internal function full linked list iteration

```
curr =head;
f(;curr!=0;curr = curr->next)
{
// code
}
```

---

external function

```
int x;
bool found = l.first(x);
while(found)
{
// code
found = l.next(x);
}
```

---

curr-> next to stand before
curr to stand on

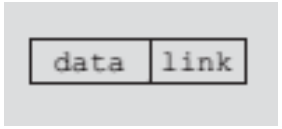
Linked Lists

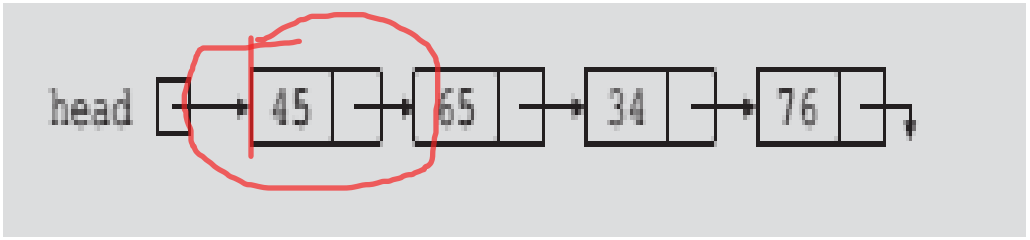
head == null only if list is empty or contain one element

What is a linked list?

A linked list is a data structure which is built from structures and pointers. It forms a chain of "nodes" with pointers representing the links of the chain and holding the entire thing together, in which the order of the nodes is determined by the address, called the link, stored in each node. A linked list can be represented by a diagram like this one.

The arrow in each node indicates that the address of the node to which it is pointing is stored in that node. The down arrow in the last node indicates that this link field is NULL.

