Data Structures and Algorithms

Application of Stacks

Stack operations

- Common operations:
  - ✓ Push – add a given element to the top of the stack.
  - ✓ Pop – read the topmost element and delete it from the stack.
  - ✓ Peek – return the topmost element.
  - ✓ Size – return the number of elements in the stack.

STACK APPLICATION: POSTPONING DATA USAGE

- **ARITHMETIC STATEMENT**
  - ✓ Example:

    A * B + C (How computers generate it???)
  - ✓ Arithmetic expression written in INFIX as above example.
  - ✓ However compiler change to POSTFIX/PREFIX for calculating purposes.
  - ✓ Three different formats:

    Infix: **A+B** the operator appears between two operands.

    Prefix: **+AB** the operator appears before the two operands.

    Postfix: **AB+** the operator appears after its two operands.

STACK APPLICATION: POSTPONING DATA USAGE

priority	arithmetic
4	^, NOT
3	*,/, AND,DIV, MOD
2	+, -, OR
1	=, <, > , <=,>=,<>

a-b*(c+d)/f^g*h

Ch	ST1	ST2
a	a	.....
-	a	-
b	ab	-
*	ab	- *
(	ab	- *(
c	abc	- *(
+	abc	- * (+
d	abcd	-*(+
)	abcd+	- *
/	abcd+*	-/
f	abcd+*f	-/
^	abcd+*f	-/ ^
g	abcd+*f g	-/^
*	abcd+*fg^/	-*
h	abcd+*fg^/h	- *
...	abcd+*fg^/h*-	....

Scan the infix expression from left to right

Test the character

1.If the character is an operand, push it to the ST1.

2. If the character is a left parenthesis, push it to the ST2.

3. If the character is a right parenthesis then pop entries from the ST2 and push them to the ST1 until a left parenthesis is popped

4.If the character is an operator

4.1. if the precedence of the scanned operator is > the precedence of the operator in ST2 push it

4.2. else, pop all operators from ST2 which are > or = to in precedence than that of the scanned operator and push them to ST1. after doing that push the scanned operator to the ST2.

At end of the expression. Pop all entries that remain in the ST2 and push them to ST1.

Exercises

Q1: Convert the infix expression 10 + (6*3 – (16/2^3)*4)*7

Into postfix form showing stacks status after every step?

Q2: Convert the infix expression ((A – (B + C)) * D) / (E +F)

Into postfix form showing stacks status after every step?

ARRAY P2

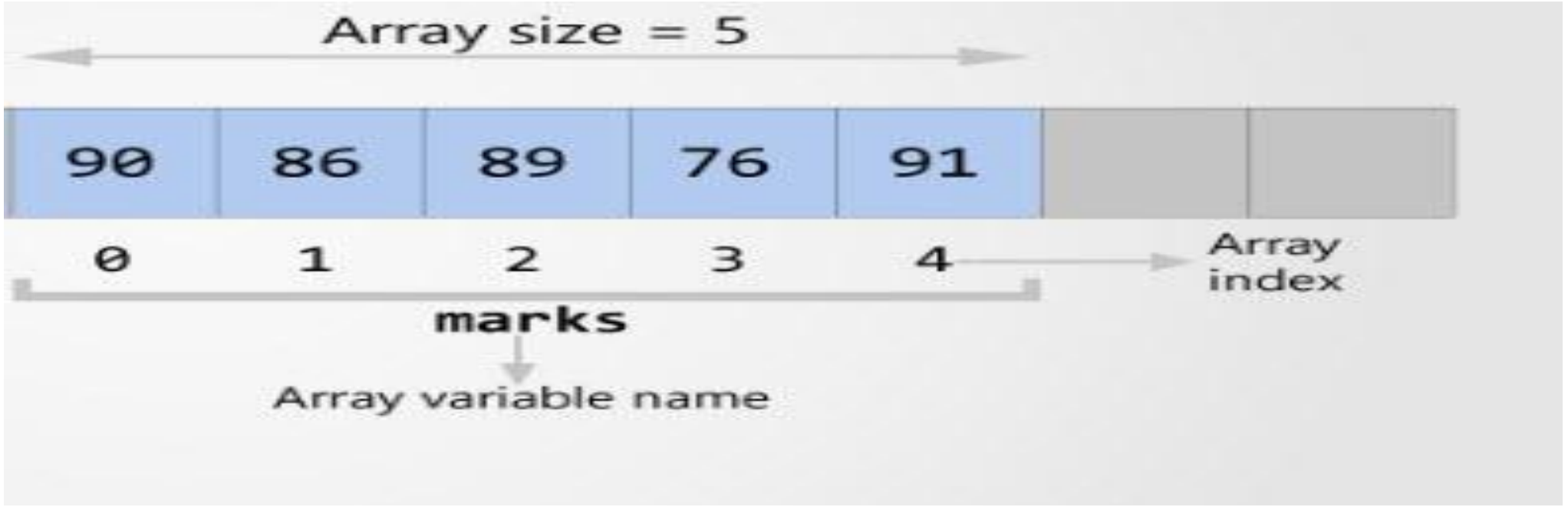
INTRODUCTION TO DATA STRUCTURE

Concept of Data Structures:

- A data structure is a way of storing and organizing data in a computer so that it can be used efficiently.
- Efficient data structures are a key to designing efficient algorithms.
- Often a carefully chosen data structure will allow the most efficient algorithm to be used. Thus, algorithms and data structures go hand in hand.

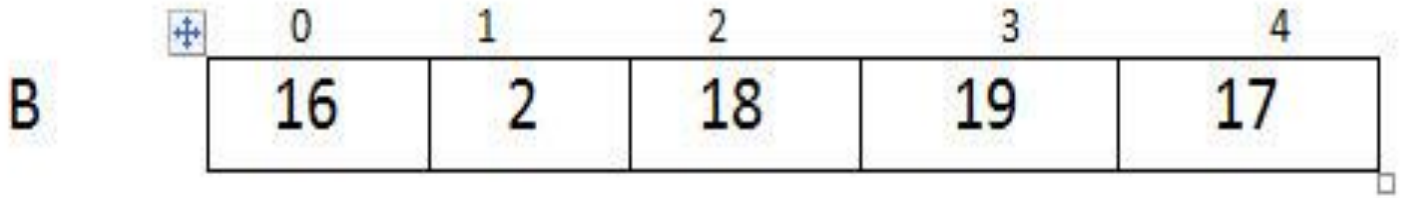
What is array?

- An array is a group of consecutive memory location with same name and data type.



Initializing array

- Int b[5]= { 16,2,18,19,17}



- Int b[0]=16
- Int b[3]=19

Array in C++

Float a[6]= {12.2,4.5,66.8,13.8}

int s[3]= {6,77,22,54} ✗

char d[5]= {‘a’, ’b’,, ’c’, ’d’} ✗

Int a[]= {3, 5,20,8}

Array in C++

Char k[]= {‘A’ ,’B’, ‘C’, ’D’}

flot z[]; ✕

Array in C++

```
Void main()
{
   int grad [] = {0, 1, 2};
    int scor [3];
    scor= grad;
}
```

Error: cannot assign one array to another

Array in C++

```
#include “iostream.h”
Void main ()
{
   Int count= 0;
   char capt[10];
For (int i=0; i< 10; i+)
    cin>> capt[i];
 for (int i=0; i<10; i++)
     if (capt[i]>= ‘A’ && capt[i] <= ‘Z’)
      count++;

cout<<count;
}
```

Input

C B d Z# 6 w * 8 H

Output

4

Writ a program to store 10 characters in an array and print the count of digits

```
#include “iostream.h”
Void main ()
{
      int count= 0;
     char capt[10];
    For (int i=0; i< 10; i+)
        cin>> capt[i];
   for (int i=0; i<10; i++)
        if (capt[i]>= ‘0’ && capt[i] <= ‘9’)
          count++;

cout<<count;
}
```

Input

C B d Z # 6 w * 8

Output

2

Writ a program to store 10 characters in an array and change each capital letter to small letter

```
#include “iostream.h”
Void main ()
{
     char capt[10];
     for (int i=0; i< 10; i+)
     cin>> capt[i];
    for (int i=0; i<10; i++)
          if (capt[i]>= ‘A’ && capt[i] <= ‘Z’)
               capt[i] = capt-‘A’ +’a’;
   for (int i=0; i< 10; i+)
   cout<<capt[i];
}
```

Input

A B d Z # 6 w * 8 H

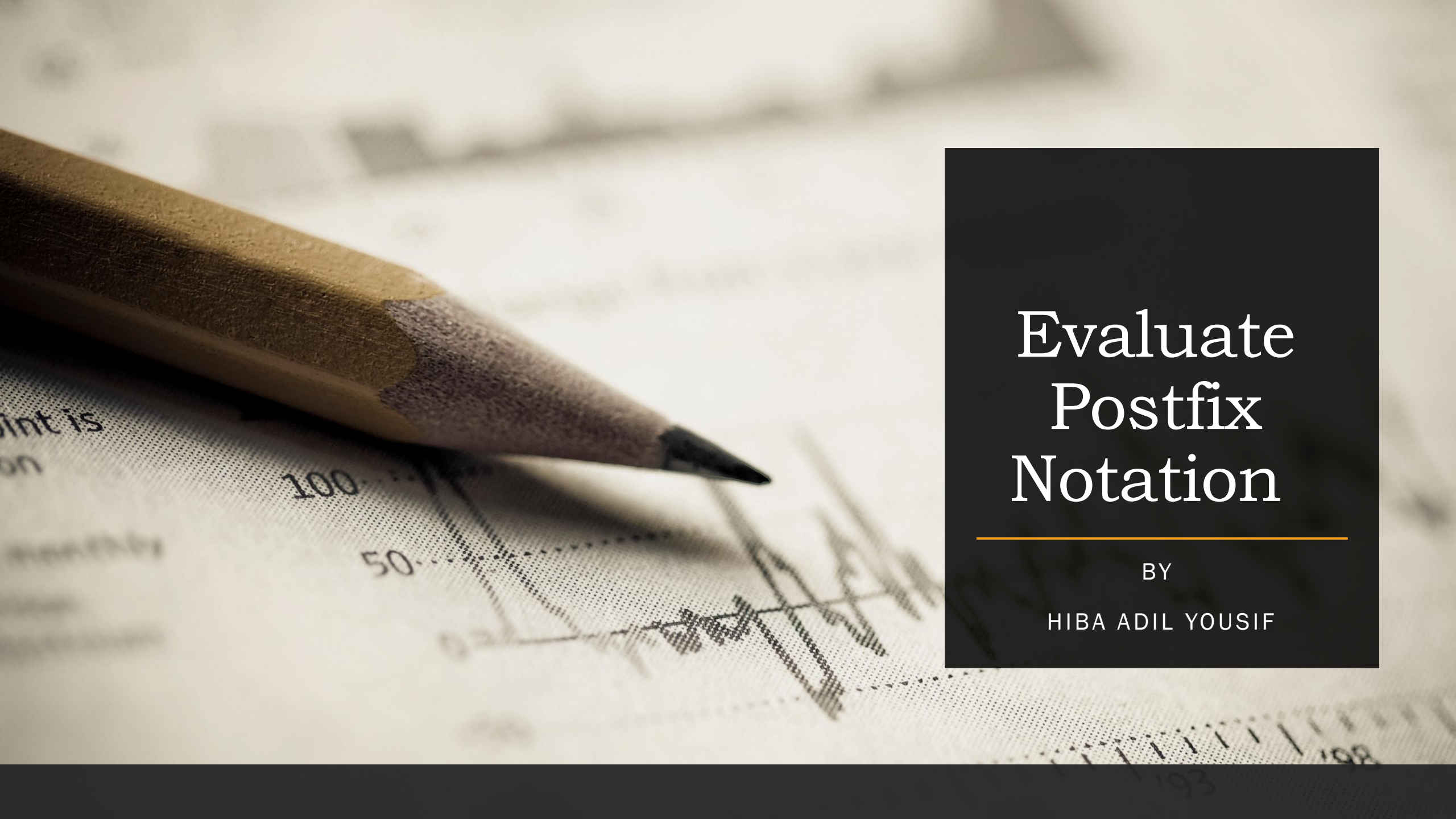
Output

a b d z # 6 w *8 h

Array in C++

Exercise

Write a program to store 30 characters in an array and print the counts of digits, capital letters, small letters, and special characters.



Evaluate Postfix Notation

BY

HIBA ADIL YOUSIF

ALGORITHMS

1. Start reading from left to right and push the operands into stack.

2. If an operator occurs, then pop last two operands from stack and perform respected operation.

3. (operand 2) operator (operand 1).

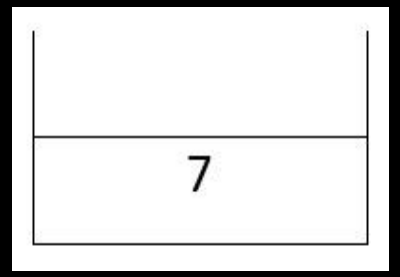
4. Push the result into stack.

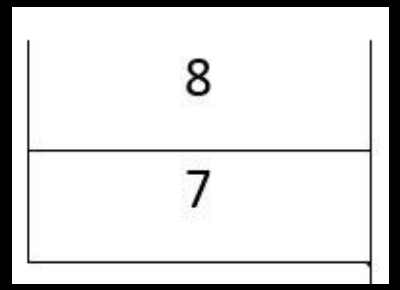
Ex: 7 8 + 6 3 * 2 / -

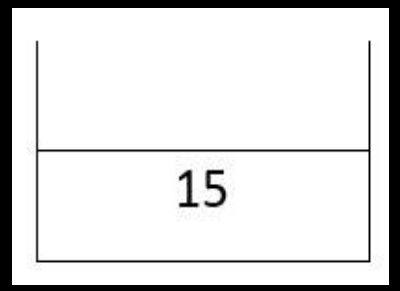
7

8

+





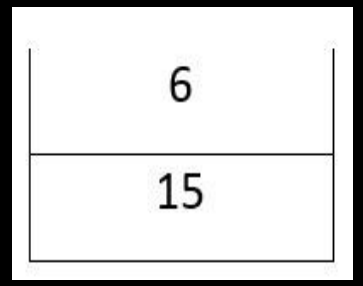


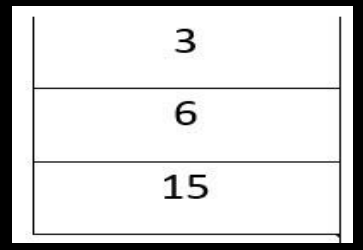
Cont.

