NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

(An Institution of National Importance under MHRD, Govt. of India)

KARAIKAL – 609 609

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Roll Number: CS19B1009 **Name** : **ARUN KUMAR R**

Semester: II **Class:** B TECH

Subject Code: CS106 **Subject Name:** DATA STRUCTURES LABORATORY

1. Infix to postfix expression

Date:22/04/20

Aim:

To convert the given infix expression to postfix expression using c program.

Algorithm:

1. Start the program.
2. Declare all the variables and functions for stacks and driver functions.
3. Get the expression from the user.
4. Iterate over each character in the expression.
5. If the character is an alphanumerical, then add to the result.
6. If it is an operator, then push it into the stack, until the precedence of the top of stack is greater than the character.
7. Add the popped operators to the result.
8. After the above operation, add the remaining operators in the stack to the result.
9. Output the result.
10. End the program.

Program:

```
//Infix to postfix expression


#include <stdio.h>

#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>

#define MAX 100

//stack operations
struct stack
{
   char STACK[MAX];
   int top;
};

typedef struct stack stk;

int isempty(stk stack)
{
   if (stack.top == -1)
      return 1;
   return 0;
}
int print_array(char array[])
{
   for (int i = 0; i < strlen(array); i++)
   {
```

```
        printf("%c", array[i]);
    }
    printf("\n");
    return 1;
}
//getting Inputs
int getin(int flag, int *num, char *exp)
{
    //gets the option from the user
    if (flag == 0)
    {
        printf("Enter the option (help - 0):\n");
        scanf("%d", num);
        return 1;
    }
    else if (flag == 1) //gets expression from the user
    {
        printf("Enter the expression\n");
        scanf("%s", exp);
        return 1;
    }
    else
    {
        return 0; // programatic error
```

```
    }
}

int push(stk *stk, int value)
{
    stk->STACK[++(stk->top)] = value;
    return 1;
}
int pop(stk *stk)
{
    return stk->STACK[(stk->top)--];
}
int peek(stk *stk)
{
    return stk->STACK[stk->top];
}
int isoperator(char dig)
{
    if (dig == '+' || dig == '-' || dig == '/' || dig == '*')
    {
        return 1;
    }
    return 0;
}
```

```
//priority of operator
int op_priority(char op)
{
   if (op == '/')
      return 4;
   if (op == '*')
      return 3;
   if (op == '+')
      return 2;
   if (op == '-')
      return 1;
   if (op == '(')
      return 0;
   return -1;
}

//infix to postfix
char *infix_to_postfix(char *exp)
{
   stk stack;
   stack.top = -1;
   int current_ptr = 0;
   //int digit;
   for(int i = 0; exp[i] != '\0';i++)
```

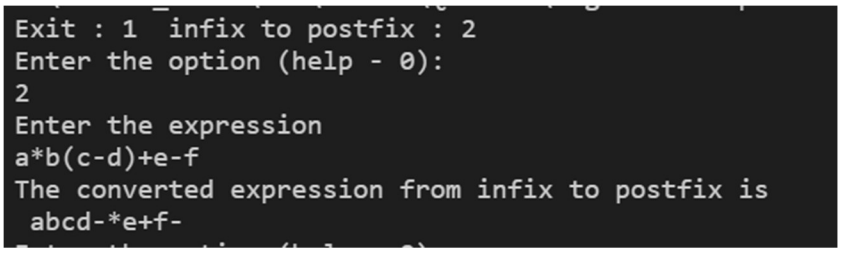
```
{
    if (isalnum(exp[i]))
    {
        exp[current_ptr++] = exp[i];
    }
    else if (exp[i] == '(')
    {
        push(&stack,exp[i]);
    }
    else if (exp[i] == ')')
    {
        while (!isempty(stack) && peek(&stack) != '(')
        {
            exp[current_ptr++] = pop(&stack);
        }
        if(!isempty(stack) && peek(&stack) != '(')
        {
            return NULL;
        }
        else
        {
            pop(&stack);
        }
```

```
        }
        else if (isoperator(exp[i]))
        {

            while (op_priority(exp[i]) < op_priority(peek(&stack)))
            {
                exp[current_ptr++] = pop(&stack);
            }
            push(&stack,exp[i]);

        }

    }

    while(!isempty(stack))
    {
        exp[current_ptr++] = pop(&stack);
    }
    exp[current_ptr] = '\0';
    return exp;
}

char *infix_to_postfix_driver(char *exp)
{
```

```
    char *result;
    getin(1, NULL, exp);
    result = infix_to_postfix(exp);
    if( result == NULL)
    {
        printf("Invalid Expression\n");
        return NULL;
    }
    printf("The converted expression from infix to postfix is\n %s \n", result);
    return result;
}
//help
int help()
{
    printf("Exit : 1  infix to postfix : 2\n");
}
//Driver function
int main(void)
{
    char input_exp[MAX];
    int options;
    help();
    while (1)
    {
```

```
        getin(0, &options, NULL);
        switch (options)
        {
        case 0:
           help();
           break;
        case 1:
           exit(0);
           break;
        case 2:
           infix_to_postfix_driver(input_exp);
           break;
        case 3:
           break;
        }
     }
     return 1;
}
```

Output:



Result:

The program was executed successfully.