Data Structures and Algorithms

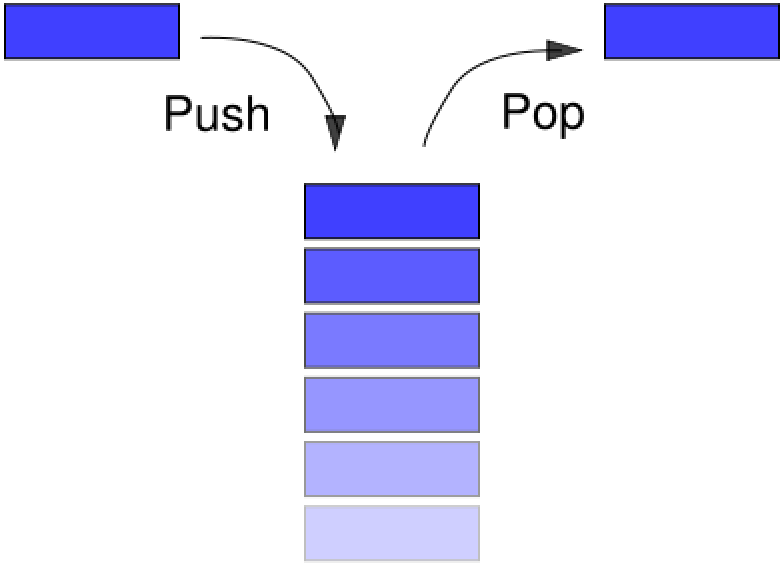
Lesson 2

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

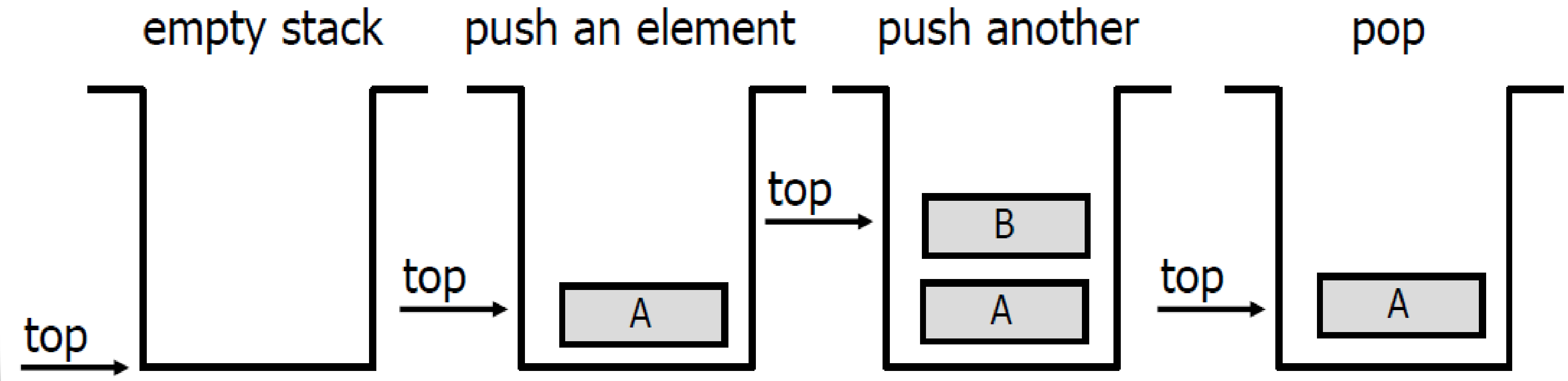


Operation On Stack

- 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

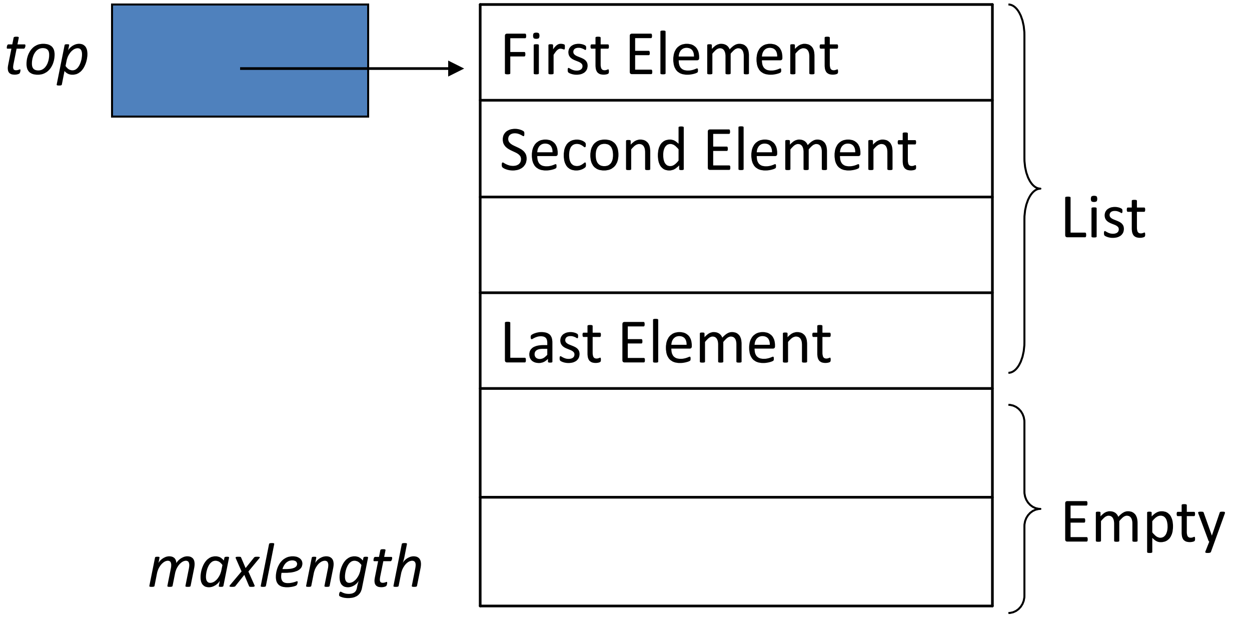
- Top
  - A pointer that points the top element in the stack.
- Stack Underflow
  - When there is no element in the stack, the status of stack is known as stack underflow.
- Stack Overflow
  - When the stack contains equal number of elements as per its capacity and no more elements can be added, the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

Stack Implementation

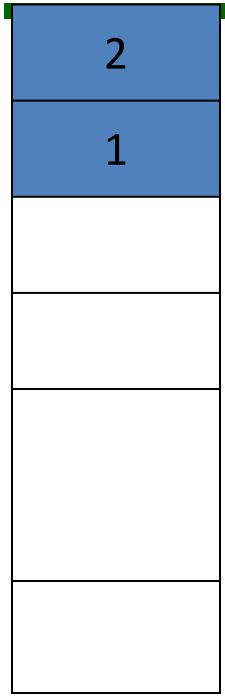
- Implementation can be done in two ways
  - Static implementation
  - Dynamic Implementation
- Static Implementation
  - Stacks have **fixed size**, and are implemented as **arrays**
  - It is also inefficient for utilization of memory
- Dynamic Implementation
  - Stack **grow in size** as needed, and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

