Southern Technical University
Technical Institute / Qurna
Dep. of Computer Systems Techniques

Second class
Subject : Data Structures
Lecturer : Israa Mahmood Hayder
Lecture no.8,9

الهياكل الموصولة

(Linked Structures)

الاسبوع الثامن- التاسع -

الثامن - تاسع	القوائم المترابطة • تعريف القائمة المترابطة • أنواع القوائم المترابطة وطرق تمثيلها • القائمة البسيطة / قراءة العناصر – طباعة القائمة – حشر عنصر في القائمة (مقدمة، موقع محدد، مؤخرة)

B// Rationale (مبررات الوحدة) :-

- A linked list is a data structure that makes it easy to rearrange data without having to move data in memory. The student will learn about Types of Storage allocation and types of linked lists

C// Central (الفكرة المركزية):-

- Types of Storage allocation
- Comparison between Sequential and Dynamic Storage allocation
- Pointers
- Operations on single Linked List

D// Objectives (أهداف الوحدة):-

After studying this unit, the student will be able to:-

- Realize the difference between Sequential and Dynamic Storage allocation
- Define pointers and use them in linked lists
- Write Operations on single Linked List

A// Storage allocation

There are two types of storage allocation depending on the structure of the data:

1- Sequential Allocation Storage

Is the simples way to store lists in memory sequentially, and from the *Base address* which is the first location of the list, we can know the location of any item in the list.

Advantages

1- Simple in representation
2- Take less memory space
3- Efficient in random access

Disadvantages

1- Hard to apply addition and deletion
2- Number of elements must be predefined

2- Dynamic Allocation Storage

The second way to store lists is to use link (or pointer), each element contain the location of the next element, so elements may not stored sequentially in memory.

Each element (node) consist of 2 parts:
1) Data
2)pointer (link) to the next address

Advantages

1- insertion and deletion is easy to implement (not need shifting)
2- Easy to merge and split by only change the pointers

Disadvantages

1- Take more memory space
2- To access any element randomly, we must start from the beginning of the list

B//Comparison between Sequential and dynamic Storage allocation:

1- Ammont of storage

The dynamic storage need more memory space because of the need to use pointer to next element.

2-Insertion and deletion operations

These operations simplest to execute in dynamical storage because they don't need shifting.

3-Random access

The sequential way is easier in accessing randomly, but the dynamic way require to start searching from the beginning of the list.

4-Merge and sort

In the dynamic storage these operations are simple to execute by only change the pointer in merging location while the sequential storage need shifting and reorganization.

Quiz1:

What are the advantages of each Dynamic and Sequential storage allocation?

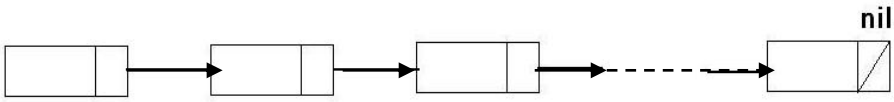
C- Types of lists :-

1- **Non-Linked List :-** do not use pointers it's structure, it use the vectors and array for representing it's structure.
2- **Linked Lists :-** a list has been defined to contain an ordered list of elements, each element (node) contains a link or pointer to the next node.

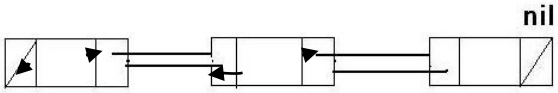
Linked lists :- A list that use pointers or link to refer to the elements of data structures, in a way that element which have logically adjacent need not to be physically adjacent in memory.

Types of linked lists:-

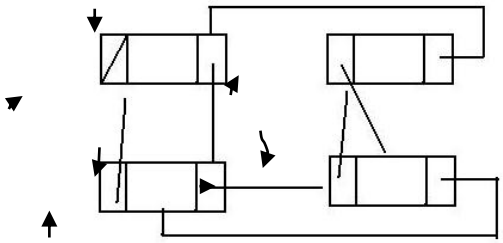
1- **Single Linked List :-** is a list contains set of elements, and each element (node) contain a link or pointer to the next node.



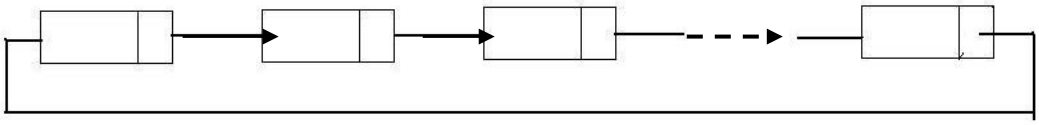
2- **Multi Linked List :-** a list has more than one pointer, like doubly linked list which has two pointers pointing to the previous and next node.



