

**b) By Stack Procedure:**

Create a class to implement a **Stack** providing appropriate operations in the class. Write a program that reads the above selected infix expression from a text file, **converts it into postfix notation and writes the postfix expression to another file**. Read the infix expression line by line and write the corresponding postfix expressions one by one to the output file. You are free to consult the algorithms available online to solve this problem. All consulted sources must be clearly mentioned/cited in your program.

Sample input file:	Sample output file:
<div>2+3*6 4+5-2 3*6</div>	<div>236*+ 45+2- 36*</div>

**SOLUTION:**

- **References:** All the information needed is delivered through the given slides by lecturer.
- **Note:** -*Comments* are also written in front of each statement for the ease of lecturer in understanding.

-Program has *modules* for each task as follows:

- 1) **M-1:** Getting data from user and saving to *Infix File*.
- 2) **M-2:** Read expressions from file and then converts into *Postfix Expression*.
- 3) **M-3:** Read data from *Infix File* and *Postfix File* and then displaying it.

-To enter data to *Infix File* program has a module in the beginning, which can be removed if we want to create a text file manually.

-All operations are carried out on mainly 2 files: *infix\_expression.txt* and *postfix\_expression.txt*.

## stack.h File:

```
1. #pragma once
2. class stack
3. {
4. private:
5.     char *arrStack;
6.     int top;
7.     int size;
8. public:
9.     stack(void);
10.    stack(int);
11.    bool isEmpty();
12.    bool isFull();
13.    void push(char);
14.    char pop();
15.    char _top();
16.    void display();
17. };
```

## stack.cpp File:

```
1. #include "stack.h"
2. #include <iostream>
3. using namespace std;
4.
5. stack::stack(void)
6. {
7.     size = 100;
8.     arrStack = new char [size];
9.     top = -1;
10. }
11.
12. stack::stack(int size)
13. {
14.     this->size = size;
15.     arrStack = new char [this->size];
16.     top = -1;
17. }
18.
19. bool stack::isEmpty()
20. {
21.     if(top == -1)
22.     {
23.         return true;
24.     }
25.     else
26.     {
27.         return false;
28.     }
29. }
30.
31. bool stack::isFull()
32. {
33.     if(top == (size-1))
```

```
34.     {
35.         return true;
36.     }
37.     else
38.     {
39.         return false;
40.     }
41. }
42.
43. void stack::push(char value)
44. {
45.     if(!isFull())
46.     {
47.         arrStack[++top] = value;
48.     }
49.     else
50.     {
51.         cout<<"Stack Overflow!!!"<<endl;
52.     }
53. }
54.
55. char stack::pop()
56. {
57.     if(!isEmpty())
58.     {
59.         return (arrStack[top--]);
60.     }
61.     else
62.     {
63.         cout<<"Stack Underflow!!!"<<endl;
64.     }
65. }
66.
67. char stack::_top()
68. {
69.     if(!isEmpty())
70.     {
71.         return arrStack[top];
72.     }
73.     else
74.     {
75.         cout<<"Stack Empty!!!";
76.     }
77. }
78.
79. void stack::display()
80. {
81.     for(int i=top; i>=0; i--)
82.     {
83.         cout<<arrStack[i];
84.     }
85. }
```

### main.cpp File:

```
1. #include "stack.h"
2. #include "string"
3. #include "string.h"
4. #include <fstream>
```

```

5. #include "conio.h"
6. #include <iostream>
7. using namespace std;
8.
9. void main()
10. {
11.     //=====Module to get Mathematical Expressions from user and then saving them to
        file=====
12.     int choice; //asks user if he wants to enter another expression and saves choice t
        o 'choice'
13.     string line_write; //string to get infix expression and then writing it to file
14.     ofstream infix_write ("infix_expression.txt"); //infix expression write file object
15.     do //loop to ask user if he wants to enter another expression
16.     {
17.         cout<<endl;
18.         cout<<"Enter Mathematical Expression: "<<endl;
19.         cin>>line_write;
20.         infix_write<<line_write; //writing string's whole expression to file
21.         cout<<endl;
22.         cout<<"Do you want to enter another expression?"<<endl;
23.         cout<<"1. YES."<<endl;
24.         cout<<"2. NO."<<endl;
25.         cin>>choice;
26.         if(choice == 1) //moves cursor to next line for new data entrance on choice 1(
            YES)
27.         {
28.             infix_write<<endl; //moves cursor to next line
29.         }
30.     }
31.     while(choice==1); //loop to ask user if he wants to enter another expression
32.     infix_write.close(); //closing infix write file
33.
34.
35.     //=====Module to read Infix Exp. from Infix File and converting it to Postfix E
        xp. and then saving it to Postfix File=====
36.     string line_read; //string to read infix expression from file
37.     ofstream postfix_write ("postfix_expression.txt");
38.     ifstream infix_read ("infix_expression.txt"); //infix read file object
39.     do
40.     {
41.         string postfix; //postfix string, declared inside so that on each iteration of new
            postfix it will be empty
42.         infix_read>>line_read; //reading string from infix file
43.         stack op( line_read.length() ); //stack to save operators of infix expression
44.         for(int i=0; i<line_read.length(); i++)
45.         {
46.             if(isdigit(line_read[i]))
47.             {
48.                 postfix += line_read[i]; //appends the operand to postfix string
49.             }
50.             else
51.             {
52.                 int precedence_stack ; //precedence of operator on top of stack
53.                 int precedence_infix; //precedence of operator in infix expression
54.                 if(!op.isEmpty())
55.                 {
56.                     switch(op._top()) //switch to track operator precedence of operator at
                        top of stack
57.                     {
58.                         case '+':

