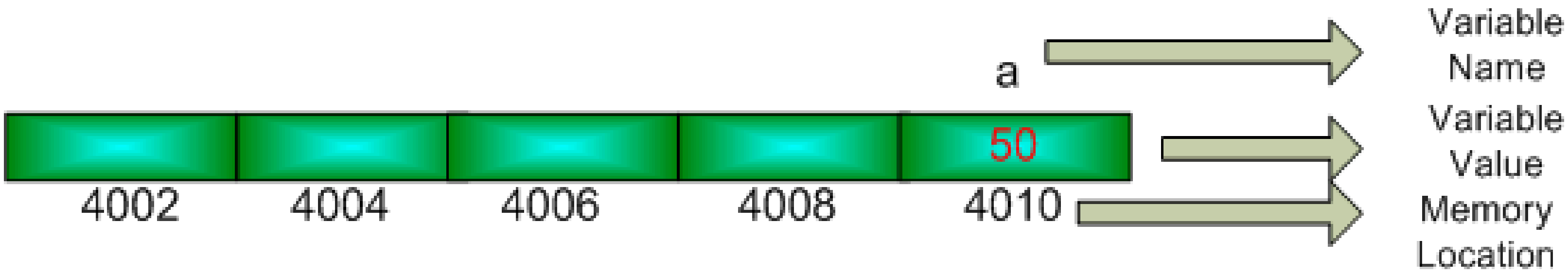
Lecture 3: Pointers, Arrays & Link Lists

Pointer

- Let us imagine that computer memory is a long array and every array location has a distinct memory location.
- int a = 50 // initialize variable a

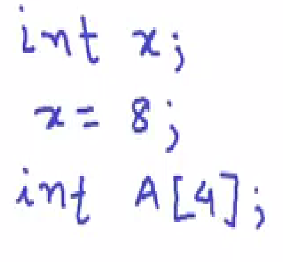


- It is like a house which has an address and this house has only one room. So the full address is-
- Name of the house: a
- Name of the person/value who live here is: 50
- House Number: 4010

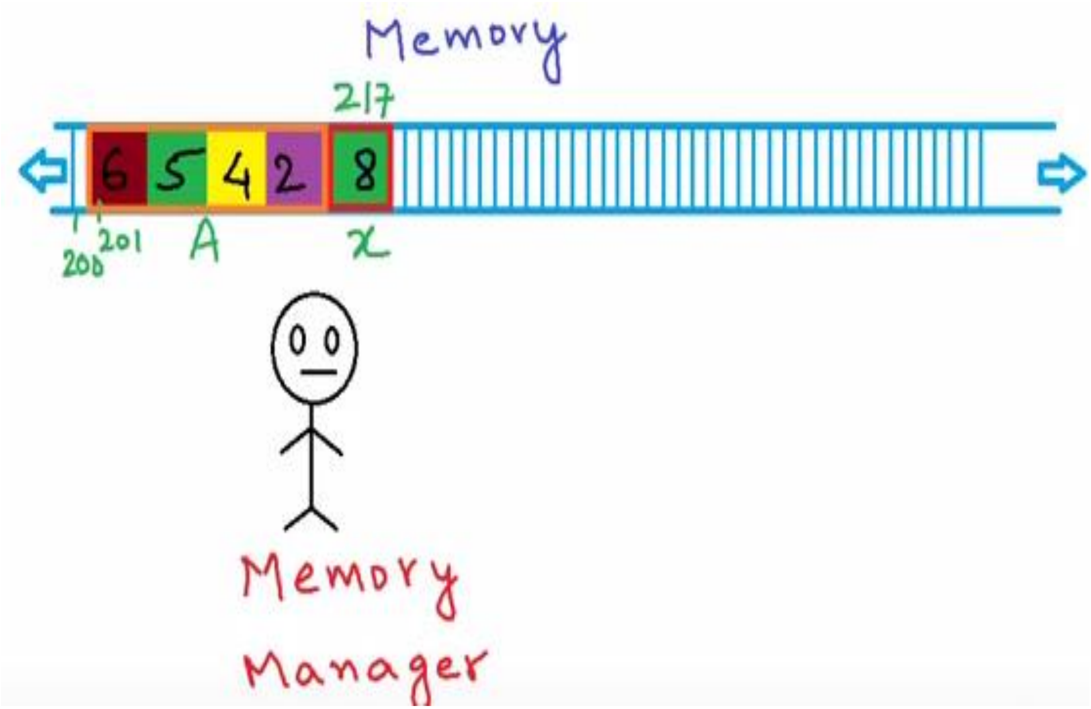
Reading Assignment

- Pointer vs Arrays
- Arrays of Pointer
- Pointer to Pointer
- Null pointer
- Void Pointer
- Invalid Pointer
- Dangling Pointer Reference Variable
- Dynamic Array (malloc, new, free, delete operators)
- Constructor in class
- Types of Constructor (Null, Default, Parametric, Overloading, Copy Constructor)

Arrays

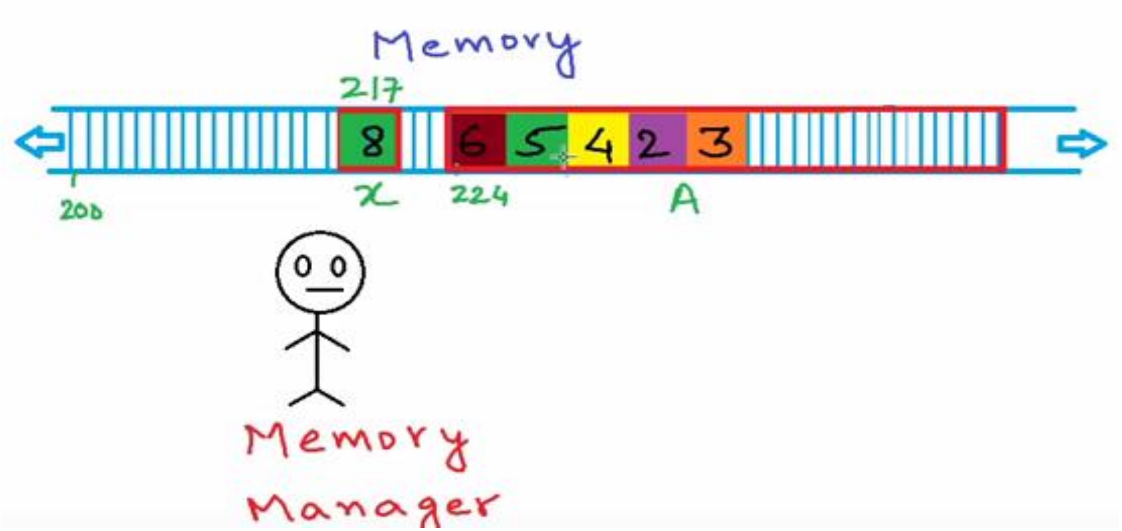


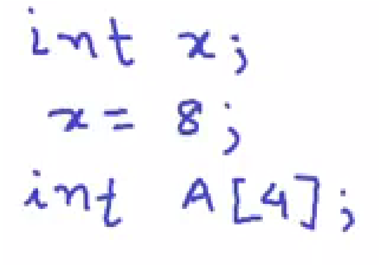
```
int x;
 x= 8;
int A[4];
```



Problems with arrays

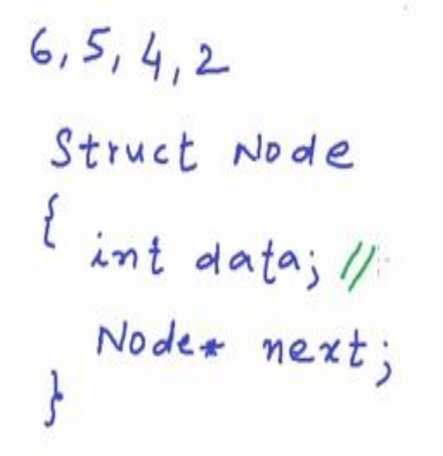
- Additional memory requirement results in copying data from old list to the new one.
- Increase in memory is double of the current memory size.
- Most of the memory locations remain unused. Low memory utilization.