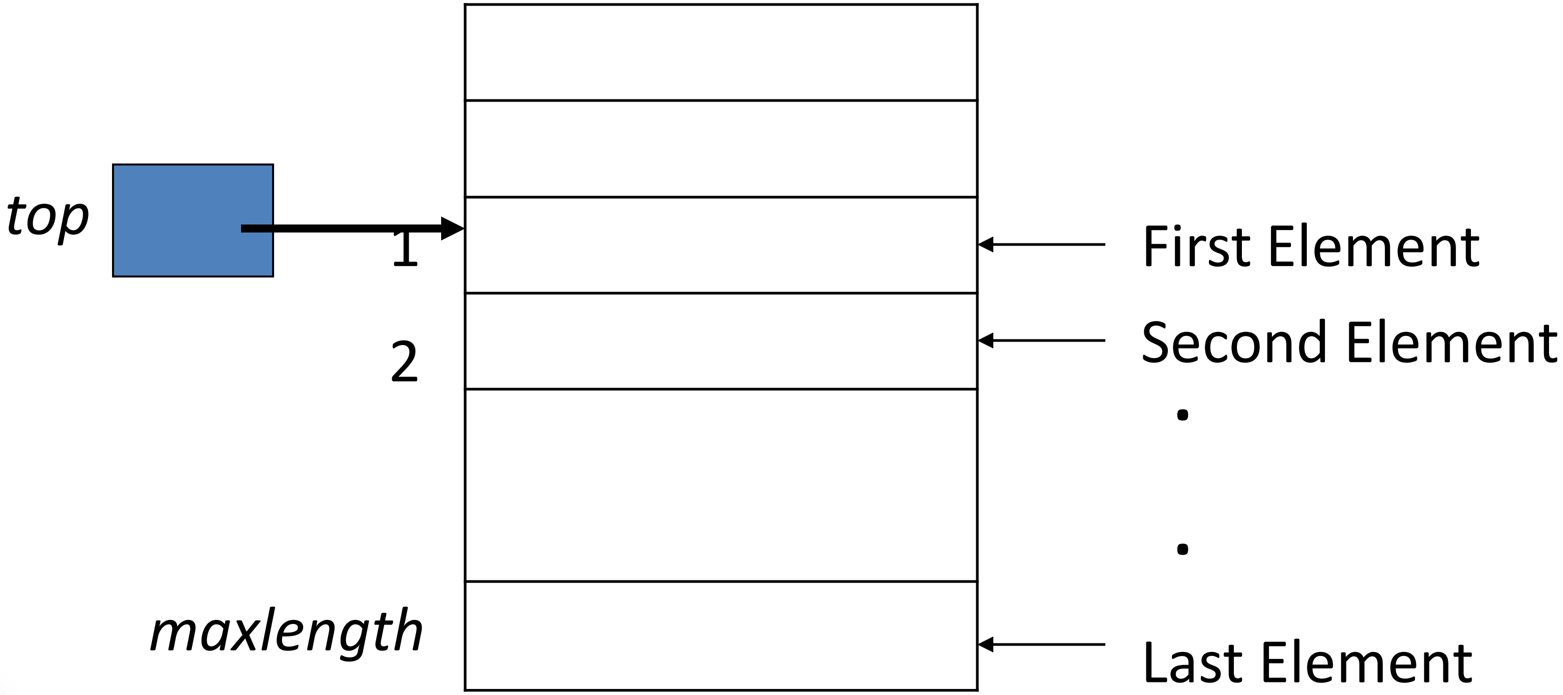
Since, in a stack the insertion and deletion take place only at the top, so…

A better Implementation:

