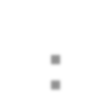
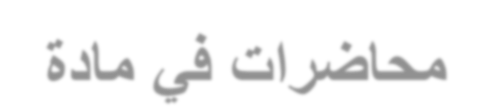
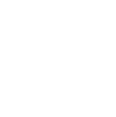
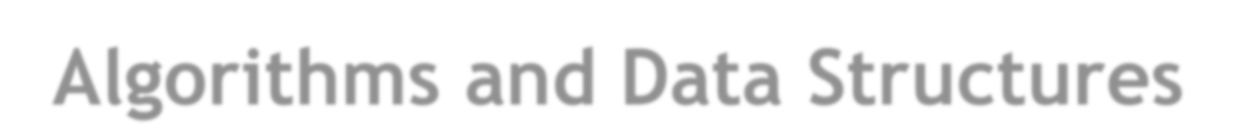
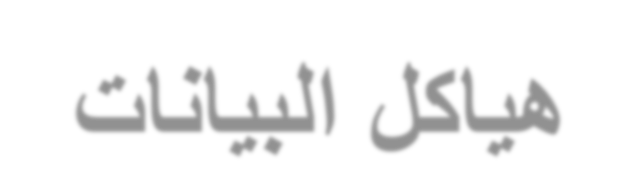
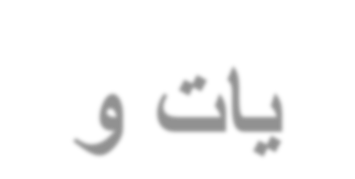
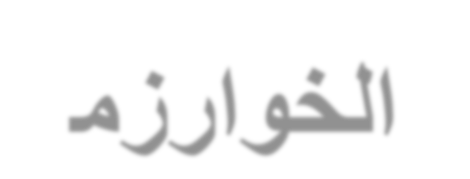
77



**Lecture# 4**

Stacks

Outline

•StackADT.

•Basic operationsof stack

Push,pop, print, etc.

•Implementations of stacks using

Array.

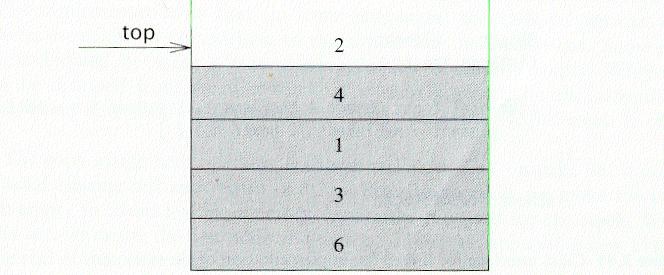
linkedlist.

Stack ADT

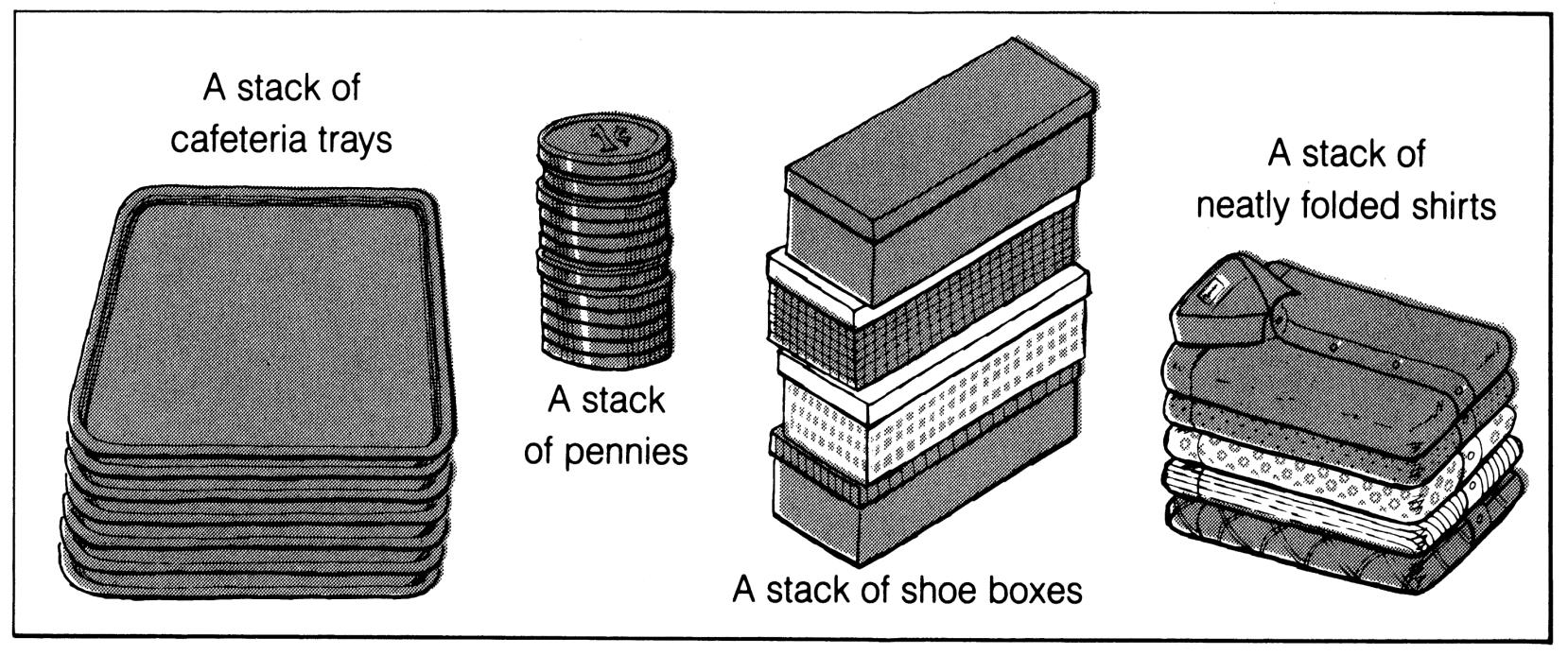
•A *stack* is an abstract data type in which elements are added and

