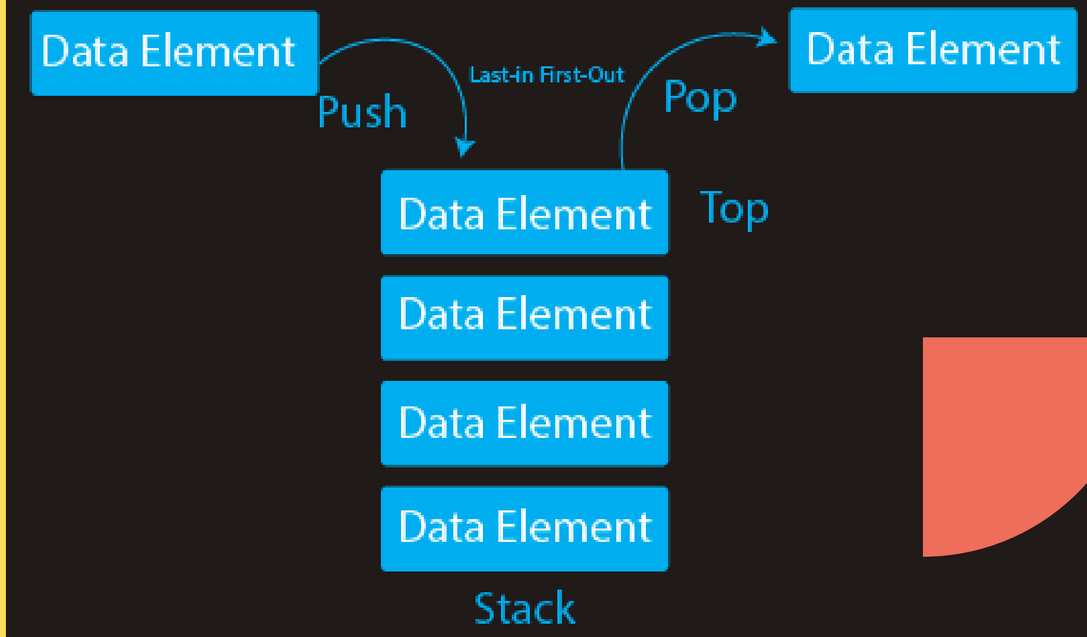
Subject: Data Structure
Using C++ (CSET243)

BENNETT UNIVERSITY
THE TIMES GROUP

Week 7 Lecture

Stack and Its Applications

What, Why and How?



