Data Structure

Stacks

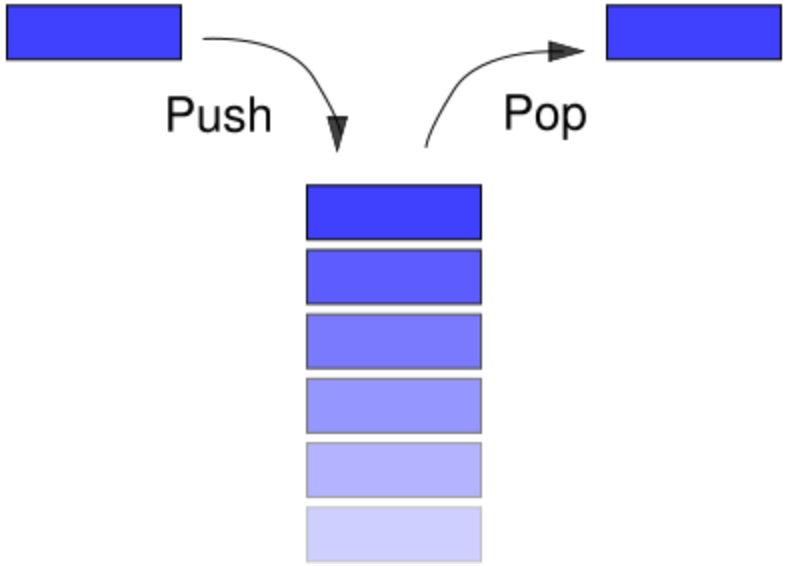
Stacks

- 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

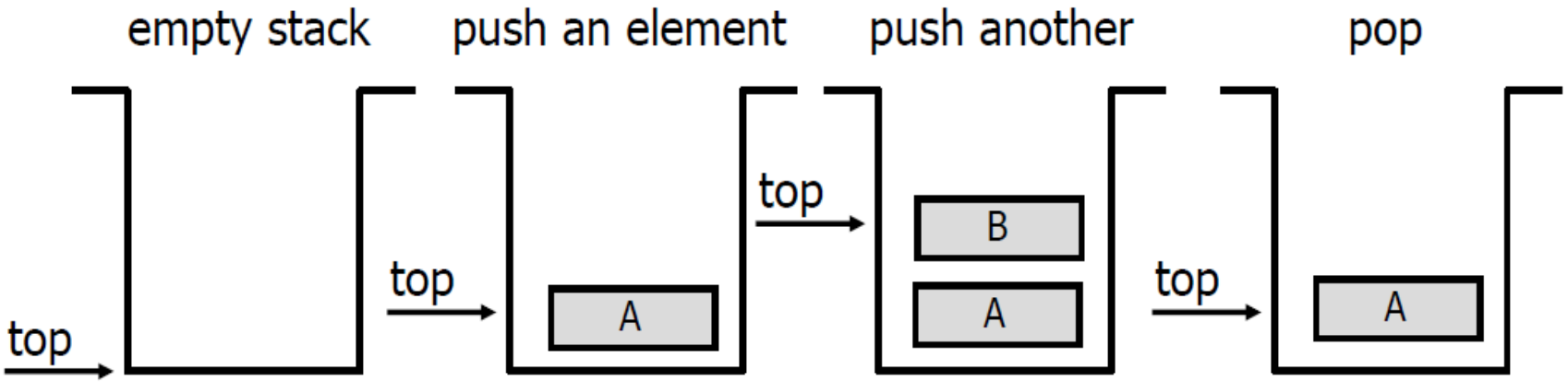


Operation On Stack

- 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

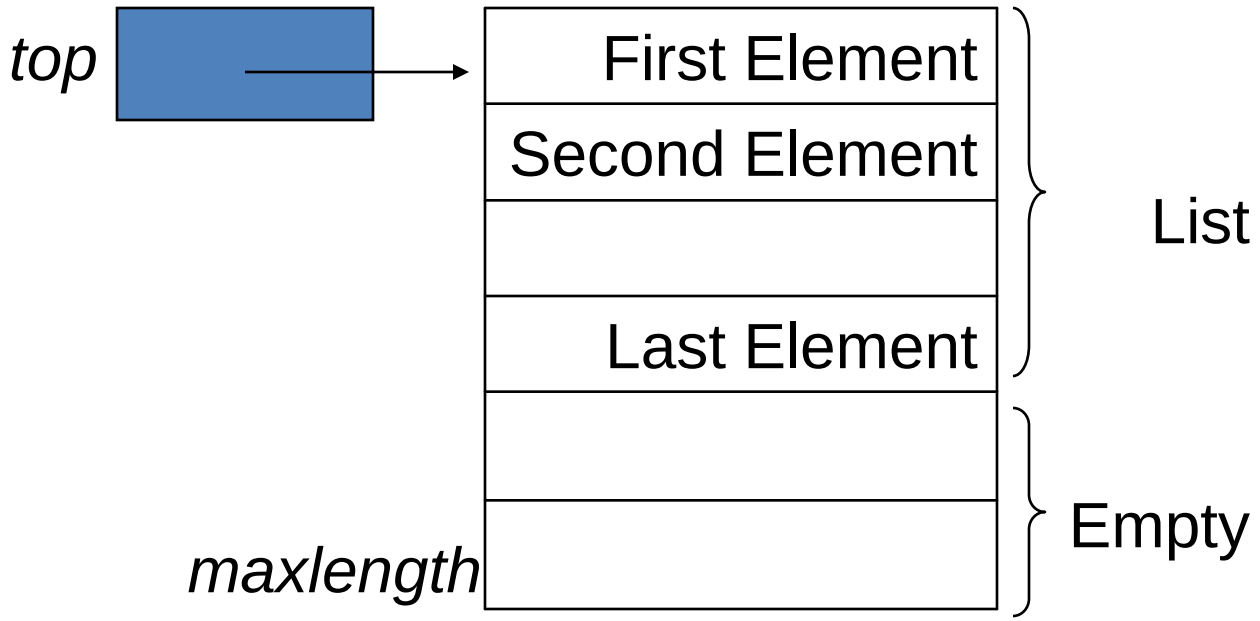
- Top
  - A pointer that points the top element in the stack.
- Stack Underflow
  - When there is no element in the stack, the status of stack is known as stack underflow.
- Stack Overflow
  - When the stack contains equal number of elements as per its capacity and no more elements can be added, the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

Stack Implementation

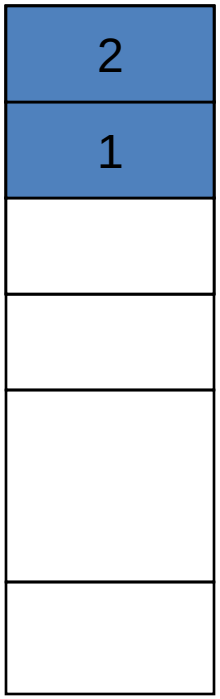
- Implementation can be done in two ways
  - Static implementation
  - Dynamic Implementation
- Static Implementation
  - Stacks have **fixed size**, and are implemented as **arrays**
  - It is also inefficient for utilization of memory
- Dynamic Implementation
  - Stack **grow in size** as needed, and 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