Removed from only one end.

This end is called *top*.



Real-lifestacks:

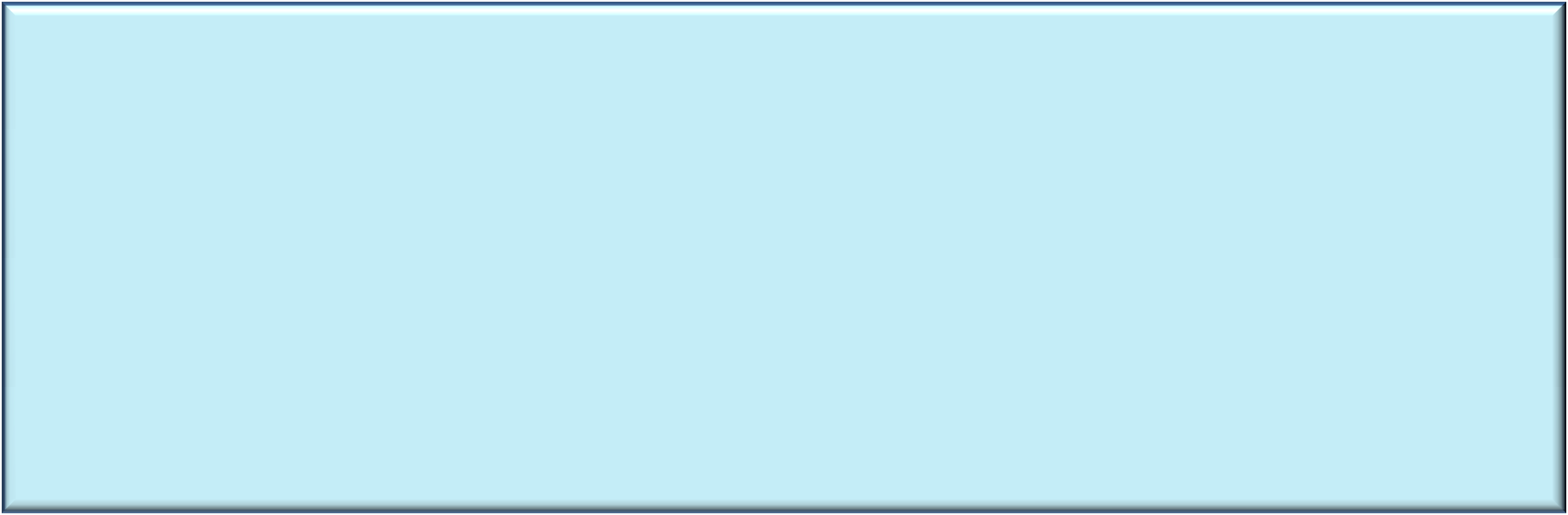


Last In First Out (LIFO)

•Stacks are known as LIFO (Last In, First Out) lists.

The last element inserted will be the first to be retrieved.