by

Dr Gaurav Kumar

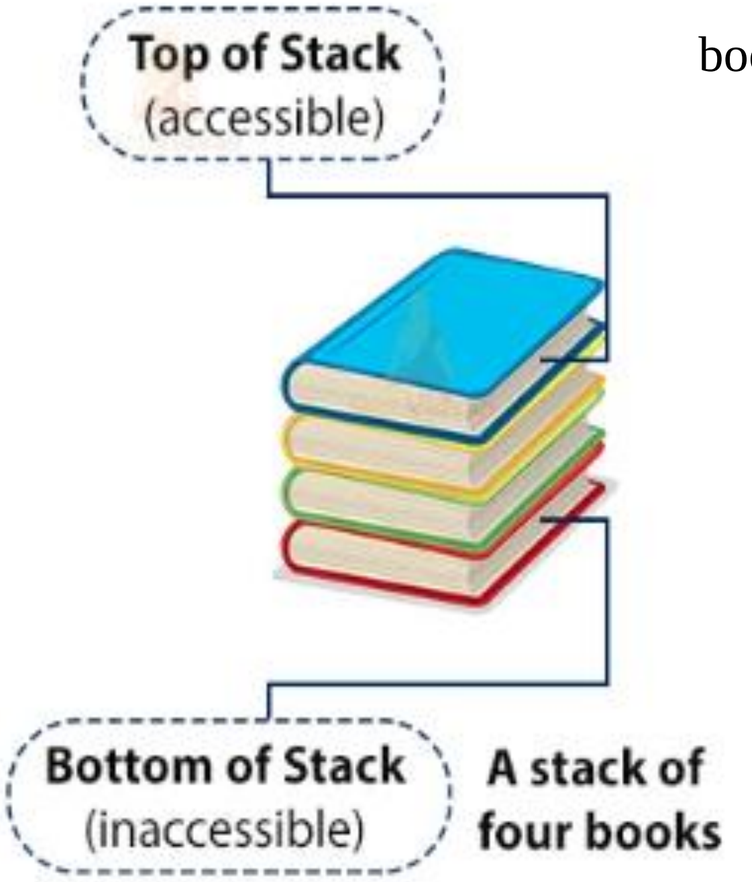
Asst. Prof, Bennett University

Quick Recap of Previous Weeks' Learnings

- Understood the Linked List and Operations

Stack Data Structure

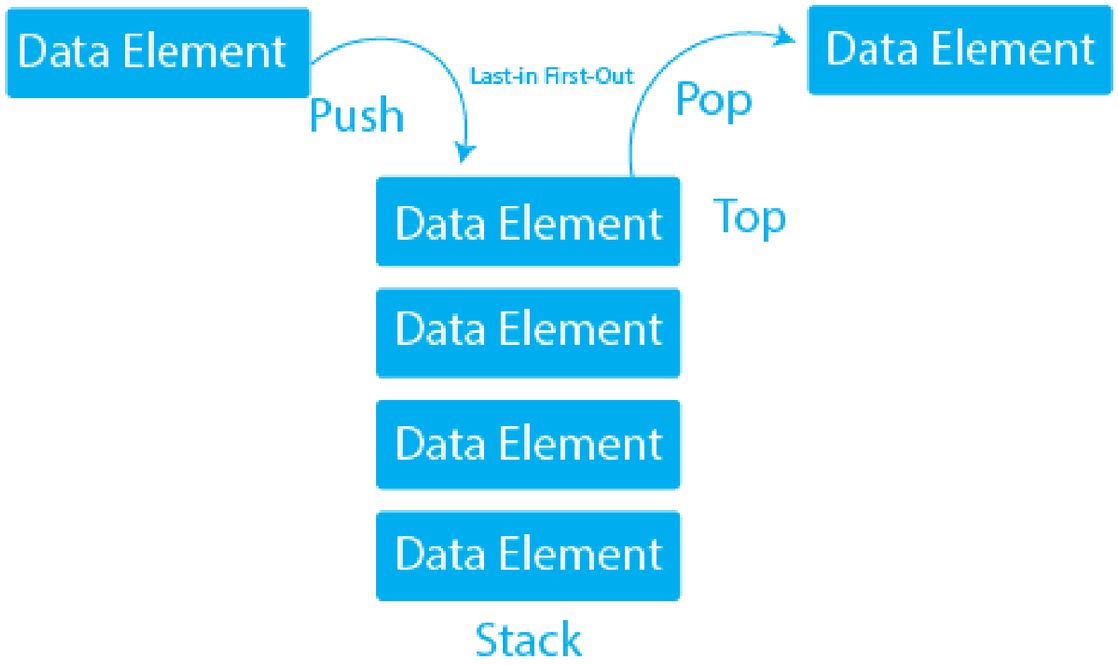
It is called as stack because it behaves like a real-world stack, piles of books, etc. It is a **linear data structure**.



Push a new
book on top

Pop a book
on top

Understanding the Stack

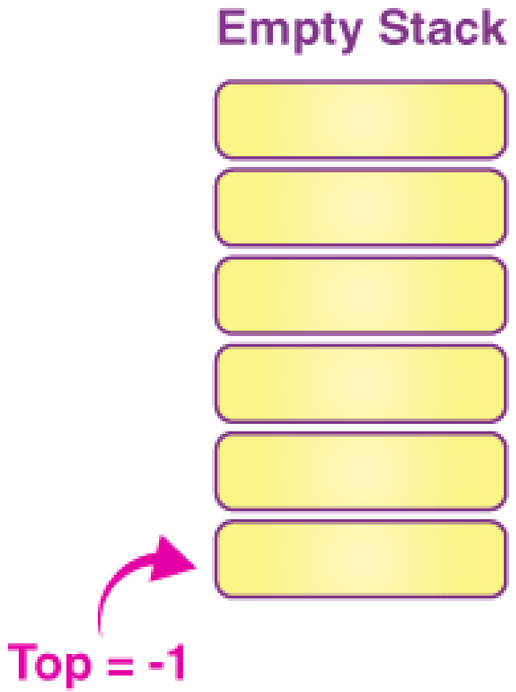


- Stack is an **abstract data type** with a pre-defined capacity.
- One side open another side closed
- Works on **LIFO (Last in First Out)** or **FILO (First in Last Out)** order to Insert or Delete the element**.**
- **Top** is a variable which contains the position of the **top-most element in the stack**.
- Stack can be defined as a container in which **insertion (push) and deletion (pop)** can be done from the **one end** known as the top of the stack.

Stack Creation: Using Array

BENNETT UNIVERSITY
THE TIMES GROUP

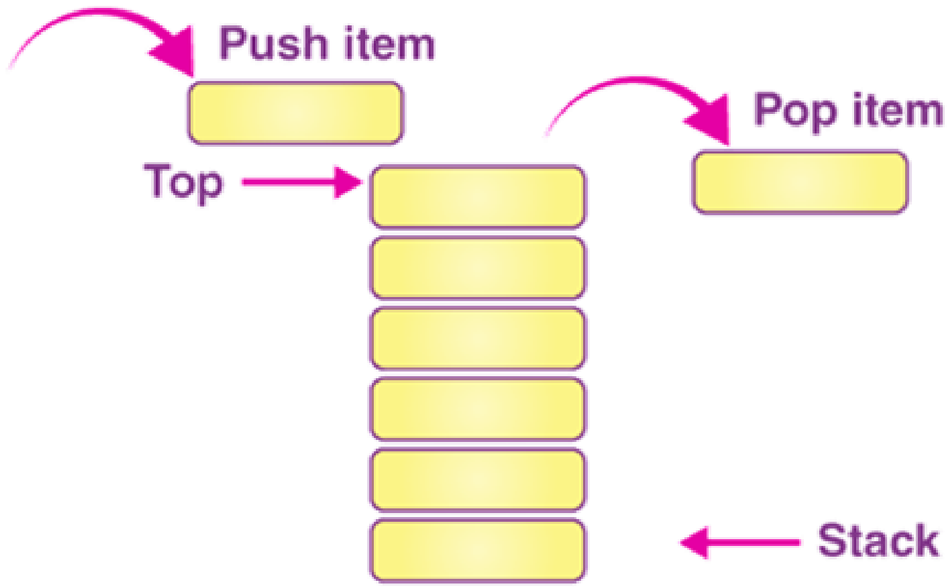
int stack[n], n = 100, top = -1;