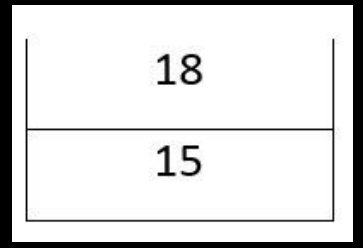
6

3

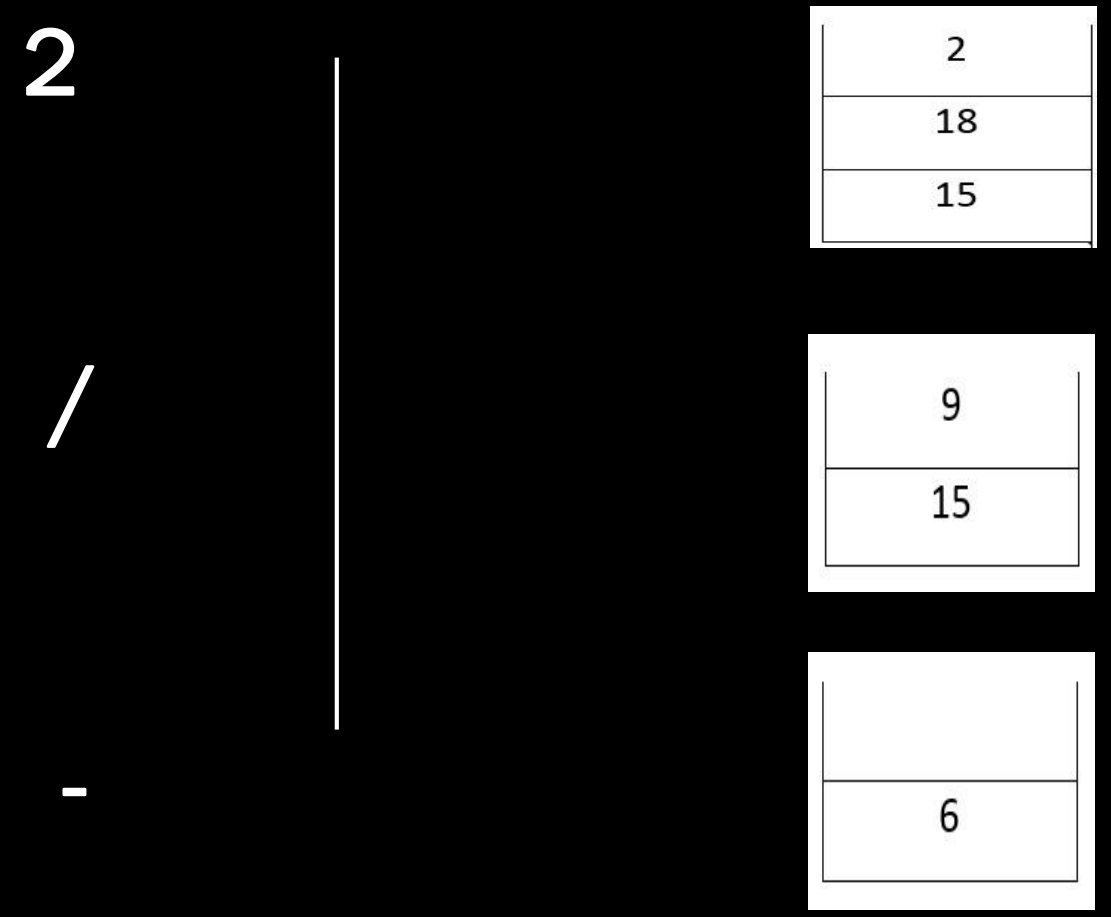
*







Cont.



EXERCISES

Q1: Evaluate the following postfix notation showing status of stack after execution of each operation:

6 2 3 + - 3 8 2 / + * 2 * 3 +

Q2: Given the following expression:

90 – 6 * (2+4) / 3 ^ 2* 9

1.Gonvert the expression to postfix notation using two stacks.

2.Evaluate the postfix notation showing status of stack after execution of each operation.

Thanks

INTRODUCTION TO DATA STRUCTURE

المرحله الثانيه مسائي
م.م هبه عادل يوسف

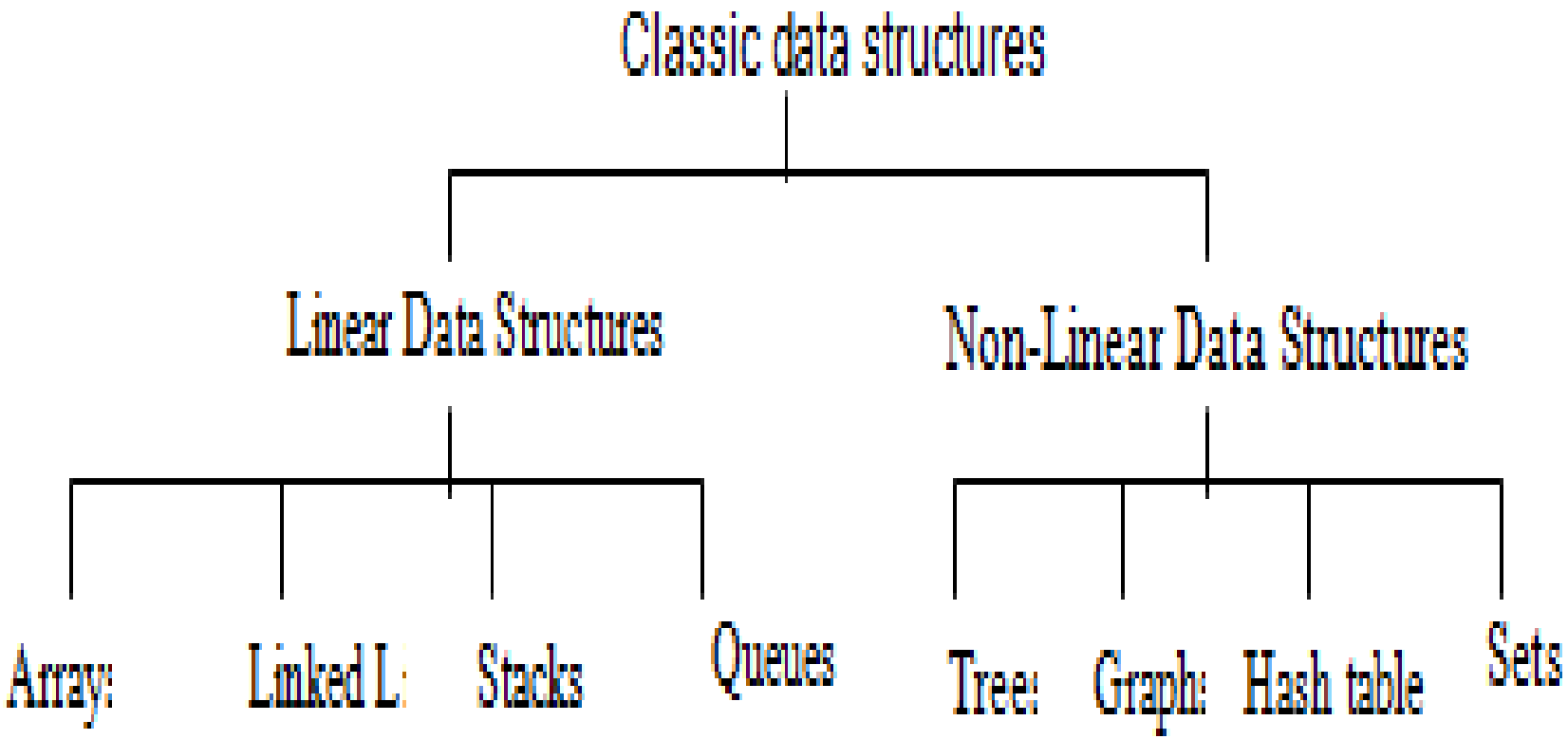
هياكل البيانات والخوارزميات

INTRODUCTION TO DATA STRUCTURE

Concept of Data Structures:

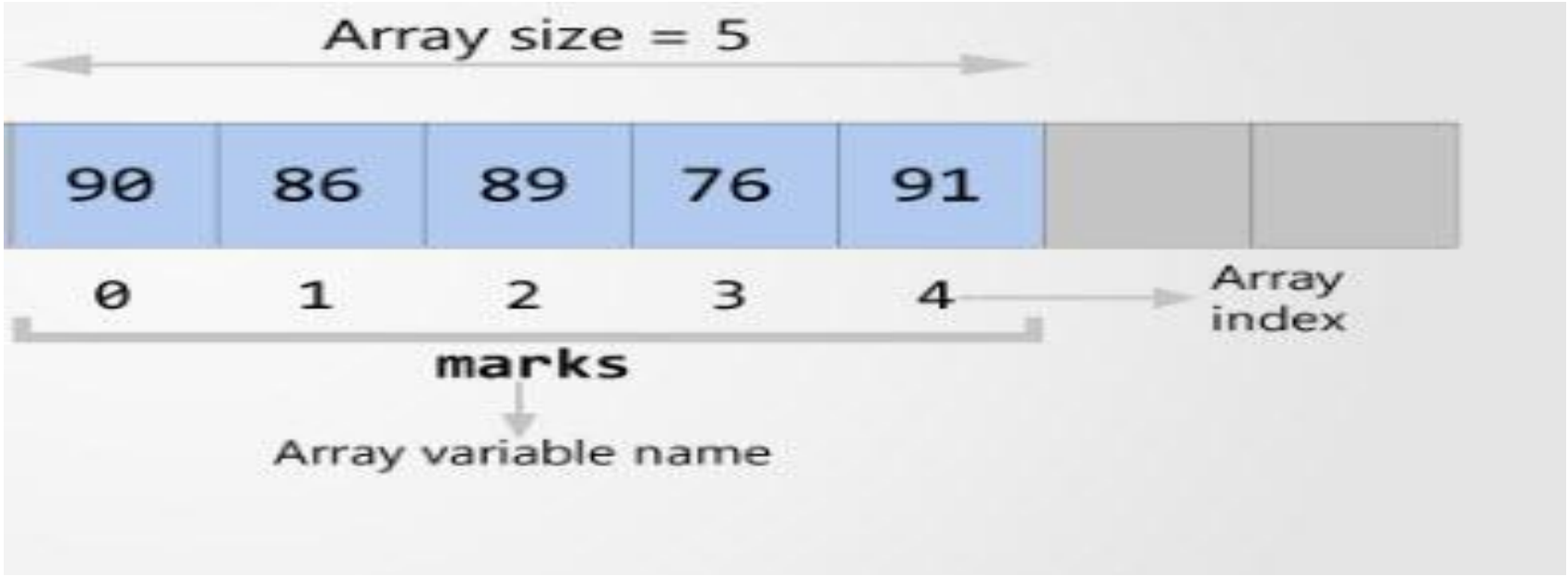
* A data structure is a way of storing and organizing data in a computer so that it can be used efficiently.
* Efficient data structures are a key to designing efficient algorithms.
* Often a carefully chosen data structure will allow the most efficient algorithm to be used. Thus, algorithms and data structures go hand in hand.

Types of Data Structures



What is array?

* An array is a group of consecutive memory location with same name and data type.



Declaration of an array

Like a regular variable, an array must be declared before it is used

Data_type array_name [size];

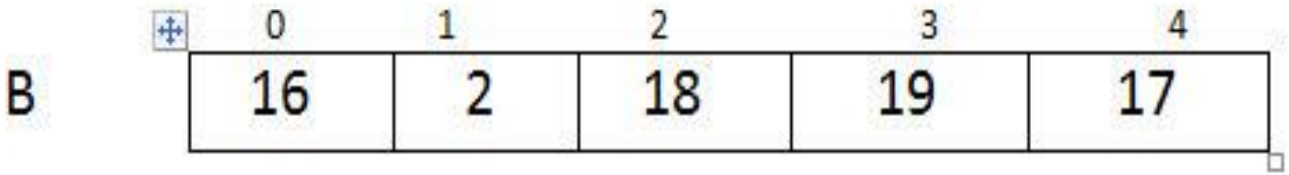
* Type is a valid type like (int, float, char…)
* Name is valid identifier

Examples:

* int a[10];
* float deg[5];
* char a[15];

Initializing array

Int b[5]= { 16,2,18,19,17}



Int b[0]=16

Int b[3]=19

Array in C++

```
#include <iostream.h>
Void main ()
{
  Float a[6];
  For (int i=0; i<6; i++)
     cin >> a[i];

  For (int i=0; i<6; i++)
    if (a[i] <50.0)
       a[i]=0.0;
For (int i=0; i<6; i++)
    cout <<a[i] <<"\n";
}
```

Input

5.7 66.8 90.2 40.0 65.8 33.3

Output

0.0
66.8
90.2
0.0
65.8
0.0

Array in C++

```
#include <iostream.h>
Void main ()
{
Float a[6], sum=0.0;
For (int i=0; i<6; i++)
        cin>> a[i];
For (int i=0; i<6; i++)
        sum= sum + a[i];
    cout << “sum=”<<sum;
}
```

Input

15.6 83.3 90.0 35.5 60.0 45.3

Output

sum=329.7

Advantages of array

* Array can store a large number of values with single name.
* Array is used to process many value easily and quickly.
* The values stored in an array can be stored easily.
* The search process can be applied on arrays easily.

Exercise

Q1: write a program to store 10 integers in an array and print the multiplication of all its integers.