OR



3- **Circuler Linked List :-** a list that last node points to the first node.



Operation on Lists :-

1- Insertion
2- Deletion
3- Search
4- Change

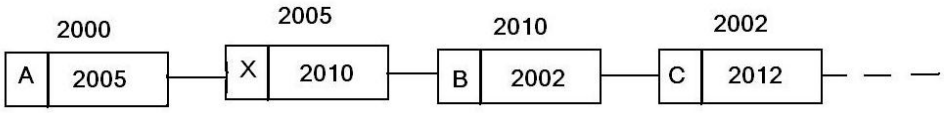
Ex Consider the following Linked List (Ordered) :-

	Address	data
1	2000	A
2	2010	B

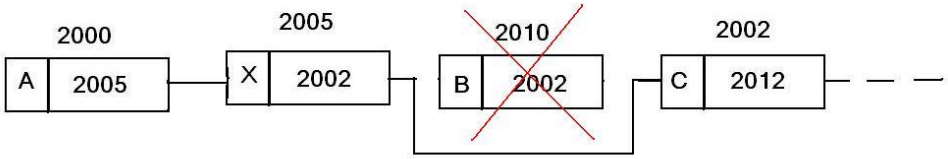
3	2002	C
4	2012	D
5	2006	E

a) Draw The List:-

b) inset node X after A at location 2005



c) Delete The Node B



Quiz2: Draw The following Linked List (Ordered) after inserting C after B with address 4003 :-

	Address	data
1	2000	A
2	2010	B
3	2012	D
4	2006	E

1- Dynamic storage

- ***Structures:***

Record is a connected data like arrays, but it can contain different types of data like record in the database. Field contains number of fields that differs in data in other records.

In C++ the record is defined as follows:

```
Name Struct
{
  fields
}
```

Example: define record "data" contain name, edge:

```
Struct data
 {
      Char nam[30];
  Int age;
  };
```

To define var of type record:

```
Struct data
{
    Type field1;
    Type field2;
                .....
 } var1;
```

Example:

```
#include<iostream.h>
Struct student
{
Char*  name;
Int  no;
};
Main()
{
Student sdt1;
Std1.name="Mohammed";
Cout<<std1.name;
}
```

When executing the program the name "Mohammed" is saved in the faild name of the variable "std1" then print it.

- ***Array of records:***

The record can be an array:

```
Struct student
{
Char*  name;
Int  no;
} data  ;
data  student[100];      // define array of type data
```

and to use the record contents use the following way:

Student[index].name & student[index].age

- ***Records and Pointers:***

After defining the record it can be pointer as follows:

Example:

```
#include<iostream.h>
Struct student
{
Char*  name[30];
Int  age;
} data  ;

Int main()
{
Data *s,std;
S=&std;   // assign std to s
Strcpy(std.name,"Talal"
Std.age=20;
Cout<<std.name<<std.age<<endl;
Return 0;
}
```

using new:

```
float *q = new float    //empty
float *q = new float(3.14)      //contain 3.14
```

Ex

```
double *p=new double;
if (p==0) then abort ( );                //full memory
else
   *p=33.2
end if
```

Delete:-

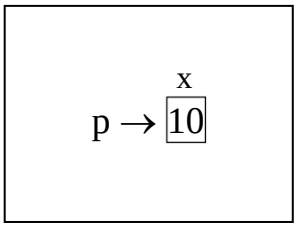
Ex:-

```
float *q =new float(3.14);
delete q;
q=5.2;              // error
```

Creating First Node In C++

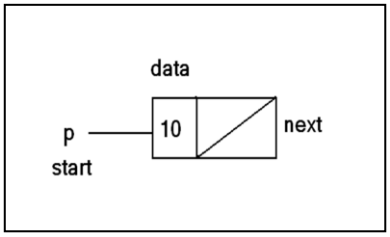
Ex1:-

```
main ( )
{
   int x;
   int *p;
   x=10;
   p=&x;
 } return 0;
```



Ex2:-

```
main( )
{
  struct node  {
                    int data;
                    struct node *next;
                 };
  struct node *p;
  struct node *start;
  p.data =10;
  p.next=nil;                    // in c++ nil is 0
  start=p;
```



3-Operation on Singly Linked List :-

Creating linked list of 2 nodes :-

Ex:-

```
new(p);
start=p;
read (p↑.data);
new(p2)
read (p2↑.data);
```

p↑.next=p2;
p2↑.next=nil;

Creating linked list of N nodes in C++ :-

```
main( )
{ int n ;
  struct node { int data;
                  struct node *next;
                }
  struct node *p= new  struct node ;
  struct node *start=p;
  struct node *p2;
   cin>>n;
for(i=1; i<=n ; i++)
   { cin>> p.data;
     if  i !=n then
         struct node *p2=new struct node;
     else
        p2=nil;
     p.next=p2;
     p=p2;}
```