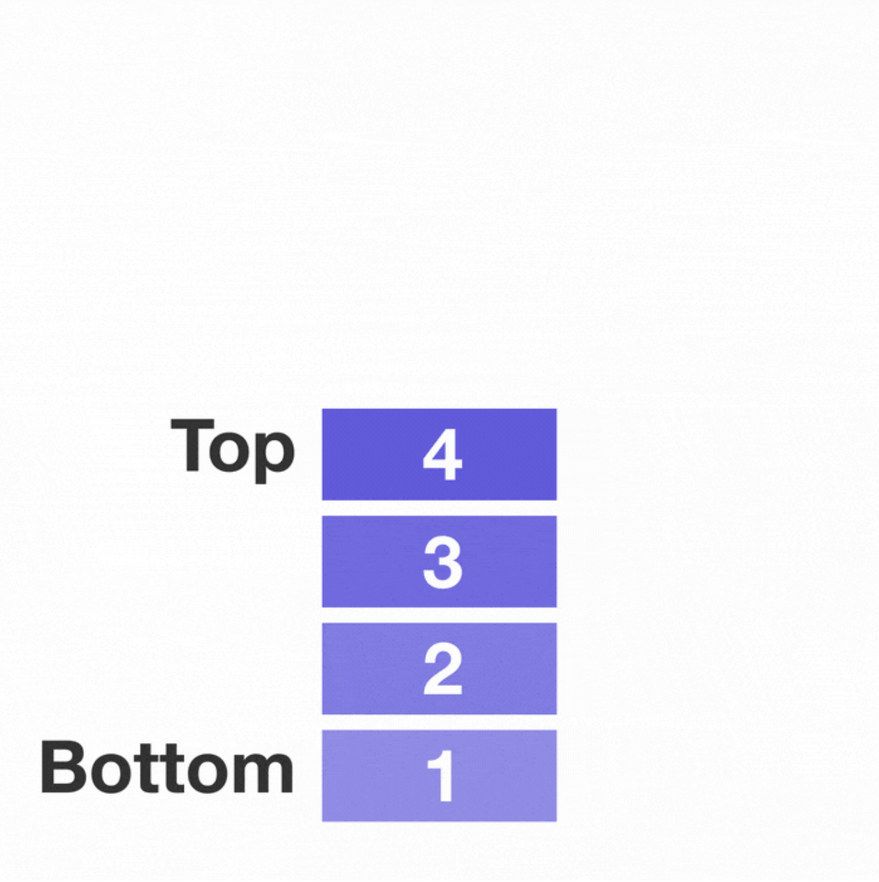


Operations on Stack

push() – (Insertion) Insert element on the stack.

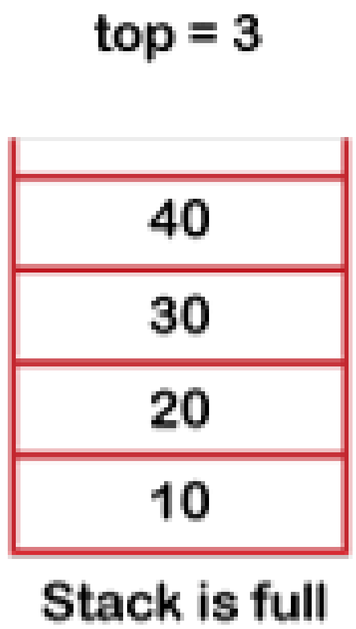
pop() – (Deletion) Removing an element from the stack.





Stack Overflow

push() – (Insertion) Insert element on the stack.



Overflow error

Push Operation Algorithm:

Step-1: If TOP = Max-1

Print "**Overflow**"

Goto Step 4

Step-2: Set **TOP= TOP + 1**

Step-3: Set **Stack[TOP]= ELEMENT**

Step-4: END

Stack Overflow

BENNETT UNIVERSITY
THE TIMES GROUP

push() − (Insertion) Insert element on the stack.