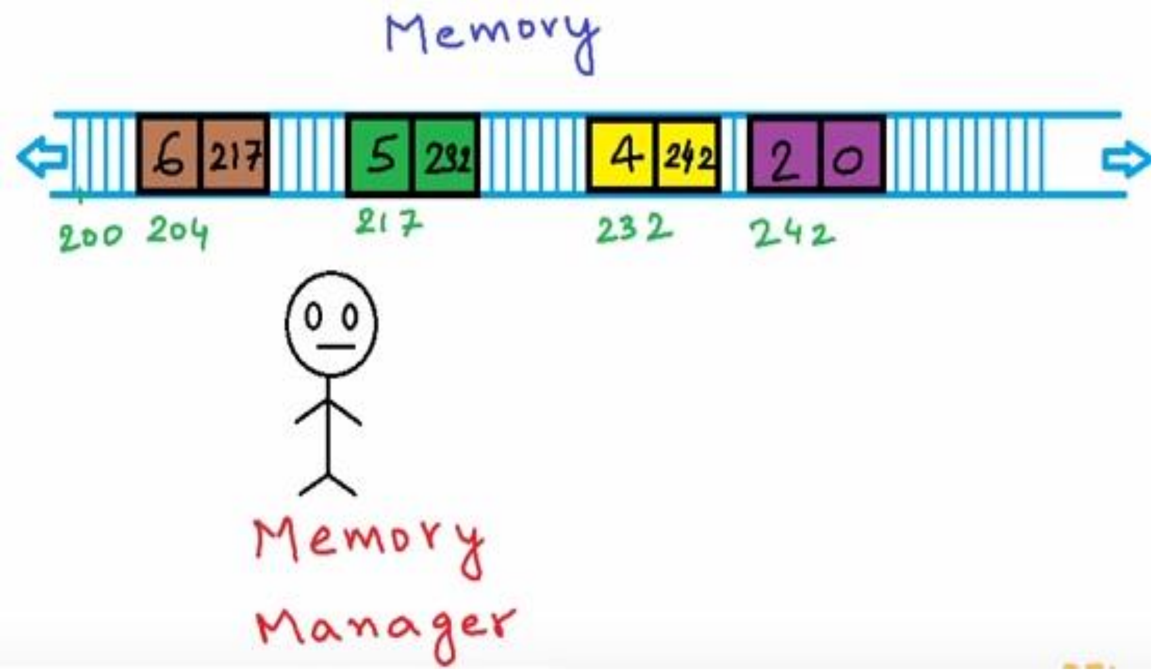


```
int x;
 x = 8;
int A[4];
```

Introduction to link list



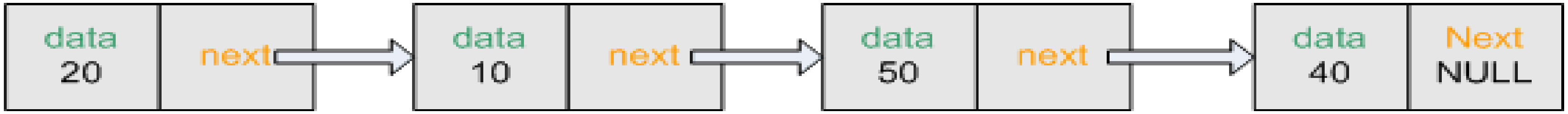
```
6,5,4,2

Struct Node
{
  int data; //
  Node* next;
}
```



Introduction to link list

- Linkedlist Node {
-     data   // The value or data stored in the node
-     next   // A reference to the next node, null for last node
- }

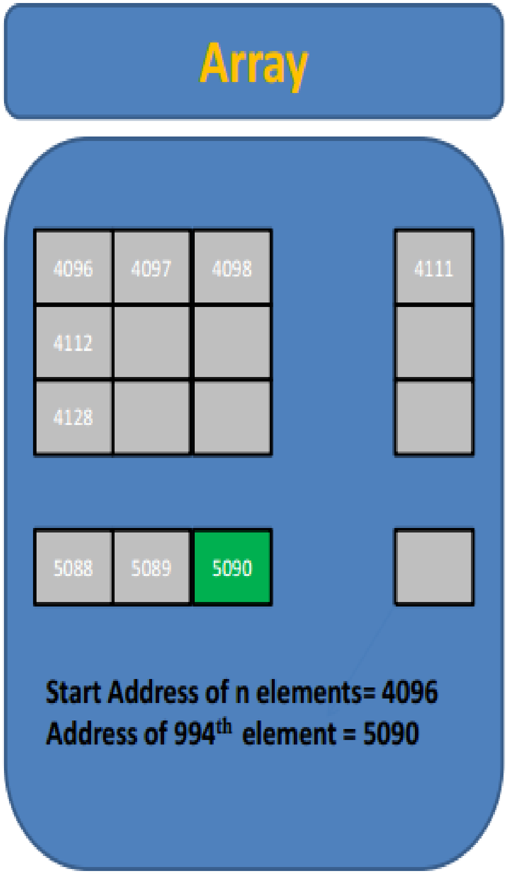
- typedef struct node
- {
-       int data;   // will store information
-       node *next;   // the reference to the next node
- };



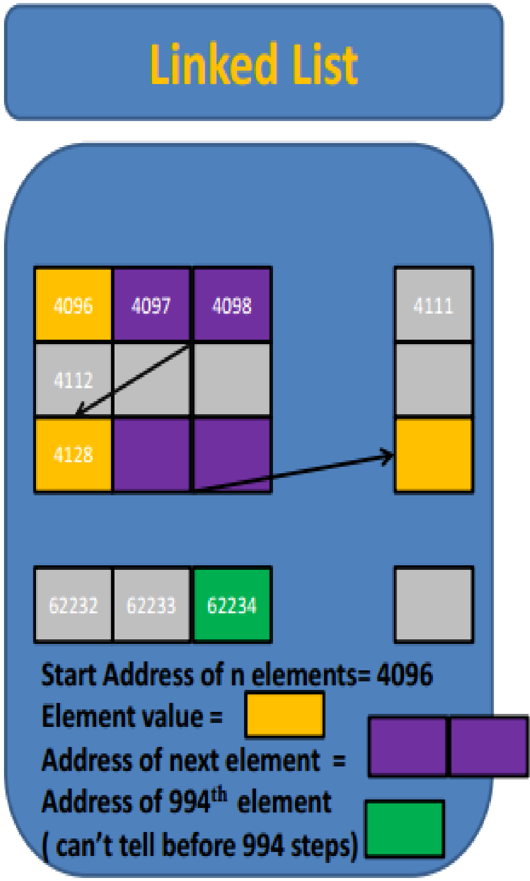
Linked list

Array vs Link list

- Let n = 10000
  r = 994
- Number of steps in accessing 994th element stored in contiguous memory (Array data structure) = 1 ~ O(1)
- Number of steps in accessing 994th element stored in non contiguous memory (Linked List data structure) = r ~ O(n) (r can be between 1-n)
- Don’t get a false impression that array is better data structure than linked list.

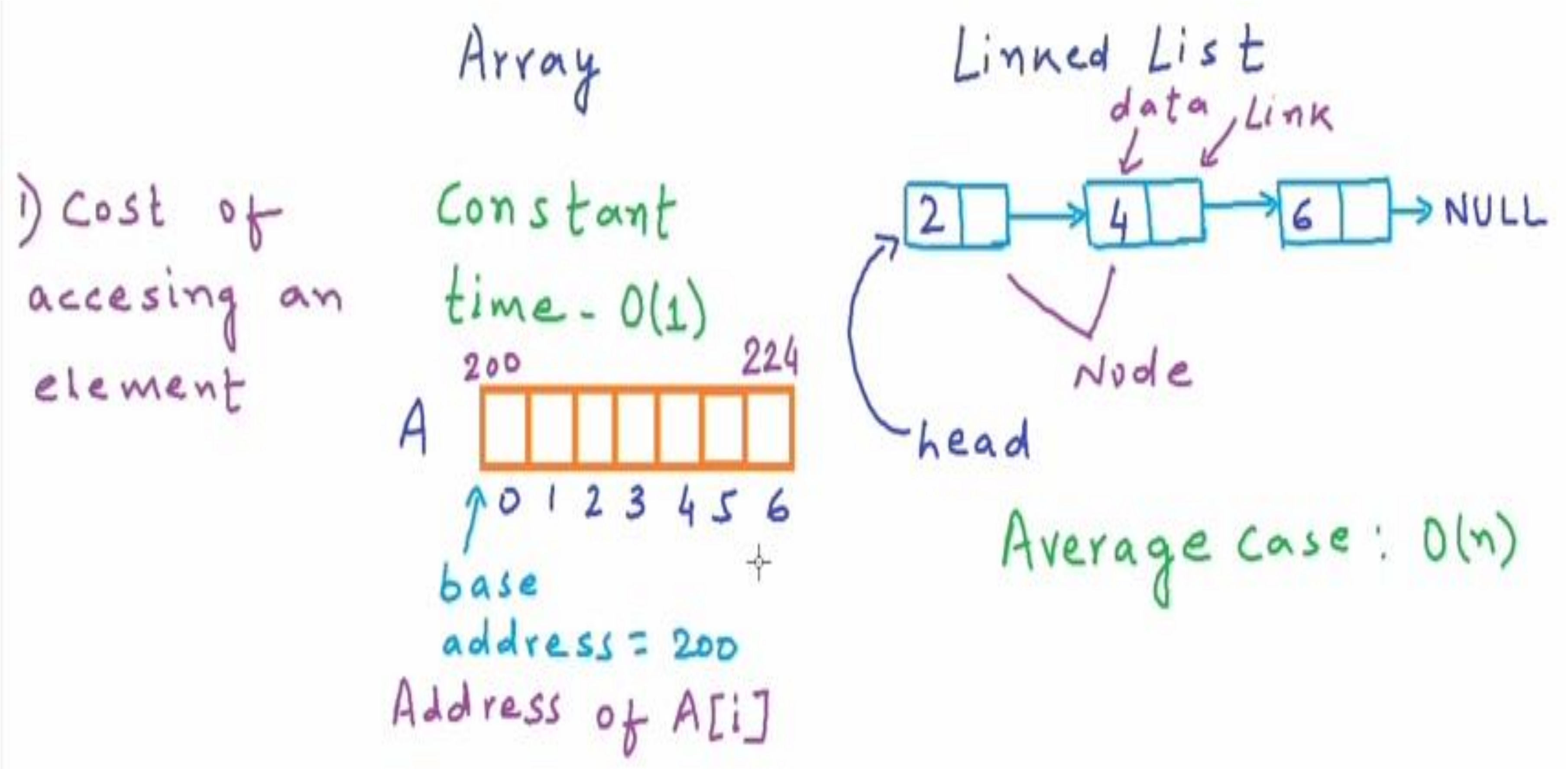


Start Address of n elements= 4096
Address of 994th element = 5090



Start Address of n elements= 4096
Element value = 🟨
Address of next element = 🟪🟪
Address of 994th element
(can’t tell before 994 steps) 🟩