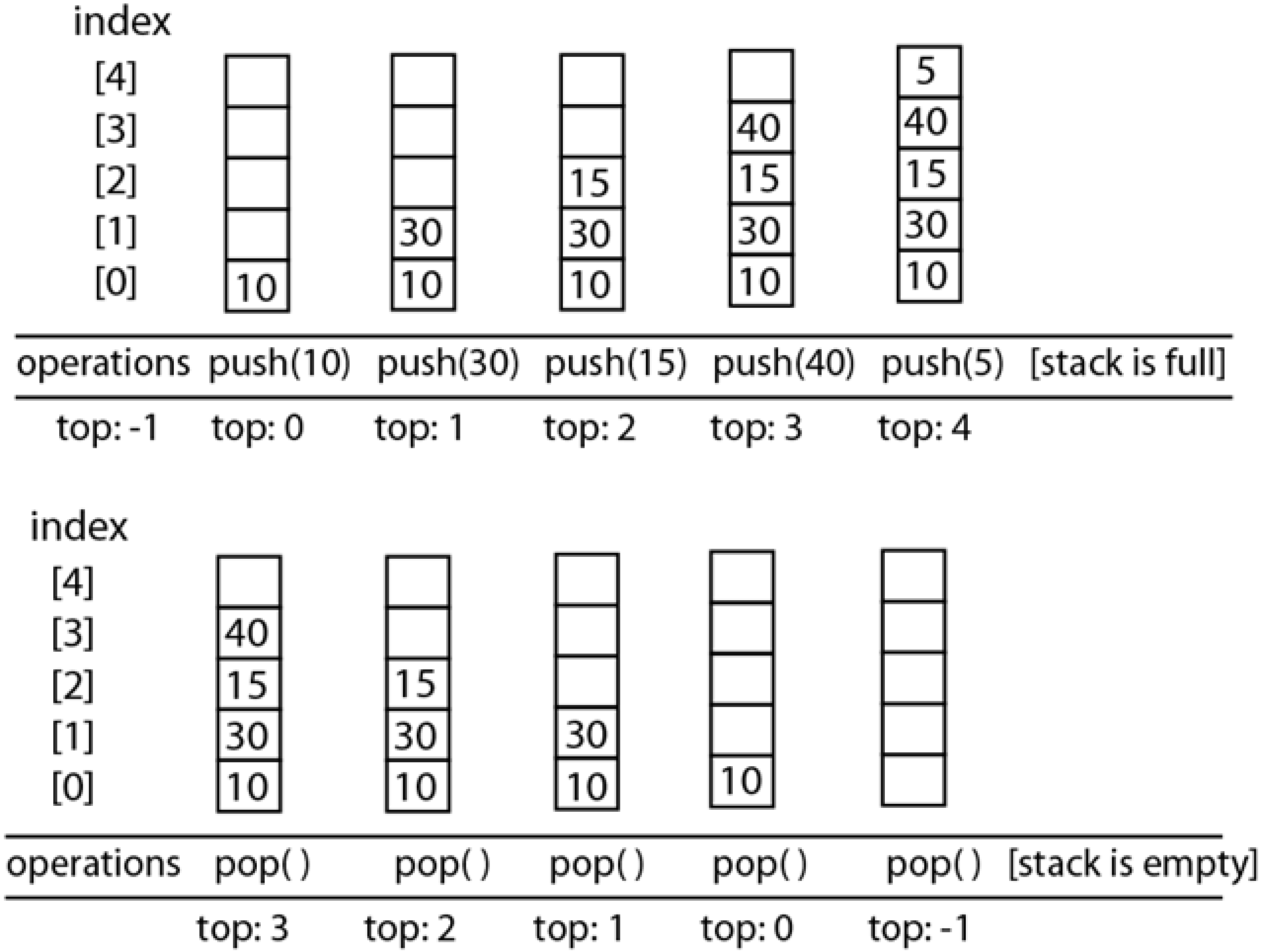
- Anchor the bottom of the stack at the bottom of the array
- Let the stack grow towards the top of the array
- *Top* indicates the current position of the first stack element.

Static Implementation

A better Implementation:



Stack Example



index						
[4]					5	
[3]				40	40	
[2]			15	15	15	
[1]		30	30	30	30	
[0]	10	10	10	10	10	
operations	push(10)	push(30)	push(15)	push(40)	push(5)	[stack is full]
top: -1	top: 0	top: 1	top: 2	top: 3	top: 4	

index						
[4]						
[3]	40					
[2]	15	15				
[1]	30	30	30			
[0]	10	10	10	10		
operations	pop( )	pop( )	pop( )	pop( )	pop( )	[stack is empty]
	top: 3	top: 2	top: 1	top: 0	top: -1	