Push Operation Algorithm:

Step-1: If TOP = Max-1

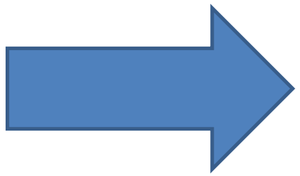
Print "**Overflow**"

Goto Step 4

Step-2: Set **TOP= TOP + 1**

Step-3: Set **Stack[TOP]= ELEMENT**

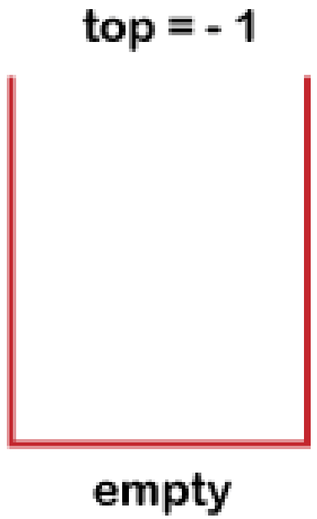
Step-4: END



```
int stack[n], n = 100, val=10, top = -1;
void push(int val, int n)
{
  if(top >= n-1)
    cout<<"Stack Overflow"<<endl;
  else {
    top++;
    stack[top] = val;
  }
```

Stack Underflow

pop() − (Deletion) Removing an element from the stack.



Underflow error

Pop Operation Algorithm:

Step-1: If **top = -1 or top= NULL**

Print "**Underflow**"

Goto Step 3

Step-2: Set **Val= Stack[TOP]**

Step-3: Set **top= top – 1**

Step-4: End

Stack Underflow

pop() − (Deletion) Removing an element from the stack.

Pop Operation Algorithm:

Step-1: If **top = -1 or top= NULL**

Print "**Underflow**"

Goto Step 3

Step-2: Set **Val= Stack[TOP]**

Step-3: Set **top= top – 1**

Step-4: End

```
int stack[n], n = 100, val=10, top = -1;
void pop()
{
  if(top <= -1)
    cout<<"Stack Underflow"<<endl;
  else
 {
    cout<<"The popped element is "<< stack[top] <<endl;
    top--;
  }
}
```

Operations on Stack

BENNETT UNIVERSITY
THE TIMES GROUP

isEmpty(): It determines whether the stack is empty or not.

isFull(): It determines whether the stack is full or not.'

isEmpty () Operation Algorithm:

Step-1: If **top = -1 or top= NULL**

Print "**Underflow**"

Step-2 : Stop

isFull () Operation Algorithm:

Step-1: If **top = Max-1**

Print "**Overflow**"

Step 2: Stop

Operations on Stack

peek() − Accessing the top data element of the stack, without removing it.

Algorithm:

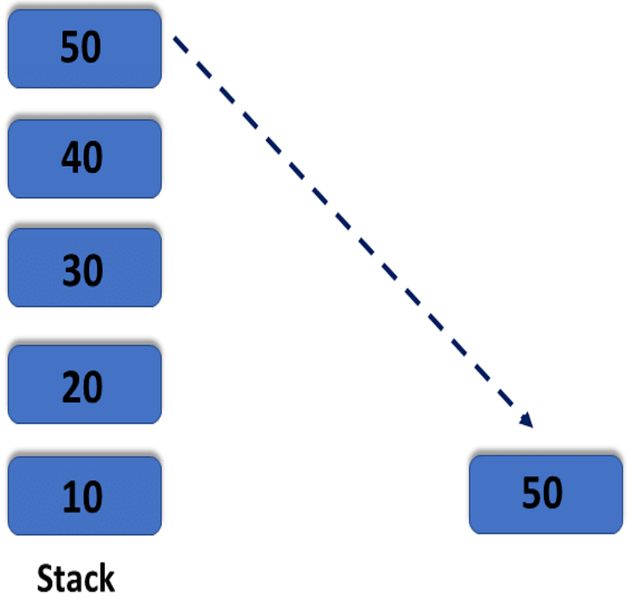
Step-1: If **TOP = NULL**

PRINT “Stack is Empty”

Goto Step 3

Step-2: Return Stack[TOP]

Step-3: END



Operations on Stack

- **count():** It returns the total number of elements available in a stack.
- **change():** It changes the element at the given position.
- **display() or Traverse():** It prints all the elements available in the stack.

Stack Operation: Traversal

BENNETT UNIVERSITY
THE TIMES GROUP

```
void traverse() {
   if(top>= 0) {
      cout<<"Stack elements are:";
      for(int i = top; i>= 0; i--)
         cout<<stack[i]<<" ";
      cout<<endl;
   }
   else
      cout<<"Stack is empty";
}
```

Applications of Stack

BENNETT UNIVERSITY
THE TIMES GROUP

1) Balancing of symbols

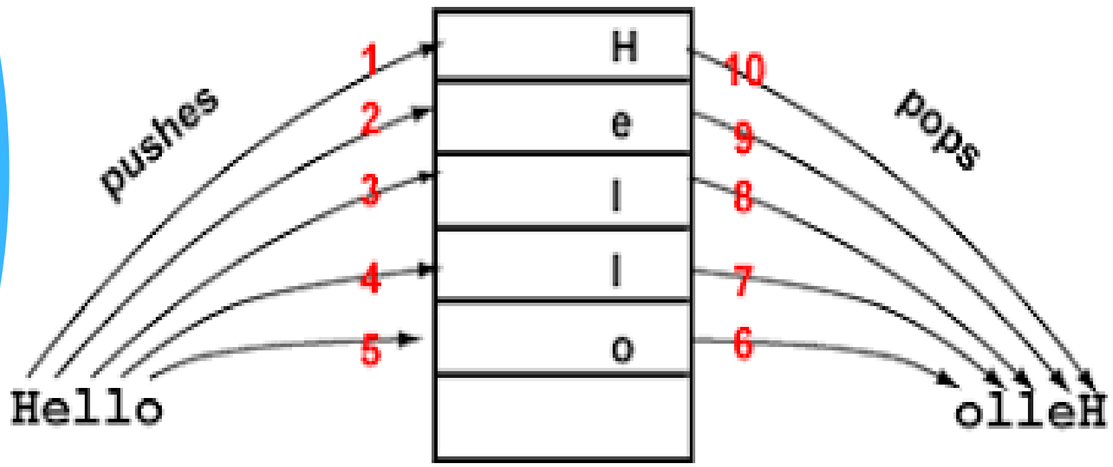
```
int main()
{
   cout<<"Hello";
   cout<<"Students";
}
```

