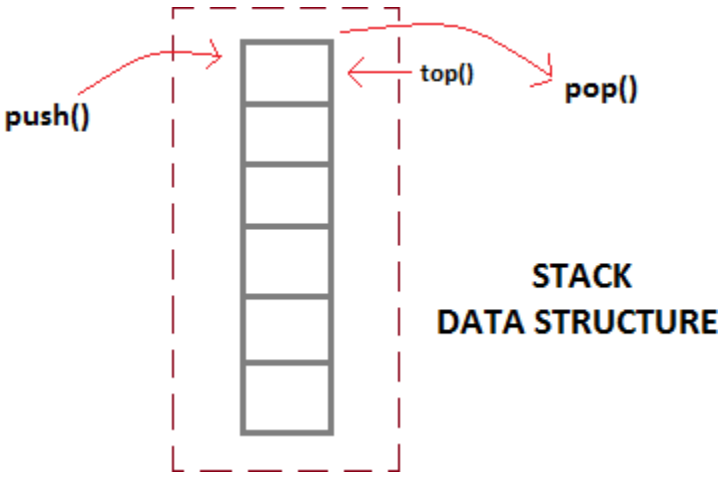
Lab 4 Stacks

4.1. What is Stack Data Structure?

Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the **top** of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.



4.2. Basic features of Stack

1. Stack is an **ordered list** of **similar data type**.
2. Stack is a **LIFO**(Last in First out) structure or we can say **FILO**(First in Last out).
3. `push()` function is used to insert new elements into the Stack and `pop()` function is used to remove an element from the stack. Both insertion and removal are allowed at only one end of Stack called **Top**.
4. Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty.

4.3. Applications of Stack

The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.

There are other uses also like:

1. Parsing
2. Expression Conversion(Infix to Postfix, Postfix to Prefix etc)

4.4. Implementation of Stack Data Structure

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using array.

4.4.1. Algorithm for PUSH operation

1. Check if the stack is **full** or not.
2. If the stack is full, then print error of overflow and exit the program.
3. If the stack is not full, then increment the top and add the element.

4.4.2. Algorithm for POP operation

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

Below we have a simple C++ program implementing stack data structure while following the object-oriented programming concepts.

Example 4-1 Stack Implementation

Code

```
/* Below program is written in C++ language */

# include<iostream>

using namespace std;

class Stack
{
    int top;
    public:
    int a[10];  //Maximum size of Stack
    Stack()
    {
        top = -1;
    }

    // declaring all the function
    void push(int x);
    int pop();
    void isEmpty();
};

// function to insert data into stack
void Stack::push(int x)
{
    if(top >= 10)
    {
        cout << "Stack Overflow \n";
    }
    else
    {
        a[++top] = x;
```

```
        cout << "Element Inserted \n";
    }
}

// function to remove data from the top of the stack
int Stack::pop()
{
    if(top < 0)
    {
        cout << "Stack Underflow \n";
        return 0;
    }
    else
    {
        int d = a[top--];
        return d;
    }
}

// function to check if stack is empty
void Stack::isEmpty()
{
    if(top < 0)
    {
        cout << "Stack is empty \n";
    }
    else
    {
        cout << "Stack is not empty \n";
    }
}

// main function
int main() {

    Stack s1;
    s1.push(10);
    s1.push(100);
    /*
        preform whatever operation you want on the stack
    */
}
```

Output

```
Element Inserted
Element Inserted
```

4.4.3. Analysis of Stack Operations

Below mentioned are the time complexities for various operations that can be performed on the Stack data structure.

```
Push Operation: O(1)
Pop Operation: O(1)
Top Operation: O(1)
Search Operation: O(n)
```

The time complexities for `push()` and `pop()` functions are `O(1)` because we always have to insert or remove the data from the **top** of the stack, which is a one step process.

4.5. Example Problem

Example 4-2 Stacks Example Problem

Problem Statement
Implement the stack using linked lists. The stack must have push, pop and top operations as follows: 1. Push operation must add a character in the stack 2. Pop operation must remove the last character that was pushed in the stack 3. Top operation must return the value of the last character that was inserted in the stack Please note that all operations (push, pop and top) must be implemented as separate functions (methods).
Solution

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};

class Stack{
      public:
      Node *top;
      int size;

      Stack()
      {
            this->top  = NULL;
            this->size = 0;
      }

      void push(int val) {
         struct Node* newnode = new Node();
         newnode->data = val;
         newnode->next = this->top;
         this->top = newnode;
      }

      void pop() {
         if(this->top==NULL)
            cout<<"Stack is empty"<<endl;
         else {
            cout<<"The popped element is "<< this->top->data <<endl;
            this->top = this->top->next;
         }
      }

      void display() {
```

```
         struct Node* ptr;
         if(this->top==NULL)
            cout<<"stack is empty";
         else {
            ptr = this->top;
            cout<<"Stack elements are: ";
            while (ptr != NULL) {
               cout<< ptr->data <<" ";
               ptr = ptr->next;
            }
         }
         cout<<endl;
      }
};

int main() {
   int ch, val;
   Stack stack;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
   cout<<"3) Display stack"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter choice: "<<endl;
      cin>>ch;
      switch(ch) {
         case 1: {
            cout<<"Enter value to be pushed:"<<endl;
            cin>>val;
            stack.push(val);
            break;
         }
         case 2: {
            stack.pop();
            break;
         }
         case 3: {
            stack.display();
            break;
         }
         case 4: {
            cout<<"Exit"<<endl;
            break;
         }
         default: {
            cout<<"Invalid Choice"<<endl;
         }
      }
   }while(ch!=4);
      return 0;
}
```

Output

```
 1) Push in stack
 2) Pop from stack
 3) Display stack
 4) Exit
 Enter choice:
 1
```

```
Enter value to be pushed:
11
Enter choice:
3
Stack elements are: 11
Enter choice:
2
The popped element is 11
Enter choice:
3
stack is empty
Enter choice:
```