



# Data Structures CS-204

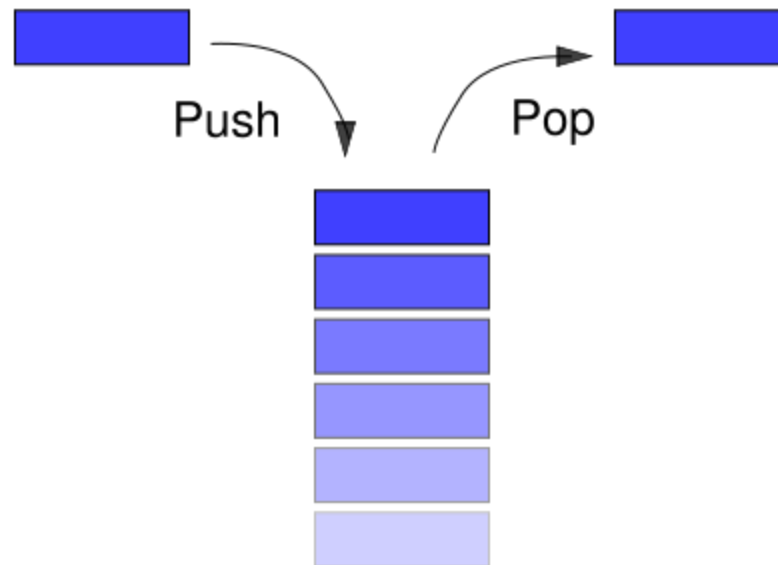
## Lecture 4

### **STACKS**

# Stacks

- A stack is a list in which insertion and deletion take place at the same end
  - This end is called top
  - The other end is called bottom
- Stacks are known as LIFO (Last In, First Out) lists.
  - The last element inserted will be the first to be retrieved
- E.g. a stack of Plates, books, boxes etc.

# Insertion and deletion on stack

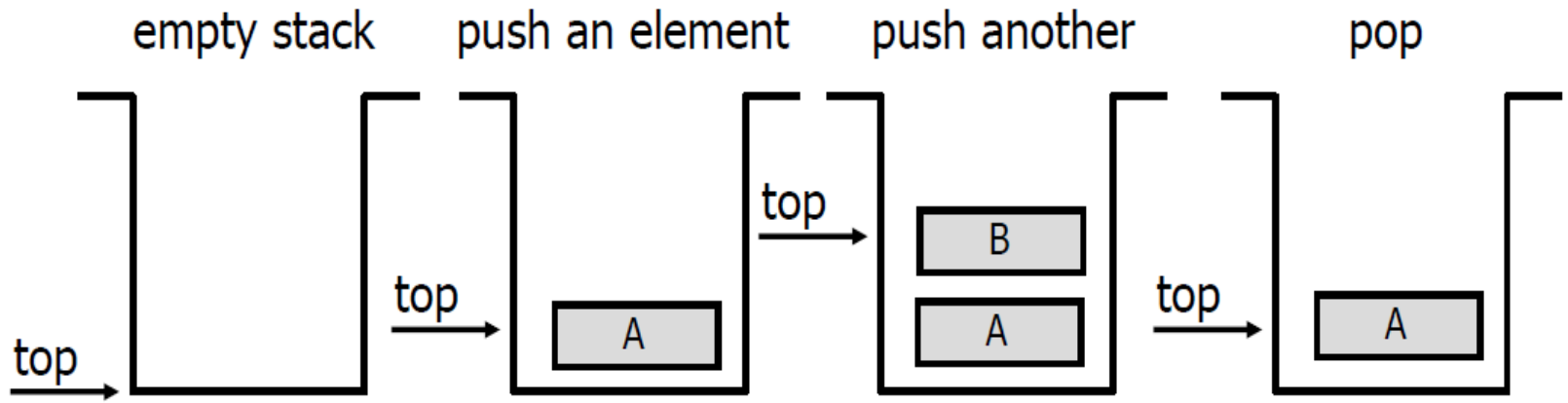


# Operation On Stack

- Creating a stack
- Checking stack---- either empty or full
- Insert (PUSH) an element in the stack
- Delete (POP) an element from the stack
- Access the top element
- Display the elements of stack

# Push and Pop

- Primary operations: Push and Pop
- Push
  - Add an element to the top of the stack.
- Pop
  - Remove the element at the top of the stack.