During compilation of program, when the opening braces come, it **push** the braces in a stack, and when the closing braces appear, it **pop** the opening braces from the stack. Therefore, the net value comes out to be zero.

If any symbol is left in the stack, it means that some **Syntax Errors occurs** in a program.

Applications of Stack

2) String reversal

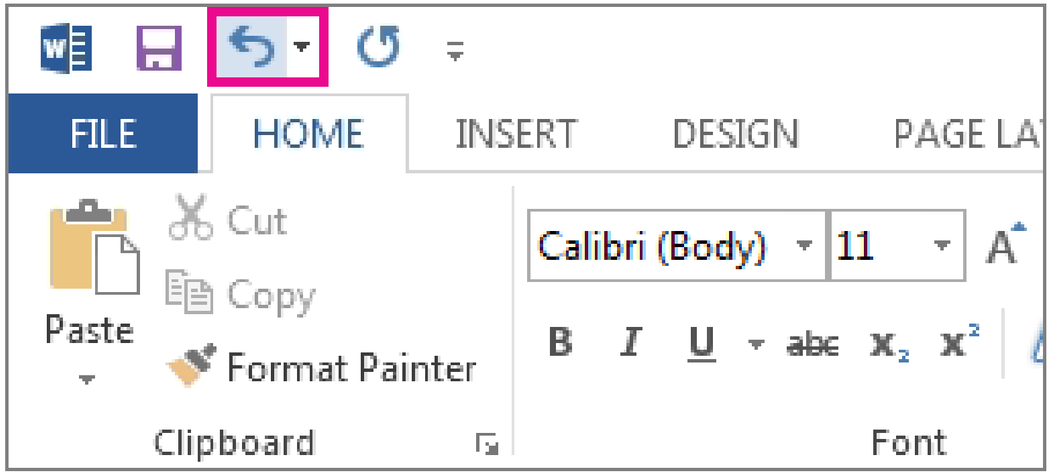


- First, we push all the characters of the string in a stack until we reach the null character.
- After pushing all the characters, we start taking out the character one by one until we reach the bottom of the stack.

Applications of Stack

BENNETT UNIVERSITY
THE TIMES GROUP

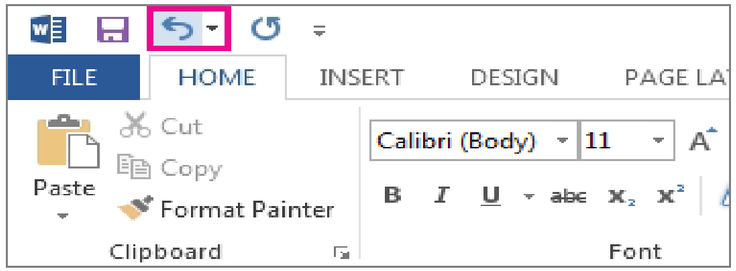
2) UNDO/REDO

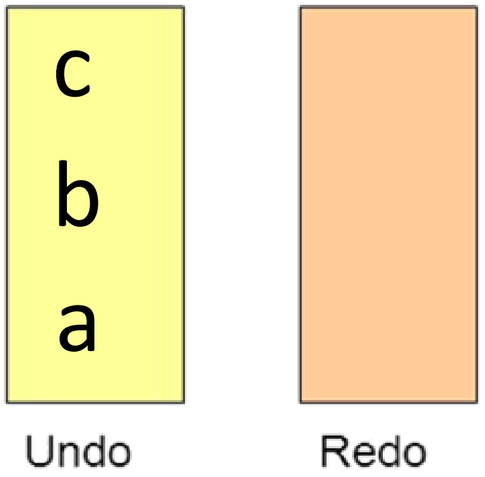


- We have an editor in which we write 'a', then 'b', and then 'c'; therefore, the text written in an editor is abc.
- There are three states, a, ab, and abc, which are stored in a stack.
- **To perform Undo** and **Redo** operations, we need 2 stacks, one stack **shows UNDO state**, and the **other shows REDO state**.

