**Operator Precedence Parse Program:**

#include<stdio.h>

#include<conio.h>

void main() {

  char stack[20], ip[20], opt[10][10][1], ter[10];

  int i, j, k, n, top = 0, col, row;

  clrscr();

  for (i = 0; i < 10; i++) {

    stack[i] = NULL;

    ip[i] = NULL;

    for (j = 0; j < 10; j++) {

      opt[i][j][1] = NULL;

    }

  }

  printf("Enter the no.of terminals :\n");

  scanf("%d", & n);

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

  scanf("%s", & ter);

  printf("\nEnter the table values :\n");

  for (i = 0; i < n; i++) {

    for (j = 0; j < n; j++) {

      printf("Enter the value for %c %c:", ter[i], ter[j]);

      scanf("%s", opt[i][j]);

    }

  }

  printf("\n\*\*\*\* OPERATOR PRECEDENCE TABLE \*\*\*\*\n");

  for (i = 0; i < n; i++) {

    printf("\t%c", ter[i]);

  }

  printf("\n");

  for (i = 0; i < n; i++) {

    printf("\n%c", ter[i]);

    for (j = 0; j < n; j++) {

      printf("\t%c", opt[i][j][0]);

    }

  }

  stack[top] = '$';

  printf("\nEnter the input string:");

  scanf("%s", ip);

  i = 0;

  printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

  printf("\n%s\t\t\t%s\t\t\t", stack, ip);

  while (i <= strlen(ip)) {

    for (k = 0; k < n; k++) {

      if (stack[top] == ter[k])

        col = k;

      if (ip[i] == ter[k])

        row = k;

    }

    if ((stack[top] == '$') && (ip[i] == '$')) {

      printf("String is accepted\n");

      break;

    } else if ((opt[col][row][0] == '<') || (opt[col][row][0] == '=')) {

      stack[++top] = opt[col][row][0];

      stack[++top] = ip[i];

      printf("Shift %c", ip[i]);

      i++;

    } else {

      if (opt[col][row][0] == '>') {

        while (stack[top] != '<') {

          --top;

        }

        top = top - 1;

        printf("Reduce");

      } else {

        printf("\nString is not accepted");

        break;

      }

    }

    printf("\n");

    for (k = 0; k <= top; k++) {

      printf("%c", stack[k]);

    }

    printf("\t\t\t");

    for (k = i; k < strlen(ip); k++) {

      printf("%c", ip[k]);

    }

    printf("\t\t\t");

  }

  getch();

}

/\*

**OutPut of Operator Precedence Parse Program in C-**

**Enter the value for \* \*:>  
Enter the value for \* $:>                                                         
Enter the value for $ i:<                                                         
Enter the value for $ +:<                                                         
Enter the value for $ \*:<                                                         
Enter the value for $ $:accept                                                    
                                                                                  
\*\*\*\* OPERATOR PRECEDENCE TABLE \*\*\*\*                                               
        i       +       \*       $                                                 
                                                                                  
i       e       >       >       >                                                 
+       <       >       <       >                                                 
\*       <       >       >       >                                                 
$       <       <       <       a  
\*/  
Enter the input string:                                                           
i\*i                                                                               
                                                                                  
STACK                   INPUT STRING                    ACTION                    
                                                                                  
$                       i\*i                     Shift i                           
$<i                     \*i                      Reduce                            
$                       \*i                      Shift \*                           
$<\*                     i                       Shift i                           
$<\*<i                                                                             
String is not accepted**

**Implementation of RECURSIVE DESCENT PARSER**

#include<stdio.h>  
#include<string.h>  
#include<ctype.h>  
   
char input[10];  
int i,error;  
void E();  
void T();  
void Eprime();  
void Tprime();  
void F();   
          main()  
          {  
i=0;  
error=0;  
                printf("Enter an arithmetic expression   :  "); // Eg: a+a\*a  
                gets(input);  
                E();  
                if(strlen(input)==i&&error==0)  
                        printf("\nAccepted..!!!\n");  
                else printf("\nRejected..!!!\n");  
                          }  
           