Array vs Link list



Array

Linked List

1) Cost of accesing an element

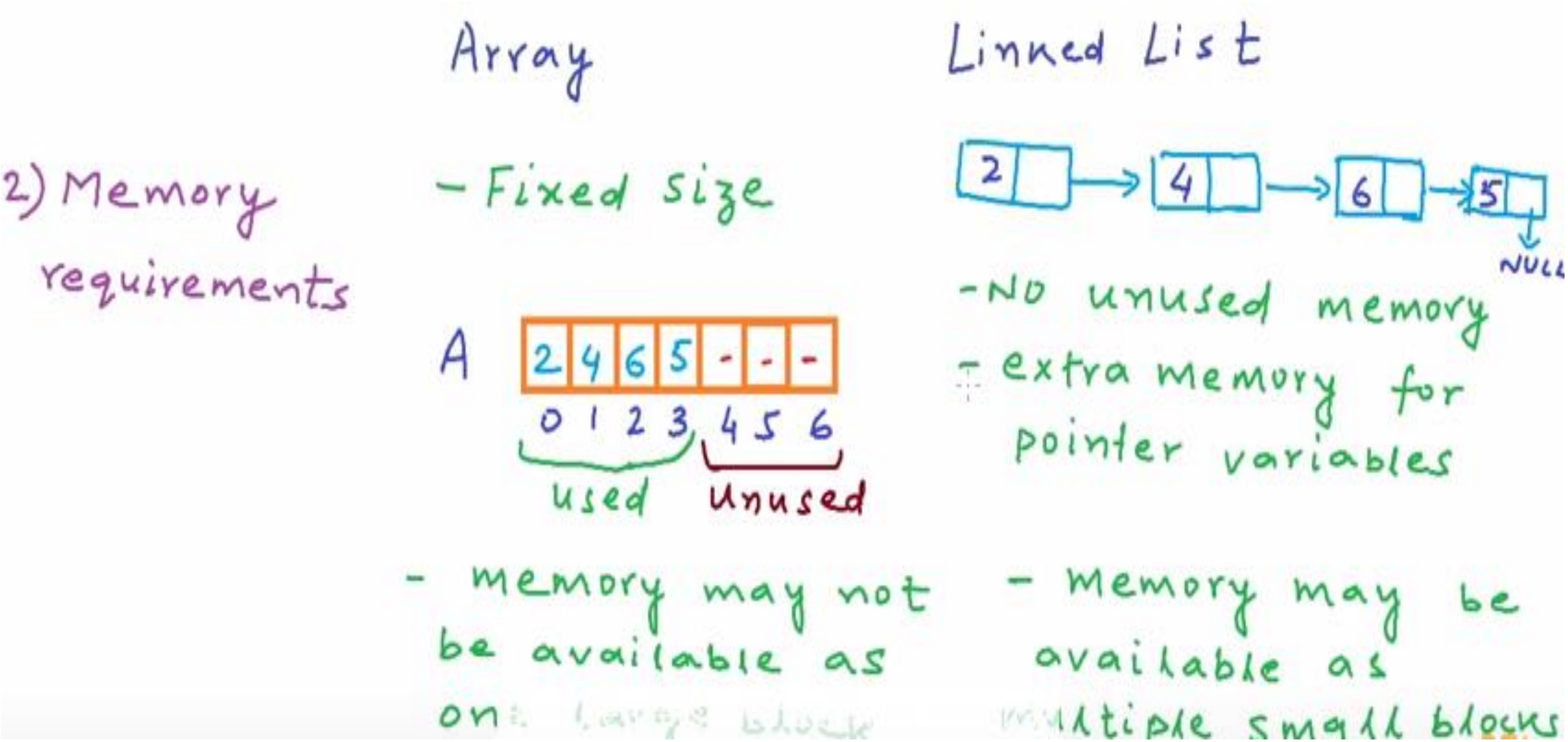
Constant time - O(1)

base address = 200

Address of A[i]

Average case : O(n)

Array vs Link list



Array

Linked List

2) Memory requirements

- Fixed size

A

2	4	6	5	-	-	-
0	1	2	3	4	5	6

used

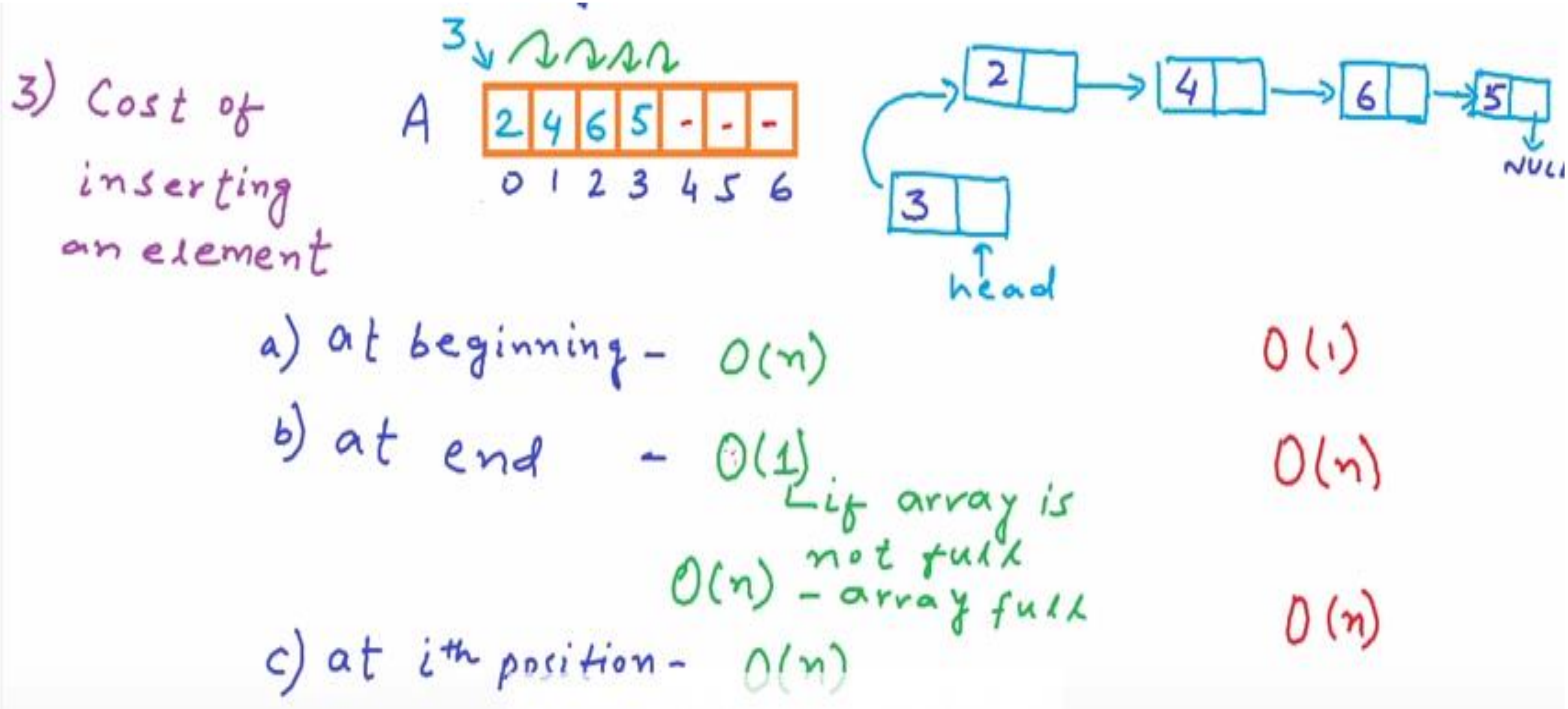
unused

- memory may not be available as one large block

2 → 4 → 6 → 5 → NULL

- No unused memory
- extra memory for pointer variables
- memory may be available as multiple small blocks