THANK YOU

Linked List

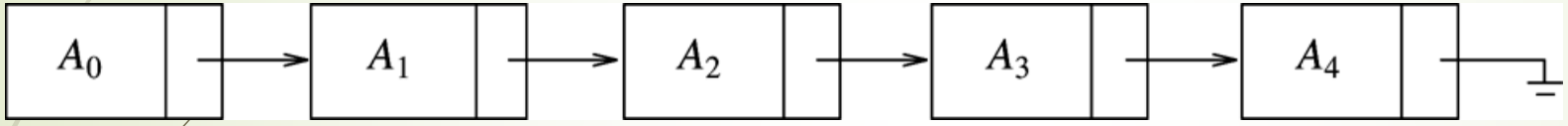
deletion

Deletion

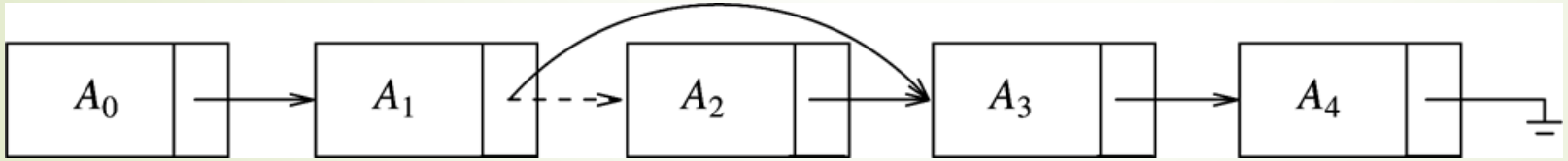
- Delete Node
  - ✓ Logically remove a node from the linked list by changing various link pointers and then physically deleting the node from dynamic memory.
  - ✓ Delete can be done at the first node, at the last node or at a specified position of the list.

Deletion

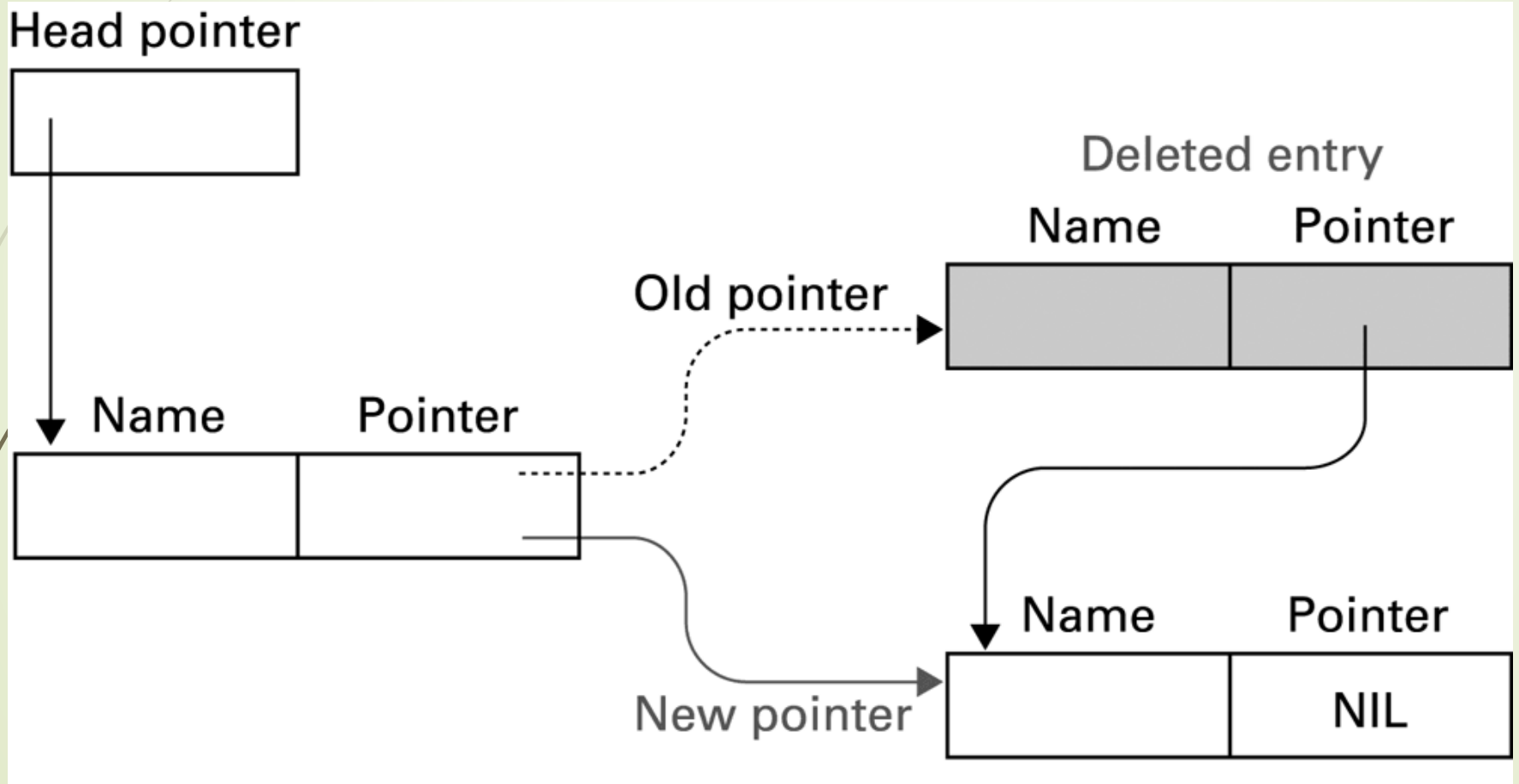
- $A_0, A_1, A_2, \ldots, A_{N-1}$



- **Operation: deletion**

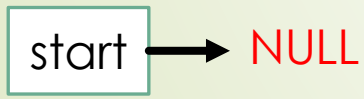


Deletion


Head pointer
Deleted entry
Name
Pointer
Old pointer
Name
Pointer
Name
Pointer
New pointer
NIL

Deletion

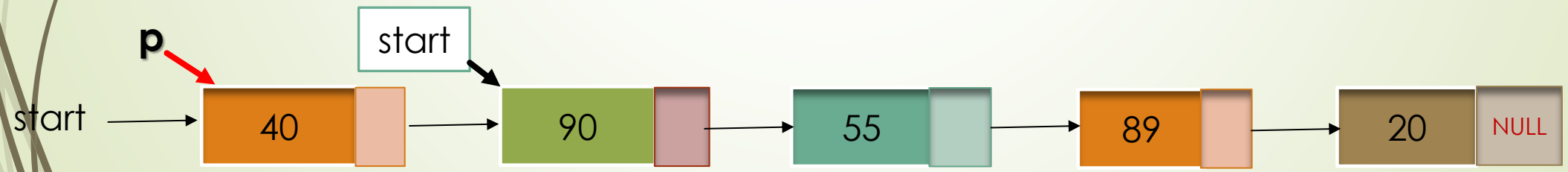
```
Struct node {
        int data;
        node *next;
   }   *start;

Void delete (int n)
{
  node *p, *p1;
  if(start ==NULL)
   return;
```



Deletion

```
Struct node {
        int data;
        node *next;
   }   *start;

Void delete (int n)
{
   node *p, *p1;
   if(start ==NULL)
    return;
   p=start;
```

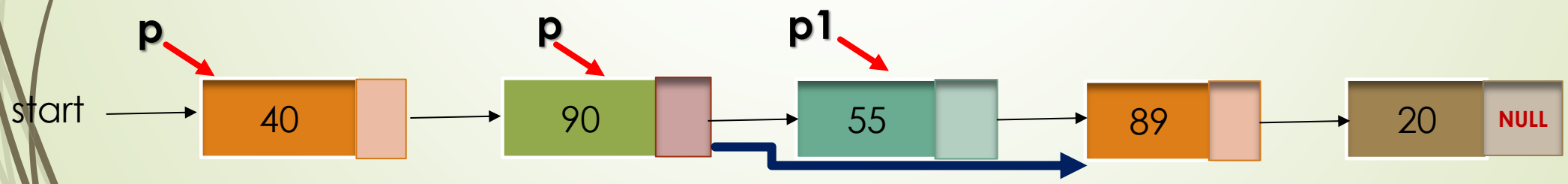
```
if (n==1) {
      Start= p->next;
      delete p;
      return;
 }
```



Deletion

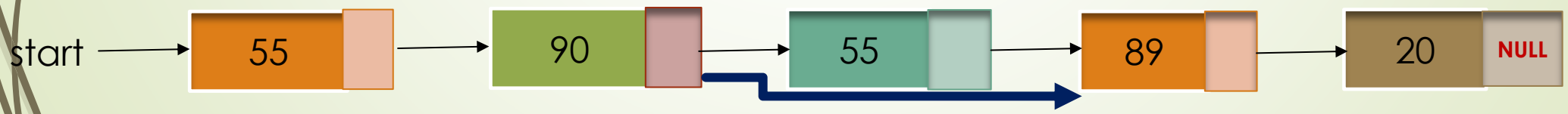
```
Struct node {
        int data;
        node *next;
   }   *start;

Void delete (int n)
{
   node *p, *p1;
   if(start ==NULL)
    return;
   p=start;
```

```
if (n==1) {
     Start= p->next;
     delete p;
     return;
 }

For ( i=1;  p!=NULL &&  i< n-1;  i++)

          p=p->next;

 If(p==NULL  | |  p->next ==NULL)
          return;

 p1=p->next;
 p->next= p1->next;
 delete p1;
 }
```



Exercise

- write a function to delete the first occurrence of a node with a given key in linked list.
- Write a complete program to delete a node by calling the above function.

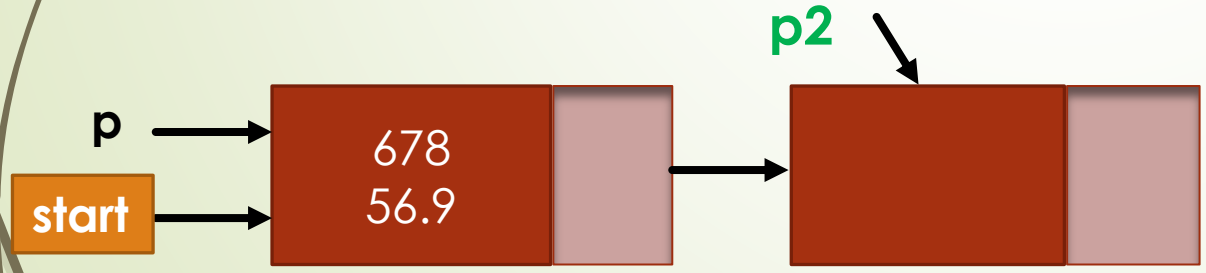


Linked List

Implementation p2

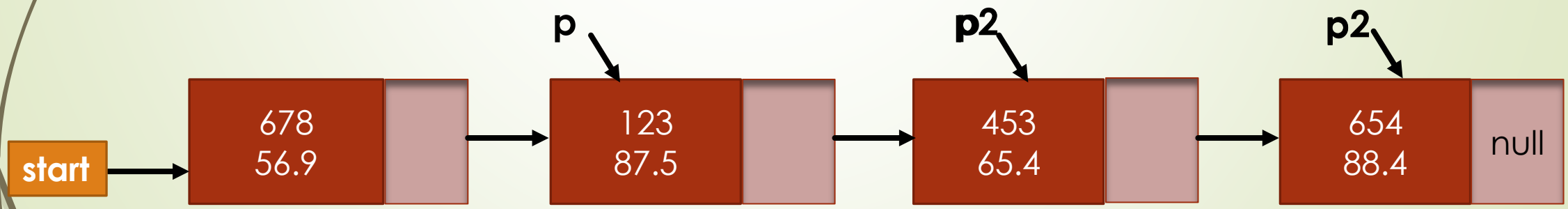
```
struct node {
int stud-id;
float average;
node *next;
};
void main()
{ node *p,*p2,* start;
     int n,i;
     P=new node;
   Start=p;
   cin>>n;
```

```
For(i=1;i<=n;i++) {
    cin>> p -> stud-id;
    cin>> p -> average;
If (i!=n) {

    p2=new node;
    p->next=p2;
    p=p2;

} else
     p2->next=NULL;

   }

}
```



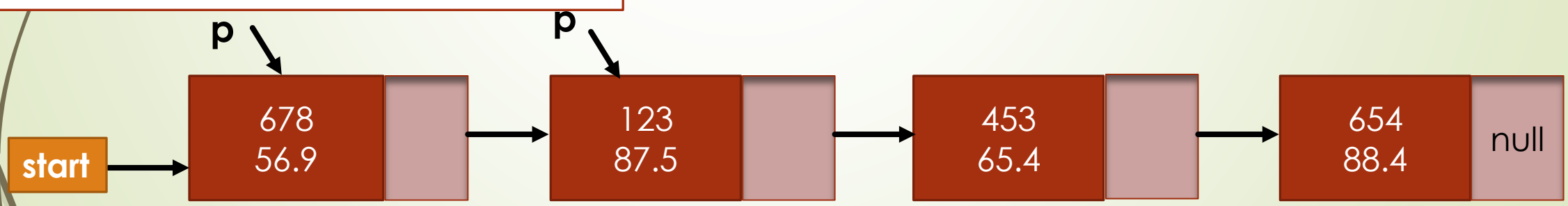
```
struct node {
int stud-id;
float average;
node *next;
};
void main()
{ node *p,*p2,* start;
     int n,i;
     P=new node;
   Start=p;
   cin>>n;
```

```
For(i=1;i<=n;i++) {
    cin>> p -> stud-id;
    cin>> p -> average;
If (i!=n) {

    p2=new node;
    p->next=p2;
    p=p2;

} else
     p2->next=NULL;

  }

}
```



```
void main()
{ node *p,*p2,* start;
      int n,i;
      P=new node;
    Start=p;
    cin>>n;
For(i=1;i<=n;i++) {
     cin>> p -> stud-id;
     cin>> p -> average;
If (i!=n) {

     p2=new node;
     p->next=p2;
     p=p2;

} else
      p2->next=NULL;
}
```

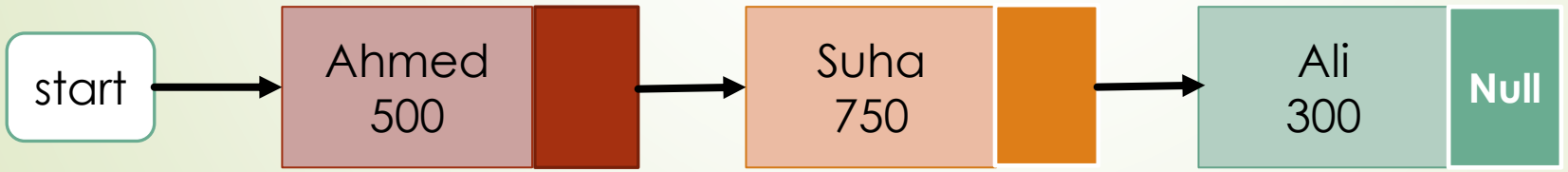
```
P=start
While( p!=NULL)
 {
      cout<<p->stud-id;
      cout<<p->average;
       p=p->next;
  }
}
```



Exercises

- Q1 write a program to create a linked list of n nodes each node containing employee information (name, salary, age). Print the name of employee that has largest salary.

