/* C6: Use YACC to generate 3-Address code for a given expression */

File: C6.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>

int var_cnt=0;
char iden[20];
%}

%token digit
%token id

%%
/* Separating the LHS and RHS of the expression. */
S:id '=' E { printf("%s = t%d\n",iden, var_cnt-1); }

/* Following the operator precedence. */
/* '+','-' have least precendece. They have to be printed after all the others 3-
Address codes are printed. */
E:E '+' T   { $$=var_cnt; var_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 );
}
 |E '-' T    {  $$=var_cnt; var_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 );
}
 |T { $$=$1; }
 ;

/* '*','/' have second least precendece. They have to be printed before the 3-
Address codes of operators '+' and '-' are printed. */
T:T '*' F  { $$=var_cnt; var_cnt++; printf("t%d = t%d * t%d;\n", $$, $1, $3 );  }
 |T '/' F { $$=var_cnt; var_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 ); }
 |F {$$=$1 ; }
 ;

/* '^' has second precendece. These 3-Address code has to be printed after the 3-
Address codes of brackets are printed. */
F:P '^' F { $$=var_cnt; var_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}
 | P  { $$ = $1;}
;

/* Brackets have highest precendece. These 3-Address codes are to be printed
before all the others 3-Address codes are printed. */
/* This recursively calls the second rule in this set of rules for printing the
3-Address codes of the expression inside the brackets. */
P: '(' E ')' { $$=$2;  }
 |digit { $$=var_cnt; var_cnt++; printf("t%d = %d;\n",$$,$1); }
 ;
```

```
%%
int main()
{
 var_cnt=0;
 printf("Enter an expression : \n");
 yyparse();
 return 0;
}

yyerror()
{
      printf("NITW Error\n");
}
```

File: C6.l

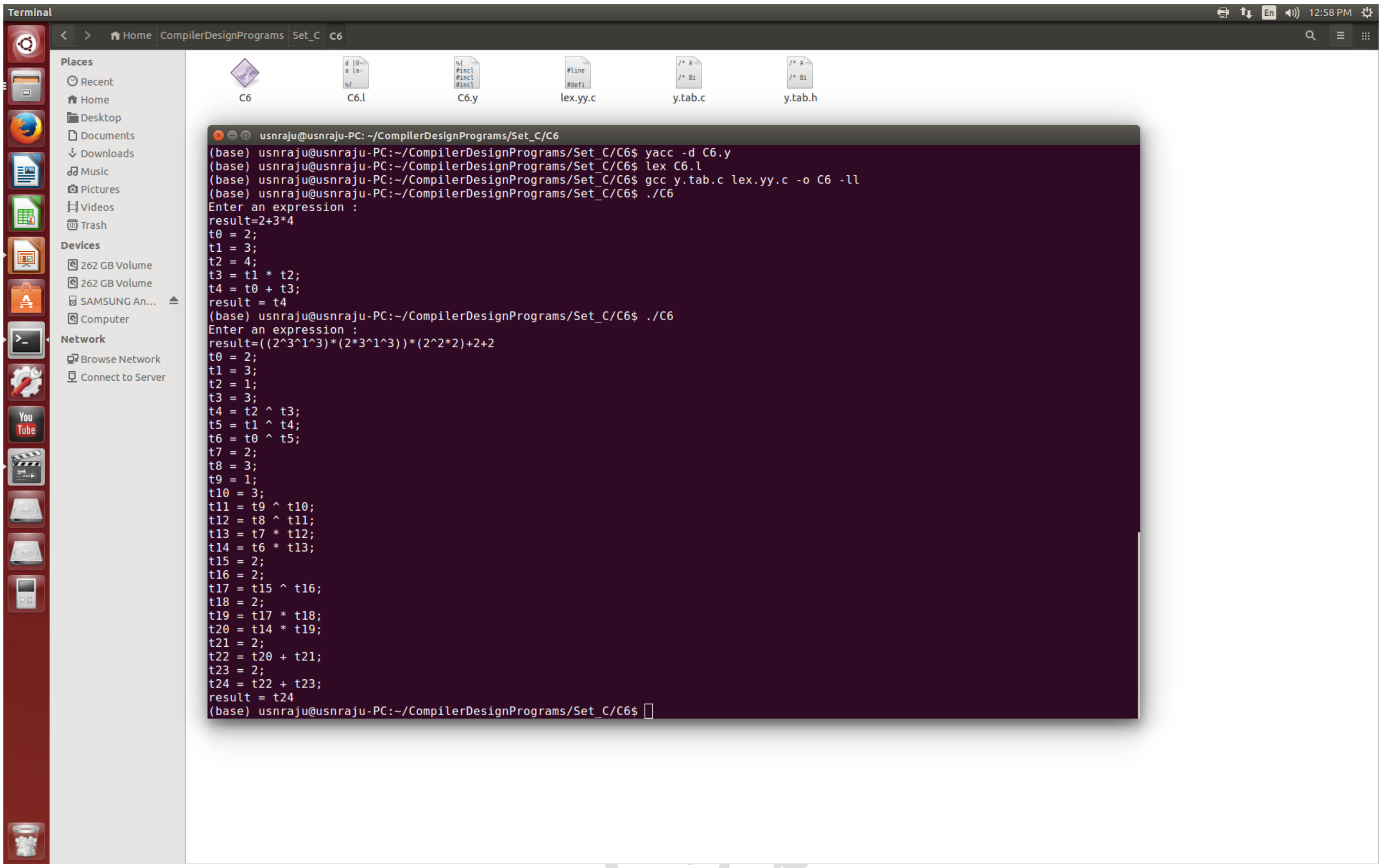
```
/* Definitions */
d [0-9]+
a [a-zA-Z]+

%{
/* Including the required header files. */
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
extern char iden[20];
%}

/*
    Rules:
        If any number is matched, make it as the yyval and send as token.
        If any word is matched, make it as the yylval and send as token.
        If any delimiter is matched, does nothing about it.
        If a new line character is encountered, end the program.
        If anything else is matched, send the first character of the matched
text.
*/

%%
{d}  {  yylval=atoi(yytext);  return digit;  }
{a}  {  strcpy(iden,yytext); yylval=1; return id; }

[ \t] {;}
\n return 0;
. return yytext[0];
%%
```



```
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ yacc -d C6.y
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ lex C6.l
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ gcc y.tab.c lex.yy.c -o C6 -ll
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ ./C6
Enter an expression :
result=2+3*4
t0 = 2;
t1 = 3;
t2 = 4;
t3 = t1 * t2;
t4 = t0 + t3;
result = t4
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ ./C6
Enter an expression :
result=((2^3^1^3)*(2*3^1^3))*(2^2*2)+2+2
t0 = 2;
t1 = 3;
t2 = 1;
t3 = 3;
t4 = t2 ^ t3;
t5 = t1 ^ t4;
t6 = t0 ^ t5;
t7 = 2;
t8 = 3;
t9 = 1;
t10 = 3;
t11 = t9 ^ t10;
t12 = t8 ^ t11;
t13 = t7 * t12;
t14 = t6 * t13;
t15 = 2;
t16 = 2;
t17 = t15 ^ t16;
t18 = 2;
t19 = t17 * t18;
t20 = t14 * t19;
t21 = 2;
t22 = t20 + t21;
t23 = 2;
t24 = t22 + t23;
result = t24
(base) usnraju@usnraju-PC:~/CompilerDesignPrograms/Set_C/C6$ 
```