Ministry of High Education and Scientific Research
Al-Furat Al-Awsat Technical University
Al-Najaf Technical Institute
Department of Computer Networks and Software Technologies
Artificial Intelligence Branch
First Year – Second Course

Data Structures

إعداد :
م.حنان عباس سلمان
Hananabbas@atu.edu.iq

2025-2026

Data Structures Syllabus Second Course / First Year – Computer Networks and Software Technologies -Artificial Intelligence Branch		
Weekly	Hours	Unit / Topic Name
1	**4**	**-Definition of Data Structure** **- Basic Principles of Data Structures**
2	**4**	**-Methods of Representing Simple Data Structures** **-Integers** **-Real Numbers** **-Strings**
3-5	**4**	**-Arrays** **-Array Representation** **-One-Dimensional Array Representation** **-Two-Dimensional Array Representation**
6-7	**4**	**- Pointers**
8-10	**4**	**- Linked Lists**
11	**4**	**- Stack**
12	**4**	**- Queue**
13-15	**4**	**- Graphs**

The pre-test for the week 1

1- What is a Data Structure?
A) Operating System
B) Method of organizing data
C) Programming Language
D) Printer

2- Why are Data Structures used?
A) To organize and process data
B) To play music
C) To delete files
D) To draw pictures

3- Which of the following is a Data Structure?
A) Array
B) Monitor
C) Mouse
D) Keyboard

4- Data Structures help in:
A) Organizing data
B) Printing only
C) Playing games
D) Watching videos

5- Which language uses Data Structures?
A) C++
B) Paint
C) Word
D) Excel

Week 1

Outline

- What is an algorithm?
- Introduction to data structure
- Why data structure?
- Types of Data structures.
- Arrays.
- Representation of arrays in a memory.

What is an algorithm?

- An algorithm is a finite set of instructions that accomplishes a particular task.
- A well-defined computational procedure that takes some value, or a set of values, as input and produces some values or a set of values as output.
- Sequence of computational steps that transform the input into the output

What is a Good Algorithm ?

- Efficient :
  - Running Time.
  - Space Used.

Introduction to data structure

- Data are simply a value or a set of values of different types such as:
  - String
  - Integer
  - Char
  - etc.
- Structure is a way of organizing information.

Introduction to data structure

- In simple words we can define a data structure as:
  - A way of organizing data so that it can be used effectively.
  - Organizing the data in some way so that later on it can be quickly and easily:
    - Accessed
    - Queried
    - And Updated

Why data structure?

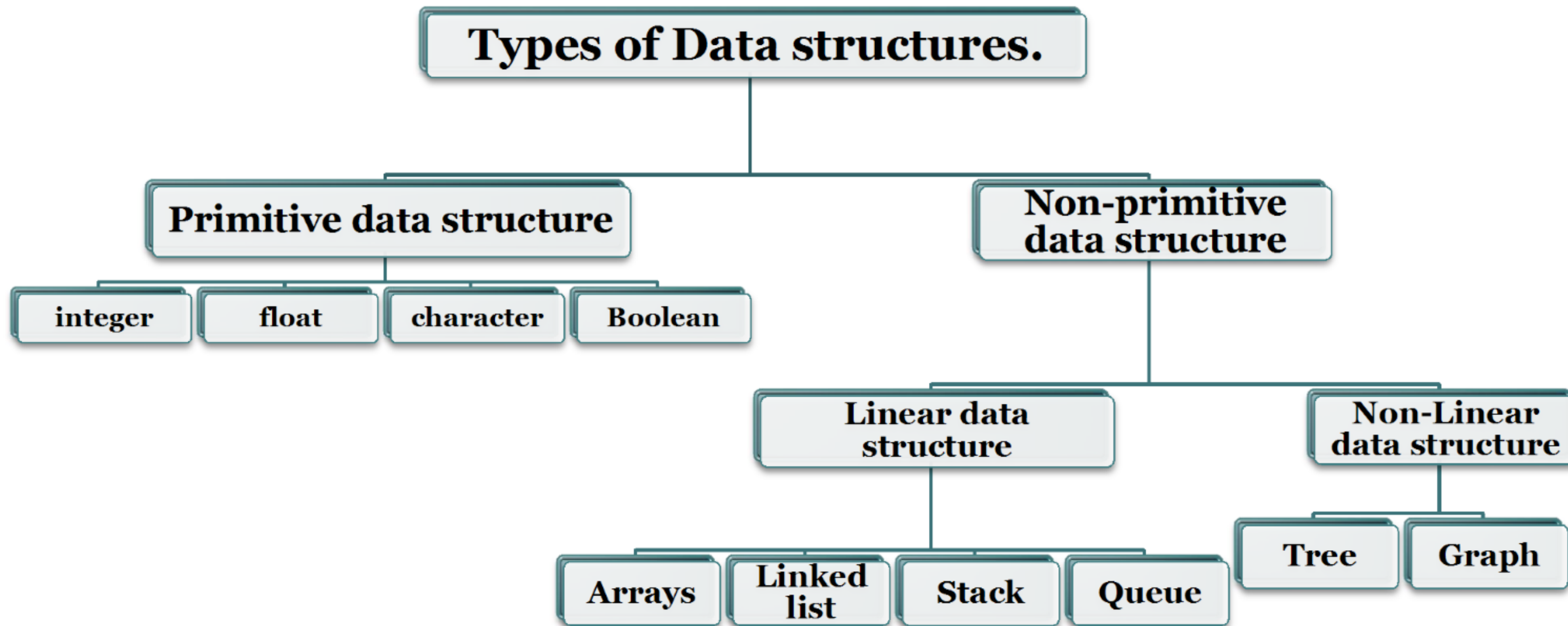
- Data structures are:
  - Essential ingredient in creating fast and powerful algorithms.
  - Help to manage and organize the data.

Advantages of organized data

- The organized data has many advantages:
  - Easy data retrieval.
  - Less data storage space.
  - Easy data operations, modification and deletion.
  - Easy to manage.
  - Avoid duplicate data.
  - Efficient software systems.

How to choose suitable data structure

- For each set of data there are different methods to organize these data in a particular data structure.
- To choose suitable data structure, we must use the following criteria:
  - Data size and the required memory
  - The dynamic nature of the data.
  - The required time to obtain any data element from the data structure.


Types of Data structures.
Primitive data structure
Non-primitive data structure
integer
float
character
Boolean
Linear data structure
Non-Linear data structure
Arrays
Linked list
Stack
Queue
Tree
Graph

Primitive Data Structure

- Some data structures are built in to the programming language itself which are readily available to the programmer for its use and these data structure are referred as primitive data structure.
- For example, the most common primitive data structures which are generally supported by almost all programming languages include
    - Integer.
    - Float.
    - Character.
    - and Boolean.

The post-test for the week 1

1- What is the importance of Data Structures?
A) Efficient data organization
B) Playing music
C) Deleting programs
D) Changing screen color
2- Which is NOT a Data Structure?
A) Stack
B) Queue
C) Keyboard
D) Array
3- Data Structures help to:
A) Access data easily
B) Stop the computer
C) Delete memory
D) Increase sound

4- Which is a principle of Data Structures?
A) Organizing data
B) Watching videos
C) Playing games
D) Drawing pictures
5- Data Structures are used in:
A) Software development
B) Photography
C) Television
D) Music

The pre-test for the week 2

1- Which type is used for whole numbers?
A) Integer
B) String
C) Float
D) Char

2- Strings are used to store:
A) Text
B) Images
C) Videos
D) Numbers only

3- Real Numbers represent:
A) Decimal values
B) Images
C) Files
D) Letters

4- Which data type stores decimal numbers?
A) Integer
B) Float
C) Character
D) Boolean

5- Integer is used for:
A) Whole numbers
B) Images
C) Text
D) Videos

Week 2

Primitive data structure

Data structure	Description	Example
Integer	Represents a number without decimal points	1, 2, 45, 1000
Float	Represents a number with decimal points	1.5, 4.5, 3.8, 2567.24
Character	Represents a single character	A, B, g, h
Boolean	Represent logical values either True or False	

Non-Primitive data structure

- The Non-Primitive data structure are user (programmer) defined data structure which also referred as user defined data structure.
- The Non-primitive data types are not pre-defined in the programming language, and therefor the programmer has to defined as per the program requirements.
- The Non-primitive data structure are derived from primitive data types by combining two or more primitive data structure. These Data structure divided as linear and Non-linear data structure.

Linear & Non-Linear Data structure

- In **Linear** data structures, the data items are arranged in memory in a linear sequential. Example **Array**, **Stack**
- In **Non-Linear** data structures, the data items are not stored in memory in sequential. Example **Tree**, graph

The post-test for the week 2

1- Which of the following is a String?
A) 25
B) "Ali"
C) 3.5
D) 10
2- Real Numbers are used for:
A) Decimal values
B) Pictures
C) Text only
D) Files
3- Which data type stores integers?
A) String
B) Integer
C) Float
D) Character

4- Which of the following is a data type?
A) Integer
B) Mouse
C) Printer
D) Monitor
5- Strings are mainly used to:
A) Store text
B) Store videos
C) Delete files
D) Draw shapes

The pre-test for the week 3-5

1- What is an Array?
A) Collection of similar elements
B) Operating System
C) Image file
D) Video file

2- One-Dimensional Array means:
A) Single dimension array
B) Double dimension array
C) Linked List
D) Queue

3- Two-Dimensional Array contains:
A) Rows and columns
B) One row only
C) One column only
D) Text only

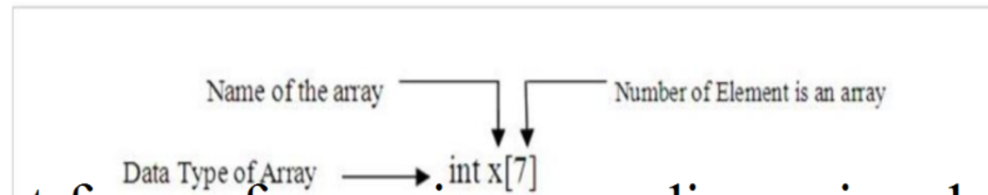
4- Arrays are used for:
A) Storing groups of data
B) Playing music
C) Printing only
D) Watching videos

5- Array elements are accessed by:
A) Index
B) Printer
C) Keyboard
D) Screen

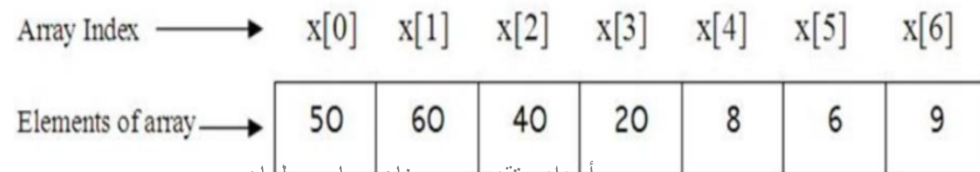
Week 3-5

Arrays

- An array is a linear data structure, it is collection of elements of same data type. An array is linear data structure because the array elements (Values) are traversed in sequential manner.