# Stack-Related Terms

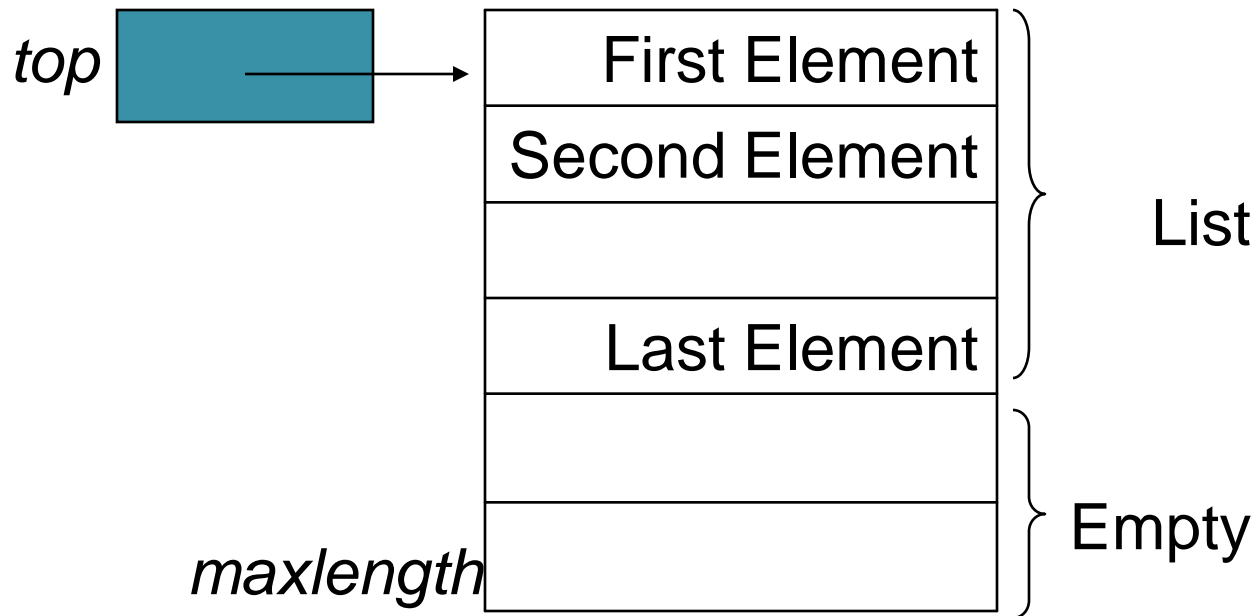
- Top
  - A pointer that points the top element in the stack.
- Stack Underflow
  - When there is no element in the stack or stack holds elements less than its capacity, the status of stack is known as stack underflow.
- Stack Overflow
  - When the stack contains equal number of elements as per its capacity and no more elements can be added, the status of stack is known as stack overflow

# Stack Implementation

- Implementation can be done in two ways
  - Static implementation
  - Dynamic Implementation
- Static Implementation
  - Stacks have **fixed size**, and are implemented as **arrays**
  - It is also inefficient for utilization of memory
- Dynamic Implementation
  - Stack **grow in size** as needed, and implemented as **linked lists**
  - Dynamic Implementation is done through pointers
  - The memory is efficiently utilize with Dynamic Implementations

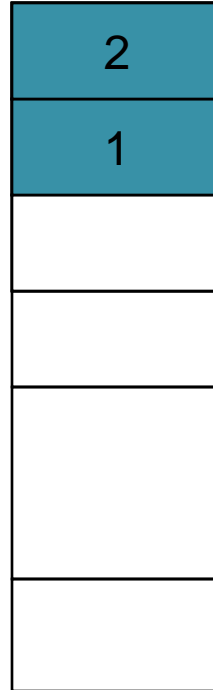
# Static Implementation

- Elements are stored in contiguous cells of an array.
- New elements can be inserted to the top of the list.