Q2 write a program to create a linked list of n nodes each node containing student information (name, average). Print the names of students that have averages less than 50.

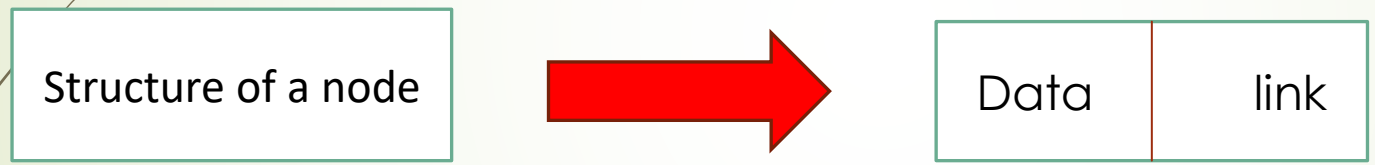


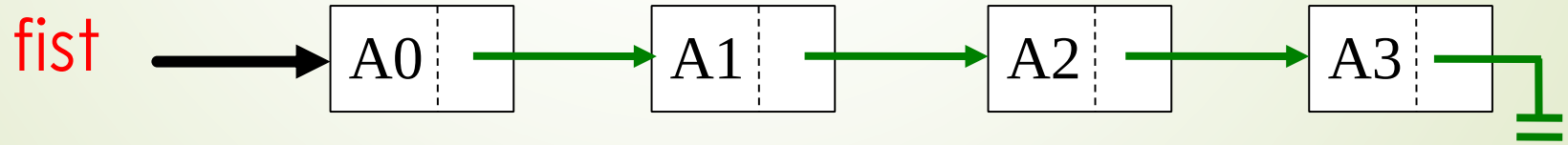
Linked List

Implementation

A List ADT using Linked List

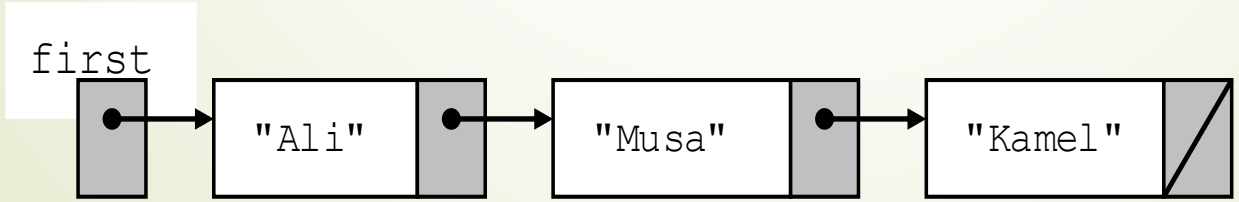
- **Definition**: A collection of nodes that form a linear list structure. Each node is a compound object that stores an element and a reference, to call the next node or another node.



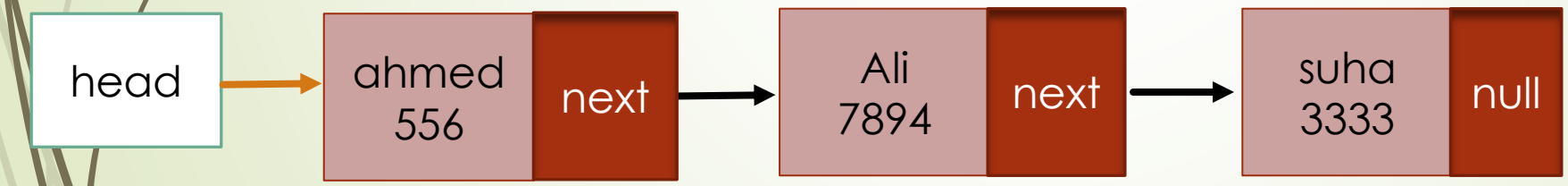


Characteristics

- Insert and delete nodes in any order
- The nodes are **connected**
- Each node has **two** components
  - Information (data)
  - Link to the next node (reference)
- The nodes are accessed through the links between them

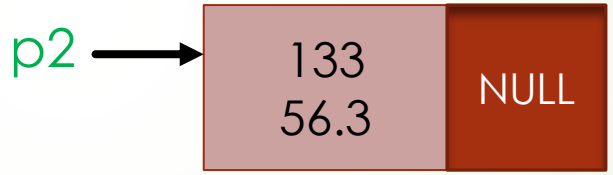


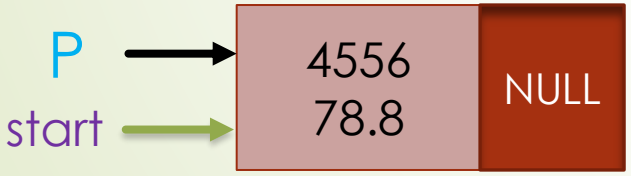
```
struct node{
char name[13];
float salary;
node*next;
};
```



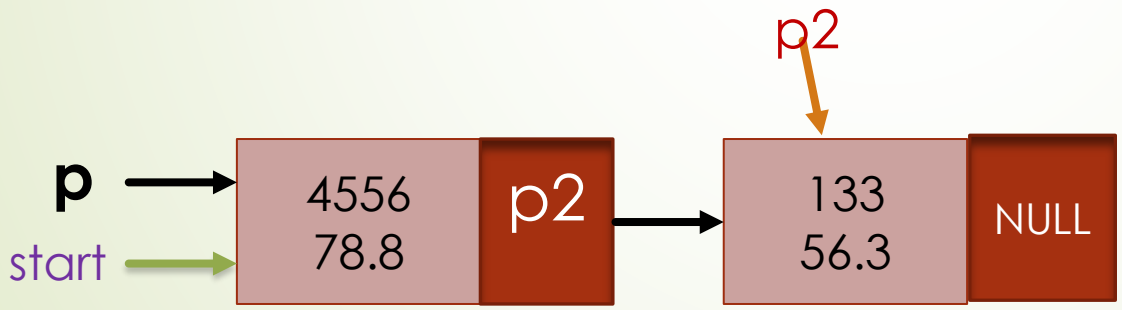
```
struct node{
int stud-id;
float average;
node *next;
};
void main()
{ node *p,* start;
P=new node;
Start=p;
cin>> p -> stud-id;
cin>> p -> average;
P ->next=null;
}
```

```
void main()
{ node *p,*p2,* start;
P=new node;
Start=p;
cin>> p -> stud-id;
cin>> p -> average;
P ->next=null;
P2=new node;
Cin>>p2->stud-id;
Cin>>p2->average;
P2->next=null;
```

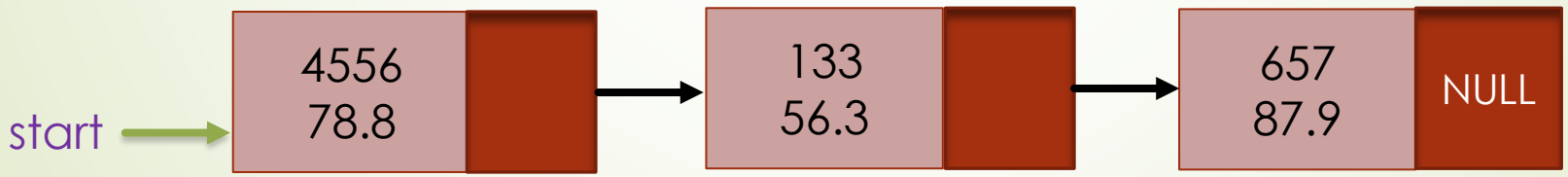




```
void main()
{ node *p,*p2,* start;
P=new node;
Start=p;
cin>> p -> stud-id;
cin>> p -> average;
P2=new node;
Cin>>p2->stud-id;
Cin>>p2->average;
P2->next=null;
P->next=p2;
```



```
void main()
{ node *p,*p2,*p3,* start;
P=new node;
Start=p;
cin>> p -> stud-id;
cin>> p -> average;
P2=new node;
cin>>p2->stud-id;
cin>>p2->average;
P->next=p2;
P3=new node;
cin>>p3->stud-id;
cin>>p3->average;
P3->next=null;
P2->next=p3;
```



Exercise

- Q1 write a program to create a linked list of 4 nodes, each node containing book information(title, author's name, year of publication)

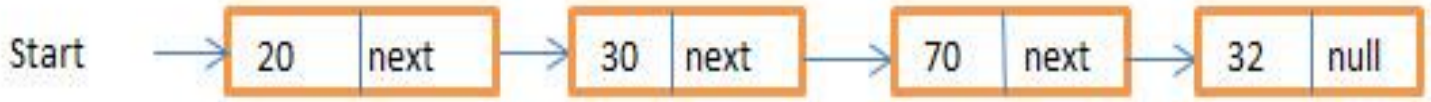
LINKED LIST- INSERTING A NODE

BY
HIBA ADIL

INTRODUCTION

Linked list: is one of the fundamental data structure, and can be used to implement other data structures. In linked list there are different numbers of nodes. Each node is consists of two fields. First field holds the value or data and the second field holds the reference to the next node or Null if the linked list is empty.

Example



OPERATION ON LINKED LIST

- Insertion
- Deletion
- Searching

ADDING AN ELEMENT TO A LINKED LIST

1. Involve two steps:

   - Finding correct location
   - Doing the work to add the node

2. Finding the correct location:

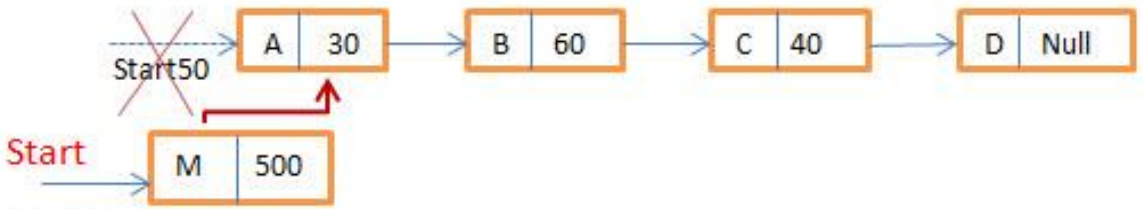
Three possible positions

- The front
- In the middle
- The end

```
Struct node {
int data;
Node *next;
} *start;
```

INSERT AT FIRST

For insert at first the New Node should point to the first node (before insertion) and start should point to the New Node.

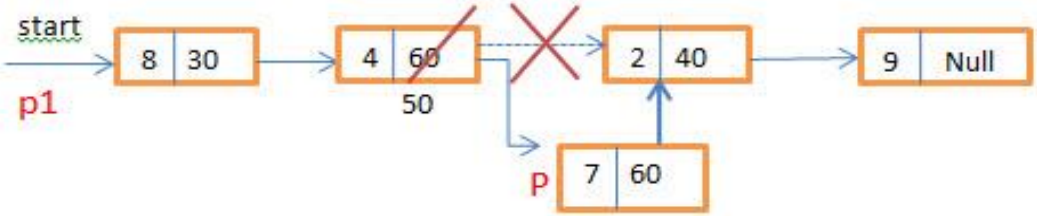


Start=90

```
Void insert_front (int item)
{
Node *p;
P=new node;
P ->data = item;
P ->next = start;
Start = p;
}
```

INSERT AT MIDDLE

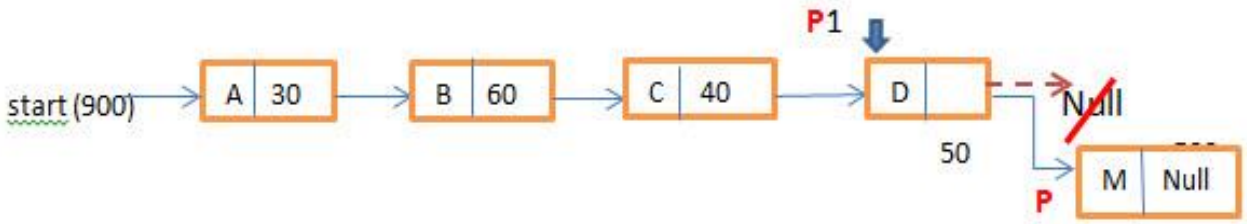
For insert at middle the New Node should point to the node next to it and the previous node should point to the New Node




```
Void insert _ after(int item, int value)
{
Node *p, *p1;
P=new node;
P -> data=item;
P1 =start;
While (p1-> data != value)
P1= p1 -> next;
If (start ==Null || p1== Null) {
Cout<<" given node must not be null
Return; }
P -> next =p1 ->next;
P1 -> next = p;
}
```

INSERT AT LAST

For insert at last the last node (before insertion) should point to the New Node and the "link" part of New Node must be Null




```
void insert_ end (int item)
{
Node *p, *p1;
P= new node;
P -> data = item;
P ->next = Null;
P1 =start;
While (p1 != Null)
P1 = p1 -> next;
P1 ->next = p;
}
```

DISPLAY

```
Void print ()
{
Node *p;
P= start;
While (p != Null) {
Cout << p -> data;
P= p -> next;

}

}
```

