

EXPERIMENT-1

Program:

To implement the c program which convert infix expression to postfix expression.

Theory:

Infix Notation: Infix notation is the standard way of writing mathematical expressions, where operators are placed between operands. For example, the infix expression $A + B * C$ indicates that you should multiply **B** and **C** first and then add the result to **A**.

Postfix Notation: Postfix notation, on the other hand, involves placing the operators after their operands. The same expression in postfix notation would be $A B C * +$. In postfix notation, the order of operations is clear, and parentheses are not needed.

Conversion Algorithm: The conversion of an infix expression to postfix can be done using a stack to keep track of operators. The algorithm typically involves scanning the infix expression from left to right and using a stack to manage operators.

1. Initialize an empty stack to store operators.
2. Start scanning the infix expression from left to right.
3. For each symbol encountered:
 - If it is an operand, output it directly.
 - If it is an operator:
 - Pop operators from the stack and output them until the stack is empty or the top operator has lower precedence.
 - Push the current operator onto the stack.
 - If it is an open parenthesis, push it onto the stack.
 - If it is a closing parenthesis:
 - Pop operators from the stack and output them until an open parenthesis is encountered.
 - Pop and discard the open parenthesis.
4. After scanning the entire infix expression, pop any remaining operators from the stack and output them.

Input:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

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

```

#include <string.h>

#define MAX 100

char stack[MAX];
int top = -1;

void push(char item)
{
    if (top >= MAX - 1)
    {
        printf("Stack Overflow\n");
        exit(1);
    } else
    {
        top++;
        stack[top] = item;
    }
}

char pop()
{
    char item;
    if (top < 0)
    {
        printf("Stack Underflow\n");
        exit(1);
    } else
    {
        item = stack[top];
        top--;
        return item;
    }
}

int is_operator(char symbol)
{
    if (symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-') {
        return 1;
    } else {
        return 0;
    }
}

int precedence(char symbol)
{
    if (symbol == '^')
    {
        return 3;
    }

```

```

    } else if (symbol == '*' || symbol == '/') {
        return 2;
    } else if (symbol == '+' || symbol == '-') {
        return 1;
    } else {
        return 0;    }}
void infix_to_postfix(char infix[], char postfix[])
{
    int i, j;
    char item;
    char x;
    push('(');
    strcat(infix, "");
    i = 0;
    j = 0;
    item = infix[i];
    while (item != '\0') {
        if (item == '(')    {
            push(item);    }
        else if (isdigit(item) || isalpha(item))    {
            postfix[j] = item;
            j++;    }
        else if (is_operator(item) == 1)
        {
            x = pop();
            while (is_operator(x) == 1 && precedence(x) >= precedence(item)) {
                postfix[j] = x;
                j++;
                x = pop();    }
            push(x);

```

```

        push(item);    }
    else if (item == ')') {
        x = pop();
        while (x != '(') {
            postfix[j] = x;
            j++;
            x = pop();    }    }
    else {
        printf("Invalid infix expression\n");
        exit(1);    }

    i++;

    item = infix[i];    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    gets(infix);
    infix_to_postfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}

```

Output:

```

Enter infix expression: a+b*c
Postfix expression: abc*+

...Program finished with exit code 0
Press ENTER to exit console.

```

EXPERIMENT-2

Program:

To implement a c program to copy the contents of one file into another file.

Theory:

Copying a file in C involves opening two file streams: one for the source file and another for the destination file. The source file is opened in read mode, and the destination file is opened in write mode. A loop is then used to read each character from the source file and write it into the destination file until the end of the source file is reached. Finally, both files are closed. This process effectively duplicates the contents of the source file into the destination file.

Input:

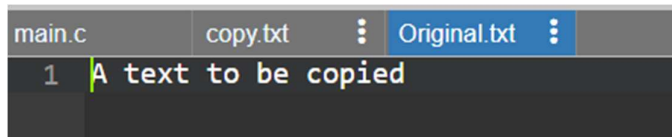
```
#include <stdio.h>

#include <stdlib.h>

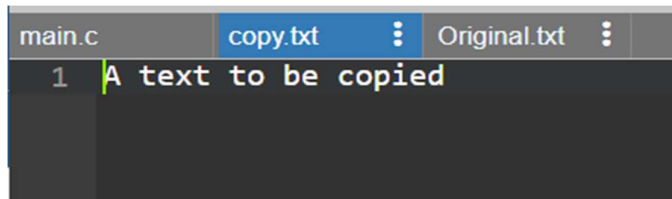
int main()
{
    char c;
    FILE *fptr1=NULL, *fptr2=NULL;
    fptr1=fopen("Original.txt","r");
    if(fptr1==NULL)
    {
        printf("Error");
        exit(1);
    }
    fptr2=fopen("copy.txt","w");
    if(fptr2==NULL)
    {
        printf("ERROR");
        exit(1);
    }
    while((c=fgetc(fptr1))!=EOF)
    {
        fputc(c,fptr2);
    }
}
```

```
}  
  
printf("Succesfully copied");  
  
fclose(fp1);  
  
fclose(fp2);  
  
}
```

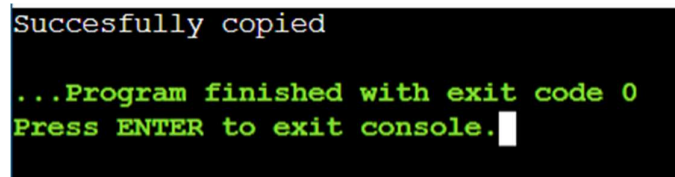
Output:



A screenshot of a code editor window. The top bar shows three tabs: 'main.c', 'copy.txt', and 'Original.txt'. The 'main.c' tab is active. The code in the editor shows line 1 with the text 'A text to be copied'.



A second screenshot of the same code editor window, showing the same content as the first screenshot.



A screenshot of a terminal window. The first line shows the output 'Succesfully copied'. The second line shows the message '...Program finished with exit code 0' in green. The third line shows the prompt 'Press ENTER to exit console.' with a cursor at the end.