# Static Implementation



## Problem with this implementation

- Every PUSH and POP requires moving the entire array up and down.

# Static Implementation

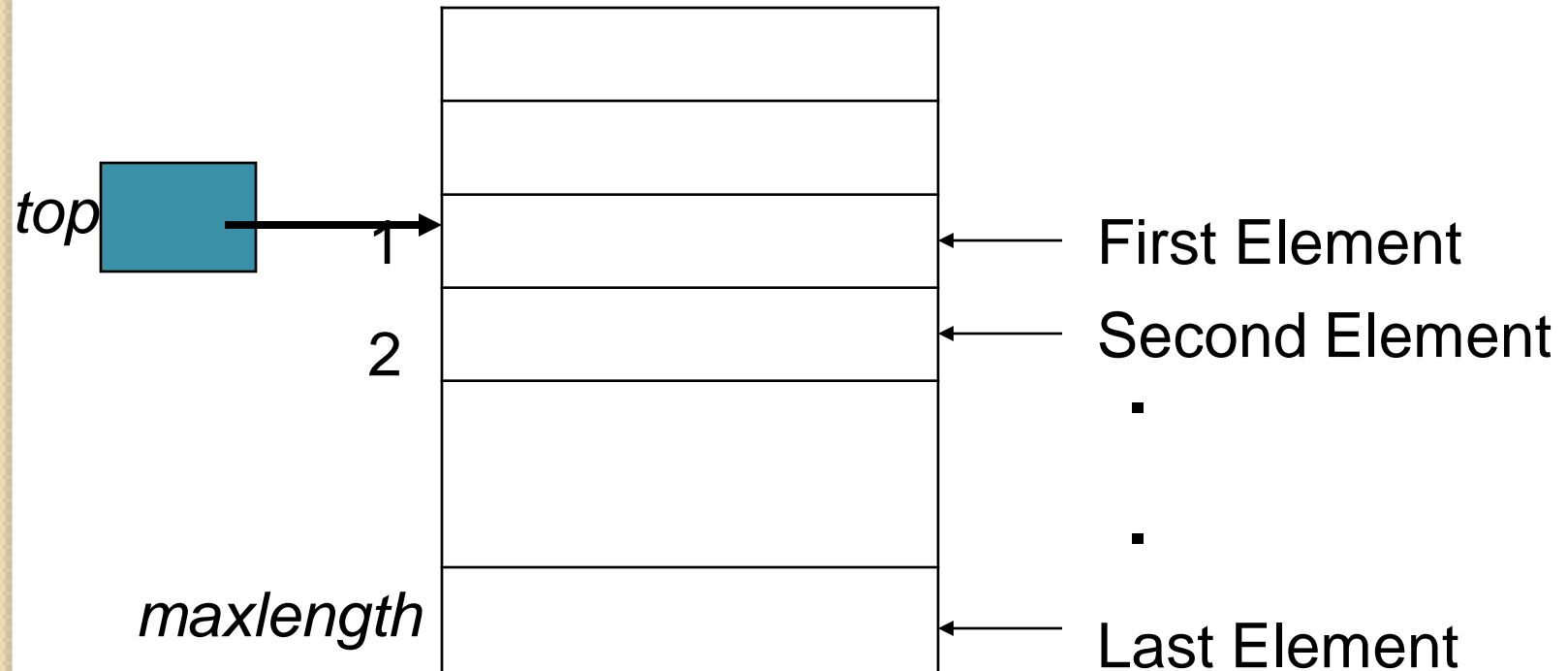
Since, in a stack the insertion and deletion take place only at the top, so...