           
   
void E()  
{  
     T();  
     Eprime();  
}  
void Eprime()  
{  
     if(input[i]=='+')  
     {  
     i++;  
     T();  
     Eprime();  
     }  
     }  
void T()  
{  
     F();  
     Tprime();  
}  
void Tprime()  
{  
     if(input[i]=='\*')  
     {  
                      i++;  
                      F();  
                      Tprime();  
                      }  
                      }  
     void F()  
     {  
          if(isalnum(input[i]))i++;  
          else if(input[i]=='(')  
          {  
          i++;  
          E();  
          if(input[i]==')')  
          i++;  
  
          else error=1;  
            }  
          
          else error=1;  
          }  
             
  
Output:  
 a+(a\*a)  a+a\*a , (a), a , a+a+a\*a+a.... etc are  accepted  
++a, a\*\*\*a, +a, a\*, ((a . . . etc are rejected.

**Identifying all Tokens using LEX**

**Program:**

**%{**

**#include<stdio.h>**

**%}**

**%%**

**"if"|"else"|"while"|"do"|"switch"|"case" {printf("Keyword");}**

**[a-zA-Z][a-z|0-9]\* {printf("Identifier");}**

**[0-9]\* {printf("Number");}**

**"!"|"@"|"\*"|"&"|"^"|"%"|"$"|"#" {printf("Special Character");}**

**%%**

**int yywrap()**

**{**

**return 1;**

**}**

**main()**

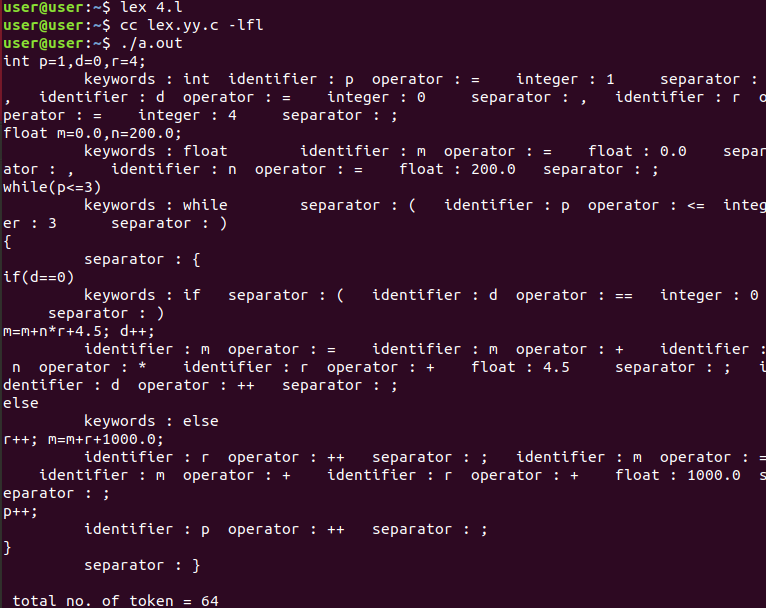
**{**

**printf("Enter a string of data\n");**

**yylex();**

**}**

**OUTPUT:**



**Implementation of Calculator using LEX**

% {

int op = 0,i;

float a, b;

% }

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

add "+"

sub "-"

mul "\*"

div "/"

pow "^"

ln \n

%%

/\* digi() is a user defined function \*/

{dig} {digi();}

{add} {op=1;}

{sub} {op=2;}

{mul} {op=3;}

{div} {op=4;}

{pow} {op=5;}

{ln} {printf("\n The Answer :%f\n\n",a);}

%%

digi()

{

if(op==0)

/\* atof() is used to convert

- the ASCII input to float \*/

a=atof(yytext);

else

{

b=atof(yytext);

switch(op)

{

case 1:a=a+b;

break;

case 2:a=a-b;

break;

case 3:a=a\*b;

break;

case 4:a=a/b;

break;

case 5:for(i=a;b>1;b--)

a=a\*i;

break;

}

op=0;

}

}

main(int argv,char \*argc[])

{

yylex();