- The simplest form of array is a one-dimensional array. It is organized in a single row consisting of number of pre-defined number of data elements



Two Dimensional Array

1	5	3	6
3	2	38	64
22	76	82	99
0	106	345	54

User's view (abstraction)

1	5	3	6	3	2	38	64	22	76	82	99	0	106	345	54

System's view
(implementation)

Representation of array in memory

- **One Dimensional array**
  - Location(X[I])=Base Address+(I-1)
  - **Example:**

Find X[4] location in memory where Base Address=500

Location(X[I])=Base Address+(I-1)

Location(X[4])=500 +(4 -1)

=503

Representation of array in memory

- **Two Dimensional array**
- Row –wise method:

Location (A[i , j]) = base address + N * (i-1) + (j-1)

- Column – wise method

Location (A[I,J]) = Base Address + M * (J -1) + (I -1)

- **Example:**

A[5,7], M=5, N=7, Base Address=900 Find A[4 , 6]

Representation of array in memory

By **Row** :

Location(A [4,6])= 900+7 *(4-1)+(6-1) = 900+21+5 =926

By **Column:**

Location(A [4,6])=900 +5 *(6 -1)+(4 -1) =900+25+3 =928

The post-test for the week 3-5

1- Which of the following represents an array?
A) int a[5]
B) float x
C) char c
D) bool y
2- A two-dimensional array contains:
A) Rows and columns
B) One element only
C) Files
D) Pictures
3- The first index of an array usually starts with:
A) 0
B) 1
C) 2
D) 5

4- Array Representation is used for:
A) Representing data in memory
B) Playing games
C) Printing documents
D) Watching videos
5- Which is NOT a type of array?
A) One-Dimensional
B) Two-Dimensional
C) Three-Dimensional
D) Keyboard

The pre-test for the week 6-7

1- What is a Pointer?
A) A memory address
B) A printer
C) A file
D) A monitor
2- Which symbol is used to get the address of a variable?
A) &
B) *
C) %
D) #
3- Which symbol is used for dereferencing?
A) *
B) &
C) +
D) /

4- A pointer stores:
A) Address of a variable
B) Video files
C) Images
D) Text only
5- Which is a correct pointer declaration?
A) int *p
B) int p
C) float p
D) char p

Week 6-7

Pointers

- To understand pointers, you should first know how data is stored on the computer.
- Each variable you create in your program is assigned a location in the computer's memory. The value the variable stores is actually stored in the location assigned.
- To know where the data is stored, C++ has an & operator.
- **The & (reference) operator gives you the address occupied by a variable.**
- **If var is a variable then, &var gives the address of that variable.**

Example 1: illustrate the Address in C++

```
#include <iostream>
using namespace std;

int main()
{
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;
    cout << &var1 << endl;
    cout << &var2 << endl;
    cout << &var3 << endl;
}
```

Output

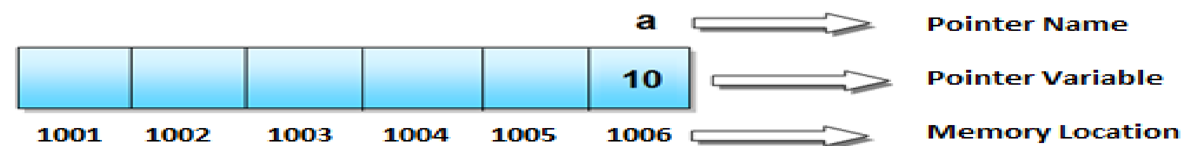
```
0x7fff5fbff8ac
0x7fff5f bff8a
8
0x7fff5f bff8a
4
```

وضح

- You may not get the same result on your system.
- The 0x in the beginning represents the address is in hexadecimal form.
- Notice that first address differs from second by 4-bytes and second address differs from third by 4-bytes.
- This is because the size of integer (variable of type int) is 4 bytes in 64-bit system.

Pointers

- A pointer is the memory address of a variable.
- A **pointer** is a variable that contains the address of a variable.
- Using pointer we can pass argument to the functions. Generally we pass them by value as a copy. So we cannot change them. But if we pass argument using pointer, we can modify them
- Let us imagine that a computer memory is a long array and every array location has a distinct memory location.
- A pointer variable contains a representation of an address of another variable (P is a pointer variable in the following):



Example: (pointer declarations)

float *p; //To declare a pointer variable p that can "point" to a variable of type float

int *K; //To declare a pointer variable K that can "point" to a variable of type int

Example: If a number variable is stored in the memory address **0x123**, and it contains a value **5**.

- The **reference (&)** operator gives the value **0x123**, while the **dereference** (*) operator gives the value **5**.

Pointer Variable Definition

Syntax: *Type *Name;*

Examples:

```
int    *P;      // P is varaible that can point to an integer var
float *Q;       // Q is a float pointer
char *R;        // R is a char pointer
```

Example:

```
int *AP[5];     /* AP is an array of 5 pointers to ints */
```

Address (&) Operator

- An address used to tell where a variable is stored in memory is a pointer
- Pointer variables must be declared to have a pointer type
- Reference operator (&) as discussed above gives the address of a variable.

The Dereferencing Operator

To get the value stored in the memory address, we use the dereference operator (*).

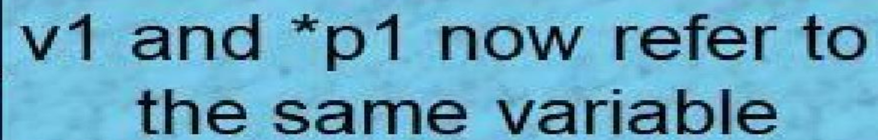
Example: p1 = &v1;

- p1 is now a pointer to v1
- v1 can be called v1 or "the variable pointed to by p1"

Example 2:

```
v1 = 0;
p1 = &v1;
*p1 = 42;
cout << v1 << endl;
cout << *p1 << endl;

output:
        42
        42
```



Example 3: C++ Pointers

C++ Program to demonstrate the working of pointer.

```
#include      <iostream>
using   namespace   std;
int main() {
int *pc, c;
  c = 5;
 cout << "Address of c (&c): " << &c << endl;
 cout << "Value of c (c): " << c << endl << endl;
  pc = &c;              // Pointer pc holds the memory address
of variable c cout << "Address that pointer pc holds (pc):
"<< pc << endl;
cout << "Content of the address pointer pc holds (*pc): " << *pc <<
endl

 c = 11;   // The content inside memory address &c is changed from
5 to 11. cout << "Address pointer pc holds (pc): " << pc << endl;
cout << "Content of the address pointer pc holds (*pc): " << *pc <<
endl;
  *pc = 2;
  cout << "Address of c (&c): " << &c << endl;
  cout << "Value of c (c): " << c << endl << endl;
  return 0;
```

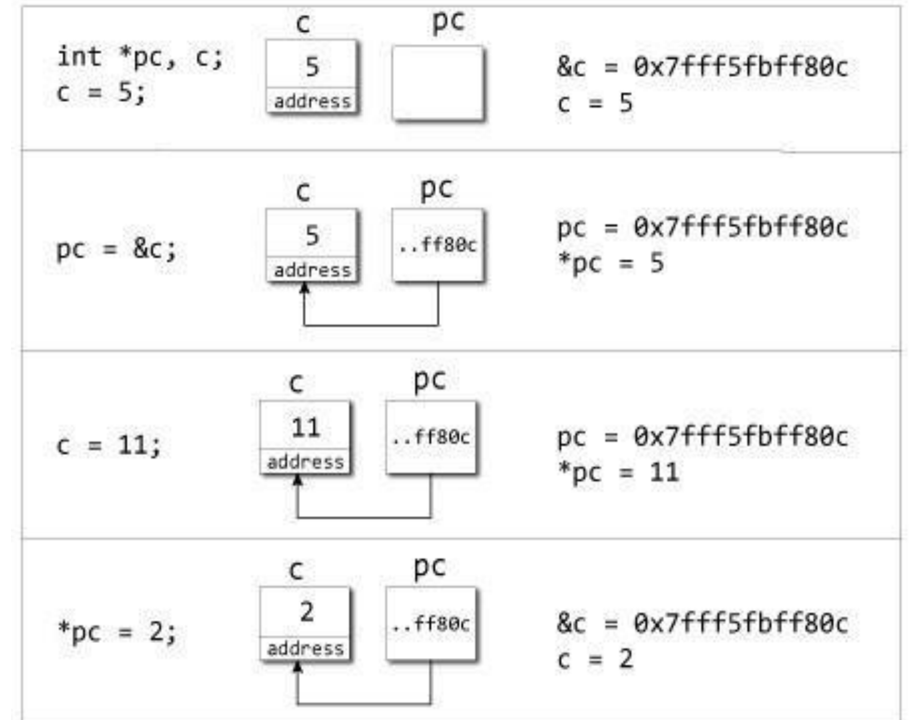
The output for Example 3

Address of c (&c): 0x7fff5fbff80c
Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff5fbff80c
Content of the address pointer pc holds (*pc): 5