PROCESS SUMMARY

1. Create a New Node
2. Apply values to the "Data" and "Link" part
3. Traverse in the linked list until we reach to the desired point
4. Insert node using swaps of link

EXERCISES

Q. writes a program to insert a new node to a linked list by displaying a menu to select one of the insertion methods. For example

1. Insert a node at the front

2. Insert a node after a given node.

3. Insert a node at the end

4. Exit

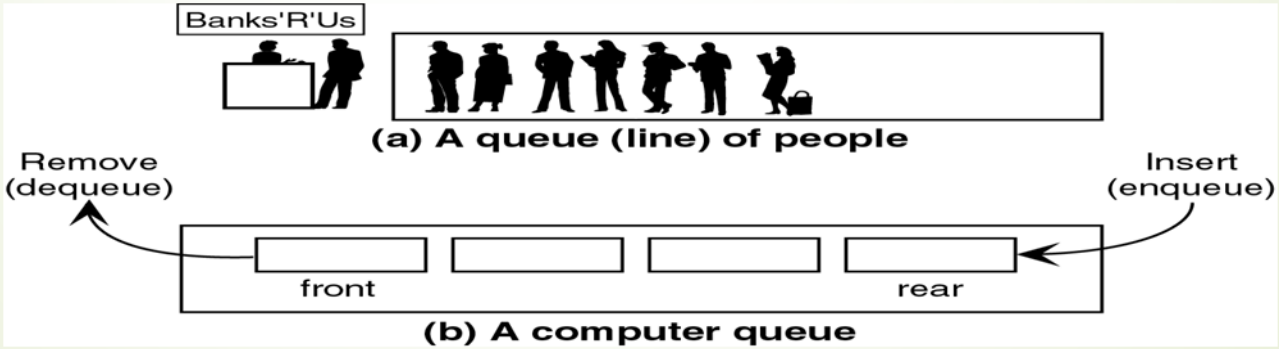
Thanks

CIRCULAR QUEUE

Data structures and algorithms

CIRCULAR QUEUE

- Queue is linear list in which data can only be inserted at one end, called the **rear (back)**, and deleted from the other end, called the **front**.
- First-In-First-Out (FIFO) concept.



CIRCULAR QUEUE

- Enqueue: add data to element in queue

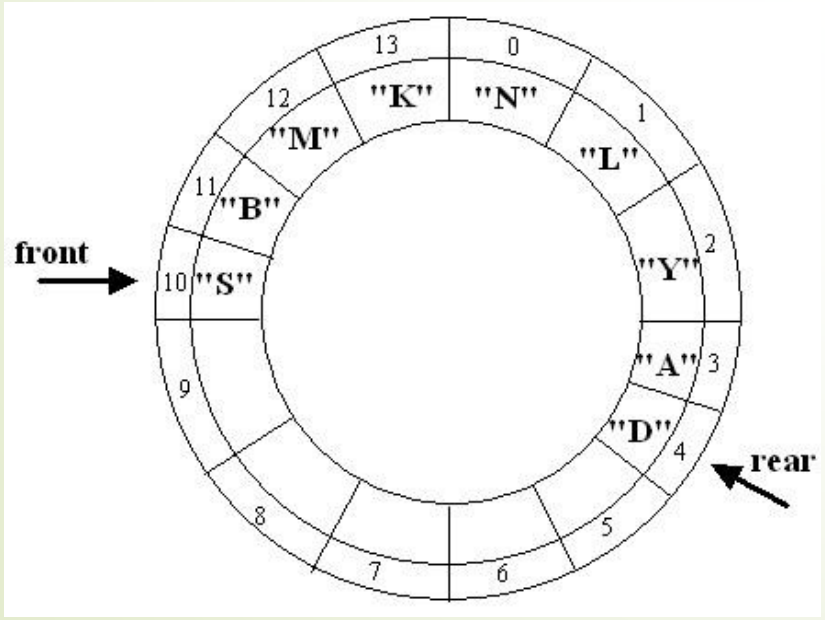
Will increased value of rear by 1

- Dequeue: take data from element in queue

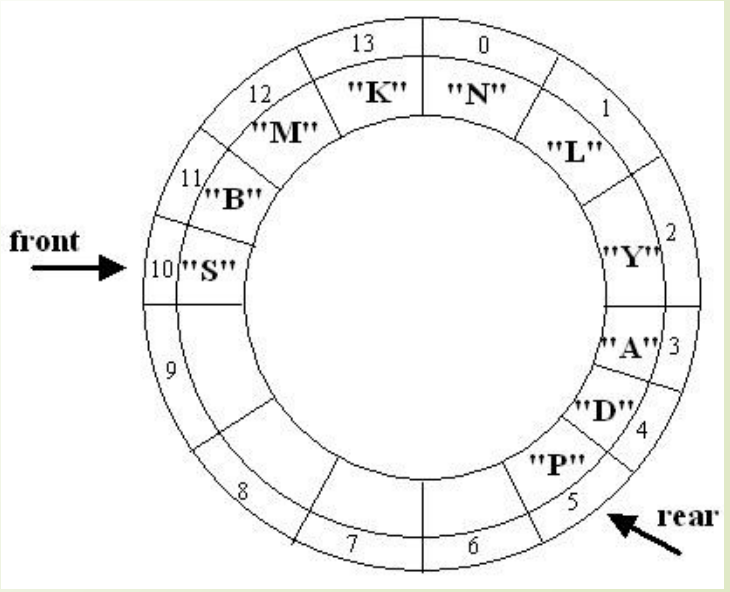
Will increased value of front by 1

CIRCULAR QUEUE

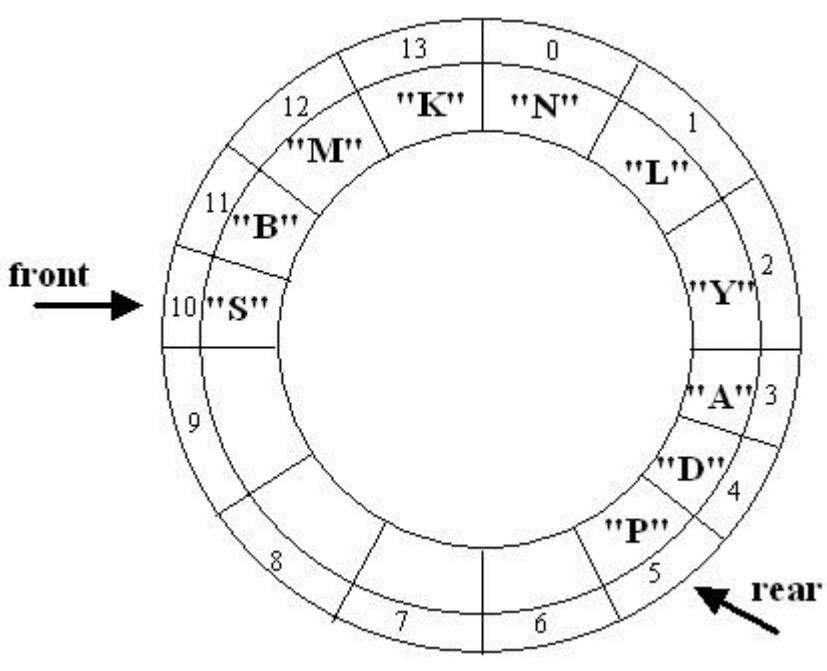
- When the queue reaches the end of the array, it "wraps around" and the rear of the queue starts from index 0. The figure below demonstrates the situation.



