Stack

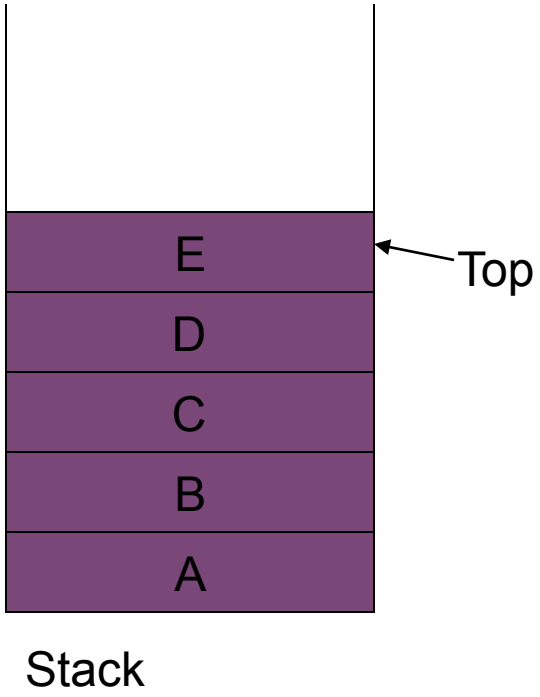
bitCode technologies

learn.innovate.share

Introduction to Stack

- A Stack is an ordered collection of items into which new items may be inserted and from which items maybe deleted at one end called the top of stack.
- The definition of the Stack provides insertion and deletion of items. So a stack is dynamically, constantly changing object.
- The definition specifies that a single end of the stack is designated as the stack top.
- New items may be put on top of the stack or items which are at the top of the stack may be removed.

The Stack Structure



- While adding the elements to stack A is added first then B, C, D, E and along with that the top of the stack keeps growing.
- Initially the Top of the stack is -1.
- While deleting the items from the stack E is deleted first and the D, C, B, A and the stack Top keeps decrementing.

Stack Operations

- You can perform following operations on the Stack.
  - Push
  - Pop
  - StackEmpty
  - StackFull

Stack Operations

Push:

The Items are put on the Stack using Push function.

If S is the Stack the items could be added to it as

```
S.Push( Item );
```

Stack Operations

Pop:

- The items from the stack can be deleted using the Pop operation.

- Using pop only the topmost element can be deleted.

e.g. S.Pop();

- The pop function returns the item that is deleted form the stack.

Stack Operations

- There is one more operation that can be performed on the Stack is to determine what the top item on the stack is without removing it.
- This operation is written as

  S.Peek();

  It returns the top element of the Stack S.
- The operation Peep is not a new operation, since it can be decomposed into Pop & a Push.
- I = S.Peek(); is equivalent to

  I = S.Pop();

  S.Push(I);
- Like the operation Pop, Peep is not defined for an empty stack.

Facts about Stack!

- There is no upper limit on the number of items that may be kept in a Stack, since the definition does not specify how many items are allowed in the Stack collection.
- If the stack does not contain any element then it is called the 'Empty Stack'.
- Although the push operation is applicable to any stack , the Pop operation can not be applied to the Empty stack.
- So it is necessary to check out whether the stack is empty before Popping the elements from the Stack.

Stack Usage

- Stacks are mainly used to support function call mechanism.
- To support or remove recursion.
- Conversion of Infix expressions to post fix expression, pre fix expression, and their evaluation.

Stack Implementation

Stack as an Object

State:

The data, Stack is holding. The stack Top position.

Identity:

Every stack will have a name & location.

Behavior:

Push the elements on Stack.

Pop the elements from Stack.

Responsibility:

Manage the data in last in first out fashion.

Stack Implementation

- Stack can be implemented in two ways
  - Using an Array
  - Using Linked list representation

Class Stack

```
class Stack {

  int * arr;
  int top;

Public:

  Stack();
  Stack( int );
  void push( int data );
  int pop();
  int StackFull();
  int StackEmpty();

};
```

Stack Implementation Using Linked List

- The Stack may be represented by a linear linked list.
- The operation of adding an element to the front of a linked list is quite similar to that of pushing an element onto a stack.
- In both the cases the element is added as the only immediately accessible item in a collection.
- A Stack can be accessed only through it's top element, and a list can be accessed through the pointer to the first element.

List as a Stack

- The operation of removing the first element from a Linked List is analogous to popping a stack.
- In both cases only immediately accessible item of a collection is removed from that collection, and the next item becomes the immediately accessible.
- The first node of the list is the top of the stack.
- The advantage of implementing a stack as a linked list is that stack is able to grow and shrink to any size.
- No space has been pre allocated to any single stack and no stack is using the space that it does not need.

Implementation

```
class Stack {

  Node * top;

Public:

  Stack ();
  void push( int data );
  int pop ();
  int stackEmpty ();

};
```