Address pointer pc holds (pc): 0x7fff5fbff80c
Content of the address pointer pc holds (*pc): 11

Address of c (&c): 0x7fff5fbff80c
Value of c (c): 2



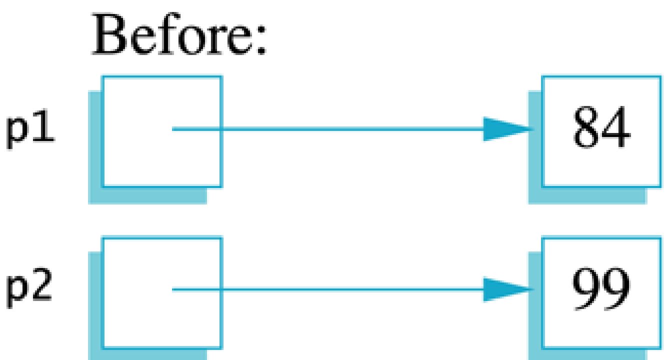
Pointer Assignment

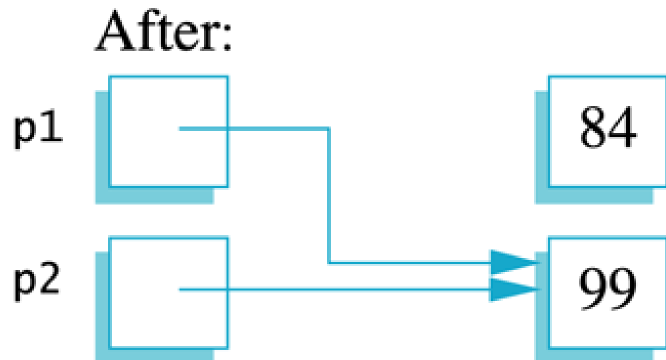
- The assignment operator = is used to assign the value of one pointer to another
    - causes *p2, *p1, and v1 all to name the same variable
  - Example: If p1 still points to v1 (previous slide) then

    p2 = p1;
- Some care is required making assignments to pointer variables
    - p1= p2; // changes the location that p1 "points"
        - to
    - *p1 = *p2; // changes the value at the location
        - that p1 "points" to

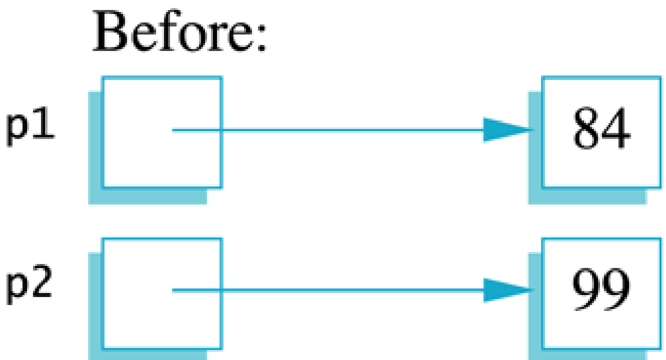
Uses of the Assignment Operator

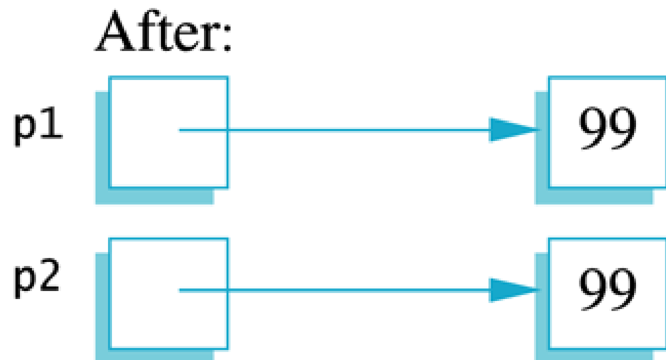
`p1 = p2;`





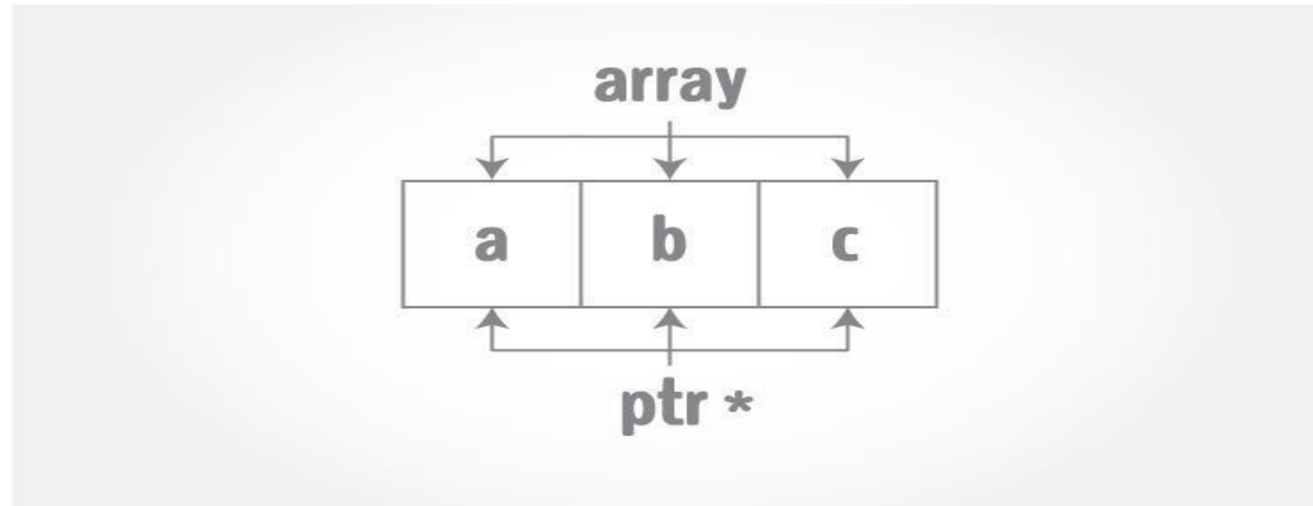
`*p1 = *p2;`





C++ Pointers and Arrays

- In this article, you'll learn about the relation between arrays and pointers, and use them efficiently in your program.



- Pointers are the variables that hold address. Not only can pointers store address of a single variable, it can also store address of cells of an array.

For example:

```
Int *ptr;
int   a[5];
Ptr = &a[2]; // &a[2] is the address of third element of a[5].
```

Figure: Array as Pointer

- Suppose, pointer needs to point to the fourth element of an array, that is, hold address of fourth array element in above case.
- Since ptr points to the third element in the above example, ptr + 1 will point to the fourth element.
- You may think, ptr + 1 gives you the address of next byte to the ptr. But it's not correct.
- This is because pointer ptr is a pointer to an **int and size of int is fixed for a operating system (size of int is 4 byte of 64-bit operating system).** Hence, the address between ptr and ptr + 1 differs by 4 bytes.
- If pointer ptr was pointer to char then, **the address between ptr and ptr + 1 would have differed by 1 byte since size of a character is 1 byte.**

Example 4: C++ Pointers and Arrays

C++ Program to display address of elements of an array using both array and pointers

```
#include <iostream>
using namespace std;

int main()
{
    float arr[5];
    float *ptr;

    cout << "Displaying address using arrays: " << endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }

    // ptr = &arr[0]
    ptr = arr;

    cout<<"\nDisplaying address using pointers: "<< endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "ptr + " << i << " = "<< ptr + i << endl;
    }


    return 0;
}
```

Output of Eample 4:

```
Displaying address using arrays:
&arr[0] = 0x7fff5fbff880
&arr[1] = 0x7fff5fbff884
&arr[2] = 0x7fff5fbff888
&arr[3] = 0x7fff5fbff88c
&arr[4] = 0x7fff5fbff890

Displaying address using pointers:
ptr + 0 = 0x7fff5fbff880
ptr + 1 = 0x7fff5fbff884
ptr + 2 = 0x7fff5fbff888
ptr + 3 = 0x7fff5fbff88c
ptr + 4 = 0x7fff5fbff890
```

قحج

In the above program, a different pointer ptr is used for displaying the address of array elements arr.

But, array elements can be accessed using

The post-test for the week 6-7

1- What is the main use of pointers?
A) Access memory locations
B) Play music
C) Delete files
D) Draw shapes
2- The symbol * is called:
A) Dereference Operator
B) Addition Operator
C) Division Operator
D) Assignment Operator
3- The symbol & is called:
A) Address Operator
B) Logical Operator
C) Arithmetic Operator
D) Assignment Operator

4- Which is a float pointer declaration?
A) float *p
B) float p
C) int p[]
D) char p
5- Pointers are used to:
A) Access memory addresses
B) Print files
C) Watch videos
D) Change screen colors

The pre-test for the week 8-10

1- A Linked List is made up of:
A) Nodes
B) Images
C) Files
D) Programs

2- Each node contains:
A) Data and Pointer
B) Keyboard
C) Printer
D) Screen

3- Linked Lists are used for:
A) Dynamic data storage
B) Drawing only
C) Playing games
D) Watching videos

4- The next element in a Linked List is accessed using:
A) Pointer
B) Monitor
C) Mouse
D) Scanner

5- Linked List is a:
A) Data Structure
B) Programming Language
C) Database
D) Operating System

Week 8-10

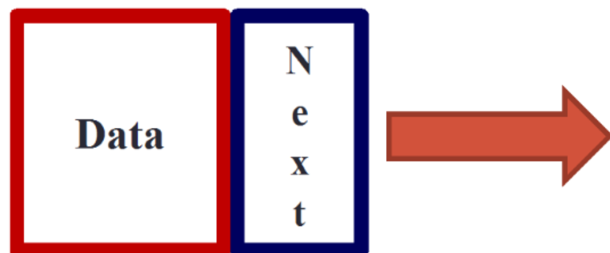
OUTLINE

- What is a linked list
- Parts of a Linked List
- Architecture of Linked list
- Applications of linked list in computer science
- Advantages and disadvantages of Linked lists
- Array vs Linked List
- Operations On Linked List
- Types of linked list
- Singly linked list

What is a linked list

- It is another type of data structure which are dynamically allocated.
- It is a collection of especially designed data elements called **nodes** linked to one another by means of **pointers**.
- Each node is divided into two parts first part contains the **Data** and the second contains **Pointer** which points to the next node.