Array vs Link list



3) Cost of inserting an element

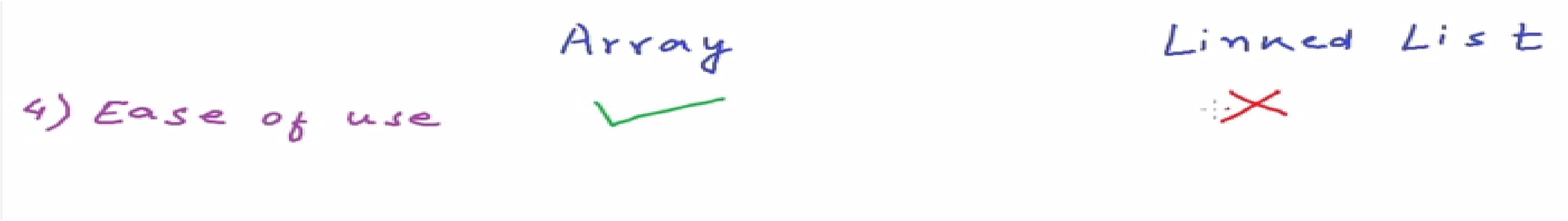
a) at beginning - O(n) O(1)

b) at end - O(1) ← if array is not full

O(n) - array full O(n)

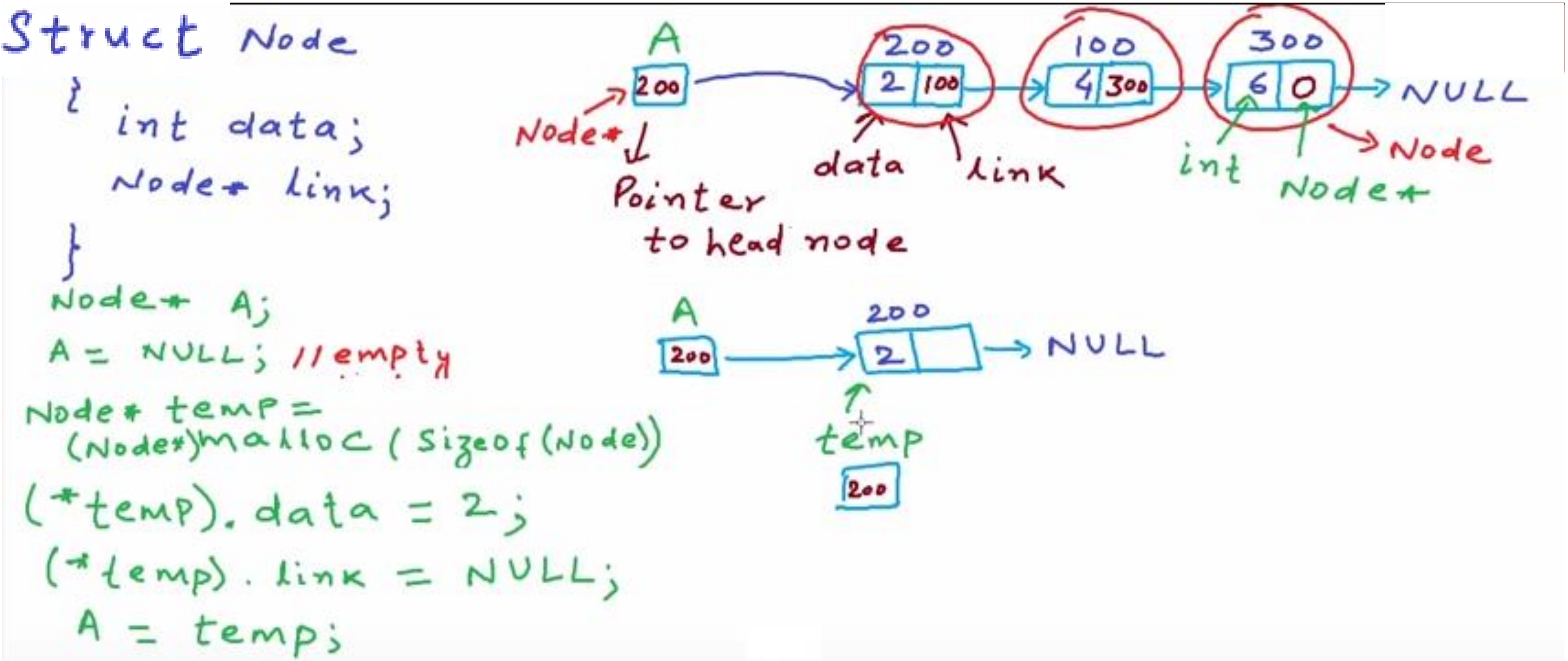
c) at i^th^ position - O(n) O(n)

Array vs Link list



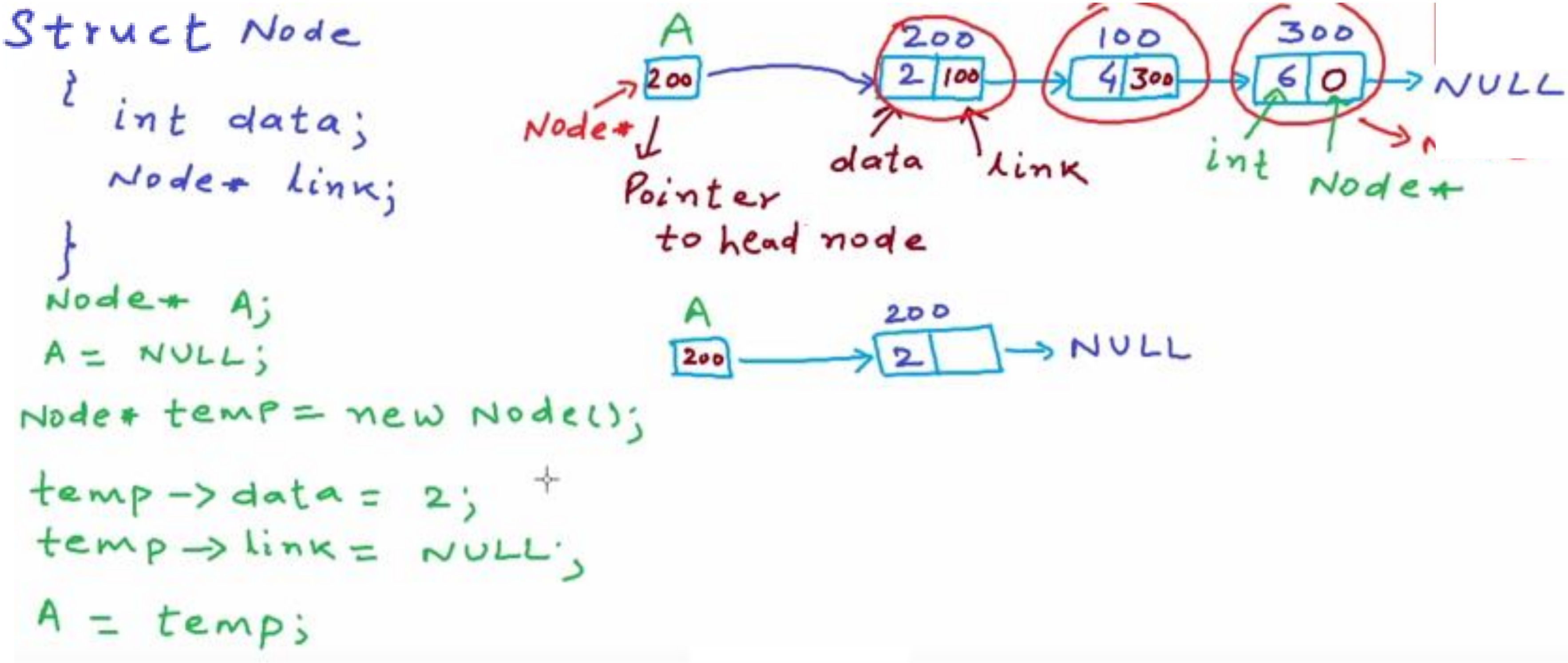
	Array	Linked List
4) Ease of use	✓	✗

Implementation of link list in C/C++ : One Method



```
Struct Node
  {
    int data;
    Node* link;
  }
  Node* A;
  A = NULL; //empty
Node* temp =
  (Node*)malloc ( Sizeof (Node))
(*temp). data = 2;
 (*temp) . link = NULL;
  A = temp;
```

Implementation of link list in C/C++ : Second Method



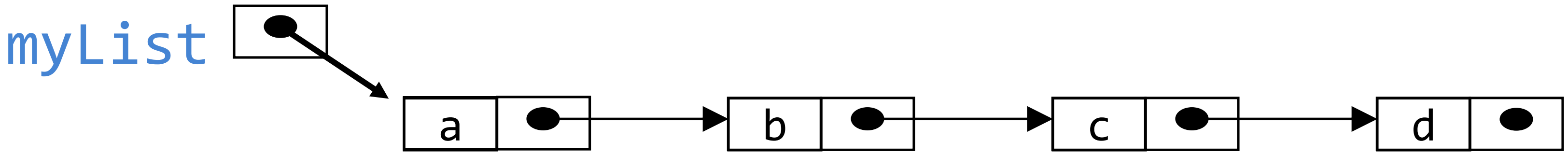
```
Struct Node
  {
    int data;
    Node* link;
  }
 Node* A;
 A = NULL;
Node* temp = new Node();

 temp -> data = 2;
 temp -> link = NULL;

 A = temp;
```

Types of link list

- Singly Link List
- Doubly Link List
- Circular Link List
- Circular Doubly Link List

Singly-linked lists

- Each node contains a value and a link to its successor (the last node has no successor)
- The header points to the first node in the list (or contains the null link if the list is empty)



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

Applications

- Linked Lists can be used to implement Stacks , Queues.
- Previous/next options used in photo viewer/media players
- Linked Lists can also be used to implement Graphs. (Adjacency list representation of Graph).
- Implementing Hash Tables :- Each Bucket of the hash table can itself be a linked list. (Open chain hashing).
- Undo functionality in Photoshop or Word . Linked list of states.
- A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term.
- However, for any polynomial operation , such as addition or multiplication of polynomials , linked list representation is more easier to deal with.
- Linked lists are useful for dynamic memory allocation.
- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running.
- All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.

