

/ C5: Use YACC to generate Syntax tree for a given expression.*/*

File: C5.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/*A structure inorder to store it's own value and left,right child indexes*/
struct tree_node
{
    char val[10];
    int lc;
    int rc;
};

int ind;
/*Nodes for syntax tree*/
struct tree_node syn_tree[100];
void my_print_tree(int cur_ind);
int mknode(int lc,int rc,char val[10]);

}%

/*declaring a digit named token.*/
%token digit

%%
/* print the tree after evaluating E */
S:E { my_print_tree($1); }
;
/*$$ is the current node's index after making the node*/
/* $n -> nth term's value in the expression */
E:E '+' T { $$= mknode($1,$3,"+"); ; }
|E '-' T { $$= mknode($1,$3,"-"); ; }
|T { $$=$1; }
;
T:T '*' F { $$= mknode($1,$3,"*"); ; }
|T '/' F { $$= mknode($1,$3,"/"); ; }
|F { $$=$1 ; }
;
F:P '^' F { $$= mknode($1,$3,"^"); ; }
| P { $$ = $1 ; }
;
P: '(' E ')' { $$=$2; }
```

```

|digit {char buf[10]; sprintf(buf,"%d", yylval); $$ = mknode(-1,-1,buf);}

%%

int main()
{
    ind=0;
    printf("Enter an expression\n");
    yyparse();
    return 0;
}

yyerror()
{
    printf("NITW Error\n");
}
/* mknode function to create new tree node with it's value as val and left and
right child node's indices as lc and rc respectively*/
int mknode(int lc,int rc,char val[10])
{
    strcpy(syn_tree[ind].val,val);
    syn_tree[ind].lc = lc;
    syn_tree[ind].rc = rc;
    ind++;
    return ind-1;
}
/*my_print_tree function to print the syntax tree in DLR fashion*/
void my_print_tree(int cur_ind)
{
    /*If it's null node,return*/
    if(cur_ind==-1) return;
    /*If, the node is a leaf node*/
    if(syn_tree[cur_ind].lc==-1&&syn_tree[cur_ind].rc==-1)
        printf("Digit Node -> Index : %d, Value : %s\n",cur_ind,syn_tree[cur_ind].val);

    else
        printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,
Right Child Index : %d \n",cur_ind,syn_tree[cur_ind].val, syn_tree[cur_ind].lc,
syn_tree[cur_ind].rc);

    /*calling left child of the current node*/
    my_print_tree(syn_tree[cur_ind].lc);
    /*calling right child of the current node*/
    my_print_tree(syn_tree[cur_ind].rc);
}

```

File: C5.1

```
%{
#include "y.tab.h"
extern int yylval;

}%

%%
/*When the token is a number,return it*/
[0-9]+ {yylval=atoi(yytext); return digit;}
/*When the token is space or tab,return nothing*/
[\t] ;
/*When the token is new line,return 0*/
[\n] return 0;
/*When the token is none of the above,return the first character*/
. return yytext[0];
%%
```

The screenshot shows a presentation slide with two parse trees and a terminal window running a C compiler's lexical analyzer.

Slide Content:

- Left Tree (Expression: $2+3*4$):** A binary tree where the root node is '+' (Index 4, Value +). Its left child is '2' (Index 0, Value 2) and its right child is '*' (Index 3, Value *). The '*' node has children '3' (Index 1, Value 3) and '4' (Index 2, Value 4).
- Right Tree (Expression: $2+3+(4*5)-6$):** A binary tree where the root node is '-' (Index 8, Value -). Its left child is '+' (Index 6, Value +) and its right child is '6' (Index 7, Value 6). The '+' node has children '2' (Index 0, Value 2) and '*' (Index 5, Value *). The '*' node has children '3' (Index 1, Value 3) and '4' (Index 3, Value 4). The '-' node also has a child '5' (Index 4, Value 5).

Terminal Window:

```
usnraju@usnraju-PC: ~/CompilerDesignPrograms/Set_C/C5
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$ yacc -d C5.y
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$ lex C5.1
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$ gcc y.tab.c lex.yy.c -o C5 -ll
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$ ./C5
Enter an expression
2+3*4
Operator Node -> Index : 4, Value : +, Left Child Index : 0, Right Child Index : 3
Digit Node -> Index : 0, Value : 2
Operator Node -> Index : 3, Value : *, Left Child Index : 1, Right Child Index : 2
Digit Node -> Index : 1, Value : 3
Digit Node -> Index : 2, Value : 4
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$ ./C5
Enter an expression
2+3+(4*5)-6
Operator Node -> Index : 8, Value : -, Left Child Index : 6, Right Child Index : 7
Operator Node -> Index : 6, Value : +, Left Child Index : 2, Right Child Index : 5
Operator Node -> Index : 5, Value : *, Left Child Index : 3, Right Child Index : 4
Digit Node -> Index : 3, Value : 4
Digit Node -> Index : 4, Value : 5
Digit Node -> Index : 7, Value : 6
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C5$
```