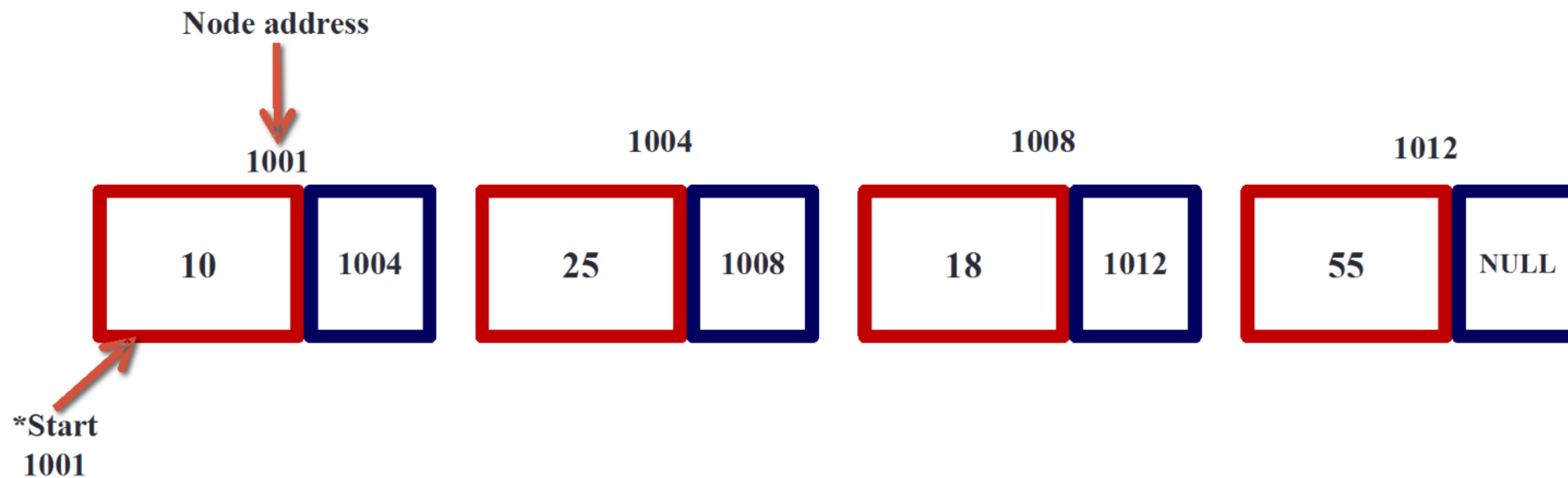
Parts of a Linked List



One node is made up of two parts: some Data that it holds, and a reference to the next node

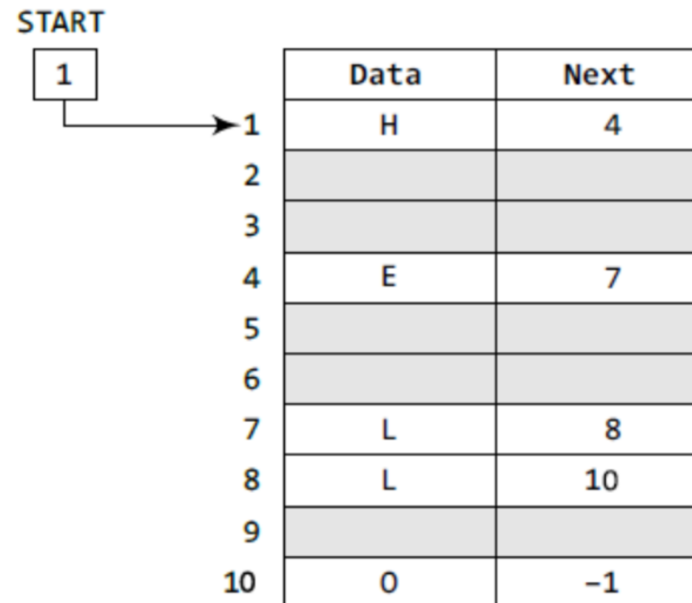
Architecture of Linked lists

- The linked list is a set of nodes, each one in it consists of two parts as follow:
  - First Part: contains data (String, Integer, Etc)
  - Second Part: Contains a pointer that refer to the next address in the memory



Example of a linked list

- START is used to store the address of the first node
- START = 1 so the first data is stored at address 1, which is H.
- The corresponding NEXT stores the address of the next node
- NEXT = 4 The second data element obtained from address 4 is E
- Next = 7 The third data element obtained from address 7 is L
- We repeat this procedure until we reach
- a position where the NEXT entry contains –1 or NULL as this would denote the end of the linked list



Applications of linked list in computer science

- Implementation of stacks and queues.
- Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
- Dynamic memory allocation : We use a linked list of free blocks.
- Maintaining directory of names.
- Performing arithmetic operations on long integers
- Manipulation of polynomials by storing constants in the node of linked list
- Representing sparse matrices

Advantages of Linked lists

- It is a dynamic data structure, i.e. a linked list can grow and shrink in size during its lifetime.
- The nodes of linked list (elements) are stored at different memory locations .
- Insertion and deletion in linked list is easy because it does not requires shifting of elements in it.

Disadvantage of Linked List

- They use more memory because of the storage used by their pointers.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Nodes are stored noncontiguous, greatly increasing the time periods required to access individual elements within the list.
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards, while doubly linked lists are somewhat easier to read, memory is consumed in allocating space for a back-pointer.
- Searching a particular element in a linked list is difficult and also consuming time.

Array vs Linked List

Array	Linked List
Fixed Size	Dynamic Size
Insertions and deletions are inefficient: elements are usually shifted	Insertions and Deletions are efficient: no shifting.
Random Access: efficient indexing	No random access Not suitable for operations requiring accessing elements using index such as sorting
No memory waste if the array is full or almost full. Otherwise may result in much memory waste	Since memory is dynamically allocated (according to our need) there is no waste of memory
Sequential access is faster [reason: elements are in contiguous memory locations]	Sequential access is slow [reason: elements are not in contiguous memory locations

Array vs Linked List cont.

Array	Linked List
It is necessary to specify the number of elements during declaration (during compile time)	It is not necessary to specify the number of elements during declaration time (memory is allocated during run time)
Occupies less memory for the same number of elements.	Occupies more memory for the same number of elements
Insertion elements at the front of the array is expensive because existing elements need to be shifted	Inserting new elements at any position can be carried out easily

Operations On Linked List

- **Creation**: Creation operation is used to create a linked list with one node.
- **Insertion**: Insertion operation is used to insert a new node at any specified location in the linked list.
- A new node may be inserted.
  a) At the beginning of the linked list
  b) At the end of the linked list
  c) At any specified position in between in a linked list
- **Deletion**: Same as Above

Operations On Linked List cont.

- **Traversing:** is the process of going through all the nodes from one end to another end of a linked list.

- **Searching:** usually searching operations is employed not only while a data item is needed but in case of insertion and deletion at specified location search operation is performed before nodes can be inserted or deleted.

Types of Linked List

- Singly linked list
- Doubly linked list
- Circular linked list

The post-test for the week 8-10

1- Which represents a node in Linked List?
A) Data + Pointer
B) Picture
C) File
D) Screen
2- One advantage of Linked Lists is:
A) Flexible insertion of elements
B) Deleting the operating system
C) Watching movies
D) Printing pictures
3- Pointers in Linked Lists are used for:
A) Connecting nodes
B) Playing music
C) Drawing shapes
D) Opening files

4- Which is NOT part of a node?
A) Data
B) Pointer
C) Address
D) Printer
5- Linked Lists allow:
A) Dynamic expansion
B) System shutdown
C) Video editing
D) Sound control

The pre-test for the week 11

1- Stack works on:
A) LIFO
B) FIFO
C) Random
D) Sequential
2- Adding an element to Stack is called:
A) Push
B) Pop
C) Delete
D) Search
3- Removing an element from Stack is called:
A) Pop
B) Push
C) Insert
D) Save

4- In Stack, the last element entered is:
A) The first to leave
B) The last to leave
C) Never removed
D) Deleted مباشرة
5- Stack is a:
A) Data Structure
B) Browser
C) Operating System
D) File

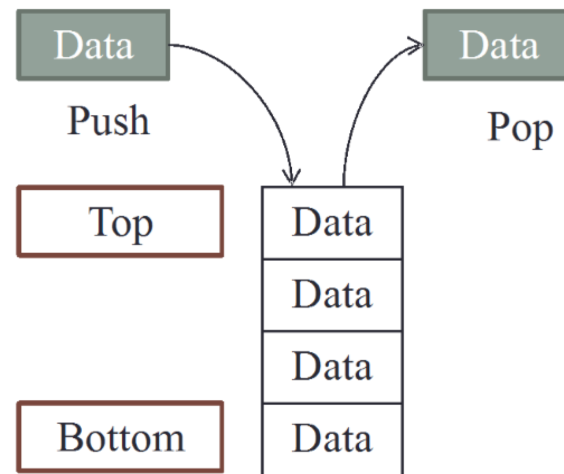
Week 11

Outline

- Stack
- Stack Operations
- Arrays to implement stacks
- Stack Applications

Stack

- Stack is a linear data structure which follows a particular order in which the operations are performed. The order is LIFO(Last In First Out)
- Objects are inserted into a stack at any time, but only the most recently inserted object (last one!) can be removed at any time
- New elements can be added and removed only at the top.