Applications of Stack

2) UNDO/REDO

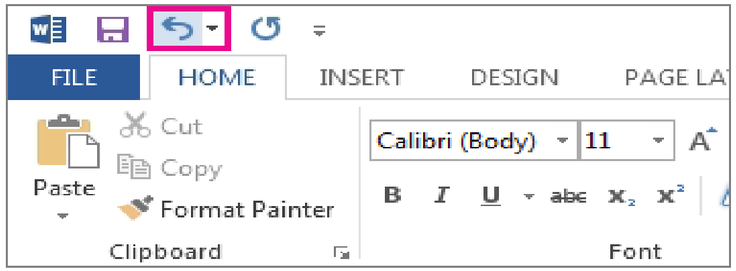


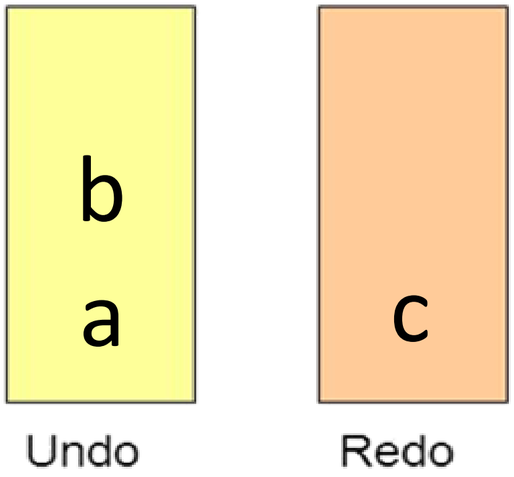


- There are three states, a, ab, and abc, If we want to perform **UNDO operation,** and want to achieve 'ab' state, then we **implement pop operation**.
- Push all operations to **Undo stack.**
- When **undo** is called, pop operations from Undo stack and push it to **Redo stack.**

Applications of Stack

2) UNDO/REDO





- There are three states, a, ab, and abc, If we want to perform **UNDO operation,** and want to achieve 'ab' state, then we **implement pop operation**.
- Push all operations to **Undo stack.**
- When **undo** is called, pop operations from Undo stack and push it to **Redo stack.**
- When **redo** is called, pop operations from Redo stack and push it to **Undo stack**.

Applications of Stack

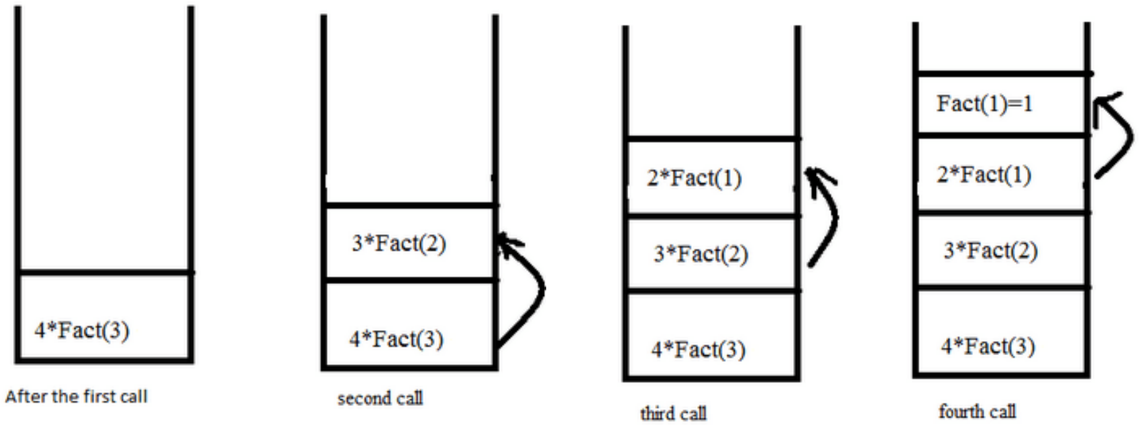
BENNETT UNIVERSITY
THE TIMES GROUP

3) Recursion

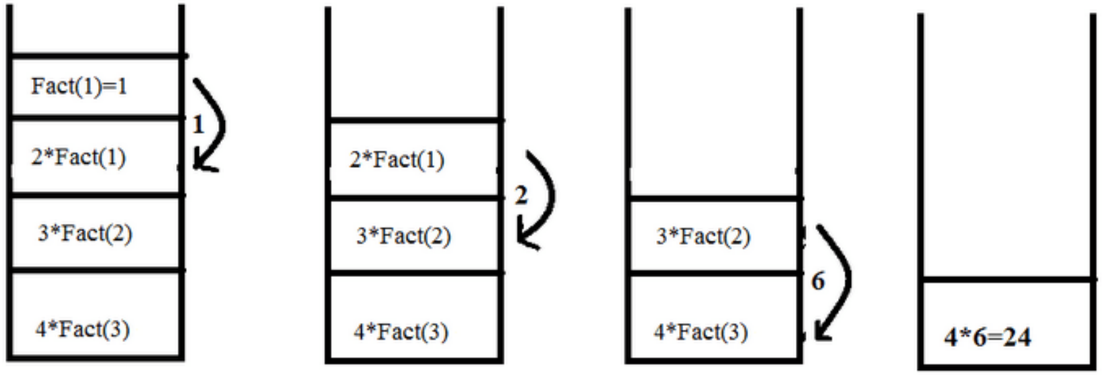
- The recursion means that the function is **calling itself again**.

