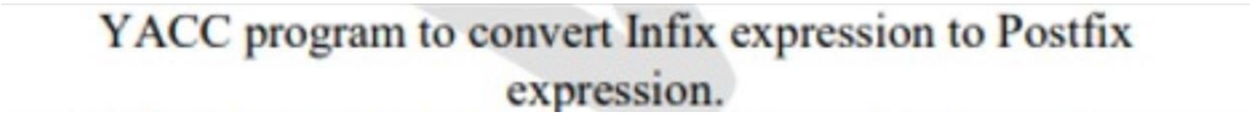
Compiler Design Lab Assignment - 6

Name : Apurba Koirala

Reg No.: 22BCE3799



YACC program to convert Infix expression to Postfix expression.

CODE:

Lexical Analyzer Source code:

intopo.l:

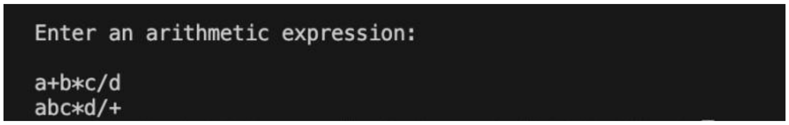
```
%{
#include <stdio.h>
#include   "y.tab.h"
extern int yylval;
%}
op "+"|"-"|"*"|"/"
%%
[a-z] { yylval=*yytext; return id; }
{op} { return (int) yytext[0]; }
\n { return(0); }
. { return err; }
%%
```

Parser Source code: intopo.y:

```
%{
#include <stdio.h>
#include <ctype.h> #define
YYSTYPE char
int f=0;
%}
%token id err
```

```
%left '-' '+'
%left '*' '/'
%%
input: /* empty string */
    | input exp {}
    | error {f=1;}
  ;
exp: exp '+' exp { printf("+"); }
| exp '-' exp { printf("-"); }
    | exp '*' exp { printf("*"); }
    | exp '/' exp { printf("/");}
    | id { printf("%c",yylval); }
  ;
%%
int main()
{
  printf("\nEnter an arithmetic expression:\n\n");
yyparse();    printf("\n");    if(f==1)
printf("Invalid Expression\n");    return 0;
}
int yywrap()
{
  return 1;
}
int yyerror(char *mes) {
return 0;
}
```

OUTPUT:



```
Enter an arithmetic expression:

a+b*c/d
abc*d/+
```

YACC program to generate 3-Address code for a given expression.

CODE:

Lexical Analyzer Source code: add3.l:

```
%{

#include "y.tab.h" extern

char yyval;

%}


%%


[0-9]+ { yylval.symbol = (char)(yytext[0]); return NUMBER; }

[a-z] { yylval.symbol = (char)(yytext[0]); return LETTER; }

. { return yytext[0]; }
```

```
\n { return 0; }

%%
```

