College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

November 27, 2020

CS431 - Compiler Design Lab · 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

```
/* definitions */
 ....

%%
/* rules */
....
%%

/* auxiliary routines */
....
```

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

```
%{

    #include<stdio.h>

    int valid=1;

%}

```

```
%token num id op

%%

start : id '=' s ';'

s :     id x

      | num x

      | '-' num x

      | '(' s ')' x

      ;

x :     op s

      | '-' s

      |

      ;

%%

int yyerror()

{

    valid=0;

    printf("\nInvalid expression!\n");

    return 0;

}

int main()

{

    printf("\nEnter the expression:\n");

    yyparse();

    if(valid)

    {

        printf("\nValid expression!\n");

    }

}
```

```
%{

    #include "y.tab.h"

%}

%%

[a-zA-Z_][a-zA-Z_0-9]* return id;

[0-9]+(\.[0-9]*)?      return num;

[+/*]                  return op;

.                      return yytext[0];

\n                     return 0;

%%

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!
```

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

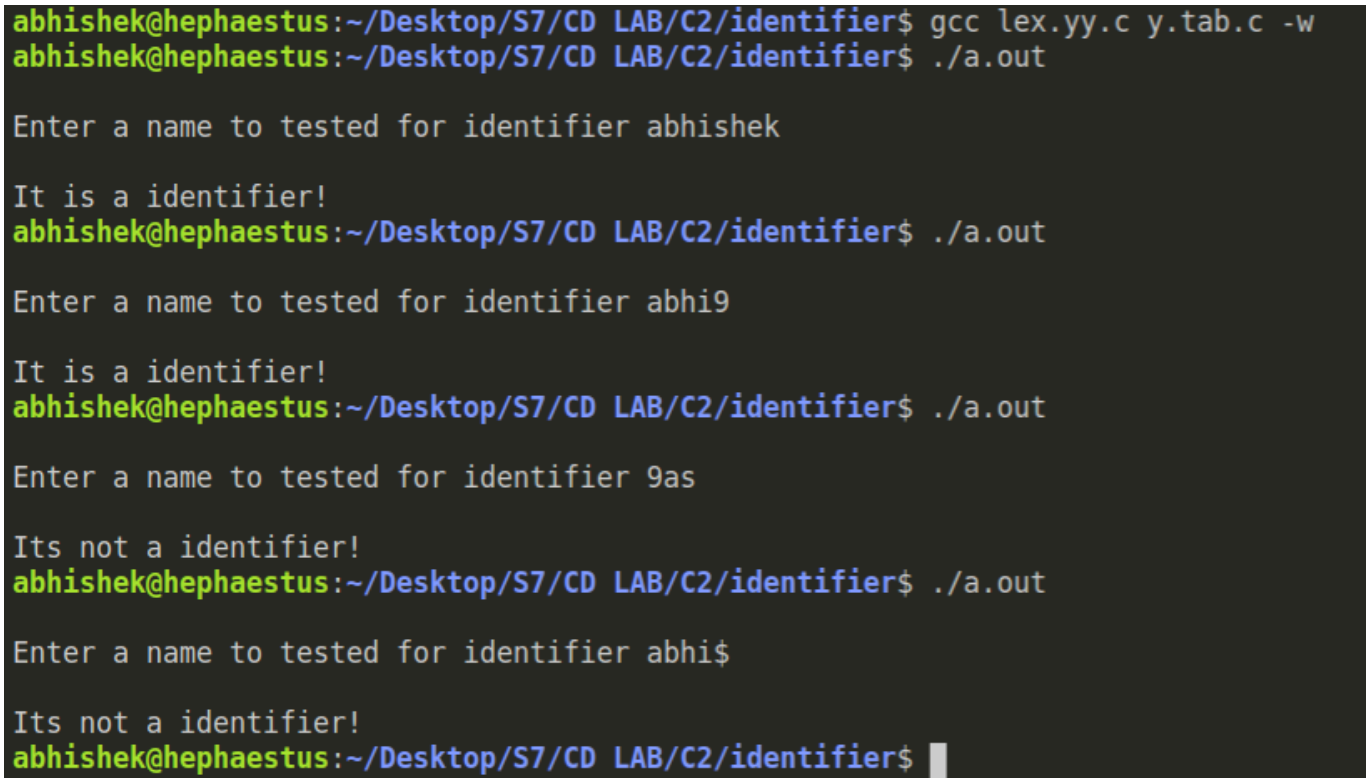
1.5.3 Code

```
%{

    #include<stdio.h>

    int valid=1;

%}

%token digit letter

%%

start : letter s

s :     letter s

      | digit s

      |

      ;

%%

int yyerror()

{

    printf("\nIts not a identifier!\n");

    valid=0;

    return 0;

}

int main()

{

    printf("\nEnter a name to tested for identifier ");

    yyparse();

    if(valid)

    {

        printf("\nIt is a identifier!\n");

    }

}
```

```
%{

```

```
    #include "y.tab.h"

%}

%%

[a-zA-Z_][a-zA-Z_0-9]* return letter;

[0-9]                       return digit;

.                       return yytext[0];

\n                      return 0;

%%

int yywrap()

{

return 1;

}
```

1.5.4 Output




```
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 programmar 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

```
%{

    #include<stdio.h>

    int flag=0;



%}

%token NUMBER



%left '+' '-'

%left '*' '/' '%'

%left '(' ')'

%%

ArithmeticExpression: E{

         printf("\nResult=%d\n",$$);

         return 0;
```

```
        };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%


void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
   yyparse();
  if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");


}
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

```
%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}


%%

[0-9]+ {

          yylval=atoi(yytext);

          return NUMBER;

       }

[\t] ;

[\n] return 0;

. return yytext[0];
```

```
%%

int yywrap()

{

return 1;

}
```

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

```
%{

#include<string.h>

#include<stdio.h>

struct quad

{

char op[5];

char arg1[10];

char arg2[10];

char result[10];

}QUAD[30];

struct stack

{









int items[100];

int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd;

extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '*' '/'
```

```
%%

PROGRAM : MAIN BLOCK

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"=");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,$1);

strcpy($$,QUAD[Index++].result);

}

;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}

| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM



;
```

```
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
```

```
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE'('CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");



strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
fp=fopen(argv[1],"r");
if(!fp)
{
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
303 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);
304
305 }
306
307 printf("\n\t\t ------------------------");
308
309 printf("\n\n"); return 0; }
310
311 void push(int data)
312
313 { stk.top++;
314
315 if(stk.top==100)
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
```

```
strcpy(QUAD[Index].arg1,arg1);

strcpy(QUAD[Index].arg2,arg2);

sprintf(QUAD[Index].result,"t%d",tIndex++);

strcpy(result,QUAD[Index++].result);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}
```

```
%{

#include"y.tab.h"

#include<stdio.h>

#include<string.h>

int LineNo=1;

%}

identifier [a-zA-Z][_a-zA-Z0-9]*

number [0-9]+|([0-9]*\.[0-9]+)

%%

main\(\) return MAIN;

if return IF;

else return ELSE;

while return WHILE;

int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext);

return VAR;}

{number} {strcpy(yylval.var,yytext);

return NUM;}

\< |

\> |

\>= |

\<= |

== {strcpy(yylval.var,yytext);

return RELOP;}

[ \t] ;

\n LineNo++;

. return yytext[0];

%%
```

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.