- To maintain the previous states, the compiler creates a **system stack** in which all the previous records of the function are maintained.

When function call happens, previous variables gets stored in the Stack



Returning value from base case to Caller function

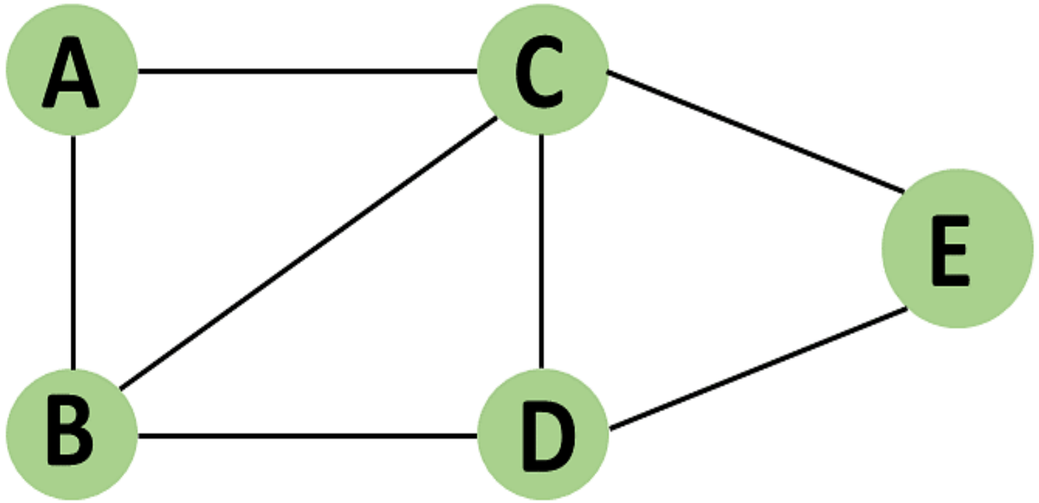


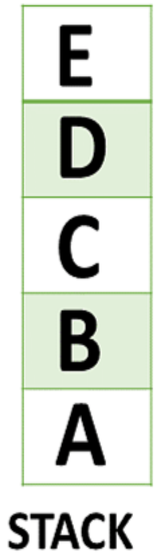
Applications of Stack

BENNETT UNIVERSITY
THE TIMES GROUP

4) DFS(Depth First Search)

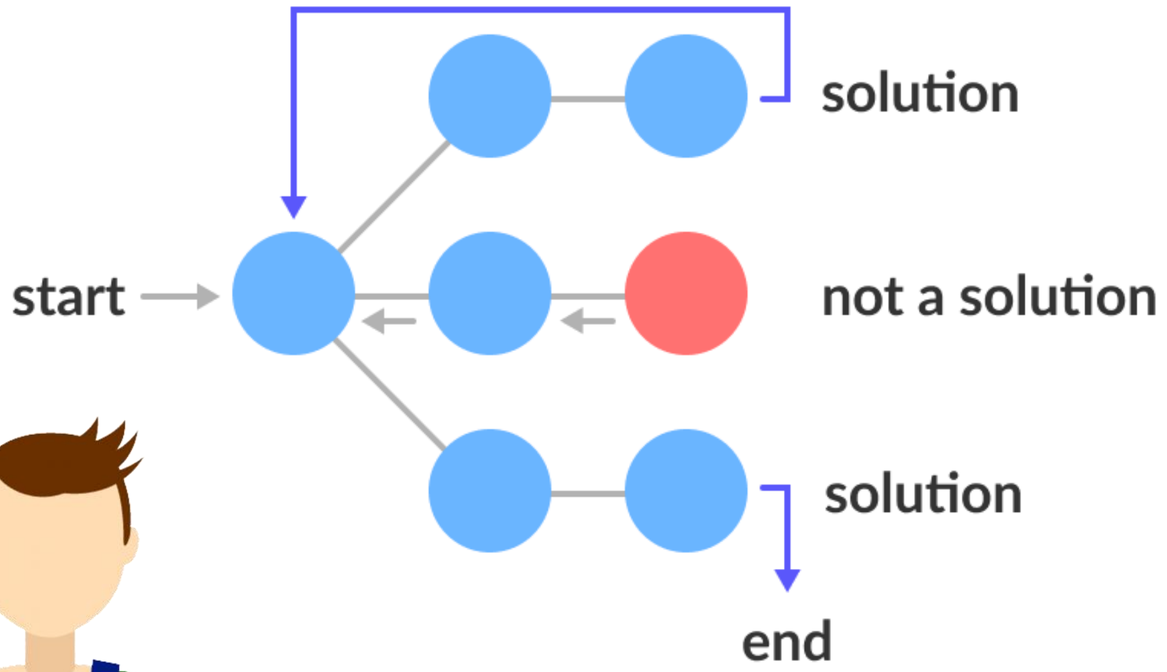
This search is implemented on a Graph, and Graph uses the stack data structure.





