# /* C9: YACC to generate DAG for expression grammar. */

## *File: C9.h*

```
#include<string>

using namespace std;

// Node structure of a node of the DAG.
struct node
{
      int number;
      node *left, *right;
      bool printed;
      const char *value;
};
```

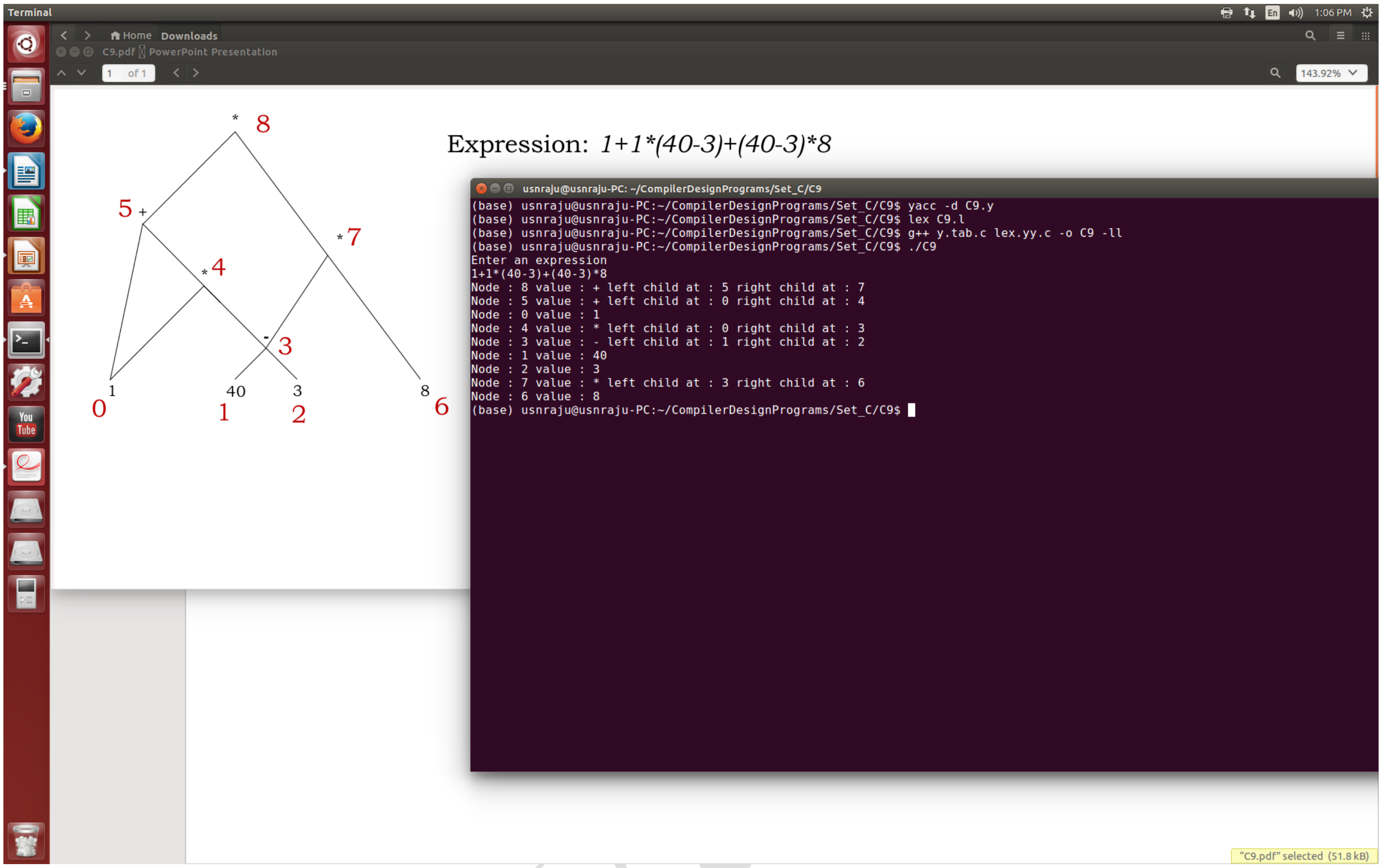
## *File: C9.y*

```
%{
#include<iostream>
#include<vector>
#include<string.h>
#include"C9.h"

using namespace std;

vector<node*> nodelist;
extern int yylex();

int node_count = 0;

void yyerror(const char *str)
{
      cerr<<"error : "<<str<<endl;
}

// This function creates a node with the value passed as input with
its left and right children as the nodes passed as parameters.
// After that the node is added into the nodelist vector.
node* make_node(node *left, const char *value, node *right)
{
      node *n = new node;
      int size = nodelist.size();
      for(int i = 0; i < size; ++i)
      {
            node *x = nodelist[i];
            if(strcmp(x->value, value) == 0 && x->left == left && x-
>right == right)
            {
                  return x;
            }
      }
      n->left = left;
      n->value = value;
      n->right = right;
      n->number = node_count++;
      n->printed = false;
      nodelist.push_back(n);
      return n;
}

// This function is used to print the DAG tree recursively.
void print_tree(node *n)
{
      if(!n || (n->printed))
      {
            return;
      }
      n->printed = true;
      cout<<"Node : "<<n->number<<" value : "<<n->value<<flush;
      if(n->left)
      {
            cout<<" left child at : "<<n->left->number<<flush;
      }
      if(n->right)
      {
            cout<<" right child at : "<<n->right->number<<flush;
      }
      cout<<endl;
      print_tree(n->left);
      print_tree(n->right);
}
%}

/* Start statet is S. */
%start S
%union
{
      char *text;
      node *n;
}

/* init the different tokens that will be used. */
%token <text> NUMBER
%token ADD SUB MUL DIV POW OPEN CLOSE
%type <n> S E T P F

%%
/* Start parsing the tree. And print at the end. */
S:
      E
      {
            print_tree($$);
      }
      ;

/* If an ADD or SUB are encountered, split it into two halves and
create the nodes of the DAG tree for the operator. */
/* Lowest precedence to ADD and SUB. */
/* Check for other operators with higher precedence in the expression
using T and for ADD or SUB again using E. */
/* If ADD or SUB isnt't there, going to operators with higher
precedence. */
E:
      E ADD T
      {
            $$ = make_node($1, "+", $3);
      }
      |
      E SUB T
      {
            $$ = make_node($1, "-", $3);
      }
      |
      T
      {
            $$ = $1;
      }
      ;

/* If an MUL or DIV are encountered, split it into two halves and
create the nodes of the DAG tree for the operator. */
/* Second lowest precedence to MUL and DIV. */
/* Check for other operators with higher precedence in the expression
using P and for MUL or DIV again using T. */
/* If MUL or DIV isnt't there, going to operators with higher
precedence. */
T:
      T MUL P
      {
            $$ = make_node($1, "*", $3);
      }
      |
      T DIV P
      {
            $$ = make_node($1, "/", $3);
      }
      |
      P
      {
            $$ = $1;
      }
      ;
```

```
/* If an POW is encountered, split it into two halves and create the
nodes of the DAG tree for the operator. */
/* Second highest precedence to POW */
/* Check for other operators with higher precedence in the expression
using F and for POW again using P. */
/* If POW isnt't there, going to operator with higher precedence. */
P:
    F POW P
    {
        $$ = make_node($1, "^", $3);
    }
    |
    F
    {
        $$ = $1;
    }
    ;

/* If an OPWN and CLOSE are encountered, recursively call state E for
parsing the expression inside the brackets. */
/* Highest precedence to OPEN and CLOSE. */
/* If OPEN and CLOSE isnt't there, make node for the number with no
children. */
F:
    OPEN E CLOSE
    {
        $$ = $2;
    }
    |
    NUMBER
    {
        $$ = make_node(NULL, $1, NULL);
    }
    ;
%%

int main()
{
    cout<<"Enter an expression\n";
      yyparse();
    return 0;
}
```