Stack Operations

- The fundamental operations involved in a stack are "**push**" and "**pop**".
  - ***push***: adds a new element on the stack
  - ***pop***: removes an element from the stack

- Checking Conditions
  - ***top>=StackSize-1 :*** Stack is full.. "Stack OVERFLOW"
  - ***top== -1 or zero***: Stack is empty.."Stack UNDERFLOW"

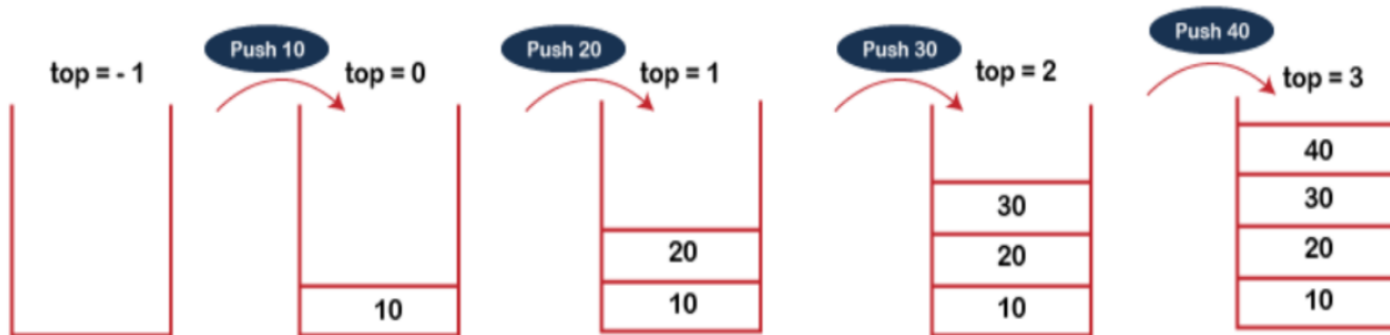
Example

- Instruction
  - Pop()
  - Push (A)
  - Push (D)
  - Push (F)
  - POP ()
  - POP()
  - PUSH(X)

PUSH operation

The steps involved in the PUSH operation is given below:

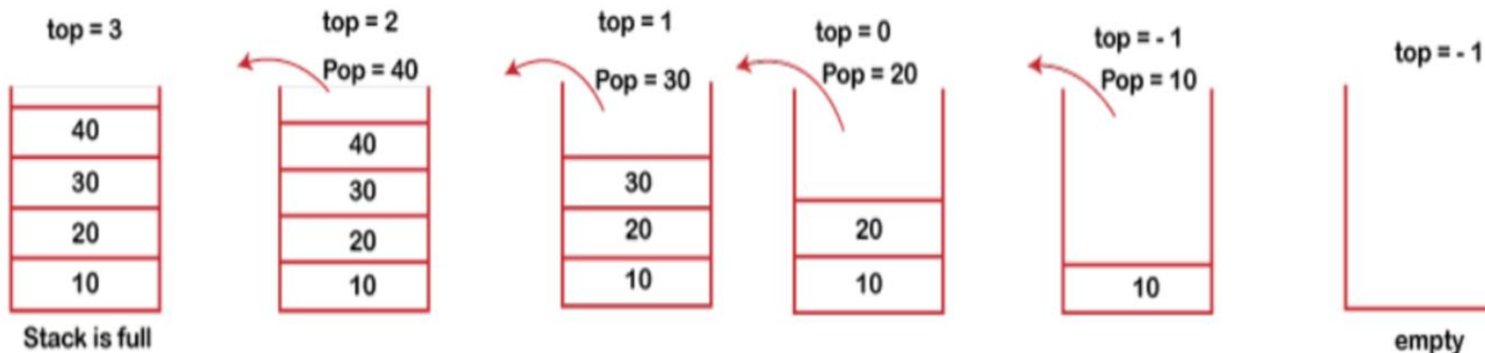
- Before inserting an element in a stack, we check whether the stack is full.
- If we try to insert the element in a stack, and the stack is full, then the ***overflow*** condition occurs.
- When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
- When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., **top=top+1**, and the element will be placed at the new position of the **top**.
- The elements will be inserted until we reach the ***max*** size of the stack.



POP operation

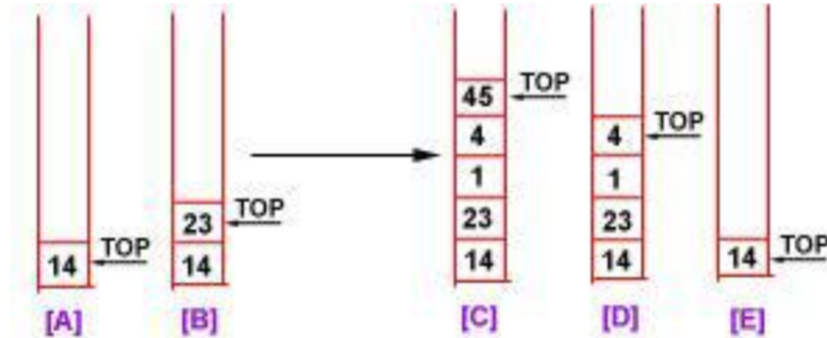
The steps involved in the POP operation is given below:

- Before deleting the element from the stack, we check whether the stack is empty.
- If we try to delete the element from the empty stack, then the ***underflow*** condition occurs.
- If the stack is not empty, we first access the element which is pointed by the ***top***
- Once the pop operation is performed, the top is decremented by 1, i.e., **top=top-1**.



Operation on a stack

- Example:
- Let us consider a stack with integers 14, 23, 1, 4, 45 and upper limit set to 5.
- Note: Every time you Push() an element inside the Stack, the TOP of the Stack gets incremented by 1 and vice versa.
- When the STACK is empty, the BOTTOM and TOP of the STACK are same and pointing to the empty STACK.
- If you try to PUSH more elements than the upper limit of the STACK, it will cause in an overflow of data and vice-versa.



- Initially, the STACK is empty and the TOP of the STACK is pointing to the BOTTOM of the STACK.
- Stage [A]: We added 14 and TOP now points to it.
- Stage [B]: 23 is PUSHed and TOP is incremented by 1.
- Stage [C]: The STACK is FULL, as the upper limit was set to 5.
- Stage [D]: The TOP most element has been POPed. The TOP gets decremented by 1.
- Stage [E]: 45, 4, 1 and 23 have been POPed and TOP is now pointing to the bottom most element

Stack functions

- To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks

- **peek()** : get the top data element of the stack, without removing it.

- **isFull()** : check if stack is full.

- **isEmpty()** : check if stack is empty.

Pushing elements on the stack

- ***Push*** operation adds a new element on the stack.

<u>Algorithm</u>

1- Start

2- Let stack [size]

3- let top=-1

4- if top<stack.size then

5- top=top+1

6- stack[top]=data

7- else

8- Print (" Stack Is Full ", "Data Over flow")

9- End

Popping Elements From Stack

- ***Pop*** operation removes a element from the stack.

Algorithm

1- Start

2- If Top>=0 then

3- print stack[top]

4- top=top-1

5- else

6- Print (“ Stack Is Empty , Data Under Flow “)

7- end

Stack Applications

- Web browser: stores the addresses of recently visited sites on a stack. Each time a user visits a new site, the address of the site is *pushed* into the stack of addresses. Use the back button the user can *pop* pack to previously visited sites.
- Text editors: powerful text editors keep text changes in a stack. The user can use the undo mechanism to cancel recent editing operations.
- Arithmetic expression: following to next slide

Implementation of Stack

- There are two ways to implement a stack:

1. Using array
2. Using linked list

Using arrays to implement stacks

- A natural way of implementing a stack is with an array.
- Top represents the index in the array at which the next item pushed onto the stack is to be stored. This increases by one when another item is pushed onto the stack, and decreases by one when an item is popped.

- The stack is empty when the top =-1, and it is full if the top pointer is greater than the maximum size of the stack array. This is because top now lies beyond the bounds of the array.

OUTLINE

- Arithmetic expressions
- Precedence ,primacy, priority
- Prefix, Postfix & Infix Notation
- Algorithm to convert infix to postfix
- Examples

Arithmetic expressions

- An arithmetic expression is one which is evaluated by performing a sequence of arithmetic operations to obtain a numeric value.
- Levels of Precedence for the usual five binary operations on arithmetic operation Q .
  - Highest: Exponentiation ^
  - Next Highest: Multiplication * and division /
  - Lowest: Addition + and subtraction -

Precedence ,primacy, priority

Operator(s)	Precedence & Associativity
( )	Evaluated first. If **nested (embedded)**, innermost first. If on same level, left to right.
* / %	Evaluated second. If there are several, evaluated left to right
+ -	Evaluated third. If there are several, evaluated left to right.
=	Evaluated last, right to left.

Prefix, Postfix & Infix Notation

- Prefix, Postfix & Infix Notation
- Infix : the operator is placed between operands.
  - Example: (A+B)*C parentheses necessary
- Prefix: the operator is placed before the operands
  - Example: *+ABC
- Postfix (Reveres polish notation): the operator is placed after the operands .
  - Example: AB+C*

Why?