Terminology

- Head (front, first node):
  - The node without predecessor, the node that starts the lists.

- Tail (end, last node):
  - The node that has no successor, the last node in the list.

- Current node: The node being processed.
  - From the current node we can access the next node.

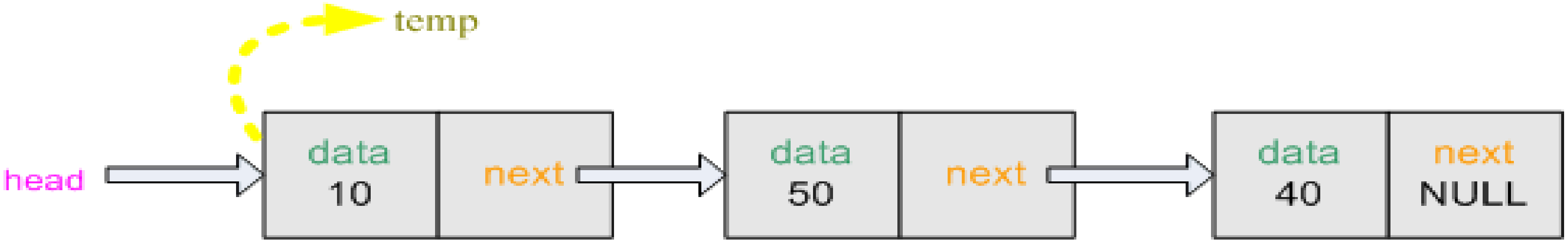
- Empty list: No nodes exist

Linked list operations

- Traverse
- Insert
- Delete
- Update

Traversing in link list

- while(temp!=NULL)
- {
- cout<< temp->data<<" ";   // show the data in the linked list
- temp = temp->next;   // transfer the address of 'temp1->next' to 'temp1'
- }



Linked list

Algorithm

- Suppose START is the address of the first node in the linked list. Following algorithm will visit all nodes from the START node to the end.

```
1. If (START is equal to NULL)
        (a) Display "The list is Empty"
        (b) Exit
2. Initialize TEMP = START
3. Repeat the step 4 and 5 until (TEMP → Next == NULL )
4. Display "TEMP → DATA"
5. TEMP = TEMP → Next
6. Exit
```

ALGORITHM FOR SEARCHING A NODE

- Suppose START is the address of the first node in the linked list and DATA is the information to be searched. If the DATA is found, POS will contain the corresponding position in the list.

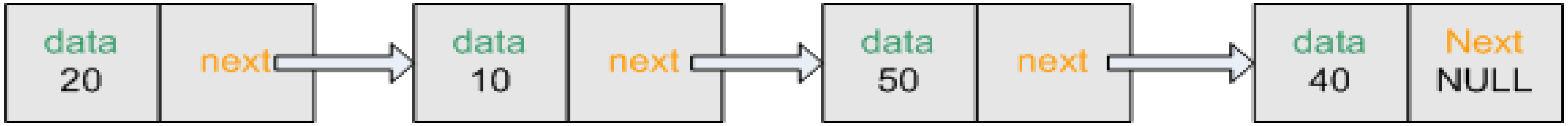
```
1. Input the DATA to be searched
2. Initialize TEMP = START; POS =1;
3. Repeat the step 4, 5 and 6 until (TEMP is equal to NULL)
4. If (TEMP → DATA is equal to DATA)
       (a) Display “The data is found at POS”
       (b) Exit
5. TEMP = TEMP → Next
6. POS = POS+1
7. If (TEMP is equal to NULL)
       (a) Display “The data is not found in the list”
8. Exit
```

Insertion in link list

01- Insert as the new first node

02- Insert as the new last node

03- Insert after specified number of nodes



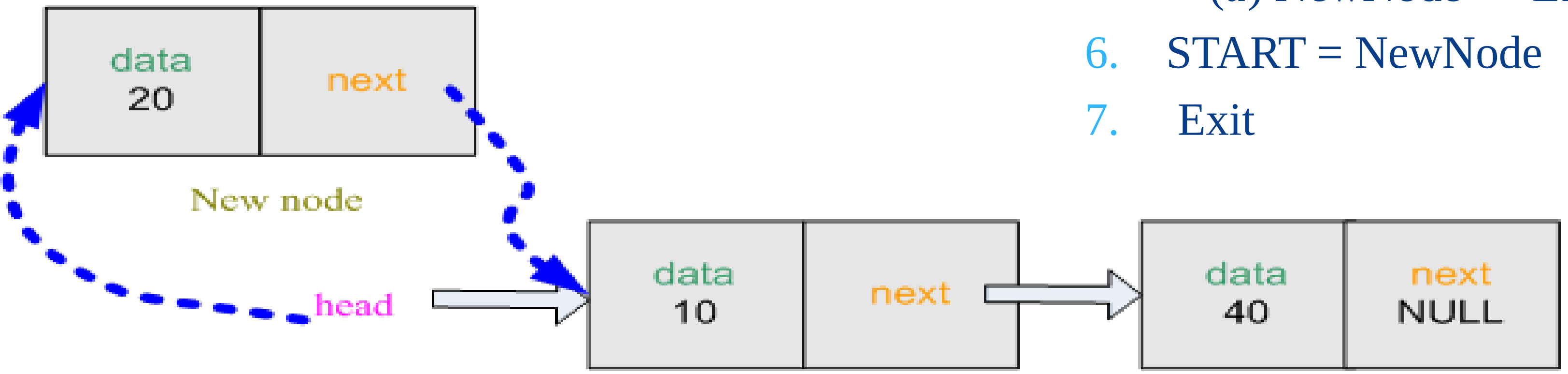
Linked list

01- Insert as the new first node

- Steps:
  - Create a Node
  - Set the node data Values
  - Connect the pointers

- **Algorithm**

```
1.  Input DATA to be inserted
2.  Create a NewNode
3.  NewNode → DATA = DATA
4.  If (SATRT equal to NULL)
        (a) NewNode → Link = NULL
5.  Else
        (a) NewNode → Link = START
6.  START = NewNode
7.  Exit
```

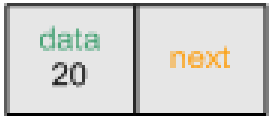


01- Insert as the new first node

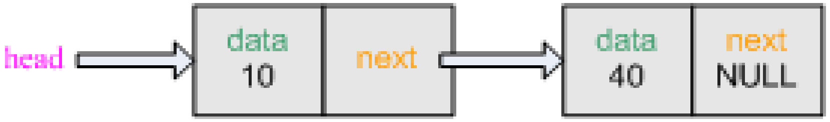
- 01- node *head = NULL; //empty linked list
- 02- node *temp; //create a temporary node
  - temp = (node*)malloc(sizeof(node)); //allocate space for node
- 03- temp->data = info; // store data (first field)
  - temp->next=head; // store the address of the pointer head (second field)
  - head = temp; // transfer the address of 'temp' to 'head'

02- Insert as the new last node

```
1.  Input DATA to be inserted
2.  Create a NewNode
3.  NewNode → DATA = DATA
4.  NewNode → Next = NULL
5.  If (SATRT equal to NULL)
6.          (a) START = NewNode
7.  Else
8.          (a) TEMP = START
9.          (b) While (TEMP → Next not equal to NULL)
10.                 (i) TEMP = TEMP → Next
11. TEMP → Next = NewNode
12. Exit
```