Creating node using new :-

Algorithm :-

```
New (p)
Start=p
Read(p↑.data)
p↑.link=nil
```

In C++ :-

```
Struct node *p = new struct node;
Struct node *start = p ;
Cin >> p.data ;  p.next = nil;
```

Creating linked list of N node using new :-

Algorithm :-

```
New (p)
Start=p
Read (n)
For i=1 to n
  Begin
     Read(p↑.data)
     If  i< > n then new (p2 )
     Else  p2=nil;
     p↑.link=p2
```

```
    p=p2
  end;
```

Print The Linked List Elements :-

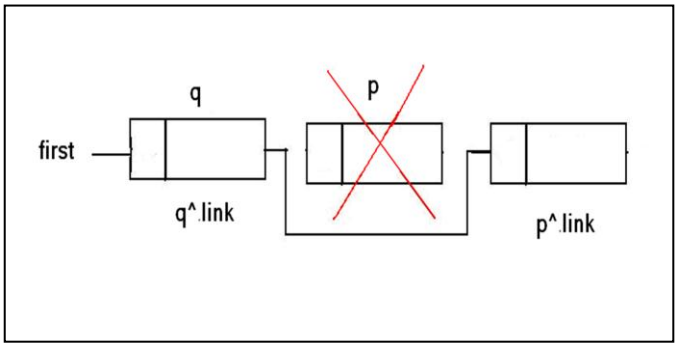
Algorithm :-

```
P= Start
While p< > nil do
  Begin
     writeln(p↑.data)
     p= p↑.link
   end;
```

Delete Element With Certain Value :-

Algorithm :-

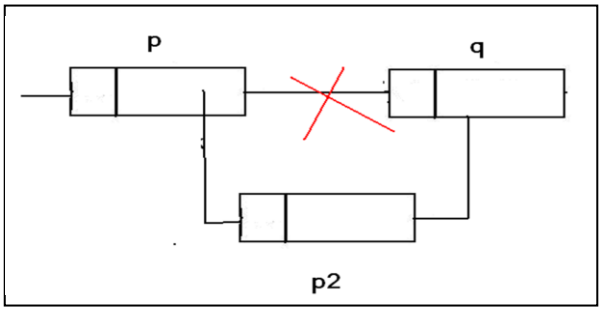
```
P= Start
While (p↑.data< > value do
  Begin
     Q=p ;   p= p↑.link
   end;
   q↑.link= p↑.link
    dispose (p);       { or delete (p)}
```



Insert element after P :-

Algorithm :-

```
New (p2)
      Read(p2↑.data)
      P2↑.link= P↑.link
      P↑.link =p2;    end;
```



Delete the first element :-

Algorithm :-

```
P= Start
Start= start↑.link
dispose (p);       { or delete (p)}
```

Delete the last element :-

Algorithm :-

```
P= head
If p↑.link=nil then
Begin
   dispose (p);       { or delete (p)}
    head = nil
```

```
end;
else
   While ( p↑.link< > nil ) do
    Begin
       Q=p ;   p= p↑.link
    end;
    q↑.link= nil
    dispose (p);       { or delete (p)}
```

Insert new node to the end of Linked List :-

Algorithm :-

```
P= Start
While p↑.link < > nil do
      p= p↑.link
new(q);
read(q↑.data)
q↑.link=nil;
p↑.link=q;
```

Insert new node at the position n in the Linked List :-

Algorithm :-

```
Read (n);
For i=1 to n do
begin
   p= p↑.link
   new(q);
   read(q↑.data)
   q↑.link= p↑.link ;
    p↑.link=q;
end if
```

Insert new node before P :-

Algorithm :-

```
   new(q);
    read (p↑.data)
   q↑.data= p↑.data
   q↑.link= p↑.link ;
    p↑.link=q;
```

delete the element P :-

Algorithm :-

```
   q== p↑.link ;
   q↑.data= p↑.data
   p↑.link= q↑.link ;
```

```
dispose (q);
```

Quiz3:

1) Delete the last element of single linked list
2) Print the single linked list

References:

1- Data Structures Demystified, by Jim Keogh and Ken Davidson, ISBN:0072253592, McGraw-Hill/Osborne © 2004

2- هياكل البيانات / الطبعة الثانية، تاليف د.عصام الصفار،اصدارات السفير للنشر/ بغداد، ٢٠٠١

3- الحقيبة التعليمية مادة "هياكل البيانات "، اعداد : نفارت الياس يوسف ،المعهد التقني كركوك-