## **A better Implementation:**

- Anchor the bottom of the stack at the bottom of the array
- Let the stack grow towards the top of the array
- *Top* indicates the current position of the first stack element.

# Static Implementation

A better Implementation:



# A Simple Stack Class

```
class IntStack{  
    private:  
        int *stackArray;  
        int stackSize;  
        int top;  
    public:  
        IntStack(int);  
        bool isEmpty();  
        bool isFull();  
        void push();  
        void pop();  
        void displayStack();  
        void displayTopElement();  
};
```

# Constructor

```
IntStack::IntStack(int size)
{
    stackArray = new int[size];
    stackSize = size;
    top = -1;
}
```

# Push( )

```
void IntStack::push()
{
    clrscr();
    int num;
    if(top>=stackSize)
        cout<<"stack Overflow"<<endl;
    else
    {
        cout<<"Enter Number=";
        cin>>num;
        top++;
        stackArray[top]=num;
    }
}
```

# Pop( )

```
void IntStack::pop()
{
    clrscr();
    if(top == -1)
        cout<<"Stack Underflow"<<endl;
    else
    {
        cout<<"Number Deleted From the stack=";
        cout<<stackArray[top];
        top--;
    }
    getche();
}
```

# Main( )

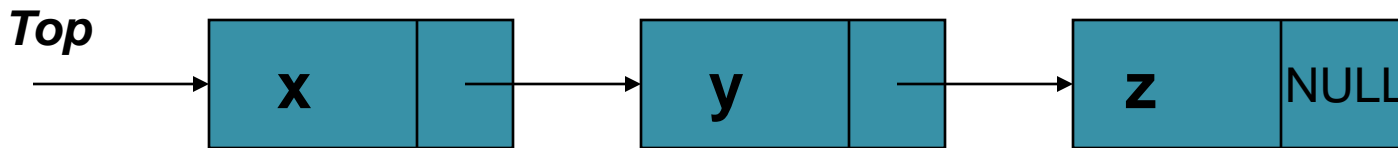
```
void main ()
{
    IntStack stack(5);
    int choice;
    do
    {
        cout<<"Menu"<<endl;
        cout<<"1-- PUSH"<<endl;
        cout<<"2-- POP"<<endl;
        cout<<"3-- DISPLAY "<<endl;
        cout<<"4-- Exit"<<endl;
        cout<<"Enter choice=";
        cin>>choice;
        switch(choice)
```

```
{
    case 1:
        stack.push(); break;
    case 2:
        stack.pop(); break;
    case 3:
        stack.displayStack();
        break;
    }
    }while(choice!=4);
    getch();
}
```



# Dynamic Implementation of Stacks

- As we know that dynamic stack is implemented using linked-list.
- In dynamic implementation stack can expand or shrink with each PUSH or POP operation.
- PUSH and POP operate only on the first/top cell on the list.



# Dynamic Implementation of Stack

## Class Definition

```
class ListStack{  
    private:  
        struct node{  
            int num;  
            node *next;  
        }*top;  
    public:  
        ListStack(){ top=NULL;}  
        void push();  
        void pop();  
        void display();  
};
```

# Push( ) Function

- This function creates a new node and ask the user to enter the data to be saved on the newly created node.

```
void ListStack::push()
{
    node *newNode;
    newNode= new node;
    cout<<"Enter number to add on stack";
    cin>> newNode->num;
    newNode->next=top;
    top=newNode;
}
```

# Pop( ) Function

```
void ListStack::pop()
{
    node *temp;
    temp=top;
    if(top==NULL)
        cout<<"Stack UnderFlow"<<endl;
    else
    {
        cout<<"deleted Number from the stack =";
        cout<<temp->num;
        temp=temp->next;
        delete temp;
    }
}
```