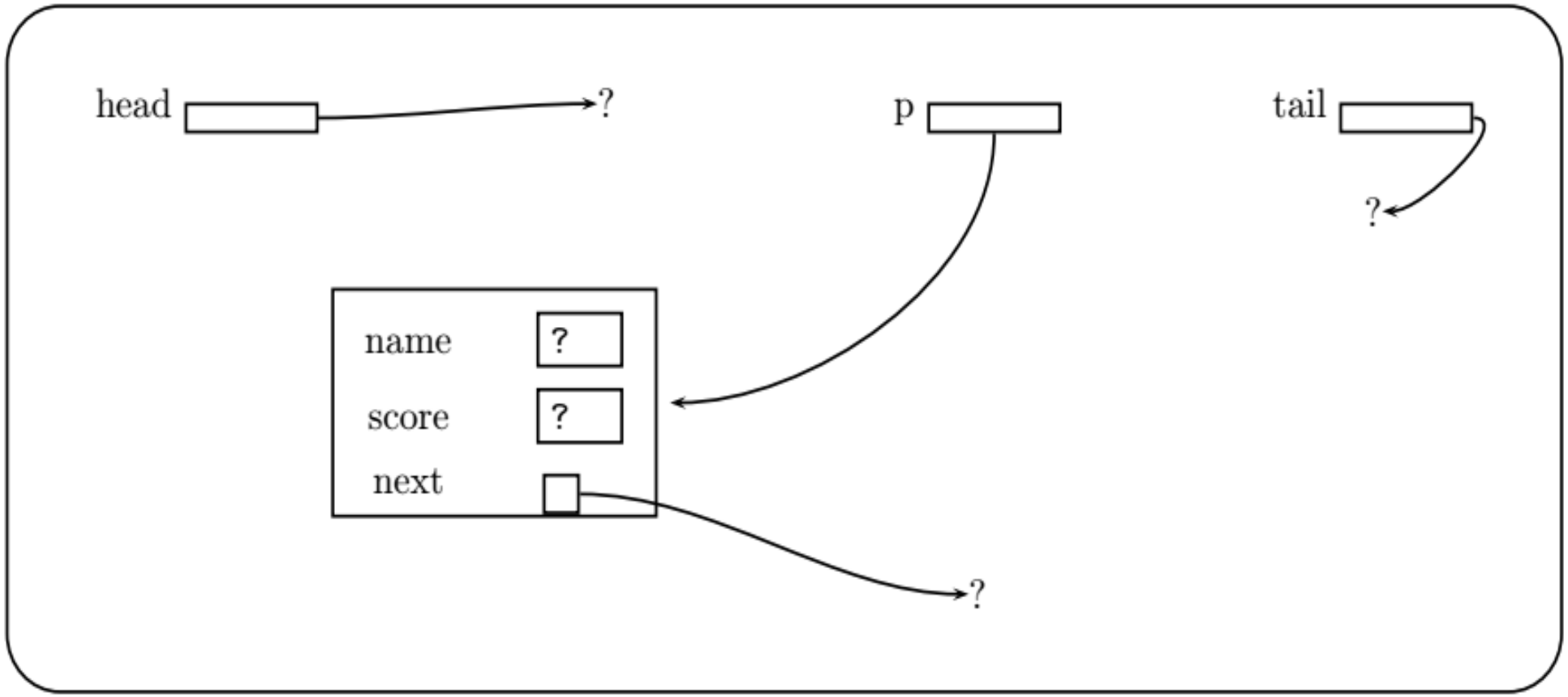




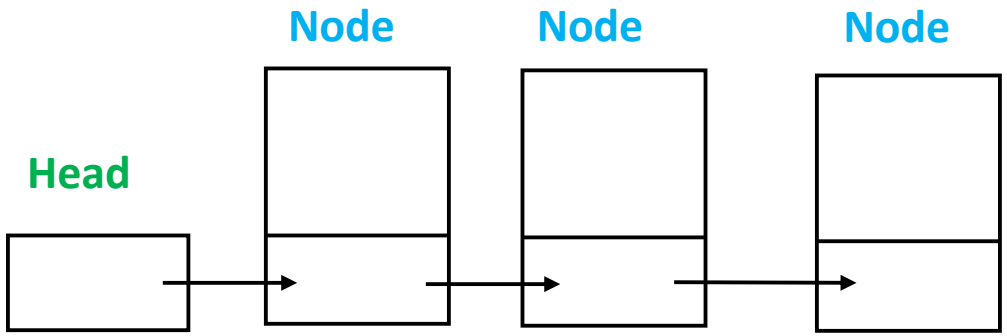
What can we do with a linked list?

1. Insert element into it
2. Remove element from it

   No shifting required
3. Traverse (iterate through) a linked list for various purposes such as displaying each element
4. Search for a particular element
5. Concatenate linked lists


head
?
p
tail
?
name
?
score
?
next
?

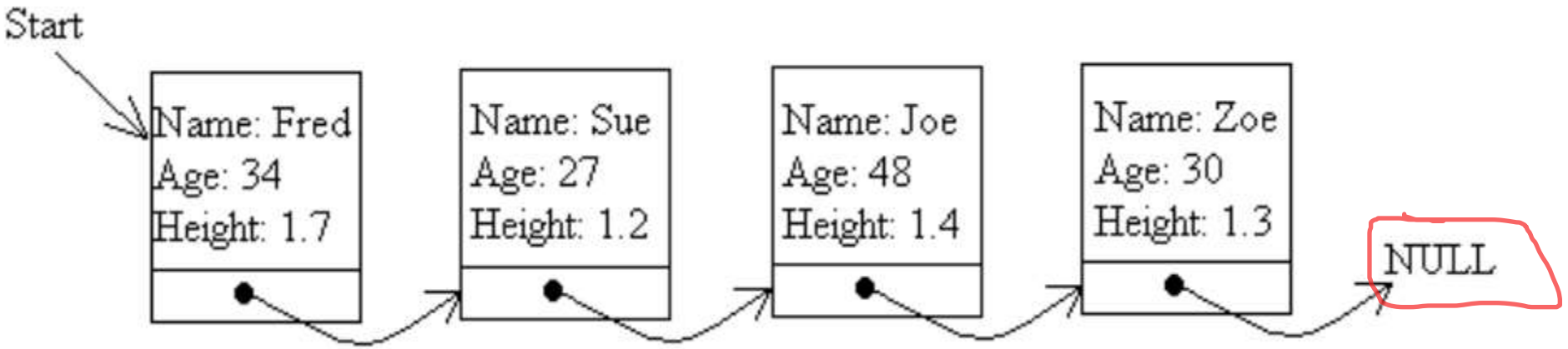
Made of **pointers** and **nodes**



not like array you reserve ahuge location array[10000]

Characteristics:

- Advantages: Flexible/unbounded size: it grows or shrinks as needed
- Disadvanteges: Sequential access
  - Elements must be accessed in some specific order dictated by the links



This linked list has four nodes in it, each with a link to the next node in the series. The last node has a link to the special value NULL The NULL pointer has been presented in the lecture and is used here, to show that it is the last link in the chain.

There is also another special pointer, called Start or head, which points to the first link in the chain so that we can keep track of it.

