# Experiment No.02

## PART A

(PART A: TO BE REFFERED BY STUDENTS)

* 1. **Aim:**

Implementation of stacks and its operations using arrays.

Implementation of sorting in stacks using recursion

###### Prerequisite: - Knowledge of any programming language

* 1. **Outcome:**

After successful completion of this experiment, students will be able to

* + 1. Explain fundamental concepts of Stacks.
    2. Understand and implement traversal, insertion and deletion on stacks
    3. Implement recursion

###### Theory:

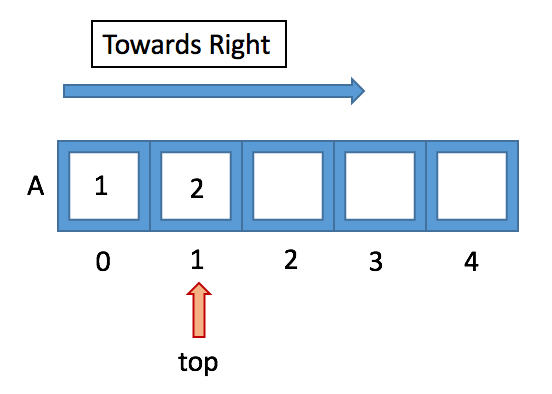
A stack is an abstract data structure that contains a collection of elements. Stack implements the LIFO mechanism i.e. the element that is pushed at the end is popped out first. Some of the principle operations in the stack are −

Push - This adds a data value to the top of the stack.

Pop - This removes the data value on top of the stack

Peek - This returns the top data value of the stack

Both insertion & deletion takes place at the top.



When we implement stack, using arrays we take the direction of the stack i.e the direction in which elements are inserted towards right.

Operations:

isempty() - to check if the Stack is empty or not. (If top=-1, then stack is empty)

push() - to insert element in the Stack. (Increment top and insert element)

pop() - to delete element from stack. (Decrement top)

show\_top() - to display element at top. (The element where top is pointing to)

**Recursion:**

A recursive function is one that calls itself. The two main principles of recursion is the stop condition and the recursion call. Every time a function is called and the base case is not reached, the function behaves recursively by calling itself again

Example:

function xyz()  
{  
 xyz();  
}

xyz();

The recursion use the stack implementation. So, when a function is called, this one go to the top of the stack. And then, as the operation of a stack requires, the function at the top of the stack will be the first to be executed. This means that the last function called will be the first one executed.

When the base case is reached, a domino effect occurs as the response obtained from the base case is used to answer the previous function calls. As soon as a call to a function has been “answered”, the stack associated with the function can be dropped!

* 1. **Tasks to be completed**
     1. Write a program in C++ to perform stack operations:

1. Push, pop, peek, isEmpty(), isFull()
2. To sort data in stack using recursion ( Do not use secondary stack)

\

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. C002 | Name: Kartik Sethi |
| Program: BTI | Division: B |
| Batch: B1 | Date of Experiment: 2/8/2024 |
| Date of Submission: | Grade : |

* 1. **Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A)***

#include <iostream>

using namespace std;

#define MAX\_SIZE 10

class SStack

{

public:

int stack[MAX\_SIZE];

int top = 0;

int choice;

int num;

void push(int n)

{

if (top >= MAX\_SIZE)

{

cout << "MAX SIZE REACHED" << endl;

}

else

{

stack[top] = n;

top++;

}

}

void pop()

{

if (isEmpty())

{

cout << "Stack is empty" << endl;

}

else

{

cout << "Popped number: " << num << endl;

num = stack[top];

top--;

}

}

void peek()

{

if (top <= -1)

{

cout << "Stack underflow";

}

else

{

cout << "Top element: " << stack[top - 1];

}

}

void display()

{

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

{

cout << stack[i] << " ";

}

cout << endl;

}

bool isEmpty()

{

return top == 0;

}

bool isFull()

{

return top>=MAX\_SIZE;

}

void insertSorted(int element)

{

if (isEmpty() || stack[top - 1] <= element)

{

push(element);

}

else

{

int temp = stack[top - 1];

pop();

insertSorted(element);

push(temp);

}

}

void sortStack()

{

if (!isEmpty())

{

int temp = stack[top - 1];

pop();

sortStack();

insertSorted(temp);

}

}

};

int main()

{

SStack stack;

int loop = 1;

while (loop == 1)

{

cout << "\n Main menu ";

cout << "\n1) Add element to stack\n";

cout << "\n2) Delete from the stack\n";

cout << "\n3) Peek from the stack\n";

cout << "\n4) Display the stack\n";

cout << "\n5) Sort the stack\n";

cout << "\nEnter choice: ";

cin >> stack.choice;

switch (stack.choice)

{

case 1:

cout << "Enter number: ";

cin >> stack.num;

stack.push(stack.num);

break;

case 2:

stack.pop();

break;

case 3:

stack.peek();

break;

case 4:

stack.display();

break;

case 5:

stack.sortStack();

cout << "Stack sorted." << endl;

break;

default:

cout << "\n Invalid choice" << endl;