## *File: C9.l*

```
%{
#include<string.h>
#include"C9.h"
#include"y.tab.h"

using namespace std;
%}
/*
    Rules:
        If any number is matched, the number is sent as the token.
        If a '+' is matched, send ADD as token.
        If a '-' is matched, send SUB as token.
        If a '*' is matched, send MUL as token.
        If a '\' is matched, send DIV as token.
        If a '^' is matched, send POW as token.
        If a '(' is matched, send OPEN as token.
        If a ')' is matched, send CLOSE as token.
        If a space, tab or new line character is matched, end the
program.
*/
%%

[0-9]+ { yylval.text = strdup(yytext); return NUMBER; }
\+ { return ADD; }
\- { return SUB; }
\* { return MUL; }
\\ { return DIV; }
\^ { return POW; }
\( { return OPEN; }
\) { return CLOSE; }
[ \n\t] { return 0; }
```



Expression: *1+1*(40-3)+(40-3)*8*

usnraju@usnraju-PC: ~/CompilerDesignPrograms/Set_C/C9

```
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C9$ yacc -d C9.y
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C9$ lex C9.l
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C9$ g++ y.tab.c lex.yy.c -o C9 -ll
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C9$ ./C9
Enter an expression
1+1*(40-3)+(40-3)*8
Node : 8 value : + left child at : 5 right child at : 7
Node : 5 value : + left child at : 0 right child at : 4
Node : 0 value : 1
Node : 4 value : * left child at : 0 right child at : 3
Node : 3 value : - left child at : 1 right child at : 2
Node : 1 value : 40
Node : 2 value : 3
Node : 7 value : * left child at : 3 right child at : 6
Node : 6 value : 8
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C9$ 
```