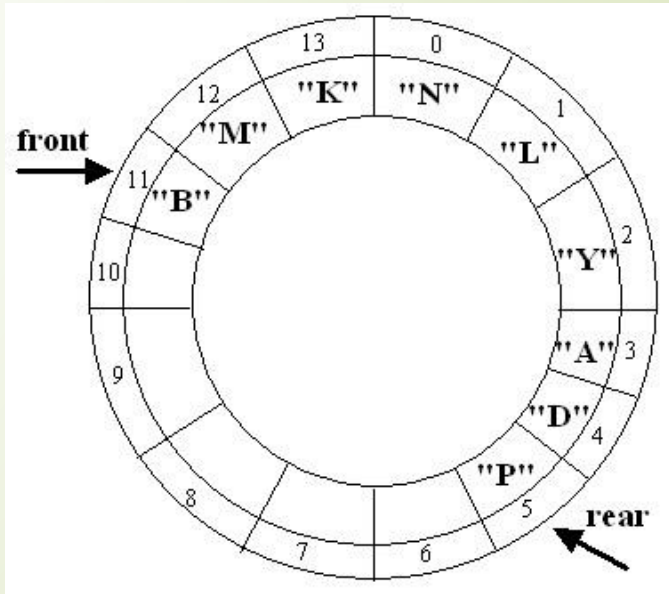
Enqueue("P") will result in …



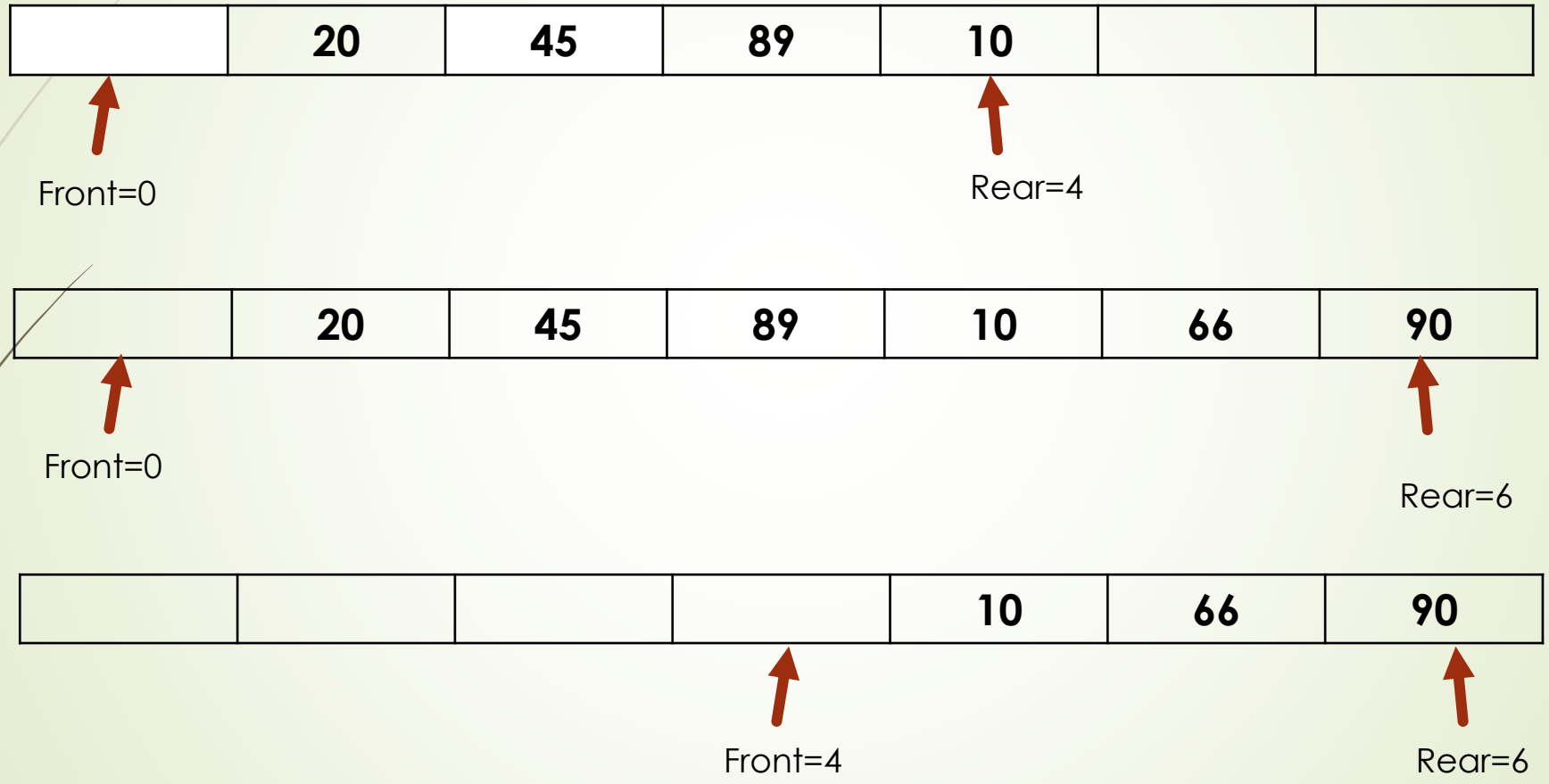
CIRCULAR QUEUE



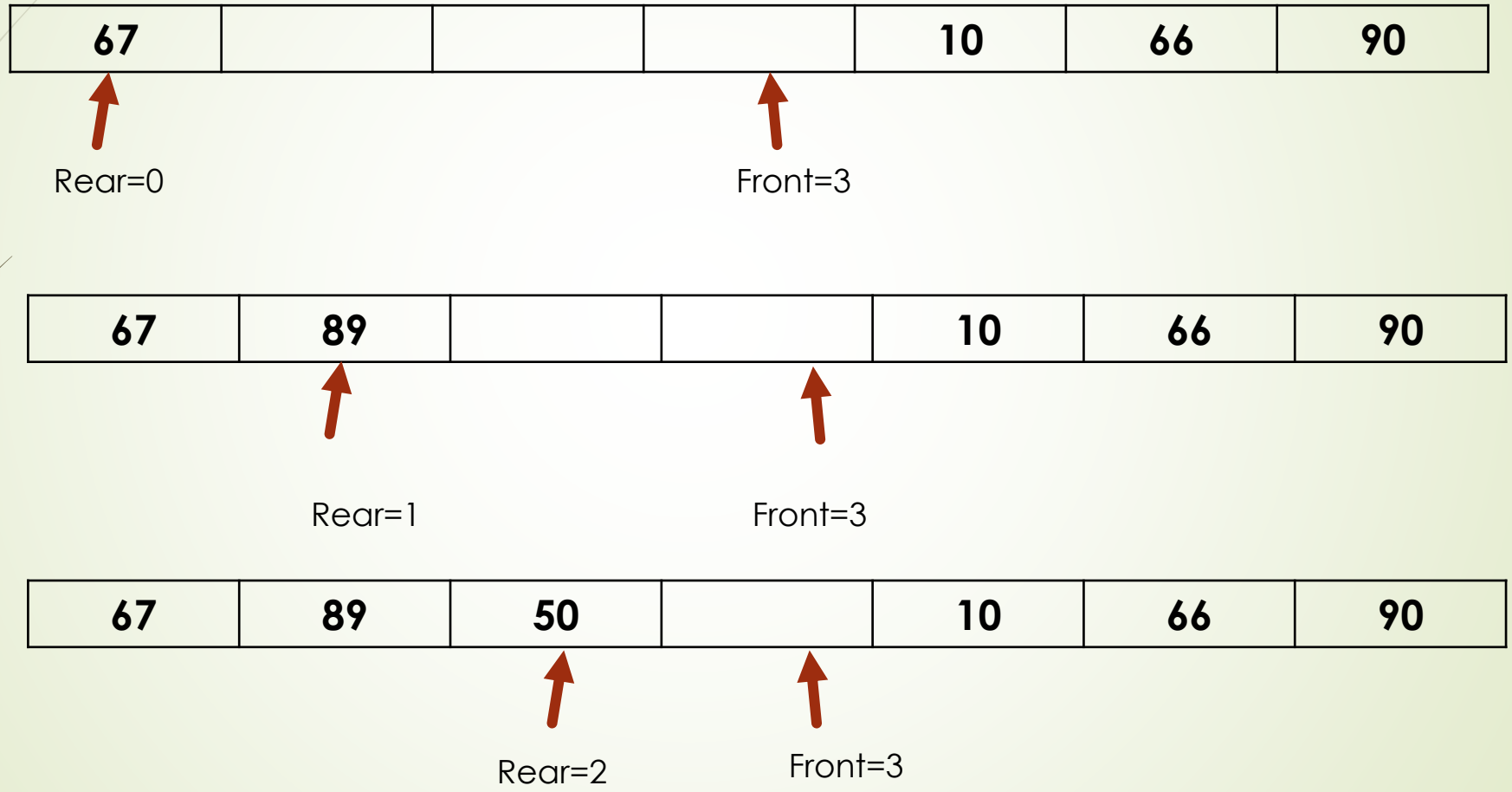
Dequeue() remove ‘s’ and <front> moves to the next item



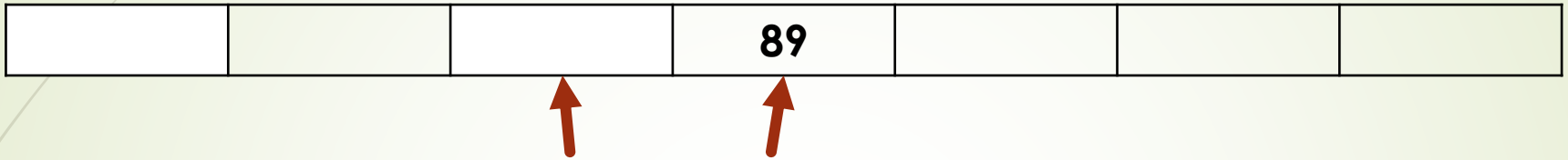
CIRCULAR QUEUE



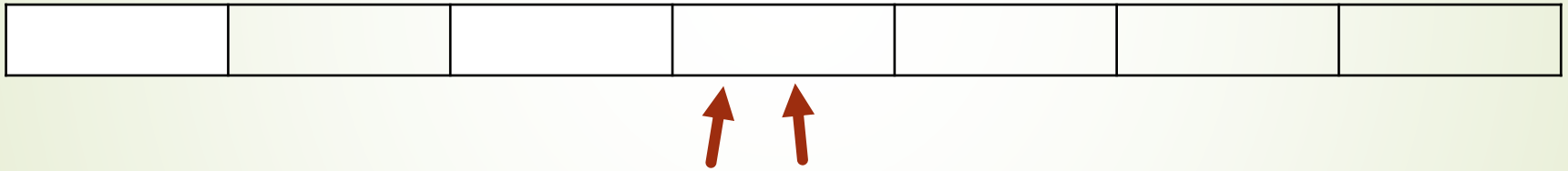
CIRCULAR QUEUE



CIRCULAR QUEUE



Front=2 Rear=3

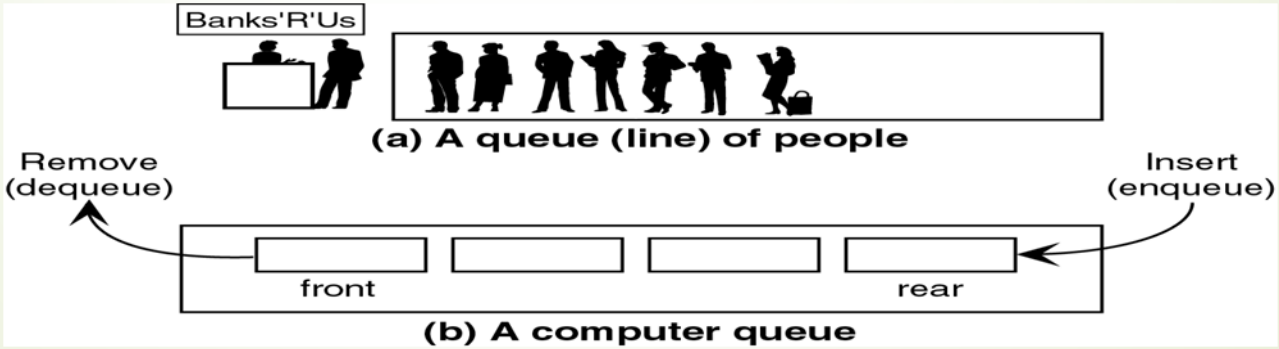


Rear=3 Front=3

Queue Implementation

Queue Implementation

- Queue is linear list in which data can only be inserted at one end, called the **rear (back)**, and deleted from the other end, called the **front**.
- First-In-First-Out (FIFO) concept.



Queue operation

- Enqueue: add data to element in queue

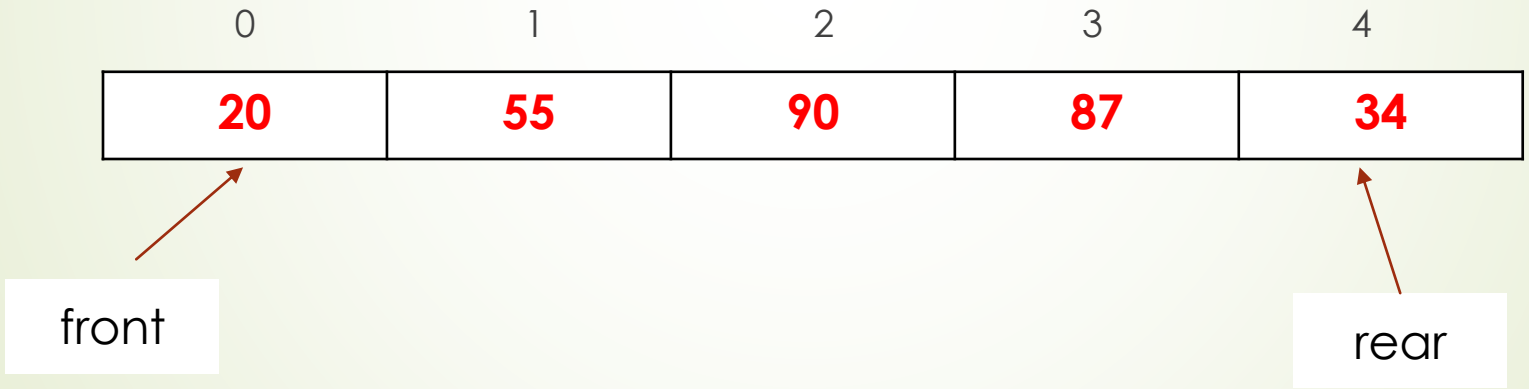
Will increased value of rear by 1

- Dequeue: take data from element in queue

Will increased value of front by 1

Queue Implementation

- int queue[size]
- int rear=front=-1
- int item



Queue Implementation: Enqueue

```
Void enqueue(int item)
{
   If (rear==size-1)
     cout<<"queue is full";
 else
   {
     Rear=rear+1;
     queue[rear]=item;
   }
  If(front==-1)
     Front=0;
}
```

Check the queue if it is full or not

- yes → write the queue is over flow

  no → then

increase the value of rear to point to the next location

Insert the new element to the queue where the rear pointer is pointing

Front=0
Rear=0

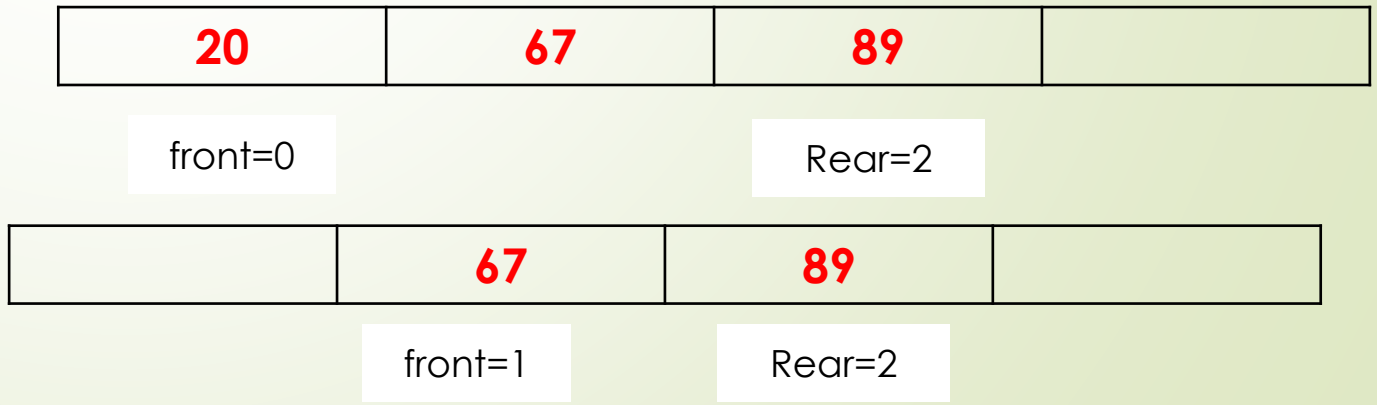
Front=0 Rear=1

Queue Implementation: Dequeue

```
Int dequeue()
{
    If (front==-1)
     cout<<"queue is empty"
     return;
else
   {
     Item=queue[front];
      If (front==rear)
        {
            front=-1;
           Rear=-1;
        }
     else
       Front=front+1;
  }
Return(item);
}
```

- check the queue is empty or not
  - yes → print queue under flow
  - no → then
    - access the item stored where the front pointer is pointing
    - increase the value of front to point to the next item

Return the item



Queue Implementation

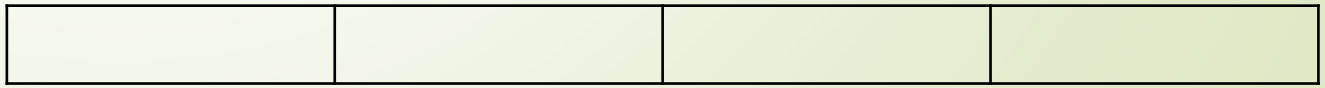
```
Int dequeue()
{
    If (front==-1)
     cout<<"queue is empty"
     return;
else
  {
    Item=queue[front];
     If (front==rear)
       {
          front=-1;
         Rear=-1;
       }
    else
      Front=front+1;
  }
Return(item);
}
```

- check the queue is empty or not
  - yes → print queue under flow
  - no → then
    - access the item stored where the front pointer is pointing
    - increase the value of front to point to the next item

Return the item

Front=3
Rear=3

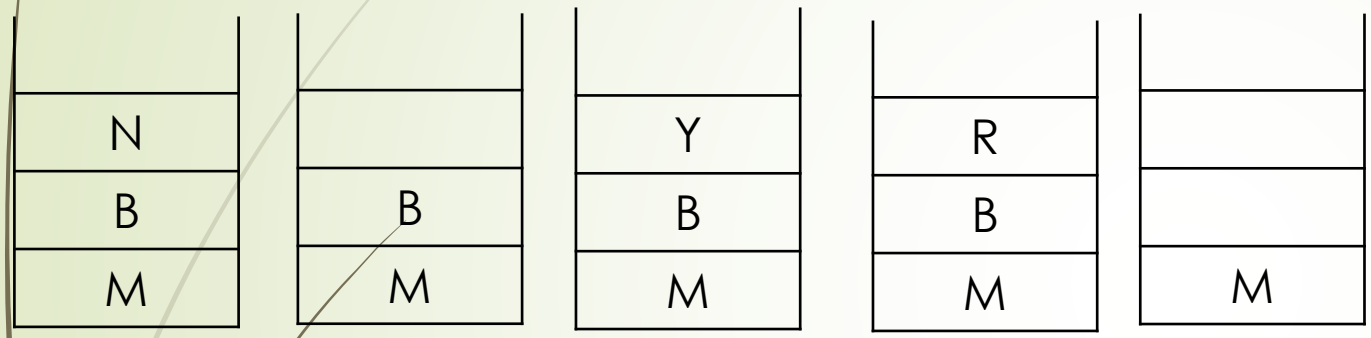
front=-1
rear=-1



Data Structures and Algorithms

Stack Exercise

EXAMPLE: If the stack stream is MBNYRI which of the following sequence of operation give the out NYBIR