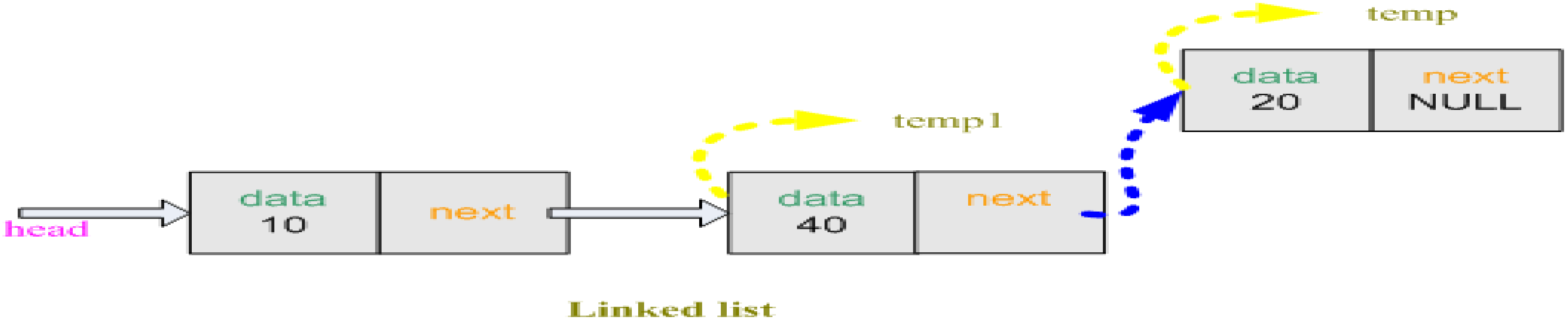
New node



Linked list

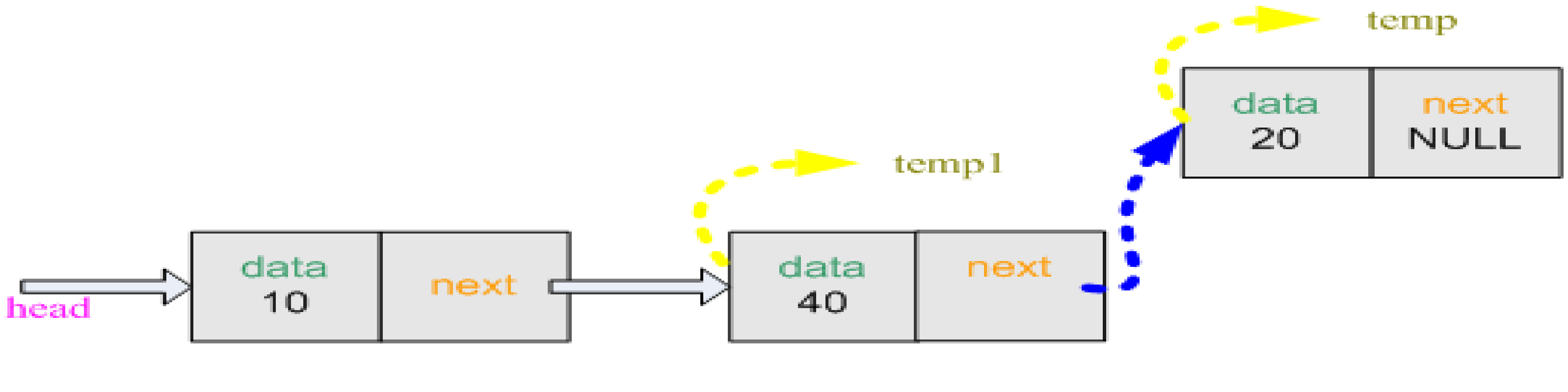
02- Insert as the new last node

- node *temp1; // create a temporary node
- temp1=(node*)malloc(sizeof(node)); // allocate space for node
- temp1 = head; // transfer the address of 'head' to 'temp1‘
- while(temp1->next!=NULL) // go to the last node
- temp1 = temp1->next; //transfer the address of 'temp1->next’ to 'temp1'



02- Insert as the new last node

- node *temp; // create a temporary node
- temp = (node*)malloc(sizeof(node)); // allocate space for node
- temp->data = info; // store data(first field)
- temp->next = NULL; // second field will be null(last node)
- temp1->next = temp; // 'temp' node will be the last node

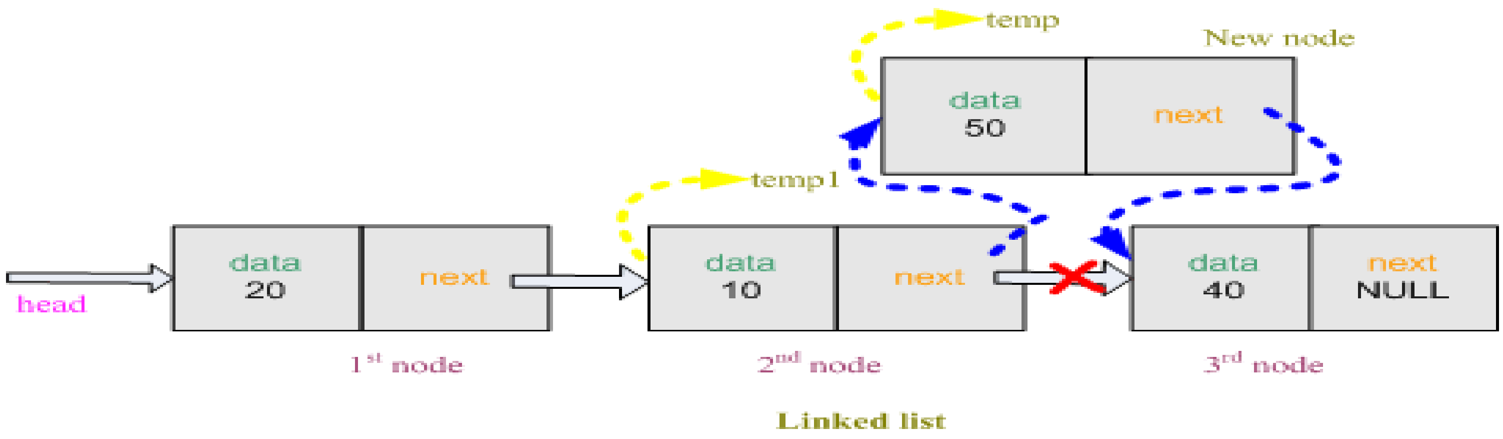


Linked list

03- Insert after specified number of nodes

```
cout<<"ENTER THE NODE NUMBER:";
cin>>node_number;                         // take the node number from user

node *temp1;                              // create a temporary node
temp1 = (node*)malloc(sizeof(node));      // allocate space for node
temp1 = head;

for( int i = 1 ; i < node_number ; i++ )
{
    temp1 = temp1->next;                              // go to the next node

    if( temp1 == NULL )
    {
        cout<<node_number<<" node does not exist"<< endl;
        break;
    }
}
```

```
node *temp;                      // create a temporary
                                   node

temp->data = info;               // store data(first field)
temp->next = temp1->next;        //transfer the address of temp1->next to
                                   temp->next

temp1->next = temp;              //transfer the address of temp to temp1->next
```



Algorithm

```
1. Input DATA and POS to be inserted
2. initialize TEMP = START; and j = 0
3. Repeat the step 3 while( k is less than POS)
       (a) TEMP = TEMP è Next
       (b) If (TEMP is equal to NULL)
               (i) Display “Node in the list less than the position”
               (ii) Exit
       (c) $k = k + 1$
4. Create a New Node
5. NewNode → DATA = DATA
6. NewNode → Next = TEMP → Next
7. TEMP → Next = NewNode
8. Exit
```

Deletion in link list

01- Delete from front

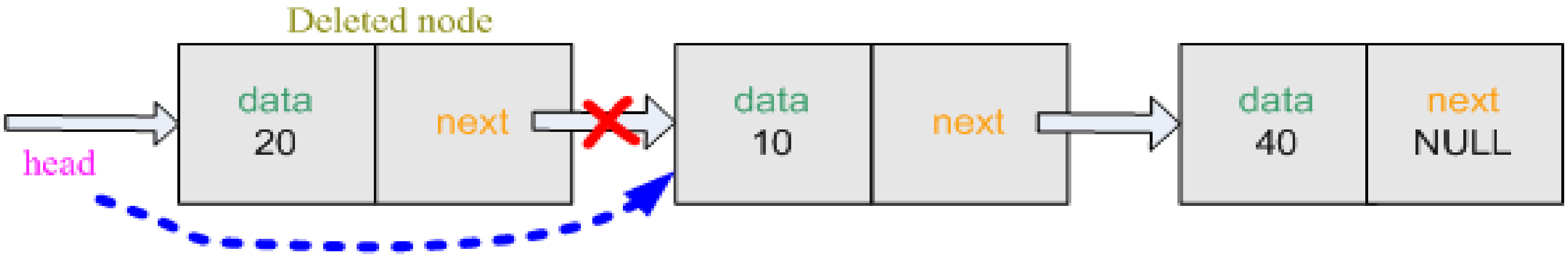
02- Delete from back

03-Delete after specified number of nodes

01- Delete from front

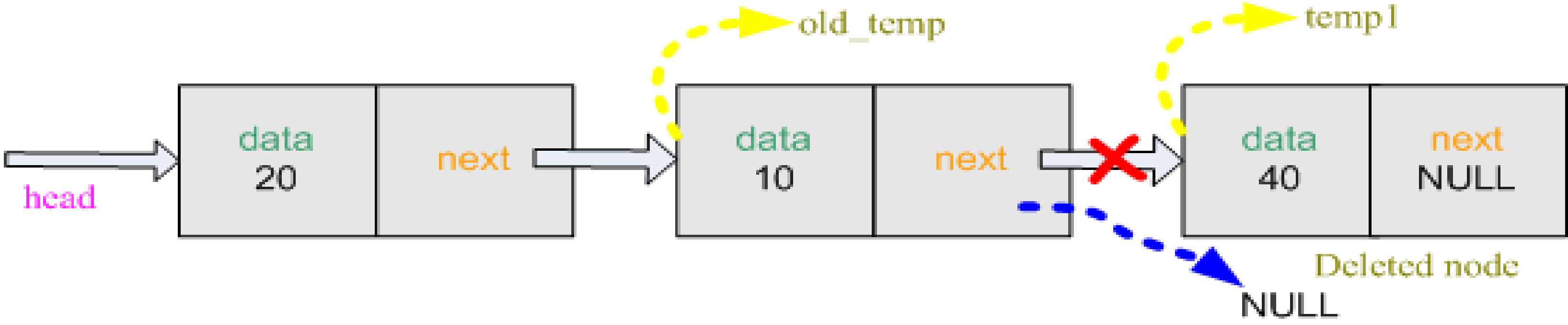
- **Steps**
  - Break the pointer connection
  - Re-connect the nodes
  - Delete the node