```

```

59.         {
60.             precedence_stack = 1;
61.             break;
62.         }
63.         case '-':
64.         {
65.             precedence_stack = 1;
66.             break;
67.         }
68.         case '*':
69.         {
70.             precedence_stack = 2;
71.             break;
72.         }
73.         case '/':
74.         {
75.             precedence_stack = 2;
76.             break;
77.         }
78.     }
79.     switch(line_read[i]) //switch to track operator precedence of operator
in infix expression
80.     {
81.         case '+':
82.         {
83.             precedence_infix = 1;
84.             break;
85.         }
86.         case '-':
87.         {
88.             precedence_infix = 1;
89.             break;
90.         }
91.         case '*':
92.         {
93.             precedence_infix = 2;
94.             break;
95.         }
96.         case '/':
97.         {
98.             precedence_infix = 2;
99.             break;
100.        }
101.    }
102.    if( precedence_infix > precedence_stack) //case if infix operator
r has high precedence than stack operator
103.    {
104.        op.push(line_read[i]);
105.    }
106.    else //case if infix operator has low/equal precedence than stack
operator
107.    {
108.        do //loop to pop operators in stack having precedence greater-
than/equal-to infix expression operator
109.        {
110.            postfix += op.pop(); //appends the char to postfix string
if its precedence is greater-than/equal-to infix expression operator
111.            if(!op.isEmpty()) //condition to check if stack is empty
112.            {
113.                switch(op._top())

```

```

114.         {
115.             case '+':
116.             {
117.                 precedence_stack = 1;
118.                 break;
119.             }
120.             case '-':
121.             {
122.                 precedence_stack = 1;
123.                 break;
124.             }
125.             case '*':
126.             {
127.                 precedence_stack = 2;
128.                 break;
129.             }
130.             case '/':
131.             {
132.                 precedence_stack = 2;
133.                 break;
134.             }
135.         }
136.     }
137. }
138. while( !op.isEmpty() && precedence_stack >= precedence_infix
); //checks stack empty or not and precedence of stack operator >= precedence of infix
operator
139.     op.push(line_read[i]); //when no more operator of higher pre
cedence in stack than operator in infix expression OR stack becomes empty it pushes the
infix operator to stack
140.     }
141. }
142. else //if stack is empty
143. {
144.     op.push(line_read[i]); //push infix operator to stack
145. }
146. }
147. }
148. while(!op.isEmpty()) //at the end pop all the operators in stack and append
them to postfix string
149. {
150.     postfix += op.pop();
151. }
152. postfix_write<<postfix; //saves postfix expression to postfix file
153. postfix_write<<endl; //moves pointer to next line in postfix file
154. }
155. while(!infix_read.eof());
156. infix_read.close();
157. postfix_write.close();
158.
159.
160. //=====Module to read and display Infix Expressions and Postfix Expressi
ons=====
161. ifstream infix_display ("infix_expression.txt");
162. ifstream postfix_display ("postfix_expression.txt");
163. cout<<endl;
164. cout<<"Postfix Expressions <--> Infix Expression"<<endl;
165. cout<<"===== "<<endl;
166. do
167. {
168.     string infix; //string to store infix expression

```

```

169.         string postfix; //string to store postfix expression
170.         infix_display>>infix; //reading infix expression from infix file
171.         postfix_display>>postfix; //reading postfix expression from postfix file

172.         cout<<" "<<infix<<" = "<<postfix<<endl; //displaying values
173.     }
174.     while( !infix_display.eof() && !postfix_display.eof() ); //read expression
    s untill end of Infix and Postfix Files
175.     infix_display.close(); //closing infix file
176.     postfix_display.close(); //closing postfix file
177.
178.     getch();
179. }