Stack Implementation (array)

```
#include <iostream>
#define STACK_SIZE 5
using namespace std;
int stackNum[STACK_SIZE];
int top=-1;
void pop();
void push(int);
void display();
```

Stack Implementation (continued)

```
int main()
{
   while(1){
     int choice,num;
     system("cls");
     cout<< "[1] - Push \n";
     cout<< "[2] - Pop \n";
     cout<< "[3] - Display \n";
     cout<< "[4] - Exit \n";
     cout<< "\n====================\n";
     cout<< "Enter your choice: ";
     cin>> choice;
```

Stack Implementation (continued)

```
switch(choice){
  case 1:
    cout<<"Enter number to push: ";
    cin>>num;
    push(num); break;
  case 2:
    pop(); break;
  case 3:
    display(); break;
```

Stack Implementation (continued)

```
  case 4:
     exit(1);
  default :
     cout<<"\nInvalid Choice";
  }
  cout<<endl<<endl;
  system("pause>0");
  return 0;
}
```

Stack Implementation (continued)

```
void pop()
{
  if(top==-1)
    cout<<"\nStack is Empty\n";
  else
    cout<<"You remove " <<
        stackNum[top--];
}
```

Stack Implementation (continued)

```
void push(int n)
{
  if(top==STACK_SIZE-1)
     cout<<"Stack is full";
  else
     stackNum[++top]=n;
}
```

Stack Implementation (continued)

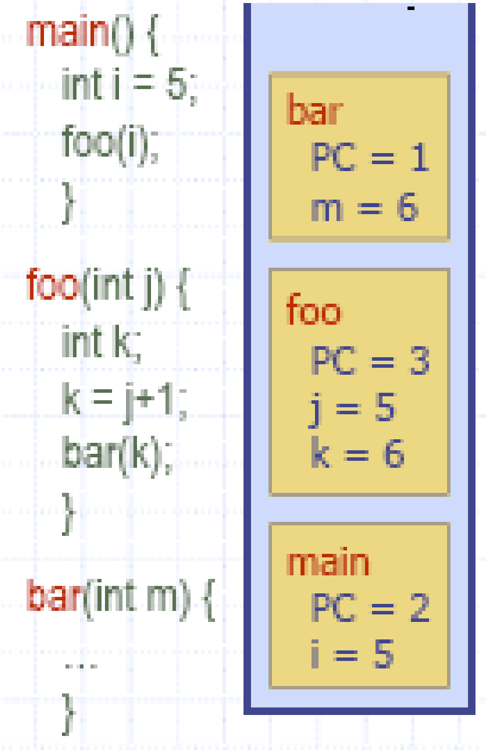
```
void display()
{
  if(top==-1)
    cout<<"\nstack is empty\n";
  else
    for(int i=top;i>=0;i--)
      cout<<stackNum[i]<<endl;
}
```

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
- Converting decimal to binary
- Reversing the order of elements in an array
- Evaluating arithmetic expression
- 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) {
  …
}
```

Infix, Prefix, and Postfix

- **Infix notation**
  - Operator is written in-between the operands.
- **Prefix notation**
  - Polish notation
  - Operators is written before the operands
- **Postfix notation**
  - Suffix notation or reverse polish notation
  - Operators is written after the operands

Operator Precedence

^	-	Exponential operator
*, /	-	Multiplication, Division
+, -	-	Addition, Subtraction

Sample Problem

- Given the expression **A + B * C**
- Solve the **infix, prefix and postfix**

Solution

- **Infix**

  A + B * C

Solution

- **Prefix**

A + (B * C)	parenthesized the expression
A + (* B C)	convert the sub expression to prefix (multiplication)
+ A (* B C)	convert to prefix (addition)
+ A * B C	remove the parenthesis

- **postfix**

A + (B * C)	parenthesized the expression
A + (B C *)	convert the sub expression to postfix (multiplication)
A (B C *) +	convert to postfix (addition)
A B C * +	remove the parenthesis