}

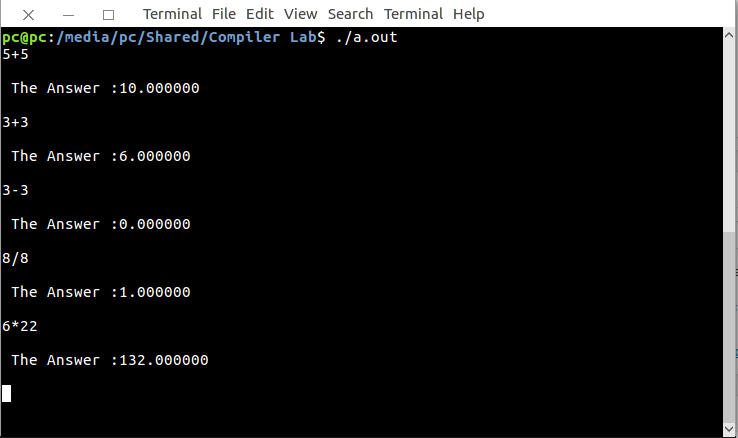
yywrap()

{

return 1;

}

**OUTPUT:**



//Implementation of Lexical Analyzer using Lex tool

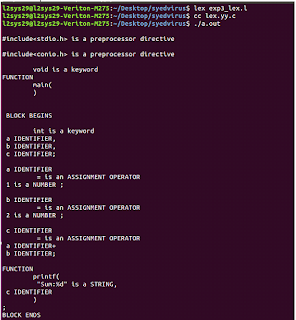
%{

int COMMENT=0;  
%}  
identifier [a-zA-Z][a-zA-Z0-9]\*  
%%  
#.\* {printf("\n%s is a preprocessor directive",yytext);}  
int |  
float |  
char |  
double |  
while |  
for |  
struct |  
typedef |  
do |  
if |  
break |  
continue |  
void |  
switch |  
return |  
else |  
goto {printf("\n\t%s is a keyword",yytext);}  
"/\*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}  
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}  
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}  
\}  {if(!COMMENT)printf("BLOCK ENDS ");}  
{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}  
\".\*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}  
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}  
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}  
\( ECHO;  
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}  
\<= |  
\>= |  
\< |  
== |  
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}  
%%

int main(int argc, char \*\*argv)  
{  
FILE \*file;  
file=fopen("var.c","r");  
if(!file)  
{  
printf("could not open the file");  
exit(0);  
}  
yyin=file;  
yylex();  
printf("\n");  
return(0);  
}  
int yywrap()  
{  
return(1);  
}

**INPUT:**  
//var.c  
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
int a,b,c;  
a=1;  
b=2;  
c=a+b;  
printf("Sum:%d",c);  
}

**OUTPUT:**



**Write a program to perform loop unrolling.**

// This program does not uses loop unrolling.

#include<stdio.h>

int main(void)

{

for (int i=0; i<5; i++)

printf("Hello\n"); //print hello 5 times

return 0;

}

**// This program uses loop unrolling.**

**#include<stdio.h>**

**int main(void)**

**{**

**// unrolled the for loop in program 1**

**printf("Hello\n");**

**printf("Hello\n");**

**printf("Hello\n");**

**printf("Hello\n");**

**printf("Hello\n");**

**return 0;**Hello

Hel

Write a program to perform constant propagation

#include<stdio.h>  
#include<string.h>  
#include<ctype.h>  
#include<conio.h>  
void input();  
void output();  
void change(int p,char \*res);  
void constant();  
struct expr  
{  
char op[2],op1[5],op2[5],res[5];  
int flag;  
}arr[10];  
int n;  
void main()  
{  
clrscr();  
input();  
constant();  
output();  
getch();  
}  
void input()  
{  
int i;  
printf("\n\nEnter the maximum number of  expressions : ");  
scanf("%d",&n);  
printf("\nEnter the input : \n");  
for(i=0;i<n;i++)  
{  
scanf("%s",arr[i].op);  
scanf("%s",arr[i].op1);  
scanf("%s",arr[i].op2);  
scanf("%s",arr[i].res);  
arr[i].flag=0;  
}  
}  
void constant()  
{  
int i;  
int op1,op2,res;  
char op,res1[5];  
for(i=0;i<n;i++)  
{  
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0) /\*if both digits, store them in variables\*/  
{  
op1=atoi(arr[i].op1);  
op2=atoi(arr[i].op2);  
op=arr[i].op[0];  
switch(op)  
{  
case '+':  
res=op1+op2;  
break;  
case '-':  
res=op1-op2;  
break;  
case '\*':  
res=op1\*op2;  
break;  
case '/':  
res=op1/op2;  
break;  
case '=':  
res=op1;  
break;  
}  
sprintf(res1,"%d",res);  
arr[i].flag=1; /\*eliminate expr and replace any operand below that uses result of this expr \*/  
change(i,res1);  
}  
}  
}  
void output()  
{  
int i=0;  
printf("\nOptimized code is : ");  
for(i=0;i<n;i++)  
{  
if(!arr[i].flag)  
{  
printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);  
}  
}  
}  
void change(int p,char \*res)  
{  
int i;  
for(i=p+1;i<n;i++)  
{  
if(strcmp(arr[p].res,arr[i].op1)==0)  
strcpy(arr[i].op1,res);  
else if(strcmp(arr[p].res,arr[i].op2)==0)  
strcpy(arr[i].op2,res);  
}  
}  
  