Atop



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | E | top |  | | |
|  |  |  | C | top |  | D  C | top |  | D  C |  |  | D  C | top |
| B | top |  | B |  |  | B |  |  | B |  |  | B |  |
| A |  |  | A |  |  | A |  |  | A |  |  | A |  |

STACKSIZE-1

STACKSIZE-2

STACKSIZE-1

STACKSIZE-2

STACKSIZE-1

\TACK\IZE-1

6

5 55

*top* ~44 4

|  |
| --- |
| 98 |
| -76 |
|  |
| 43 |
| 15 |
| 23 |
| 15 |
| 22 |
| 41 |
| 34 |

|  |
| --- |
|  |
|  |
|  |
|  |
| +------!  1--1  1--1  1--1  1--1 |

|  |  |
| --- | --- |
|  |  |
|  |
| . |
|  |
|  |
| -23 | |
|  | 15 |
| 22 |
| 41 |
| 34 |

3 3 3

22 2

1 1

1

0

00

*AStackwith5ele111e11ts A/111/stack*

*(top=./,co1111t=5)(top=STACKSIZE-1,count=ST.4CKSIZEJ*

*AnemptyStack*

*(top=-1,count=O)*

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.

empty stack

pushan element

pushanother

pop

top

top

top

A



B

top

A A

Implementation of Stacks

•Any list implementation could be used to implement a stack, such as:

Arrays (static: the size of stack is given initially).

Linked lists (dynamic: never become full).

•Let’s see how to use an array to implement a stack.

**Stack**class

**classStack{**

**public:**

**Stack(intsize=10); //constructor**

**~Stack(){ delete[]values;} //destructor**

**boolIsEmpty(){return(top==-1);}**

**boolIsFull(){return(top==maxTop);}**

**doubleTop(); //retrievethetopelement**

**voidPush(constdoublex);**

**doublePop();**

**voidDisplayStack();**

**private:**

**intmaxTop; //maxstacksize=size-1 inttop; //currenttopofstack**

**double\*values; //elementarray**

**};**

•Attributes of **Stack**:

**maxTop**: the max size of stack.

**top**: the index of the top element of stack.

**values**: point to an array which stores elements of stack.

•Operations of **Stack**:

**Is Empty**: return true if stack is empty,return false otherwise.

**Is Full**: return true if stack is full, return false otherwise.

**Top**: return the element at the top of stack.

**Push**: add an element to the top of stack.

**Pop**:delete the element at the top of stack.

**Display Stack**: print all the data in the stack.

Create Stack

•The constructor of **Stack**:

Allocate a stack array of **size**.By default,

**size= 10**.

Initially**top** is set to**-1**.It means the stack is empty.

When the stack is full,**top** will have its maximum value,i.e.

**size–1**.

**Stack::Stack(intsize/\*=10\*/){ values=newdouble[size]; top=-1;**

**maxTop= size-1;**

**}**

Although the constructor dynamically allocates the stack array,the stack is still static.The size is fixed after the initialization.

Push Stack

•**voidPush(const double x)**

Push an element on to the stack.

Note:**top**always representstheindex of thetopelement.Before pushinganelement,increment**top**.

**voidStack::Push(constdoublex){**

**if(IsFull())//ifstackisfull,printerror cout<<"Error:thestackisfull."<<endl;**

**else**

**{**

**}**

**}**

**top++;**

**values[top]=x;**

•**doublePop()**

PopStack

Popandreturn theelementat thetopof thestack.

Don’tforgotto decrement**top**, afterpoppinganelement, decrement**top**.

**doubleStack::Pop(){**

**if(IsEmpty()){//ifstackisempty,printerror cout<<"Error:thestackisempty."<<endl;**

**return-1;**

**}**

**else{**

**}**

**}**

**doublex=values[top];**

**top--;**

**returnx;**

•**doubleTop()**

StackTop

Returnthetopelementofthestack.

Unlike **Pop**, this function does not remove the top element.

**doubleStack::Top(){**

**if(IsEmpty()){**

**cout<<"Error:thestackisempty."<<endl;**

**return-1;**

**}**

**else**

**}**

**returnvalues[top];**

Printing all the elements

•**voidDisplayStack()**

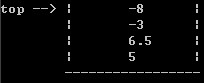
Print all theelementsin thestack.

**voidStack::DisplayStack(){**

**cout<<"top-->";**

**for(inti= top;i>=0;i--)**

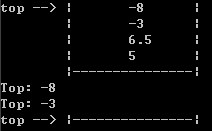
**cout<<"\t|\t"<<values[i]<<"\t|"<<endl;**



**cout<<"\t|---------------|"<<endl;**

**}**

Using **Stack**



Result

|  |  |  |
| --- | --- | --- |
|  |  | |
| **intmain(void){**  **Stackstack(5); stack.Push(5.0); stack.Push(6.5); stack.Push(-3.0);** |  |  |
| **stack.Push(-8.0);**  **stack.DisplayStack();**  **cout<<"Top:"<<stack.Top()<<endl;**  **doublez;**  **z= stack.Pop();**  **cout<<"Top:"<<stack.Top()<<endl;**  **while(!stack.IsEmpty())z=stack.Pop();**  **stack.DisplayStack();**  **return0;**  **}** | |  |