```

## DISPLAY CONSOLE OUTPUT:

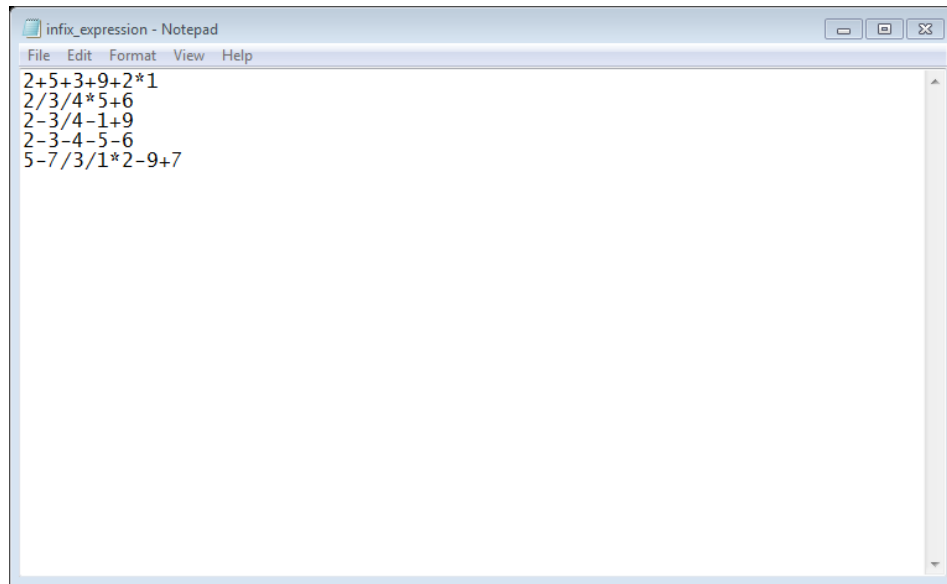
```

c:\users\administrator\documents\visual studio 2010\Projects\Assignment1\Debug\Assignment1.exe
Enter Mathematical Expression:
2+5+3+9+2*1
Do you want to enter another expression?
1. YES.
2. NO.
1
Enter Mathematical Expression:
2/3/4*5+6
Do you want to enter another expression?
1. YES.
2. NO.
1
Enter Mathematical Expression:
2-3/4-1+9
Do you want to enter another expression?
1. YES.
2. NO.
1
Enter Mathematical Expression:
2-3-4-5-6
Do you want to enter another expression?
1. YES.
2. NO.
1
Enter Mathematical Expression:
5-7/3/1*2-9+7
Do you want to enter another expression?
1. YES.
2. NO.
2
Postfix Expressions <--> Infix Expression
=====
2+5+3+9+2*1 = 25+3+9+21*+
2/3/4*5+6 = 23/4/5*6+
2-3/4-1+9 = 234/-1-9+
2-3-4-5-6 = 23-4-5-6-
5-7/3/1*2-9+7 = 573/1/2*-9-7+

```

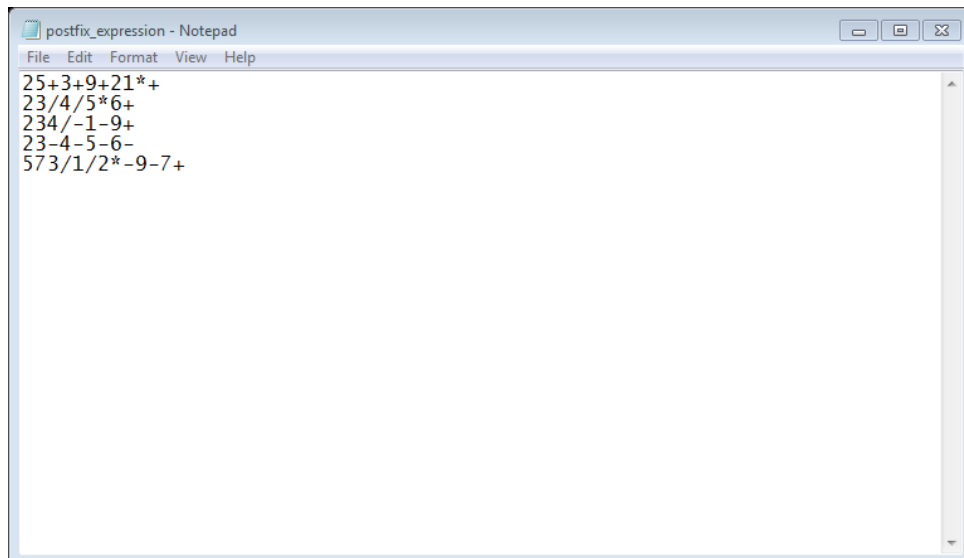
## OUTPUT TEXT FILES:

### infix\_expression.txt File:



```
infix_expression - Notepad
File Edit Format View Help
2+5+3+9+2*1
2/3/4*5+6
2-3/4-1+9
2-3-4-5-6
5-7/3/1*2-9+7
```

### postfix\_expression.txt File:



```
postfix_expression - Notepad
File Edit Format View Help
25+3+9+21*+
23/4/5*6+
234/-1-9+
23-4-5-6-
573/1/2*-9-7+
```

T H E - E N D