# Main( ) Function

```
void main()
```

```
{
```

```
    clrscr();
```

```
    ListStack LS;
```

```
    int choice;
```

```
    do{
```

```
        cout<<"Menu "<<endl;
```

```
        cout<<"1.Push" <<endl;
```

```
        cout<<"2.Pop"<<endl;
```

```
        cout<<"3.Show"<<endl;
```

```
        cout<<"4.EXIT"<<endl;
```

```
        cin>>choice;
```

```
    switch(choice){
```

```
        case 1:
```

```
            LS.push();
```

```
            break;
```

```
        case 2:
```

```
            LS.pop();
```

```
            break;
```

```
        case 3:
```

```
            LS.display();
```

```
            break;
```

```
    }
```

```
    }while(choice!=4);
```

```
}
```

# Stack applications

- “Back” button of Web Browser
  - History of visited web pages is pushed onto the stack and popped when “back” button is clicked
- “Undo” functionality of a text editor
- Reversing the order of elements in an array
- Saving local variables when one function calls another, and this one calls another, and so on.

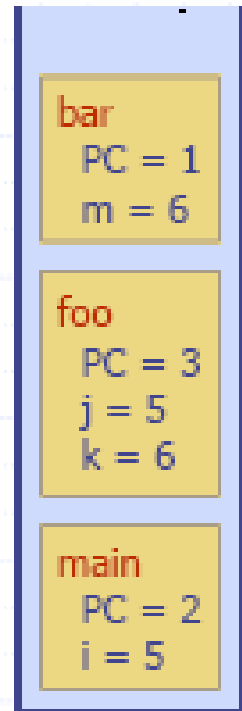
# C++ Run-time Stack

- The C++ run-time system keeps track of the chain of active functions with a stack
- When a function is called, the run-time system pushes on the stack a frame containing
  - Local variables and return value
  - Program counter, keeping track of the statement being executed
- When a function returns, its frame is popped from the stack and control is passed to the method on top of the stack

```
main() {  
    int i = 5;  
    foo(i);  
}
```

```
foo(int j) {  
    int k;  
    k = j+1;  
    bar(k);  
}
```

```
bar(int m) {  
    ...  
}
```





# Pointers

“new” & “delete” Operators



# Dynamic Variables: 'new' operator

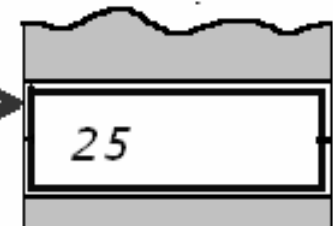
A **dynamic variable** is created and destroyed while the program is running

```
int *p ;
```

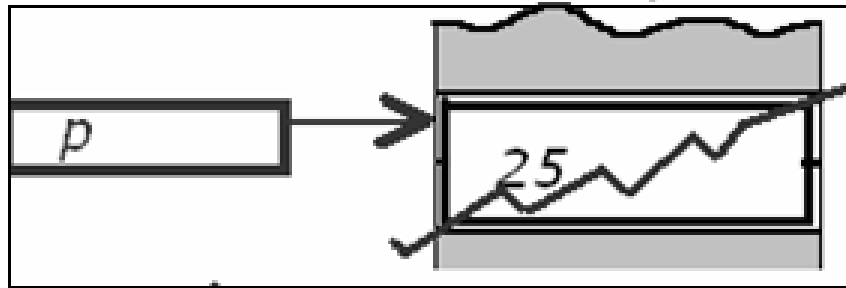
```
p = new int ;
```

creates a new dynamic integer variable and leaves p to point to this variable

```
*p = 25 ;
```



# Dynamic Variables: 'delete' operator



**delete p** ; destroys the dynamic variable pointed by p  
After delete p, p becomes an undefined pointer variable: **a dangling pointer**.

**Take care:** before using \* again, be sure p points to something and is not a dangling pointer. Otherwise unpredictable effects.