Applications of Stack

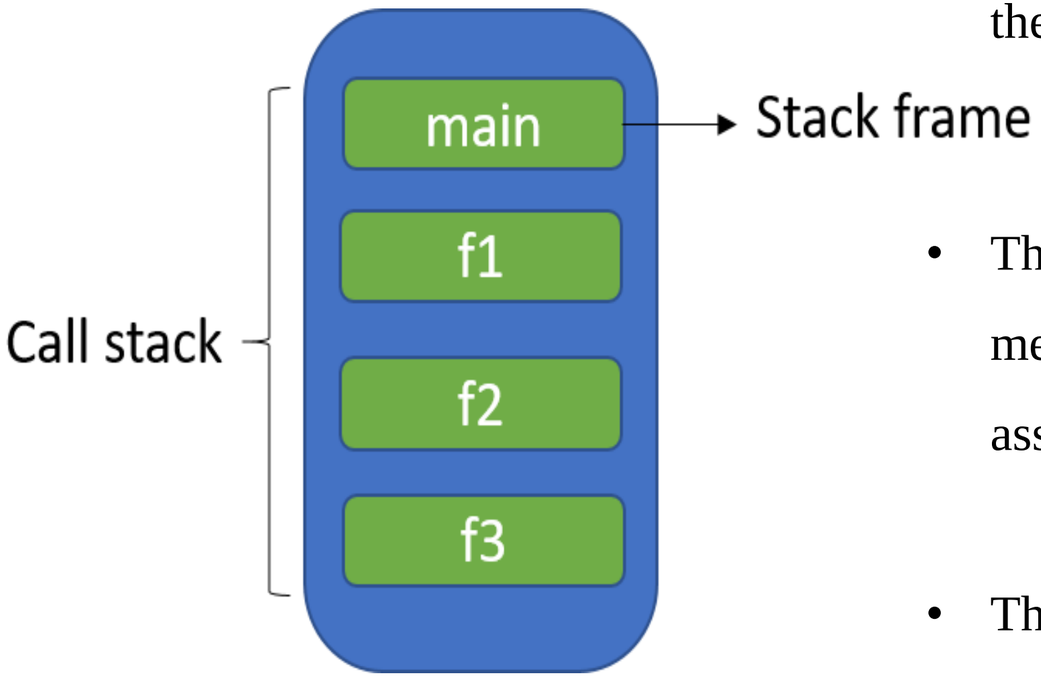
5) Backtracking



- Exploring a route from **home to Bennett University**.
- If we are moving in a particular path, and we realize that we come on the wrong way.
- In order to come at the beginning of the path to explore another path, we use the stack data structure.

Applications of Stack

6) Memory management



- When the function is created, all its variables are assigned in the stack memory. When the function completed its execution, all the variables assigned in the stack are released.
- The memory is assigned in the contiguous memory blocks. The memory is known as stack memory as all the variables are assigned in a function call stack memory.
- The memory size assigned to the program is known to the compiler.

Applications of Stack

BENNETT UNIVERSITY
THE TIMES GROUP

7) Expression Conversion

Ex- 2 + 4

Stack is also used for **expression conversion**. This is one of the most important applications of stack.

Ex- 2 + 4 → 24+

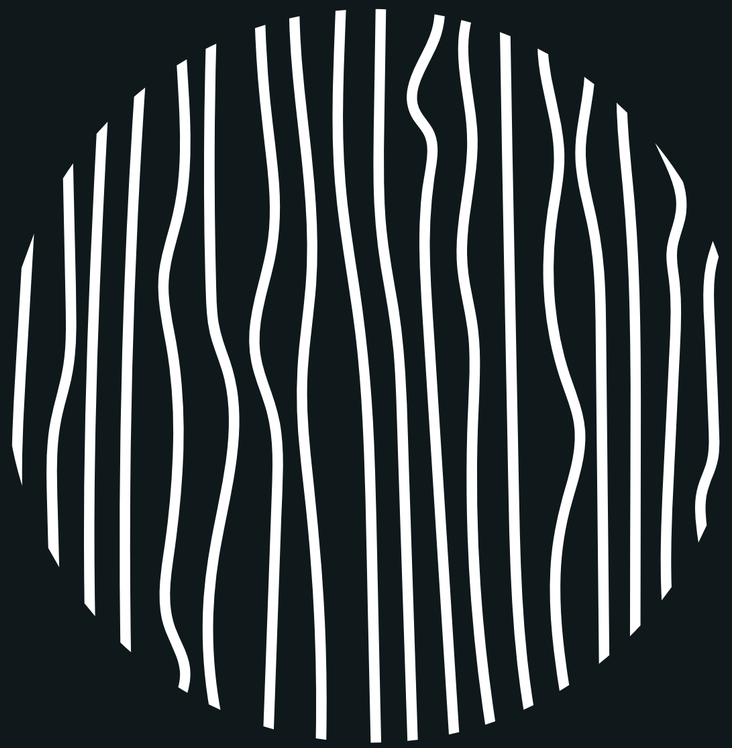
(Infix Notation) (Postfix Notation)

Next Week Readings

BENNETT UNIVERSITY
THE TIMES GROUP

- Polish Notation
- Infix-Postfix Conversion
- Postfix Evaluation

BENNETT UNIVERSITY
THE TIMES GROUP



Any Queries?

Office MCub311
Discussion Time: 3-5 PM
Mob: +91-8586968801