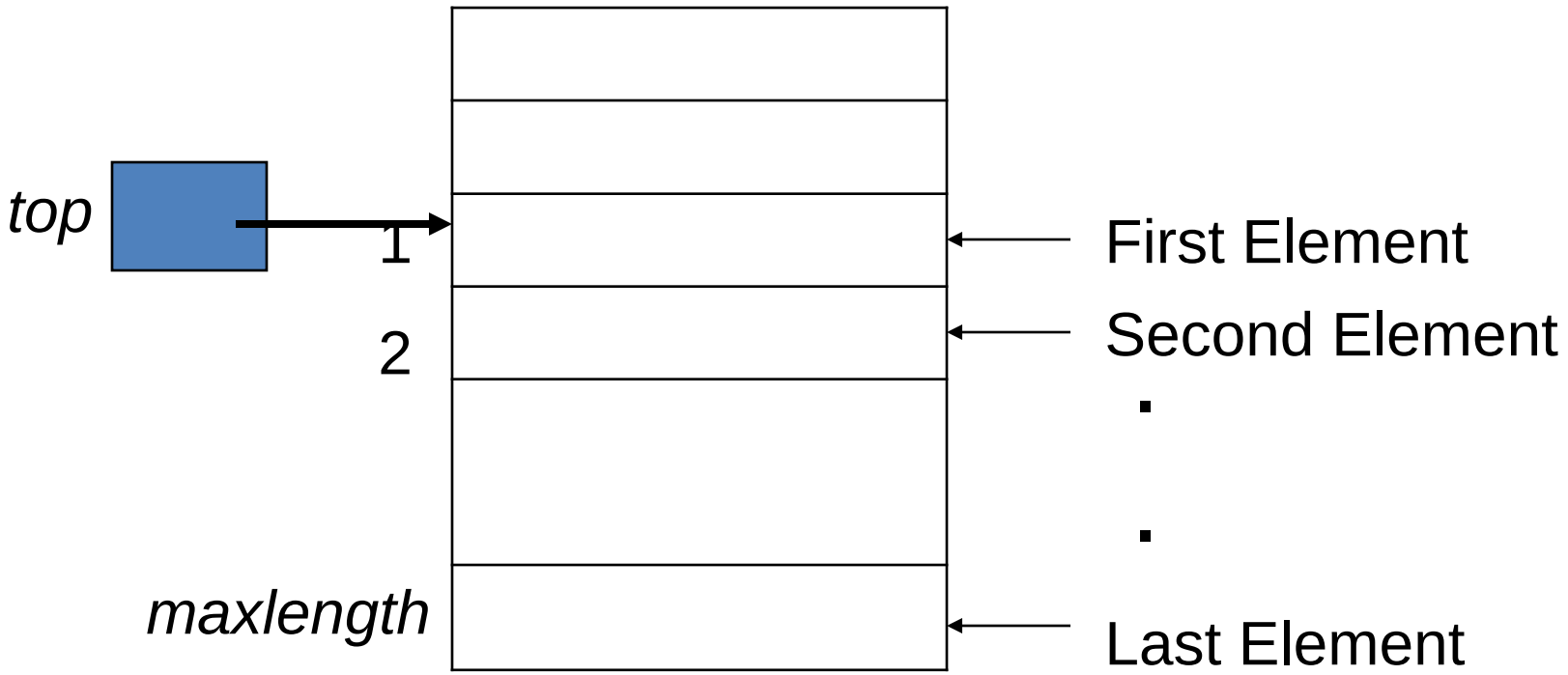
Since, in a stack the insertion and deletion take place only at the top, so…

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:



Push( )

```
void push()
{
   int element;
   top = top + 1;
   if( top < MAX )
   {
     printf("\nEnter a Number to push into Stack : ");
     scanf("%d",&element);
     st[top] = element;
     printf("\n%d is inserted into the Stack ",element);
   }
   else
   {
     printf("\nStack Overflow ( FULL ) No more elements can be Added");
     top = top - 1;
   }
}
```

Pop( )

```
void pop()
{
   int x;

   if(top>=0)
   {
     x = st[top];
     printf("\n\nPopped Element from stack is %d ",x);
     top = top-1;
   }
   else
     printf("\n\nStack is Empty, no elements present " );
}
```

display( )

```
void display()
{
   int j;
   if( top == -1)
     printf("\n\nStack is Empty, no elements present " );
   else
   {
     printf("\n\nElements of the Stack are : ");
     for(j=top; j>=0; j--)
       printf("\n%d ",st[j]);
   }
}
```

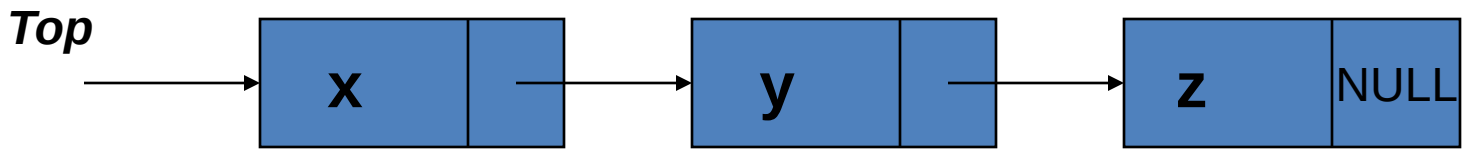
Main( )

```
#define MAX 10
#include<stdio.h>
void display();
void push();
void pop();

int st[MAX], top=-1;

int main()
{
   int i, n, ch;

   printf("\n\n Program to Implement Stack using Arrays : ");
   printf("\n\n MAX Size of the stack is %d elements : ",MAX);

   while(1)
   {
   printf("\n\n 1. Push an element into Stack ");
   printf("\n 2. Pop an Element from Stack ");
   printf("\n 3. Display elements of Stack ");
   printf("\n 4. EXIT the Program ");
   printf("\n\nEnter your Option : ");
   scanf("%d",&ch);
```

```
switch(ch)
   {
      case 1 : push();
               break;

      case 2 : pop();
               break;

      case 3 : display();
               break;

      case 4 : return(0);

      default : printf("\nInvalid Choice [ Enter 1 to 3 ]
       ");
               break;

   }

   }
   printf("\n\n\n");
}
```

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<<top->num;
        top=top->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