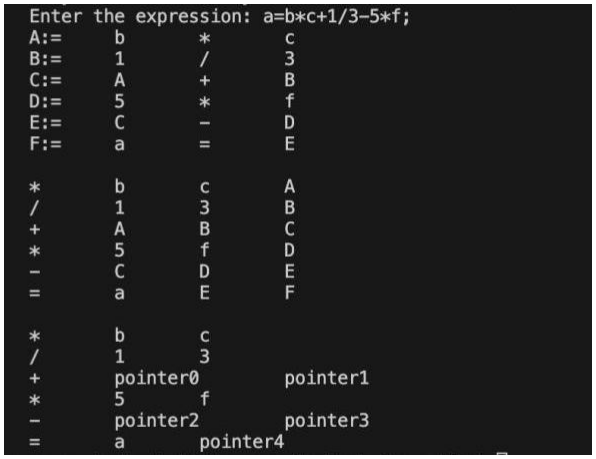
Parser Source code: add3.y:

```
%{
#include "y.tab.h"
#include <ctype.h> #include
<stdio.h> char addtotable(char,
char, char);

int index1 = 0; char
temp = 'A' - 1;

struct expr {
char operand1;
char operand2;
char operator;
char result;
};

%}

%union{
   char symbol;
}

%left '+' '-'
%left '/' '*'

%token <symbol> LETTER NUMBER
%type <symbol> exp

%%

statement: LETTER '=' exp ';' { addtotable((char)$1, (char)$3, '='); };
exp: exp '+' exp { $$ = addtotable((char)$1, (char)$3, '+'); }    | exp '-
' exp { $$ = addtotable((char)$1, (char)$3, '-'); }
   | exp '/' exp { $$ = addtotable((char)$1, (char)$3, '/'); }
   | exp '*' exp { $$ = addtotable((char)$1, (char)$3, '*'); }
   | '(' exp ')' { $$ = (char)$2; }
   | NUMBER { $$ = (char)$1; }
   | LETTER { $$ = (char)$1; };

%%

struct expr arr[20];

void yyerror(char *s) {
printf("Errror %s", s);
}
```

```
char addtotable(char a, char b, char o) {
temp++;     arr[index1].operand1 = a;
arr[index1].operand2 = b;
arr[index1].operator = o;
arr[index1].result = temp;     index1++;
return temp;
}

void threeAdd() {
   int i = 0;     char temp = 'A';
while (i < index1) {
printf("%c:=\t", arr[i].result);
printf("%c\t", arr[i].operand1);
printf("%c\t", arr[i].operator);
printf("%c\t", arr[i].operand2);
      i++;
      temp++;
printf("\n");
   }
}

void fouradd() {
   int i = 0;     char temp = 'A';
while (i < index1) {
printf("%c\t", arr[i].operator);
printf("%c\t", arr[i].operand1);

printf("%c\t", arr[i].operand2);

printf("%c", arr[i].result);        i++;

      temp++;

printf("\n");

   }

}


int find(char l) {

   int i;

   for (i = 0; i < index1; i++)      if

(arr[i].result    ==    l)    break;

return i;

}


void triple() {     int i = 0;     char temp = 'A';

while (i < index1) {        printf("%c\t",

arr[i].operator);        if

(!isupper(arr[i].operand1))

printf("%c\t", arr[i].operand1);        else {

printf("pointer");           printf("%d\t",

find(arr[i].operand1));

      }

      if (!isupper(arr[i].operand2))

printf("%c\t", arr[i].operand2);        else {
```

```
printf("pointer");        printf("%d\t",
find(arr[i].operand2));
    }
i++;
    temp++;
printf("\n");
  }
}

int yywrap() {
return 1;
}

int main() {      printf("Enter the
expression:  ");          yyparse();
threeAdd();             printf("\n");
fouradd();   printf("\n");   triple();
return 0;
}
```

OUTPUT:



```
Enter the expression: a=b*c+1/3-5*f;
A:=     b       *       c
B:=     1       /       3
C:=     A       +       B
D:=     5       *       f
E:=     C       -       D
F:=     a       =       E

*       b       c       A
/       1       3       B
+       A       B       C
*       5       f       D
-       C       D       E
=       a       E       F

*       b       c
/       1       3
+       pointer0        pointer1
*       5       f
-       pointer2        pointer3
=       a       pointer4
```

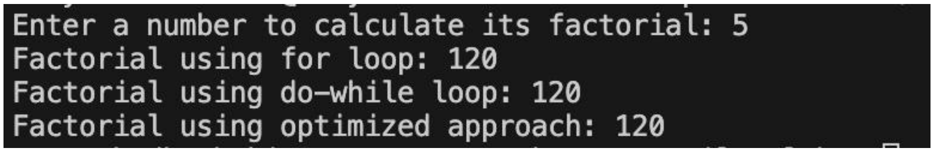
C Program for implementation of Code Optimization Technique.

CODE:

```
#include <stdio.h>


int factorial_for(int n) {

int fact = 1;   int

unused_variable = 0;
```

```
    for (int i = 1; i <= n; i++) {
fact *= i;
    }

    return fact;
}

int factorial_do_while(int n) {
int fact = 1, i = 1;

    do {
        fact *= i;
i++;
    } while (i <= n);

    return fact;
}

int optimized_factorial(int n) {
int fact = 1;

    for (int i = 1; i <= n; i++) {
fact = fact * i;
    }
    return fact;
}

int main() {
    int n;

    printf("Enter a number to calculate its factorial: ");
scanf("%d", &n);

    printf("Factorial using for loop: %d\n", factorial_for(n));     printf("Factorial
using do-while loop: %d\n", factorial_do_while(n));     printf("Factorial using
optimized approach: %d\n", optimized_factorial(n));

    return 0;
}
```

OUTPUT:



```
Enter a number to calculate its factorial: 5
Factorial using for loop: 120
Factorial using do-while loop: 120
Factorial using optimized approach: 120
```