Defining the data structure for a linked list

The key part of a linked list is a structure, which holds the data for each node (any items in the list, and most importantly, a pointer to the next node). Here given the structure of a typical node:

```
struct Node
{ char name[20];
  int number;
    Node *next;        // Pointer to next node
        };
```

data

The important part of the structure is the line before the closing curly brackets. This gives a pointer to the next node in the list. This is the only case in C++ where you are allowed to refer to a data type (in this case Node) before you have even finished defining it!

Properties of linked lists

The linked list has nodes where,

1- The address of the first node is stored in the pointer **head.**

2- Each node has two components: info, to store the info, and link, to store the address of the next node.

3- **Tail**, the last node.

Consider the node of a linked list as follows:

Struct node

{ int info;

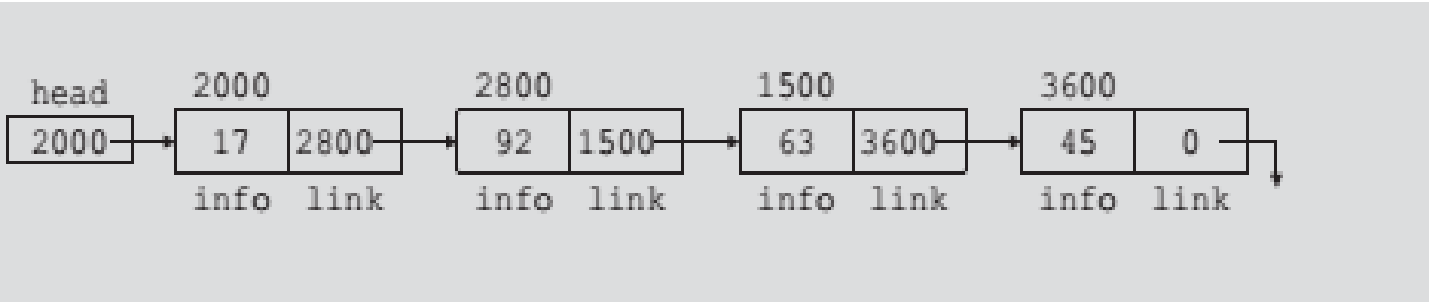
node *next; };

The variable declaration is as follows:

*node * head;*

If the value of the head is 2000, the data part of the first node is 17, and the link component of the first node contains 2800, the address of the second node, the data part of the second node is 92, and the link component of the second node contains 1500, the address of the third node and so on. We will use the arrow notation whenever we draw the figure of a linked list.

Head= 2000, *head→info=2800*



Declaration of Class of linked list

```
class  linked_list
{ private:
struct node;                    //line 1

typedef  node * link;           // line 2
struct  node { //now we define node
                elemtype  elem;
                link  next;     };
link  head, tail, current;  };
```

3 Pointers

typedef x y

x -> y

Notes

Line 1 : its purpose is to introduce the name (node) to the compiler so that it can be used later

Line 2 : (typedef node * link) means that a link is defined to be a pointer to a node

Line 3 : new pointers (head, tail and current)

Pointer =0, indicates null pointer.

Linked list class (cont.)

If the elements of list will be integers

```
typedef  int  elemtype ;
class  linked_list  {private: struct node;
                       typedef  node * link;
struct  node { //now we define node
                    elemtype  elem;   link  next;      };
   link  head, tail, current;
Public :
linked_list ( );              //constructor
Void insert( const elemtype &e);
bool  first ( elemtype & e);
bool  next ( elemtype &e);  };
```

Feature	Reference (`&`)	Pointer (`*`)
Syntax	`int &ref = a;`	`int *ptr = &a;`
Memory	No extra memory for the reference itself	Requires memory to store the address (pointer)
Reassignable	Cannot be reassigned to refer to another object	Can be reassigned to point to a different object
Nullability	Cannot be null	Can be null (`nullptr`)
Modifies the Original	Yes, directly modifies the original object	Yes, using dereferencing (`*ptr`)
Use in Functions	Often used for passing by reference (modify original variable)	Often used for passing by address (pointer)
Memory Access	Direct access to the object being referred to	Access using dereferencing (`*ptr`)
Size	Same size as the type being referenced (no additional memory for reference)	Typically 4 or 8 bytes depending on the system (for storing the address)

this is allias for the varible
passed deal with it by refrance

its linke other name for varible

Functions of linked list

```
void insert (x)        true
void insert (3)        false
void insert (x =3)     true
```

there is no allias for constant values

Notes on functions of linked list

insert( ): function is used to insert an element in the list calling by reference, and use const to prevent the function to change it.

first ( ) : it is used to return the first element in the list, if the list is empty, it will return false.

next ( ) : it is used to return the next element to the current node, if there is not a next node (current is tail), it will return false.