- temp = head; // transfer the address of 'head' to 'temp‘
- head = temp->next; // transfer the address of 'temp->next‘ to 'head‘
- free(temp);



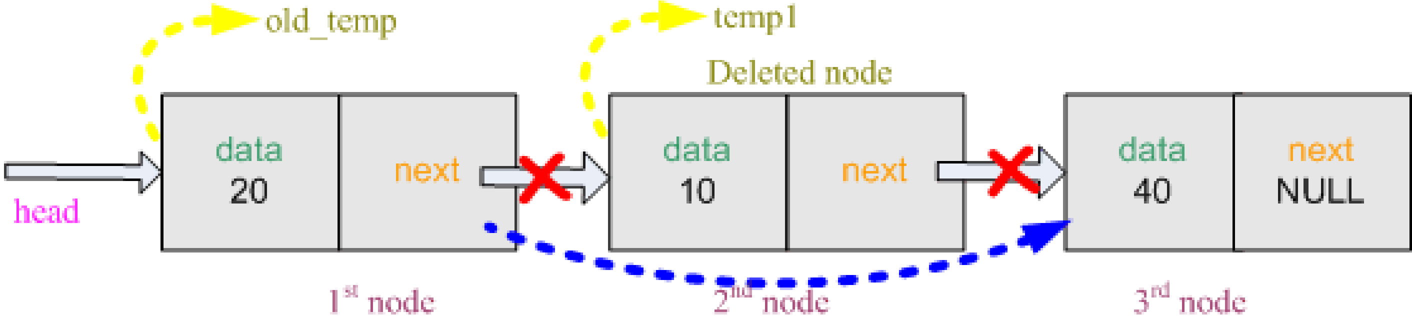
02- Delete from back

```
node *temp1;                          // create a temporary node

temp1 = head;                         //transfer the address of head to temp1

node *old_temp;                       // create a temporary node

while(temp1->next!=NULL)              // go to the last node
{
    old_temp = temp1;                 // transfer the address of 'temp1' to 'old_temp'
    temp1 = temp1->next;              // transfer the address of 'temp1->next' to 'temp1'
}
old_temp->next = NULL;                // previous node of the last node is null
free(temp1);
```



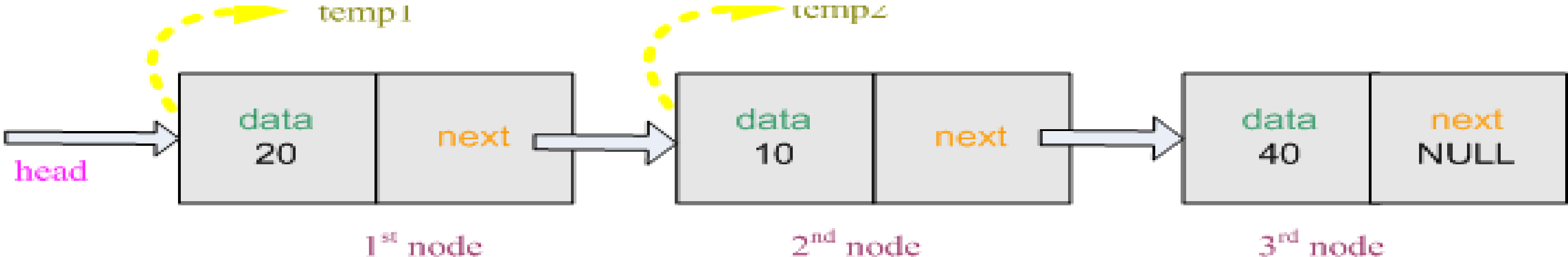
03-Delete specified number of node

```
node *temp1;                     // create a temporary node
temp1 = head;                    // transfer the address of 'head' to 'temp1'

node *old_temp;                  // create a temporary node
old_temp = temp1;                // transfer the address of 'temp1' to 'old_temp‘

cout<<"ENTER THE NODE NUMBER:";
cin>>node_number;                // take location
for( int i = 1 ; i < node_number ; i++ )
{
    old_temp = temp1;             // store previous node
    temp1 = temp1->next;         // store current node
}
old_temp->next = temp1->next;    // transfer the address of 'temp1->next' to 'old_temp->next‘
free(temp1);
```



04- Sort nodes

```
node *temp1;                  // create a temporary node

node *temp2;                  // create a temporary node

int temp = 0;                // store temporary data value

for( temp1 = head ; temp1!=NULL ; temp1 = temp1->next )
{
    for( temp2 = temp1->next ; temp2!=NULL ; temp2 = temp2->next )
    {
        if( temp1->data > temp2->data )
        {
            temp = temp1->data;
            temp1->data = temp2->data;
            temp2->data = temp;
        }
    }
}
```



Applications of Link List

Problem :

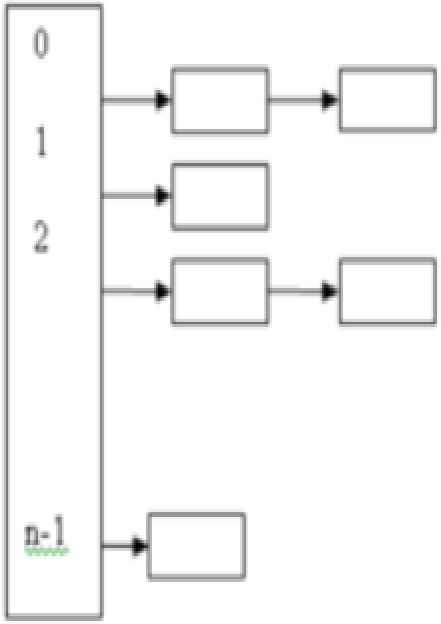
Suppose you need to program an application that has a pre-defined number of categories, but the exact items in each category is unknown.

Solution:

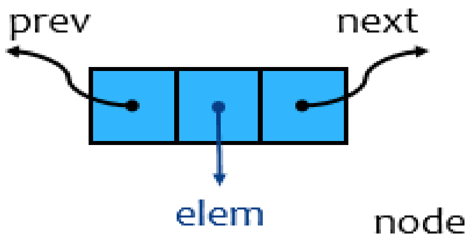
Pre-defined number of categories implies that we can use a simple static structure like array to represent the categories. Since we do not know the number of items in each category, we can represent items in each category using a linked list. So what we need is an array of linked lists .

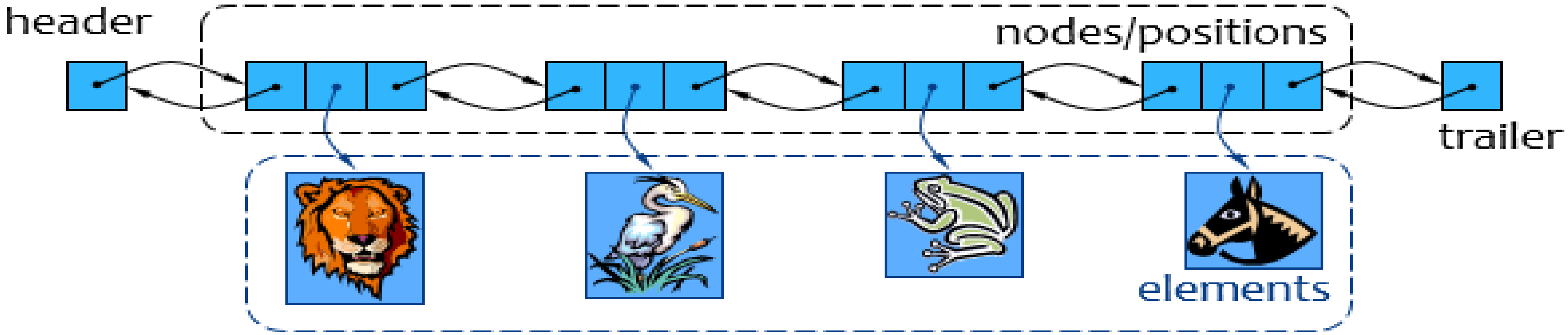
More Examples :

You can also think of representing a web index using an array of linked lists, where array contains the keywords and linked lists contains the web URL's where that keyword occurs.