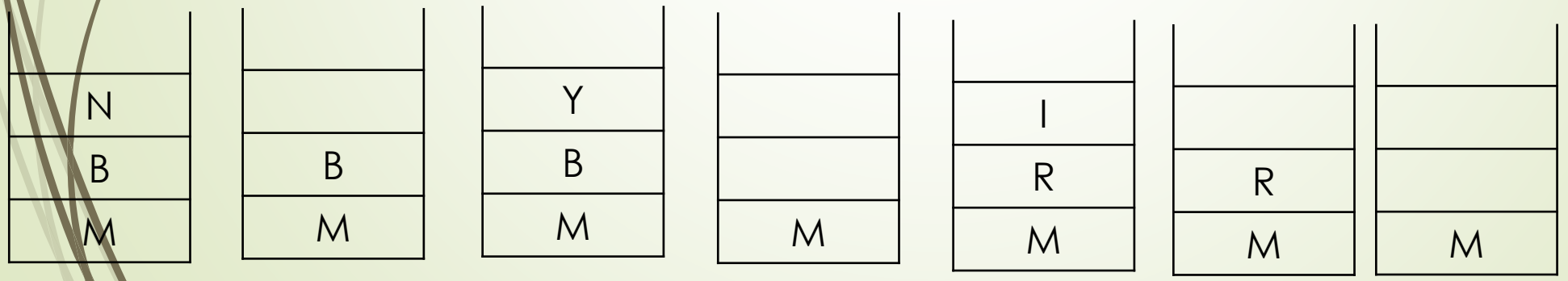
✗

✓

UUUOUOUOOUO

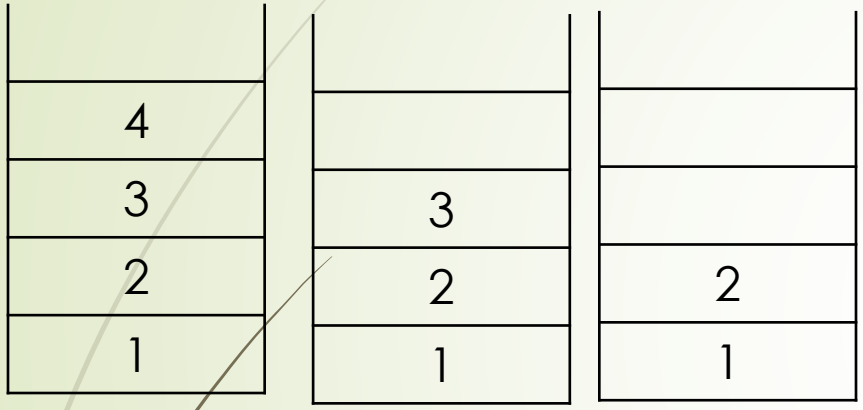
UUUOUOOUUOO

N Y R

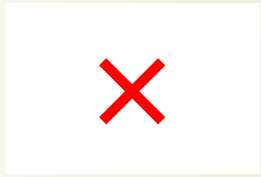


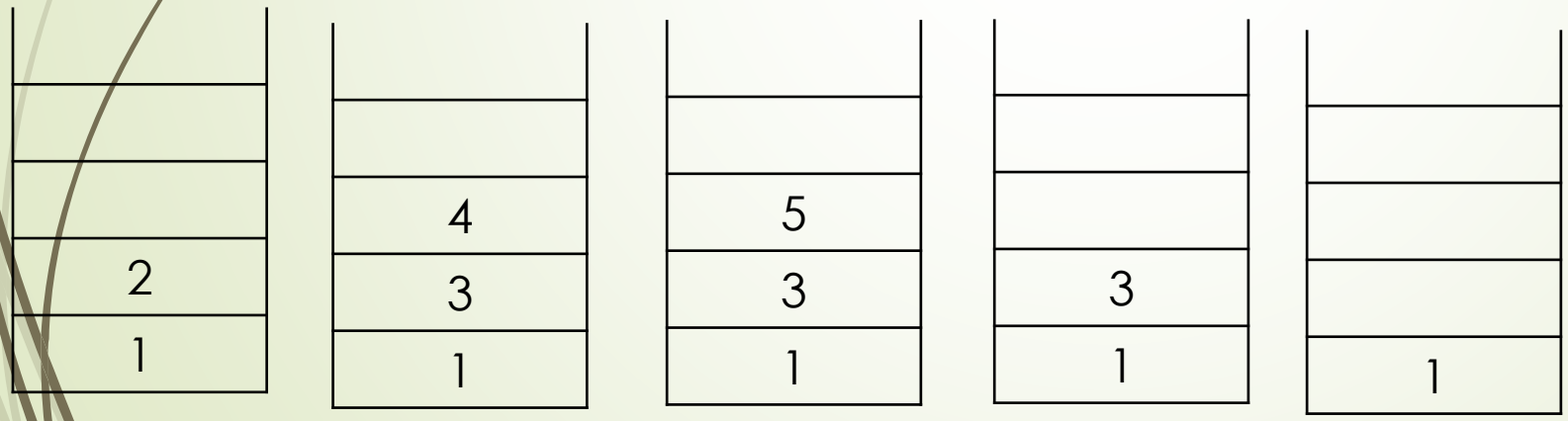
N Y B I R

EXAMPLE: If the stack stream is 12345 which of the following permutation can be obtained as out put stream

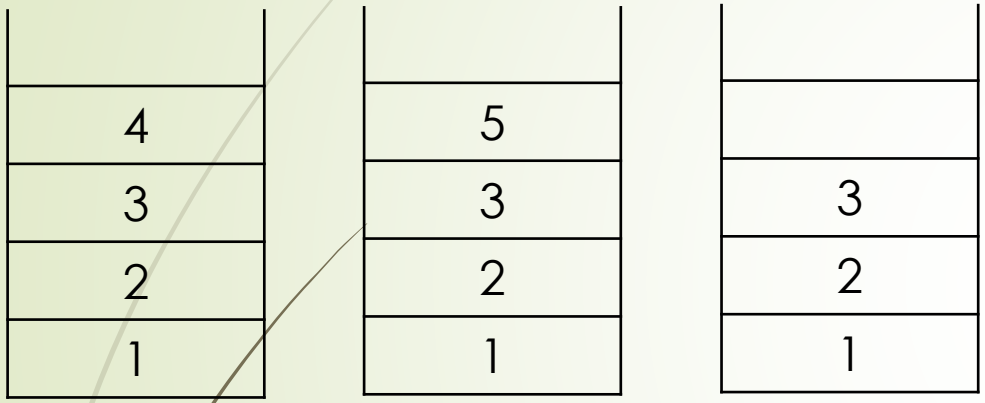


4 2 3 1 5





24531



4 5 1 2 3

Exercises

- Q: if the stack stream is Z W Y D K F which of the following sequence of operations gives the output Y W D K F Z let U for pushing an item in the stack and O for popping an item.
  - UUUOOUUUOOOO
  - UUUOOUOUOUOO
  - UUUOOUUOUOOO
  - UUUOUOOUUUOO
- Q: if the order of the stack input is 1 2 7 4 5 8 which of the following permutation can be obtained as output stream ?
  - 5 4 8 1 7 2
  - 2 7 5 8 1 4
  - 7 2 1 5 8 4
  - 2 1 4 8 7 5

Exercises

Write a program to perform the undo and redo operations using stacks. The program should display the following list.

1. Undo
2. Redo
3. Exit

Data Structures and Algorithms

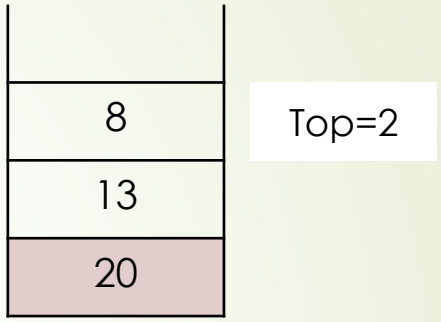
Stack Implementation

Introduction

- Stack is a linear list in which data items can only be accessed at one end, called the **top** of the stack.
- The topmost element is the **most recent** element added to the stack
- Data item: Simple data types.
- Last-In–First-Out (LIFO) concept

STACK USING ARRAY-BASED IMPLEMENTATION

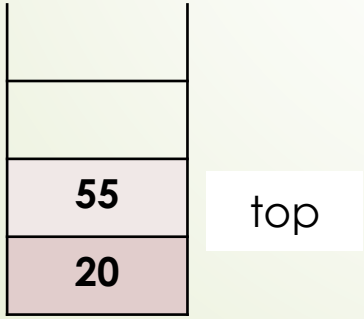
- int stack[size];
- int top= -1;
- int item;

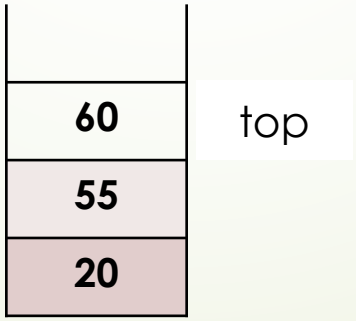


STACK USING ARRAY-BASED IMPLEMENTATION

```
Void push (int item )
{
   if (top==size-1)
      cout<<"the stack is over flow"<<"\n";
   else
      {
       top=top+1;
        stack[top]=item;
      }
}
```

Stack[++top]=item





- Check if the stack is full or not

  yes → write the stack is over flow

  no → then

  increase the value of top to point to the next location

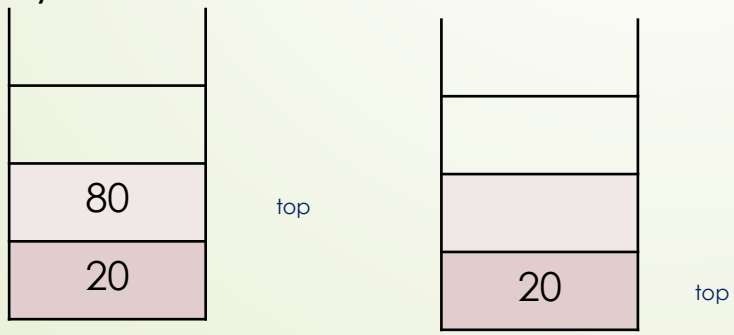
  [top=top+1]

  Insert the new element to the stack

STACK USING ARRAY-BASED IMPLEMENTATION

```
Void pop ()
{
   if (top==-1)
    {
     cout<<"stack under flow"<<"\n";
      return
    }
   else
    {
        item=stack[top];
        top=top-1;
    }
     return(item)
}
```

Item=stack[top--]



- check the stack is empty or not

  yes ⟶ print stack under flow

  no ⟶ then

  take the item where the top is pointing to

  decrease the value of the top to point to the previous item

  [Top =top-1]

Ex: write a program to insert 10 characters into a stack and print them in reverse order

```
Void push(char item)
{
}
Char pop ( )
{
}
Void main()
{ char ch;

For ( int i=0; i<10; i++){
cin>>ch;
Push(ch); }
```

Input:

H D S M a O t R C p

Output:

p C R t O a M S D H

```
while (top>-1)
  cout<< pop();
  }
```

EX2: write a program to insert 7 items into a stack each item represents an employee's salary. Print the item that its value greater than 200

```
Void push (float item)
{
}
Float pop ( )
{
}
Void main(  ){
Float salary, res;
For (int i=0; i< 7; i++){
cin<<salary;
push(salary)
}
```

Input:
500 203 102 90 70 400 80

Output:
400 203 500

```
While (top>-1){
Res= pop();
If (res>200)
  Cout<<res;
}
}
```

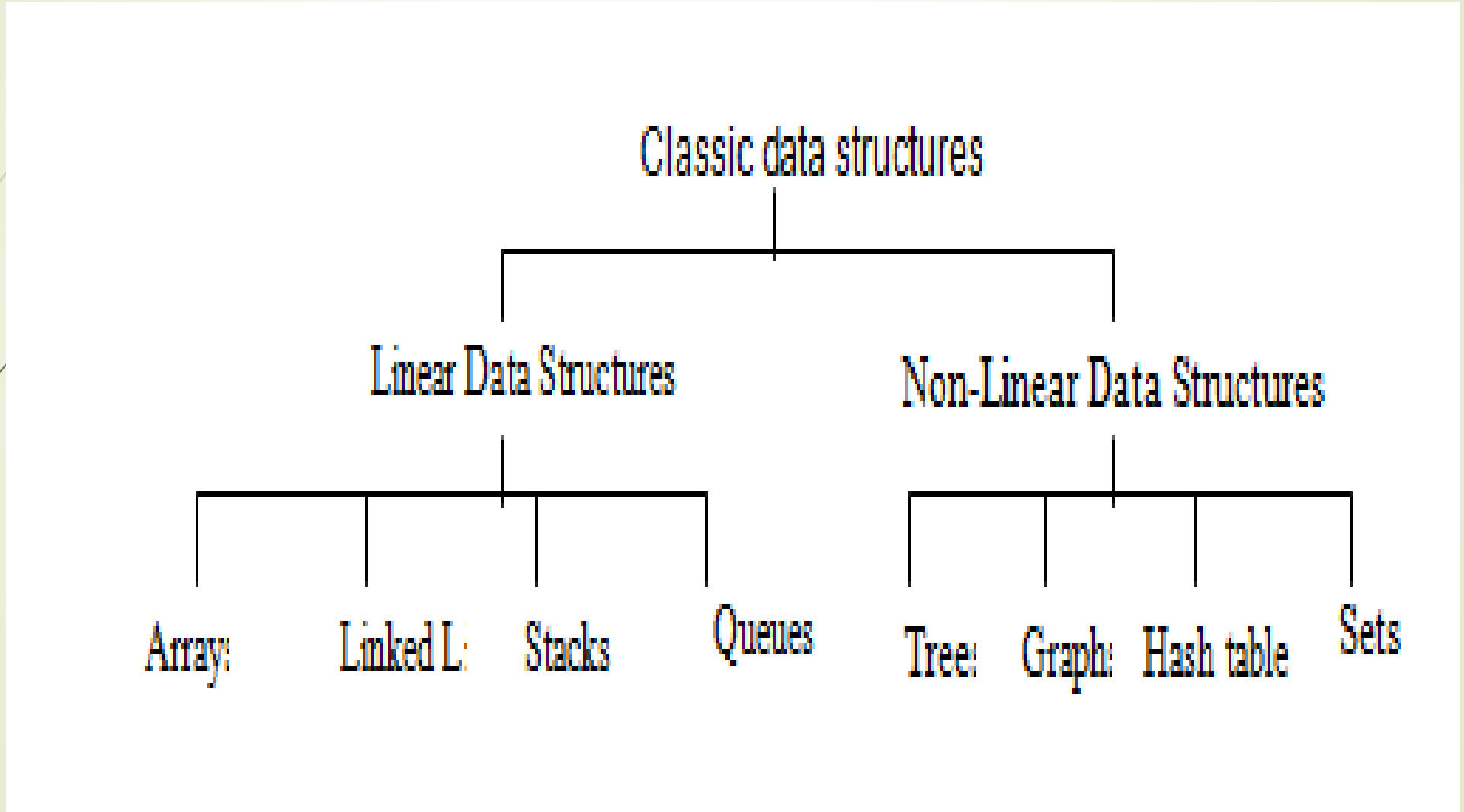
- Q1: write a program to manage a stack manually by displaying a menu to select one of stack operations. For example,

1. insert an element.
2. delete an element.
3. display the stack contents.
4. exit.