Linked list class (cont.)

Constructor function:

linked_list :: linked_list ( )

{ //initialize an empty list

head = 0; tail = 0; current = 0; }

1) assert(condeition) if condition is false programme stops
2) // Checks if memory allocation succeeded

insert function, to insert node at the end of the list

void linked_list :: insert (const elemtype & e)

{ link addnode (new node); // dynamic memory allocation of a node

assert (addnode); // to be sure that a node was allocated, if no space is not available, it returns 0

addnode-→elem= e;

If(head ==0) head = addnode;

else tail→next= addnode;

tail = addnode;

addnode→next=0; }

Modify the insert( ) to insert node at the beginning of the list

```
void linked_list:: insert(int e,bool F)
{
   link added_node = new node;
   assert(added_node);
   added_node->value = e;


   if(head == 0)
   {
       head = tail = added_node;
       tail->next  = 0;
   }
   else if(!F)
   {
      tail->next = added_node; // instead of null
      tail = added_node; // tail shift
      tail->next = 0; // pointing to null
   }
   else
   {
      added_node->next = head;
      head = added_node;
   }
}
```

first function

```
bool  linked_list :: first (elemtype & e)
 { // after calling first, current points to first node (head)
   if( head == 0) return false;
      else { e = head → elem;  current = head;  return true;}}
```

flag = l.first(var)
1) flag check if the list is empty or not
2) curr is pointing to the head
3) var hold the value of the first node

next function

```
bool  linked_list :: next ( elemtype & e)
{ assert ( current); //for proper use, current should be nonzero
 // after each call, current always points to the item that next has just returned
If ( current → next ==0) return false;
else { current = current → next;  e = current → elem;
  return  true; } }
```

2 important checks

1- is linked list empty : head == 0
2- does i reach last node : curr -> next = 0

flag = l.next(var)
1) flag check if the next node is found or not
2)curr point to the next node
3) var hold the value of the next node

Ex_1: Write a main function that uses the class linked list, reads 10 elements, inserts them in the list and prints them

```
int main ( )
 { linked_list L ;
   elemtype x , y;  int n,m;
     cin>>n;
      for( int i=0; i<n ; i++)
    { cin>>x; L. insert (x); }
   cout <<" elements of the list "<<endl;
   bool notempty =L. first (y);
   while (notempty) { cout <<y<<"  "; notempty = L. next(y); } }
```

cant use currr head or tail here bec they are private members

Output if insert 5 elements

insert 5 elements in the list

10 20 30 40 50

elements of the list

10 20 30 40 50

Ex_2: Add a member function to the class linked list to print the elements of the list

```
void linked_list:: print( )
 { elemtype y;  bool notempty =first (y);
  while (notempty) { cout <<y<<"  "; notempty = next(y); } }
```

Another solution:

```
void linked_list:: print( )
{ current=head;
 cout<< current->elem<<endl;
 while( current->next!=0) {current=current->next;
  cout<< current->elem <<endl;}   }
```

```
void print()
  {
  for(curr = head ; curr!=0;curr=curr->next) cout<<curr->value<<endl;  /// iterate on all linked list
  }
```

Ex_3: Add search function as member function to the class linked list to search for certain element in the list and returns the number of its node

first occurance

```
int linked_list :: search (elemtype &e)
 {elemtype x; int  n=1;
        bool notempty = first(x);    cout<<x<<endl;
        while (notempty)  if (x == e)  return n;
        else  {n++; notempty = next(x); }
        return -1;}
int main ( )   { linked_list  L; elemtype x , y;  int n,m;
    cin>>n;       for( int i=0; i<n ; i++)     { cin>>x; L. insert (x); }
  cout <<" elements of the list "<<endl;
```

linked list print

```
  bool notempty =L. first (y);    while (notempty) { cout <<y<<"   "; notempty = L. next(y); } }
 cin>> y;       m= L. search( y);       if (m<0)  cout<<" element was not found"<<endl;
   else cout<<" element was found at node  "<<m<<endl<<endl;
```

Output of Ex_3 if elements of the list are: 10, 20 ,30 ,40 ,50, and search for certain elements

```
n      5
insert  elements in the list
10 20 30 40 50
elements of the list
10 20 30 40 50
enter element to search  10
element was found at node number  1
enter element to search  30
element was found at node number  3
enter element to search  50
element was found at node number  5
enter element to search  66
element was not found
```