- Why to use PREFIX and POSTFIX notations when we have simple INFIX notation?
- INFIX notations are not as simple as they seem especially while evaluating them. To evaluate an infix expression we need to consider Operators' Priority and Associative property
  - Example:
  - expression 3+5*4 evaluate to 32 i.e. (3+5)*4 or to 23 i.e. 3+(5*4).
- To solve this problem Precedence or Priority of the operators were defined. Operator precedence governs evaluation order. An operator with higher precedence is applied before an operator with lower precedence.

Algorithm to convert infix to postfix

- A summary of the rules follows:

1. Print operands as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.
3. If the incoming symbol is a left parenthesis, push it on the stack.
4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.
7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.
8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

Rules

- Four rules you should know to convert from infix to postfix

1. Priority of operands
2. No two operands of same priority can stay in a stack
3. Lowest priority can not be placed after highest priority
4. If the symbol is a close parenthesis then the operands in between should be popped out the stack.

Examples of infix to prefix and postfix

Infix	Prefix	Postfix
(A+B)/D	/+ABD	AB+D/
(A+B)/(D+E)	/+AB+DE	AB+DE+/
(A-B/C+E)/(A+B)	/+-A/BCE+AB	ABC/-E+AB+/
B^2-4*A*C	-^B2**4AC	B2^4A*C*-
A-B/(C*D^E)	-A/B*C^DE	ABCDE^*/-

Example: postfix expressions

- Postfix notation is another way of writing arithmetic expressions.
- In postfix notation, the operator is written after the two operands.
  - infix: 2+5 postfix: 2 5 +
- Expressions are evaluated from left to right.
- Precedence rules and parentheses are never need

Infix	Postfix	evaluation
2 - 3 * 4 + 5	234*-5+	-5
(2 - 3) * (4 + 5)	23-45+*	-9
2- (3 * 4 +5)	234*5+-	-15

Converting between notations

- **1. Infix to Prefix:**
- (A+B)-(C*D)
- - Do the first brace: (A+B) ,the prefix is +AB
- - Do the second brace : (C*D) , the prefix is *CD
- - The end is operator-: +AB - *CD
- The prefix is: - + A B * C D

Converting between notations

- **2. Infix to Postfix:**
- (A+B)-(C*D)
- - Do the first brace: (A+B) , the Postfix is AB+
- - Do the second brace : (C*D), the Postfix is CD*
- - The end is operator-: AB+ - CD*
- The postfix is A B + C D * -

The post-test for the week 11

1- What does LIFO mean?
A) Last In First Out
B) First In First Out
C) Last In Last Out
D) First In Last Out
2- Which operation removes an element from Stack?
A) Pop
B) Push
C) Insert
D) Print
3- Stack is commonly used in:
A) Function calls
B) Video editing
C) Printing only
D) Music playing

4- Which operation adds a new element?
A) Push
B) Pop
C) Delete
D) Save
5- Stack depends on:
A) Order of elements
B) Images
C) Videos
D) Audio files

The pre-test for the week 12

1- Queue works on:
A) FIFO
B) LIFO
C) Random
D) Stack
2- In Queue, the first element entered is:
A) The first to leave
B) The last to leave
C) Never removed
D) Deleted directly
3- Adding an element to Queue is called:
A) Enqueuers
B) Push
C) Pop
D) Delete

4- Removing an element from Queue is called:
A) Dequeuers
B) Insert
C) Push
D) Search
5- Queue is a:
A) Data Structure
B) Screen
C) File
D) Game

Week 12

OUTLINE

- What is a queue
- Queue operations
- Applications of Queue
- Types Of queue
- Circular Queue
- Double ended queue
- Priority queue

What is a queue

- Is a linear data structure.
- Contains elements that are inserted and removed according to the first-in-first-out (FIFO) principle .
- The new elements are added at the end ("the rear") and elements are removed from the other end ("the front").

Queues Operations

- It supports two fundamental methods:
  - Enqueue: inserts an element at the end of the queue
  - Dequeue: removes an element from the front of the queue

- Checking Conditions:
  - Queue Overflow :If rear = maxsize-1
  - Queue Empty: If front = -1

Operations on Queues … Cont

- The queue has two pointers:
  - FRONT: containing the location of the front element.
  - REAR: containing the location of the rear element.
- The new element will add itself at the 'rear' end, then Queue's **‘REAR' value increments by one.**
- The element leaves the queue from the 'front' end, so the Queue's **‘FRONT' value increments by one.**

Algorithm for inserting an element into a queue

Step1: Start

Step2: let queue[Size]

Step3: let front=-1 , Rear=-1 // Initialization Queue

Step4: If (rear = queue . Size -1) Then

print (Queue Is Full“Data Data Over flow “)

Else

rear=rear+1

queue[rear]=item

Step5: If front = -1 Then

Front= 0

Step6: End

Algorithm for deleting an element from a queue

Step1: Start

Step 2: IF (front = -1) then

Print (“ Queue Is Empty “ , “ Data Underflow “)

ELSE

print(queue[front])

Step 3: IF (front=rear) then

front = -1 , rear=-1

Else

front =front+1

Step 4:End

Applications of Queue

- Used in scheduling the jobs to be processed by the processor.
- A queue schedules the order of the print files to be printed.
- A server maintains a queue of the client requests to be processed

Queue Types

- Linear Queue
- Circular Queue
- Double ended Queue
- Priority Queue

Circular Queue

- A circular queue is a Queue but with a particular implementation of a queue.
  - They have a circular structure.
  - There is no space lost.
- The properties of this type of queues is :-
  - Front pointing to the first item.
  - Rear pointing to the last item.
  - when Rear arrives to the end of the queue make it wrap to the beginning of queue (rear =0) , also this with front.

0	1	2	3	4	5	6	7
A	B	C	D				

Front = 0 Rear = 3

0	1	2	3	4	5	6	7
		C	D	E	F	G	H

Front = 2 Rear = 7

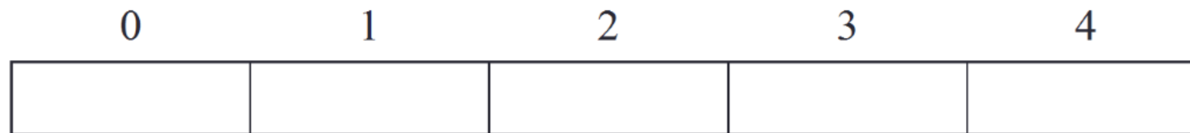
0	1	2	3	4	5	6	7
I		C	D	E	F	G	H

Rear = 0 Front = 2

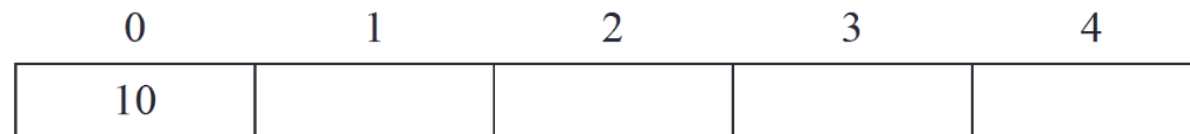
Example

- Consider the following circular queue with size 5

1- Initially Rear =-1 Front = -1

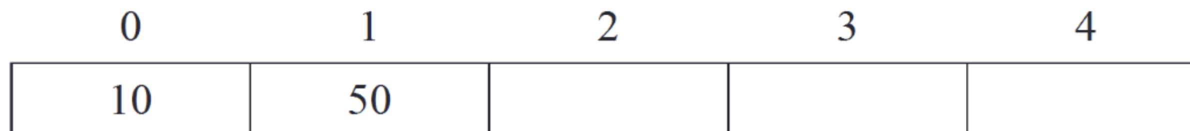


2- Insert 10, Rear = 1 Front =1



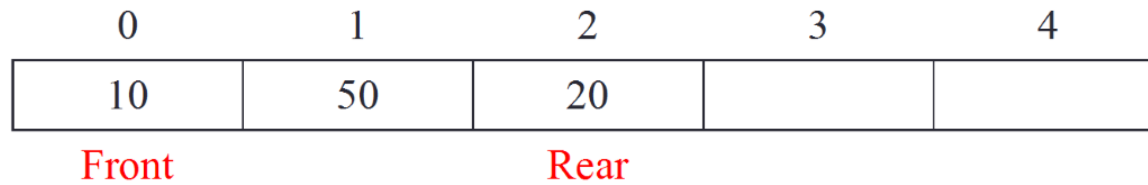
Rear Front

3- Insert 50 Rear = 1 Front =0

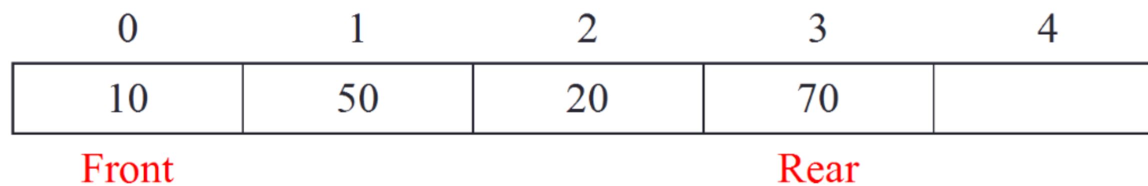


Front Rear

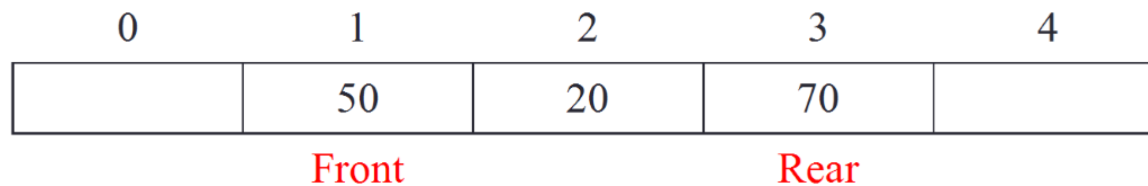
4- Insert 20, Rear = 2, Front =0



5- Insert 70, Rear = 3, Front =0



6- Delete front, Rear = 3, Front =1



7- Insert 100, Rear = 4, Front =1

0	1	2	3	4
	50	20	70	100
	Front			Rear

8- Insert 40, Rear =0, Front =1

0	1	2	3	4
40	50	20	70	100
Rear	Front			

9- Insert 140, Rear = 0, Front =1, As Front = Rear – 1 so the queue is full (overflow)