- Q2: Suppose an array implementation of a stack that stores 20 integers. Write a program to print the count of even numbers on the stack that their values greater than 10.

Data Structures and Algorithms

Types of Data Structures



Introduction

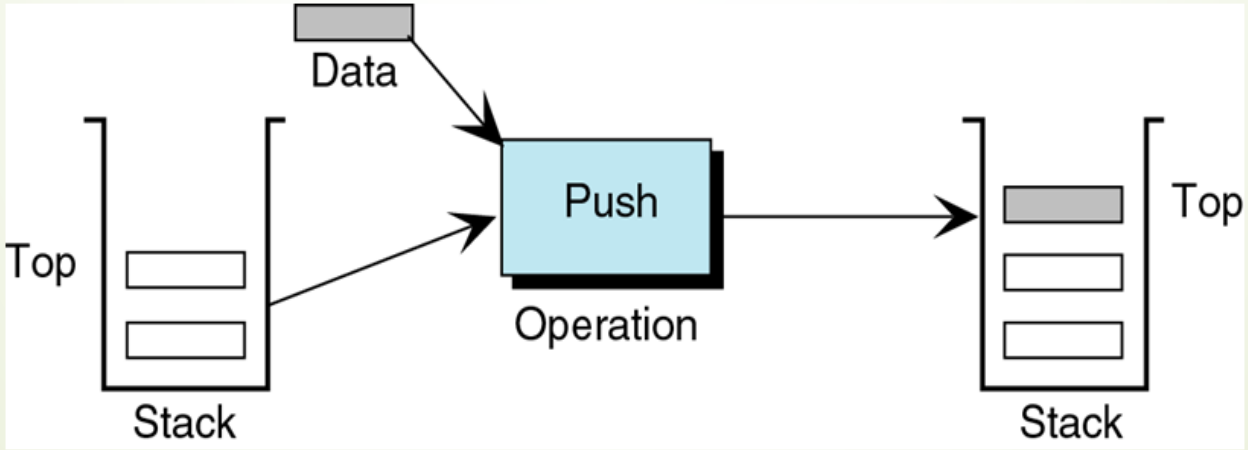
- Stack is a linear list in which data items can only be accessed at one end, called the **top** of the stack.
- The topmost element is the **most recent** element added to the stack
- Data item: Simple data types.
- Last-In–First-Out (LIFO) concept.

Stack operations

- Common operations:
  - ✓ Push – add a given element to the top of the stack.
  - ✓ Pop – read the topmost element and delete it from the stack.
  - ✓ Peek – return the topmost element.
  - ✓ Size – return the number of elements in the stack.

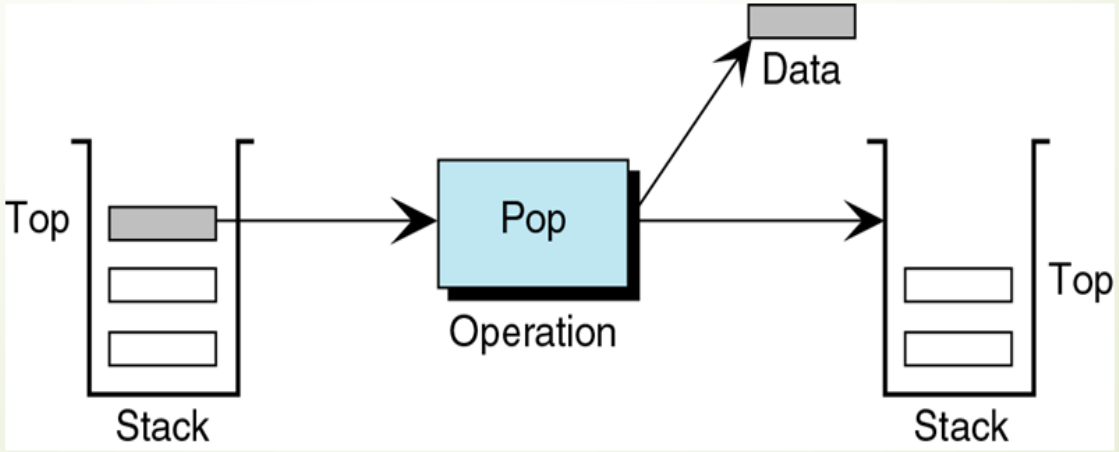
Stack operations

- Push - add a given element to the top of the stack.
- If stack is full, resize or return error if not succeed



Stack operations

- ✓Pop – remove the top element of the stack and return the element to the caller.
- ✓Will not succeed if the stack is empty - return error



Push operation

- Check if the stack is full or not
  - yes → write the stack is over flow
  - no → then

    increase the value of top to point to the next location

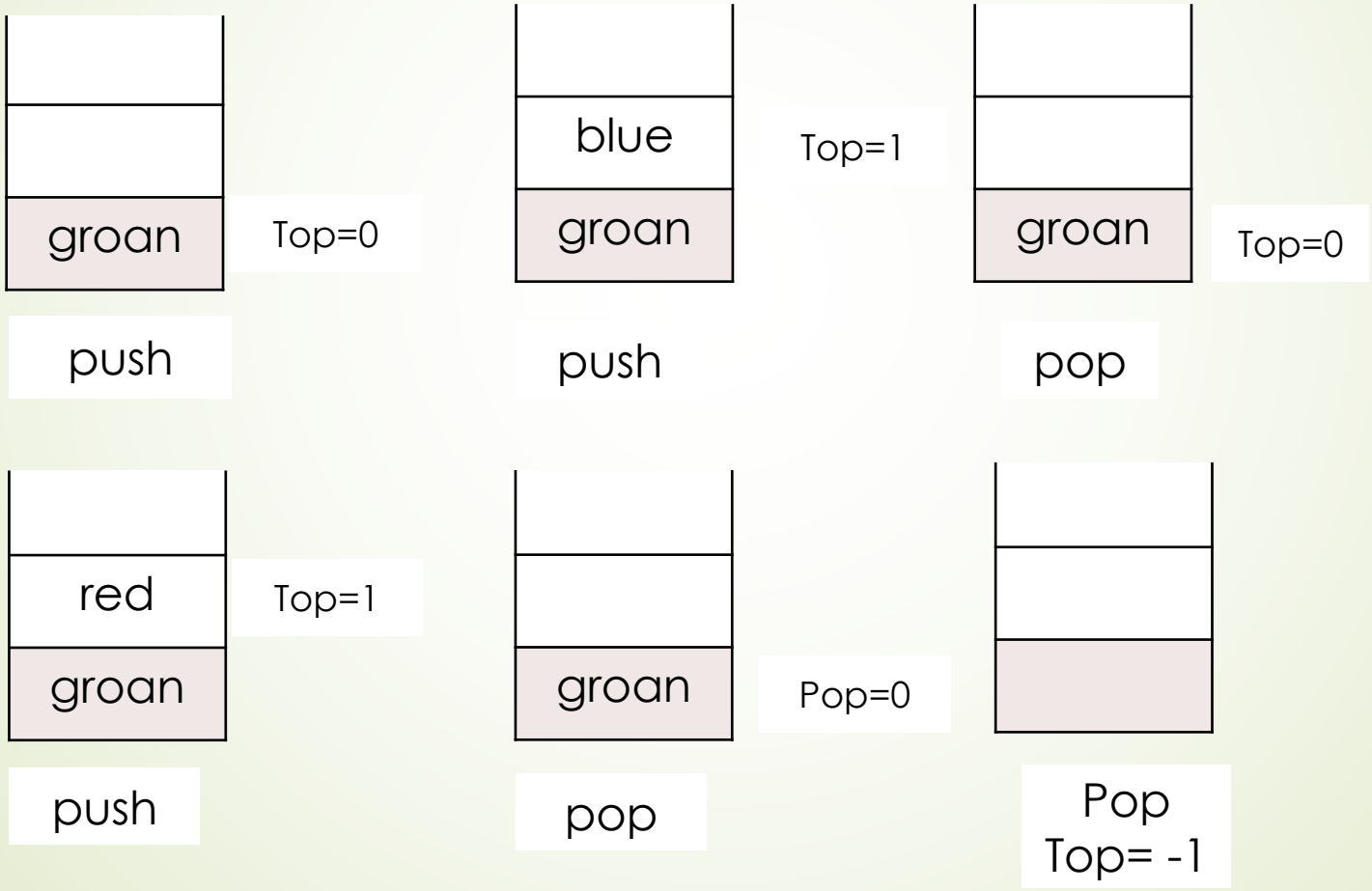
    [top=top+1]

    Insert the new element to the stack

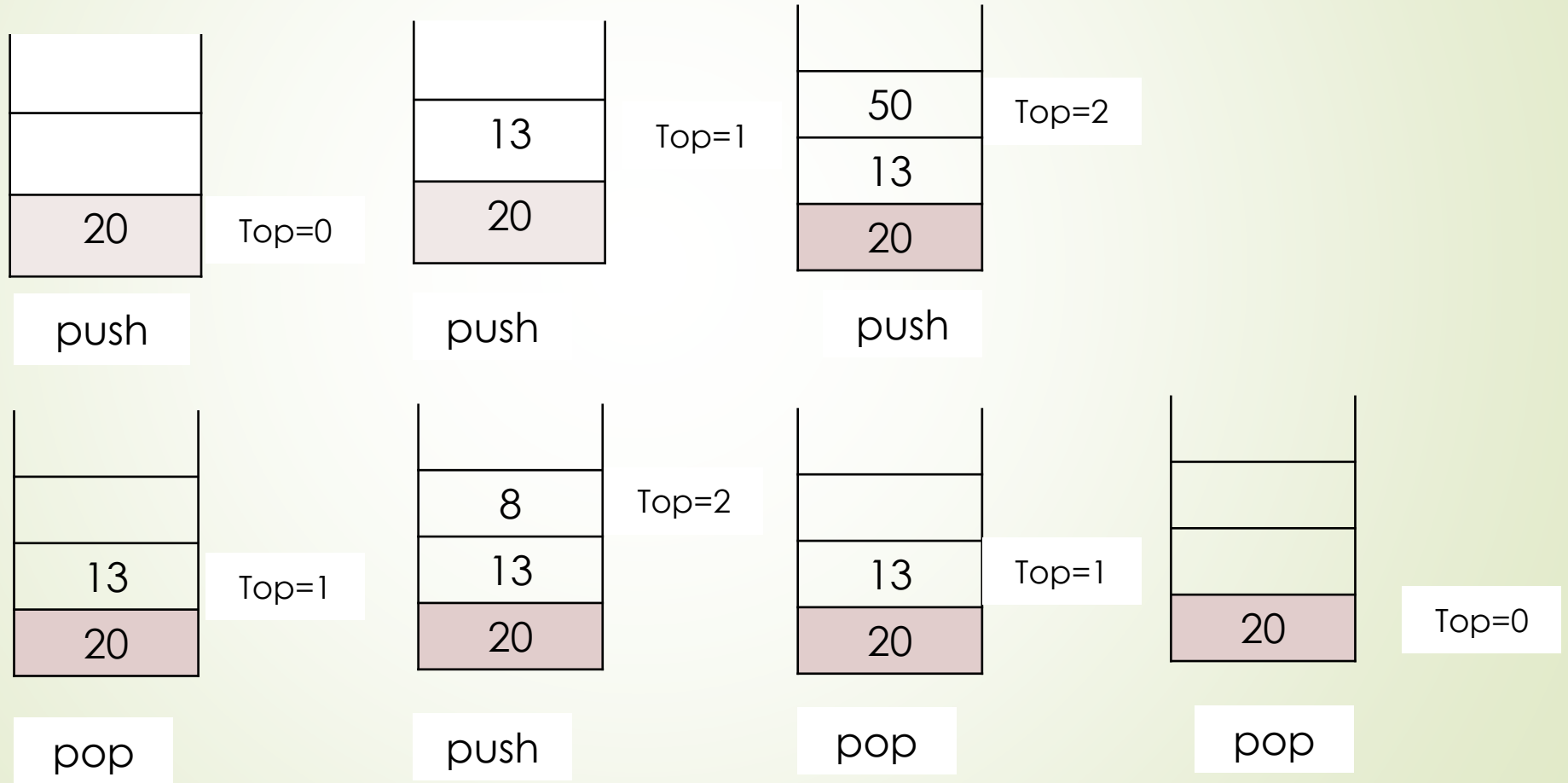
Pop operation

- check the stack is empty or not
  - yes → print stack under flow
  - no → then
    - take the item where the top is pointing to
      - decrease the value of the top to point to the previous item
      - [Top =top-1]

Example - Given stack S with list of operations as below. Illustrate the stack operations step by step:
push (green), push (blue), pop( ), push(red), pop( ), pop( )



Example - Given stack S with list of operations as below. Illustrate the stack operations step by step:
push (20), push (13),push (50), pop( ), push(8), pop( ), pop( )



Exercise

- Consider the following sequence of stack operations:

  push (A), push(X), pop(), push(M), pop(), push(G), pop()

  what is the sequence of popped values and what is the final state of the stack?