Another solution for example_3

```
int linked_list :: search_2 (elemtype &e)
  {int n=1;
  if (head->elem ==e) return n;
      current= head;
     while (current->elem!= e&& current-> next!=0)
     {n++; current= current->next;}
     if (current->elem ==e) return n;
     return -1;}
```

Try to write search function as external function

Ex_4: Search for certain element and print the address of the found node , if not found , prints 0

```
int linked_list :: search_2 (elemtype &e)
  {  int n=1;
  if (head->elem ==e) {cout<<"  address of found node "<<head<<endl; return n;}
       current= head;
     while (current->elem!= e&& current-> next!=0)
     {n++; current= current->next;}
     if (current->elem ==e) {cout<<"  address of found node "<<current<<endl;return n;}
     cout<<"  not found, address is  0 "<<endl;return -1;}
```

print node adderss

In the main

```
for(int i=0; i<3;i++) {cin>>x;
    int   found=L.search_2(x);
     if (found>0) cout<<" element was found at node "<<found<<endl;
     else cout<<"  element was not found  "<<endl;}
```

Output of example_4

```
 insert 5 elements in the list
10 20 30 40 50
Enter the required element to search
40
  address of found node 0xf01830
 element was found at node 4
10
  address of found node 0xf017a0
 element was found at node 1
55
  not found, address is  0
  element was not found
```

Write search function as external function to search for certain element in the list (we must use first( ) and next( ) functions)

```
int search_ext (linked_list A,elemtype &e)
 { elemtype x; int k= -1;  int n=1;
   bool notempty = A.first(x);
        while (notempty) {
        if (x == e) {k=1; return n;}
        else  {n++; notempty = A.next(x); }}
        return k;}

In main( ) {cout<<"  enter element to search   "<<endl;
     int m;  cin>>x;        m=search_ext(L, x);
  if( m<0)cout<<" element was not found"<<endl;      else cout<<" element was found
            at position  "<<m<<endl<<endl;
```

Ex_5 :Add a member function to the class linked list to count the number of the nodes of the list

```
int linked_list::count_node( )
{if(head==0) return 0;
int n=1;
current= head;
while(current->next !=0)
{n++; current=current->next;}
return n;}
```

Solution of Ex_5

No. of elements 7

insert elements in the list

66 49 30 12 33 30 90

elements of the list

66 49 30 12 33 30 90

use count_node function to count the number of the nodes of the list

number of nodes of the list 7

Another solution of Example_5

```
int linked_list::count_node( )
        {elemtype x; int n=0;
         bool found=first(x);
         if(!found)return 0;
         while( found){ n++;
                  found= next(x);}
                  return n;}
```

Ex_6 : Write the count function as external function

```
int count_2(linked_list &A)
        {elemtype x; int n=0;
         bool found=A.first(x);
         if(!found)return 0;
         while( found){ n++;
                  found= A.next(x);}
                  return n;}
In main( )
{ linked_list  A; int  n=count_2(L );
```

Print( ) function as member function of the class linked list

```
void linked_list:: print( )
{ current=head;
  cout<< current->elem<<endl;
  while( current->next!=0) {current=current->next;
    cout<< current->elem <<endl;}   }

In main( ) {…….
cout<<" elements of the list through function print( ) "<<endl;
     L.print( );……..……}
```

Example_7: the node can contain many items as follows:

```
typedef  int  elemtype ;
class  linked_list  {private: struct node;
                              typedef  node * link;
struct  node { //now we define node
                    elemtype  elem1;   float elem2;  link  next;       };
  link  head, tail, current;
Public:   ……..
In main( )  { elemtype e; float y; for(   ) cin>>e<<y;
                  head->elem1=e;   head->elem2=y;
                    ……..
```

Function to search for certain element and delete its node

```
bool linked_list :: remove (elemtype &e)
  {   link p;
  cout<<"element to search  "<<e<<endl;
  if (head->elem ==e) {current= head->next;
    delete head; head=current;return true; }
  current=head;
  while (current->next->elem!= e&& current->next!=0)
       { current= current->next;
         if(current->next==0)break;
         }// it is used to not ask the loop condition of current->next->elem
            if(current->next== 0)return false;
           if (current->next->elem ==e)
   {     p=current->next;
     current->next= p-> next;        delete p;        return true;     }  else return false;}
```

Output

```
10   20     30     40     50
 Enter the required element to delete   40
element to search  40          found   1
 element was found and removed
 the new list after delete node             10   20     30     50
element to search  50              found   1
 element was found and removed
 the new list after delete node             10   20     30
element to search  55             found   0
  element was not found
 the new list after delete node             10   20     30
element to search  20          found   1
 element was found and removed
 the new list after delete node               10   30
```