0	1	2	3	4
40	50	20	70	100
Rear	Front			

10- Delete front, Rear = 0, Front = 2

0	1	2	3	4
40		20	70	100
Rear		Front		

11- Delete front, Rear = 0, Front =3

0	1	2	3	4
40			70	100
Rear			Front	

12- Delete front, Rear = 0, Front=4

0	1	2	3	4
40				100
Rear				Front

Algorithm to insert an element in a circular queue

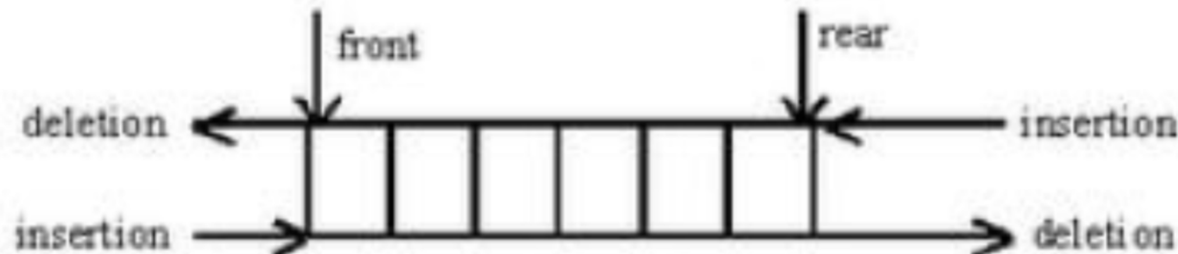
```
Step1: Start
Step2: p = (Rear+1) MOD CQueue.length;
Step3: If (p = Front) Then
                    print(“ Queue Overflow ");
          else
                    Rear = p;
                    CQueue[Rear] = Item;
Step4: if (Front = -1)
                    Front = 0;
Step5: End
```

Algorithm to delete an element from a circular

```
Step1: Start
Step2: If(Front =-1)
                    print ("CQUEUE IS EMPTY")
          Else
                    Print(CQueue[Front])
Step3:  If ( Front == Rear )
                    Front = Rear = -1
          Else
                    Front=(Front+1) Mod CQueue.length
Step4: END
```

Double Ended queue

- It is a linear list in which elements are added or removed at either end but not in middle.
- A double-ended queue is a data structure that supports the following operations:
  - enq_front
  - enq_back
  - deq_front
  - deq_back



Double Ended queue

- Double Ended Queue can be represented in two ways:

1) Input restricted De-Queue :- allows insertions at only one end but allows deletions on both ends of the list .

2) Output-restricted de-queue:- allows deletions at only one end but allows insertions at both ends of the list.

Priority queue

- A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the following
- rules:

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with the same priority are processed according to the order in which they were added to the queue.

The post-test for the week 12

1- What does FIFO mean?
A) First In First Out
B) Last In First Out
C) First In Last Out
D) Last In Last Out
2- Queue is used in:
A) Process scheduling
B) Painting
C) Watching videos
D) Playing games
3- Which operation inserts an element into Queue?
A) Enqueue
B) Dequeue
C) Delete
D) Save

4- Which operation removes an element from Queue?
A) Dequeue
B) Push
C) Insert
D) Print
5- Queue depends on:
A) Time order
B) Images
C) Audio
D) Videos

The pre-test for the week 13-15

1- A Graph consists of:
A) Vertices and Edges
B) Images
C) Files
D) Programs

2- Vertices mean:
A) Nodes
B) Pictures
C) Files
D) Text

3- Edges represent:
A) Connections
B) Images
C) Videos
D) Programs

4- Graph is a:
A) Data Structure
B) Operating System
C) Browser
D) Application

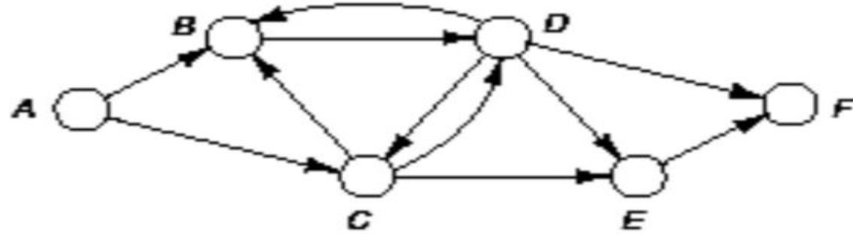
5- Graphs are used in:
A) Networks
B) Printing
C) Drawing only
D) Audio editing

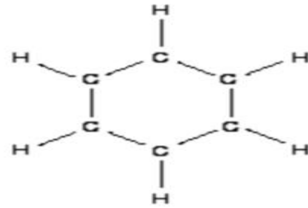
Week 13-15

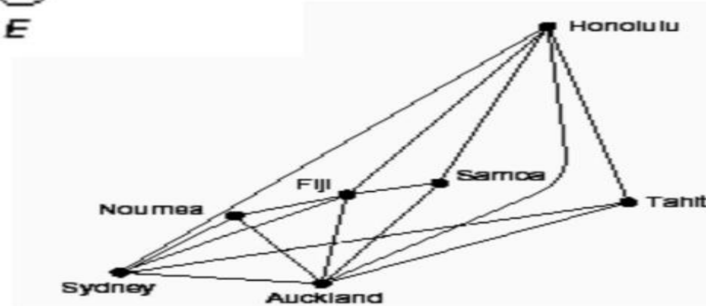
1- Examples and Definitions

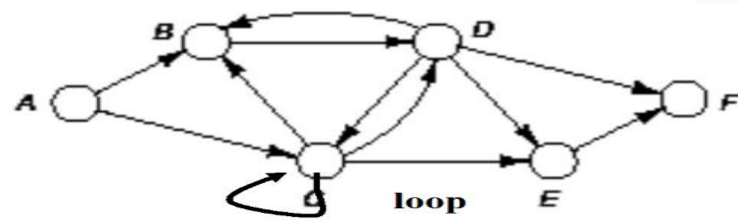
A graph can be thought of as collection of points joined together by ***lines*** or ***Arcs***.

- The points on a graph are called ***Vertices*** or ***Nodes***.





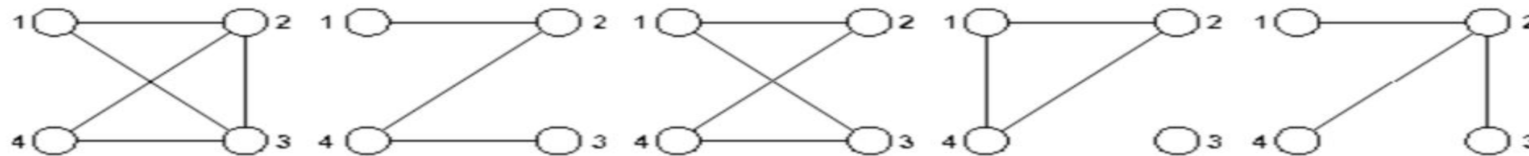




- A node which is not adjacent to any other node called is an ***Isolated node***.
- A graph is called ***connected*** if there is a path from any vertex to any other vertex.

- If graph is disconnected, we shall refer to a maximal subset of connected vertices as a ***component***.
- ***Path*** from one node to Another is a sequence of arcs.
- ***Loop*** is an edge of the graph which join node to itself.
- ***Cycle:*** is the path that begin and end with the same vertex and no arcs occurs more than once in the path.
- ***Cyclic Graph:*** is a graph that have a cycle.
- ***Acyclic Graph:*** is a graph that have no cycle in it.
- ***Simple path:*** path does not contain cycle.
- ***Adjacent nodes:*** Nodes that are connected by an edge in the graph.

Length of the path: is the number of arcs appearing in the sequence of the path.



(a) Connected (b) Path (c)Cycle (d)Disconnected (e) Tree

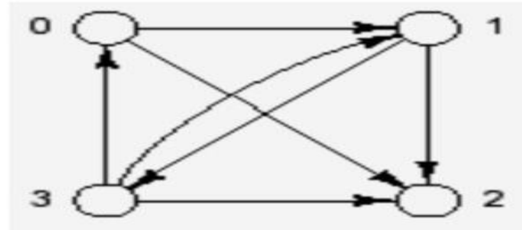
Various kinds of undirected graphs

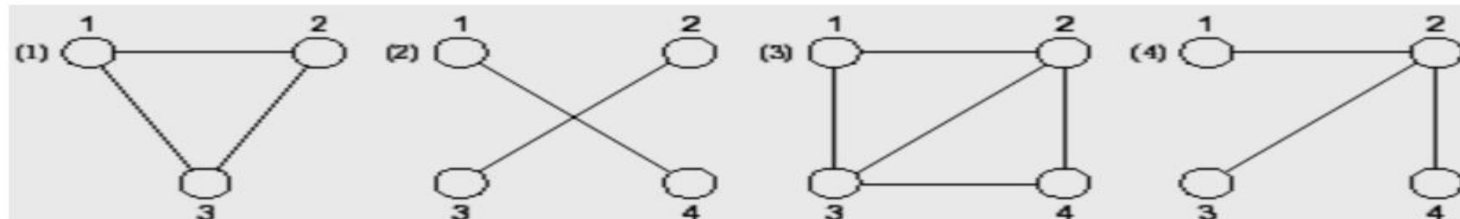
- ***Graph*** is collection of vertices (V) connected with edges (G)

G=(E,V)
In figure (17-c):
Vertics V(G)= { A, B, C, D, E, F }
Edges E(G)= { (A,B), (B,D), (D,F), (D,B), (D,C),(C,B)(C,D),(A,C),(C,E), (C,D) }

2- Directed and Undirected Graphs:

If the pairs are unordered, then G is called an ***undirected graph***; if the pairs are ordered, then G is called a ***directed graph***. The term *directed graph* is often shortened to ***digraph***, and the unqualified term *graph* usually means *undirected graph*. The natural way to picture a graph is to represent vertices as points or circles and edges as line segments or arcs connecting the vertices. If the graph is directed, then the line

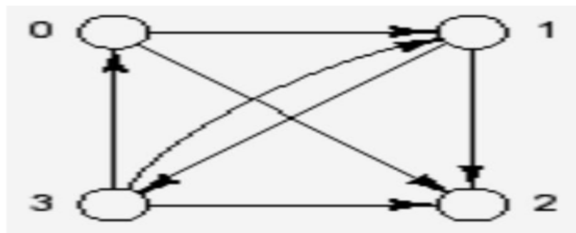




Definitions:

Directed graph: Is a graph that the relation between its edges are ordered, the direction is important to define the relation.