  
**INPUT:**  
  
Enter the maximum number of  expressions : 4  
  
Enter the input :  
= 3 - a  
+ a b t1  
+ a c t2  
+ t1 t2 t3  
  
**OUTPUT:**  
Optimized code is :  
+ 3 b t1  
+ 3 c t2  
+ t1 t2 t3

HeOU

**program to implement intermediate code generation for simple expression.**

#include<stdio.h>

#include<conio.h>

#include<string.h>

**int**i=1,j=0,no=0,tmpch=90;

**char**str[100],left[15],right[15];

**void**findopr();

**void**explore();

**void**fleft(**int**);

**void**fright(**int**);

**struct**exp

{

**int**pos;

**char**op;

}k[15];

**void**main()

{

printf("\t\tINTERMEDIATE CODE GENERATION\n\n");

printf("Enter the Expression :");

scanf("%s",str);

printf("The intermediate code:\n");

findopr();

explore();

}

**void**findopr()

{

**for**(i=0;str[i]!='\0';i++)

**if**(str[i]==':')

{

k[j].pos=i;

k[j++].op=':';

}

**for**(i=0;str[i]!='\0';i++)

**if**(str[i]=='/')

{

k[j].pos=i;

k[j++].op='/';

}

**for**(i=0;str[i]!='\0';i++)

**if**(str[i]=='\*')

{

k[j].pos=i;

k[j++].op='\*';

}

**for**(i=0;str[i]!='\0';i++)

**if**(str[i]=='+')

{

k[j].pos=i;

k[j++].op='+';

}

**for**(i=0;str[i]!='\0';i++)

**if**(str[i]=='-')

{

k[j].pos=i;

k[j++].op='-';

}

}

**void**explore()

{

i=1;

**while**(k[i].op!='\0')

{

fleft(k[i].pos);

fright(k[i].pos);

str[k[i].pos]=tmpch--;

printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);

printf("\n");

i++;

}

fright(-1);

**if**(no==0)

{

fleft(strlen(str));

printf("\t%s := %s",right,left);

getch();

exit(0);

}

printf("\t%s := %c",right,str[k[--i].pos]);

getch();

}

**void**fleft(**int**x)

{

**int**w=0,flag=0;

x--;

**while**(x!= -1 &&str[x]!= '+' &&str[x]!='\*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':')

{

**if**(str[x]!='$'&& flag==0)

{

left[w++]=str[x];

left[w]='\0';

str[x]='$';

flag=1;

}

x--;

}

}

**void**fright(**int**x)

{

**int**w=0,flag=0;

x++;

**while**(x!= -1 && str[x]!= '+'&&str[x]!='\*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')

{

**if**(str[x]!='$'&& flag==0)

{

right[w++]=str[x];

right[w]='\0';

str[x]='$';

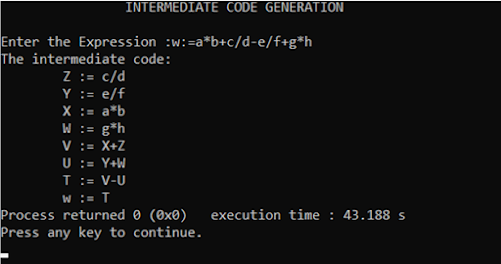
flag=1;

}

x++;

}

}

[](https://lh3.googleusercontent.com/-9nbeUxFSS8I/X-v3u0CTf-I/AAAAAAAAfhI/GY4vLcf4m8EjwtzfNGkZrAD1R7FBjGbwwCLcBGAsYHQ/image.png)