ;

```

```

210
211 WHILEST: WHILELOOP{
212
213 Ind=pop();
214
215 sprintf(QUAD[Index].result,"%d",StNo);
216
217 Ind=pop();
218
219 sprintf(QUAD[Index].result,"%d",Index);
220
221 }
222
223 ;
224
225 WHILELOOP: WHILE '(' CONDITION ')' {
226
227 strcpy(QUAD[Index].op,"=");
228
229 strcpy(QUAD[Index].arg1,$3);
230
231 strcpy(QUAD[Index].arg2,"FALSE");
232
233
234
235
236
237 strcpy(QUAD[Index].result,"-1");
238
239 push(Index);
240
241 Index++;
242
243 }
244
245 BLOCK {
246
247 strcpy(QUAD[Index].op,"GOTO");
248
249 strcpy(QUAD[Index].arg1,"");
250
251 strcpy(QUAD[Index].arg2,"");
252
253 strcpy(QUAD[Index].result,"-1");
254
255 push(Index);
256
257 Index++;
258
259 }
260
261 ;
262
263 %%
264
265 extern FILE *yyin;
266
267 int main(int argc,char *argv[])
268
269 {
270
271 FILE *fp;
272
273 int i;
274
275 if(argc>1)
276
277 {
278
279 fp=fopen(argv[1],"r");
280
281 if(!fp)
282
283 {
284
285 printf("\n File not found");

```

```

286
287 exit(0);
288
289 }
290
291 yyin=fp;
292
293 }
294
295 yyparse();
296
297 printf("\n\n\t\t -----" "\n\t\t Pos Operator \tArg1 \tArg2 \tResult" "\n
\t\t-----");
298
299 for(i=0;i<Index;i++)
300 {
301
302 printf("\n\t\t %d\t %s\t %s\t %s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
303
304 }
305
306 printf("\n\t\t -----");
307
308 printf("\n\n"); return 0; }
309
310 void push(int data)
311 {
312 stk.top++;
313
314 if(stk.top==100)
315 {
316
317 {
318
319 printf("\n Stack overflow\n");
320
321 exit(0);
322
323 }
324
325 stk.items[stk.top]=data;
326
327 }
328
329 int pop()
330 {
331 {
332
333 int data;
334
335
336
337
338
339 if(stk.top== -1)
340 {
341
342
343 printf("\n Stack underflow\n");
344
345 exit(0);
346
347 }
348
349 data=stk.items[stk.top--];
350
351 return data;
352
353 }
354
355 void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
356 {
357
358
359 strcpy(QUAD[Index].op,op);
360

```

```

361 strcpy(QUAD[Index].arg1, arg1);
362
363 strcpy(QUAD[Index].arg2, arg2);
364
365 sprintf(QUAD[Index].result, "t%d", tIndex++);
366
367 strcpy(result, QUAD[Index++].result);
368
369 }
370
371 yyerror()
372 {
373
374
375 printf("\n Error on line no:%d", LineNo);
376
377 }

```

```

1  %{
2
3  #include "y.tab.h"
4
5  #include <stdio.h>
6
7  #include <string.h>
8
9  int LineNo=1;
10
11  %}
12
13  identifier [a-zA-Z][_a-zA-Z0-9]*
14
15  number [0-9]+|([0-9]*\.[0-9]+)
16
17  %%
18
19  main\(\) return MAIN;
20
21  if return IF;
22
23  else return ELSE;
24
25  while return WHILE;
26
27  int |
28
29  char |
30
31  float return TYPE;
32
33  {identifier} {strcpy(yylval.var, yytext);
34
35  return VAR;}
36
37  {number} {strcpy(yylval.var, yytext);
38
39  return NUM;}
40
41  \< |
42
43  \> |
44
45  \>= |
46
47  \<= |
48
49  == {strcpy(yylval.var, yytext);
50
51  return RELOP;}
52
53  [ \t] ;
54
55  \n LineNo++;
56
57  . return yytext[0];
58
59  %%

```

### 1.7.4 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/BNF$ ./a.out test.c
```

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	==	t0	FALSE	5
2	+	a	b	t1
3	=	t1		a
4	GOTO			5
5	<	a	b	t2
6	==	t2	FALSE	10
7	+	a	b	t3
8	=	t3		a
9	GOTO			5
10	<=	a	b	t4
11	==	t4	FALSE	15
12	-	a	b	t5
13	=	t5		c
14	GOTO			17
15	+	a	b	t6
16	=	t6		c

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/BNF$ ./a.out test.c
```

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	==	t0	FALSE	5
2	+	a	b	t1
3	=	t1		a
4	GOTO			5
5	<	a	b	t2
6	==	t2	FALSE	10
7	+	a	b	t3
8	=	t3		a
9	GOTO			5
10	<=	a	b	t4
11	==	t4	FALSE	15
12	-	a	b	t5
13	=	t5		c
14	GOTO			17
15	+	a	b	t6
16	=	t6		c

### 1.8 Result

Implemented the program for Intermediate code generation(3 Address code). It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.