Undirected graph: Is a graph that the relation between nodes is not ordered (undirected), i.e the relation (A,D) is as (D,A).



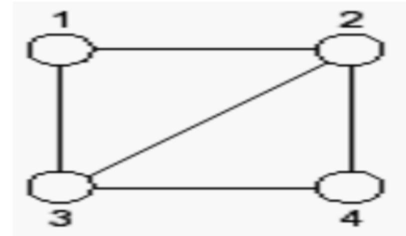


Figure 18.3: (a) directed graph (b) undirected graph

Graph Degree In Directed Graph:

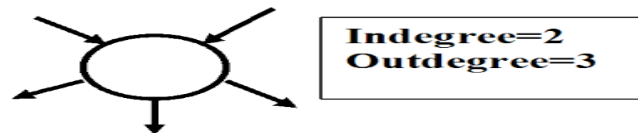
Indegree(Degree): Number of edges any node have as terminal node (input edges).

Outdegree: Number of edges any node have as initial node (output edges).

Totaldegree: Number of edges any node have as initial and terminal node (input and output edges).

Graph degree: the most degree node.

Ex: consider the following node:



3- Graphs Representation:

There are wo methods for graph representation:

A-The Set Representation

Graphs are defined in terms of sets, and it is natural to look first to sets to determine their representation as data. First, we have a set of vertices, and, second, we have the edges as a set of pairs of vertices. Rather than attempting to represent this set of pairs directly, we divide it into pieces by considering the set of edges attached to each vertex separately.

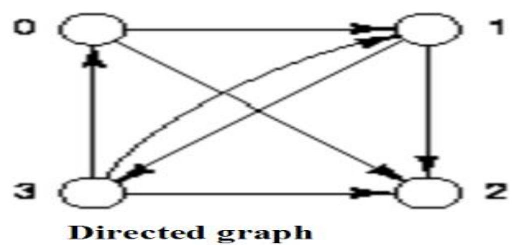
1. Implementation of Sets

There are two general ways for us to implement sets of vertices in data structures and algorithms. One way is to represent the set as a *list* of its elements, the other implementation, often called a ***bit string***, keeps a Boolean value for each potential element of the set to indicate whether or not it is in the set.

2. Adjacency Tables (Matrix)

In the foregoing implementation, the structure Set is essentially implemented as an array of **bool** entries. Each entry indicates whether or not the corresponding vertex is a member of the set. If we substitute this array for a set of neighbors, we find that the array neighbors in the definition of **class** Graph can be changed to an array of arrays, that is, to a two-dimensional array, as follows:

The adjacency table has a natural interpretation: adjacency[v][w] is true if and only if vertex v is adjacent to vertex w. If the graph is directed, we interpret adjacency [v][w] as indicating whether or not the edge from v to w is in the graph. If the graph is undirected, then the adjacency table must be symmetric; that is, adjacency [v][w] = adjacency[w][v] for all v and w. The representation of a graph by adjacency sets and by an adjacency table is illustrated in figure below.



Directed graph

vertex	Set
0	{ 1, 2 }
1	{ 2, 3 }
2	∅
3	{ 0, 1, 2 }

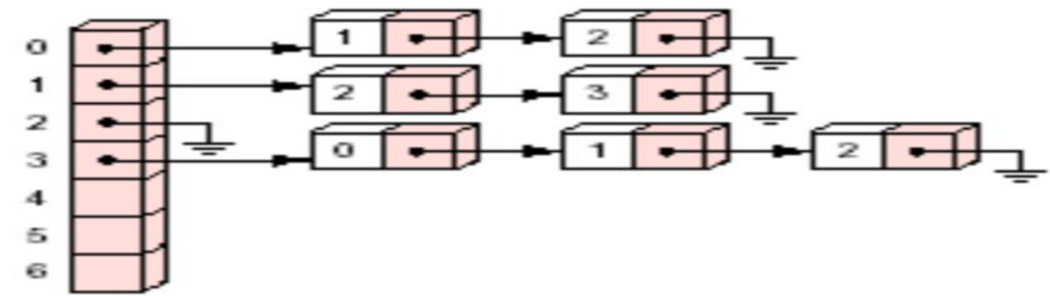
Adjacency sets

	0	1	2	3
0	F	T	T	F
1	F	F	T	T
2	F	F	F	F
3	T	T	T	F

Adjacency table

3- Adjacency Lists

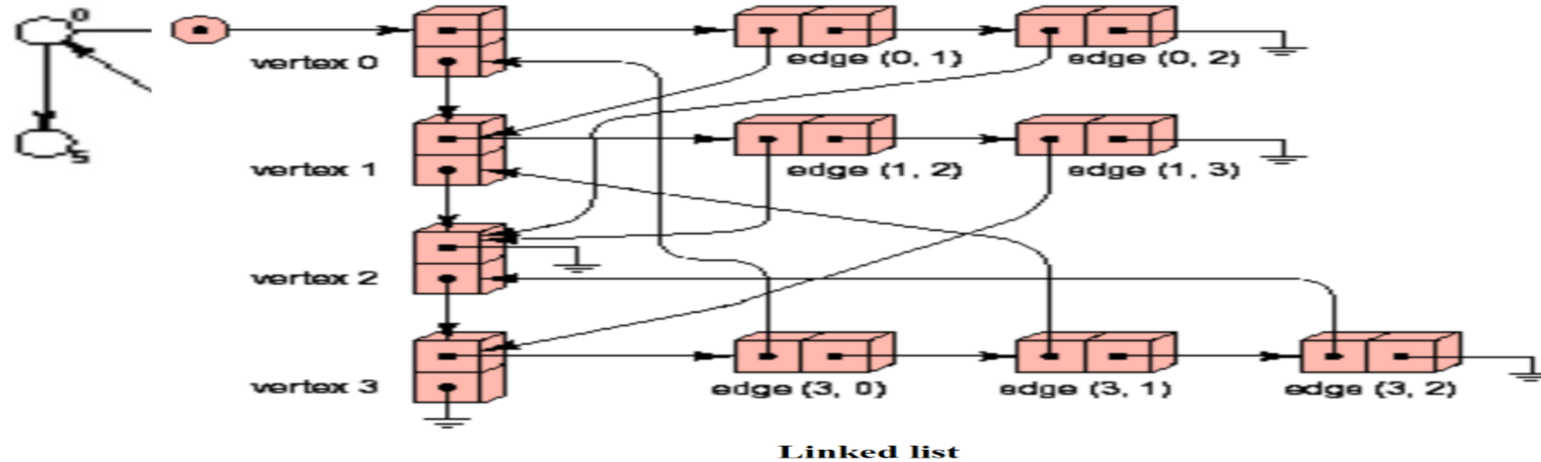
Another way to represent a set is as a *list* of its elements. For representing a graph, we shall then have both a list of vertices and, for each vertex, a list of adjacent vertices. We can consider implementations of graphs that use either contiguous lists or simply linked lists. For more advanced applications, however, it is often useful to employ more sophisticated implementations of lists as binary or multiway search trees or as heaps. Note that, by identifying vertices with their indices in the previous representations, we have *ipso facto* implemented the vertex set as a contiguous list, but now we should make a deliberate choice concerning the use of contiguous or linked lists.



Adjacency Lists

B- Linked Implementation

Greatest flexibility is obtained by using linked objects for both the vertices and the adjacency lists. This implementation is illustrated in the following figure and results in a definition such as the following:



Linked list

The post-test for the week 13-15

1- Which of the following represents a Graph?
A) Nodes and Edges
B) Keyboard
C) Monitor
D) Printer
2- Graphs are used for:
A) Representing relationships
B) Deleting files
C) Playing music
D) Watching videos
3- A Vertex represents:
A) Node
B) Connection
C) File
D) Picture

4- An Edge represents:
A) Connection between nodes
B) Screen
C) Text
D) Audio
5- Graphs are important in:
A) Computer Networks
B) Video games only
C) Painting
D) Music editing

Answer Keys

Week 1

Pre-test

Q1	B
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	C
Q3	A
Q4	A
Q5	A

Week 2

Pre-test

Q1	A
Q2	A
Q3	A
Q4	B
Q5	A

Post-test

Q1	B
Q2	A
Q3	B
Q4	A
Q5	A

Week 3-5

Pre-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	D

Week 6-7

Pre-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Week 8-10

Pre-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	A
Q3	A
Q4	D
Q5	A

Week 12

Pre-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Week 13-15

Pre-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Post-test

Q1	A
Q2	A
Q3	A
Q4	A
Q5	A

Thank you