}

cout << "\nDo you want to do more operations on the stack (1 for yes, any other key for exit): ";

cin >> loop;

}

return 0;

}

***Output:***

***For Push:***

***A screen shot of a computer

Description automatically generated***

***For Pop:***

***A black screen with white text

Description automatically generated***

***For Peek:***

***A screenshot of a computer program

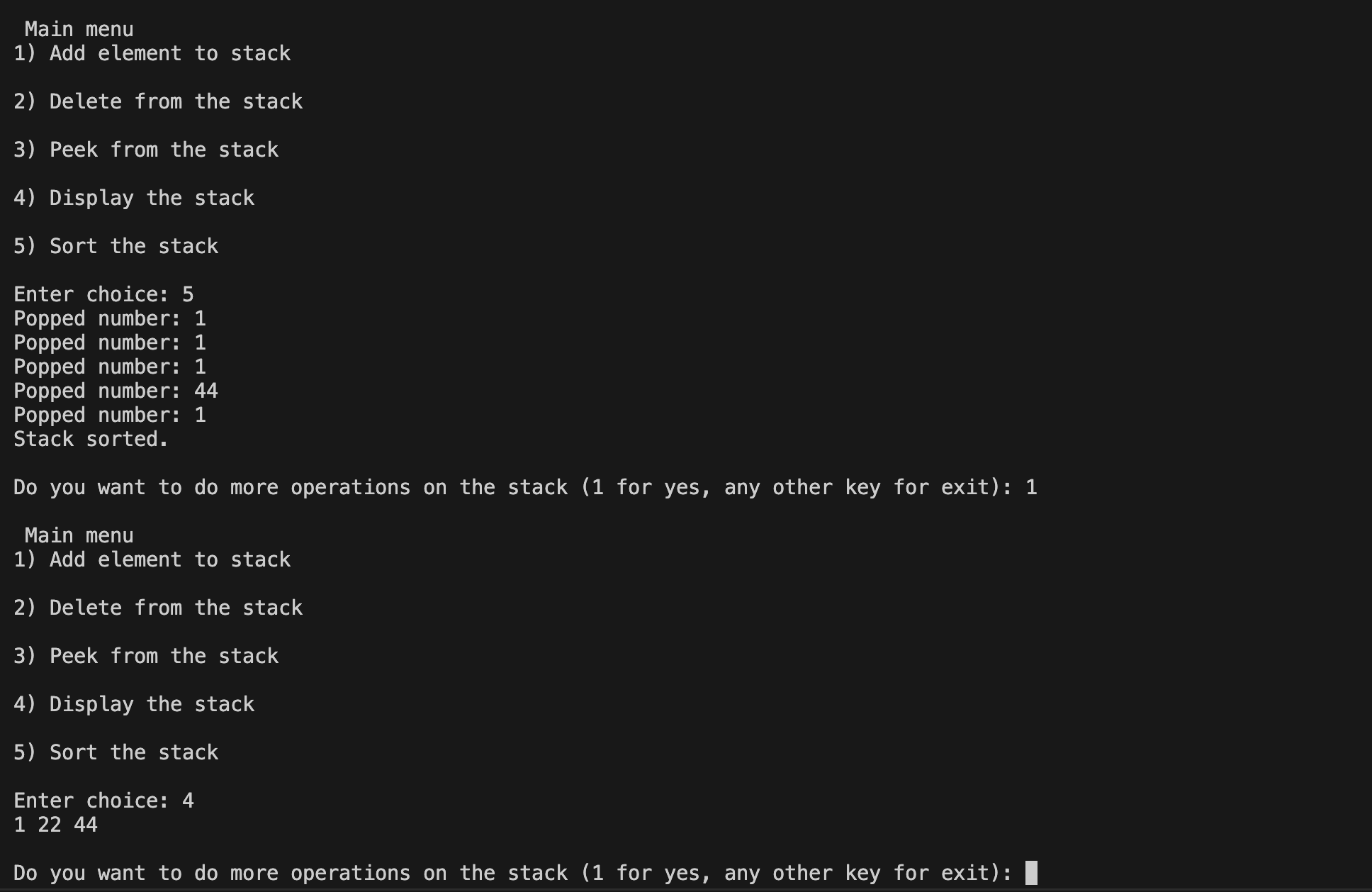
Description automatically generated***

***For Display:***

***A black screen with white text

Description automatically generated***

***FOR SORTING:***

******

* 1. **Observations and Learning:**

*(****Students must write the observations and learning based on their understanding built about the subject matter and inferences drawn)***

I learned how to perform stack operations and how to implement sorting a stack using recursion.

###### Conclusion:

*(****Students must write the conclusive statements as per the attainment of individual outcomes listed above and learning/observation noted in section B.2)***