Doubly-linked lists





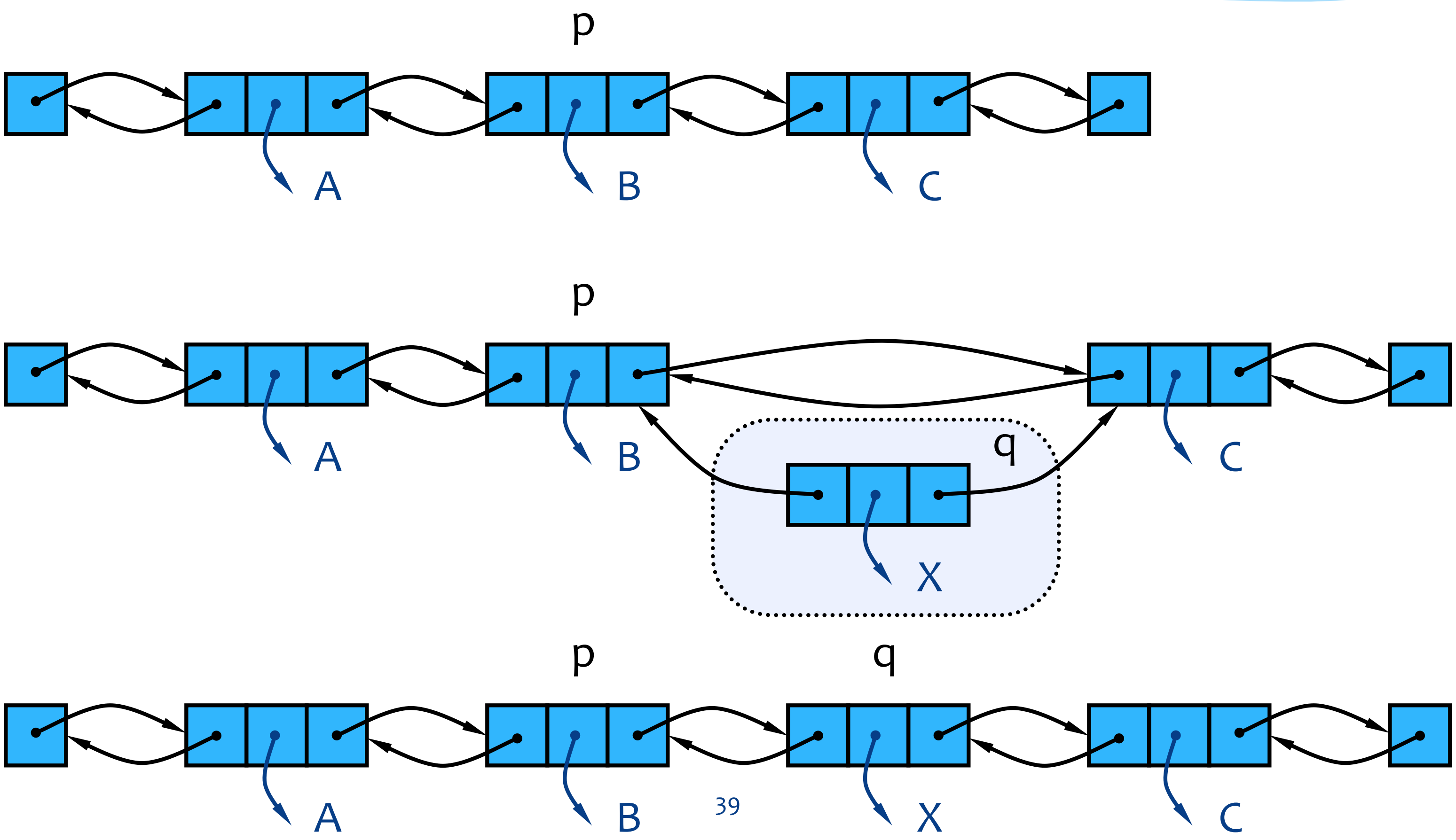
- Each node contains a value, a link to its successor (if any), *and* a link to its predecessor (if any)
- The header points to the first node in the list *and* to the last node in the list (or contains null links if the list is empty)

Doubly Linked Lists

- **Applications:**
  - Applications that have a Most Recently Used (MRU) list (a linked list of file names)
  - A stack, hash table, and binary tree can be implemented using a doubly linked list
  - Previous/next options used in photo viewer/media players
  - It is also used to represent various states of a game

- **Advantages:**
  - Convenient to traverse the list backwards.
  - Simplifies insertion and deletion because you no longer have to refer to the previous node.
- **Disadvantage:**
  - Increase in space requirements.

Insertion



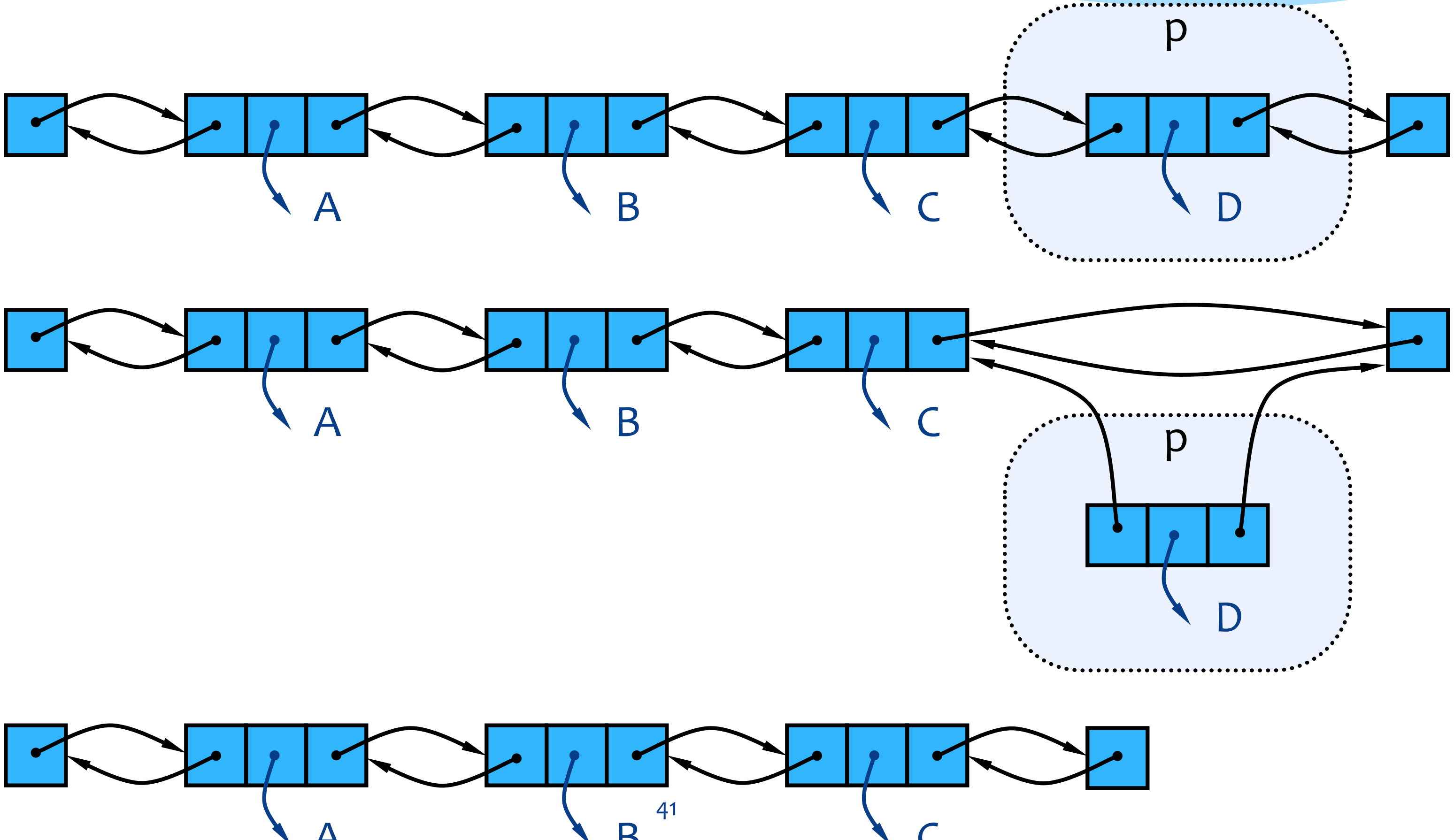
Algorithm

- Suppose START is the first position in linked list. Let DATA be the element to be inserted in the new node. POS is the position where the NewNode is to be inserted. TEMP is a temporary pointer to hold the node address.

```
1.  Input the DATA and POS
2.  Initialize TEMP = START; i = 0
3.  Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
4.  TEMP = TEMP → RPoint; i = i +1
5.  If (TEMP not equal to NULL) and (i equal to POS)
        (a) Create a New Node
        (b) NewNode → DATA = DATA
        (c) NewNode → RPoint = TEMP → RPoint
        (d) NewNode → LPoint = TEMP
        (e) (TEMP → RPoint) → LPoint = NewNode
        (f ) TEMP → RPoint = New Node
6.  Else
        (a) Display “Position NOT found”
7.  Exit
```

Deletion

- remove(p), where p == last()



Algorithm

- Suppose START is the address of the first node in the linked list. Let POS is the position of the node to be deleted. TEMP is the temporary pointer to hold the address of the node. After deletion, DATA will contain the information on the deleted node.

```
1.  Input the POS
2.  Initialize TEMP = START; i = 0
3.  Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
4.  TEMP = TEMP → RPoint; i = i +1
5.  If (TEMP not equal to NULL) and (i equal to POS)
        (a) Create a New Node
        (b) NewNode → DATA = DATA
        (c) NewNode → RPoint = TEMP → RPoint
        (d) NewNode → LPoint = TEMP
        (e) (TEMP → RPoint) → LPoint = NewNode
        (f ) TEMP → RPoint = New Node
6.  Else
        (a) Display "Position NOT found"
7.  Exit
```

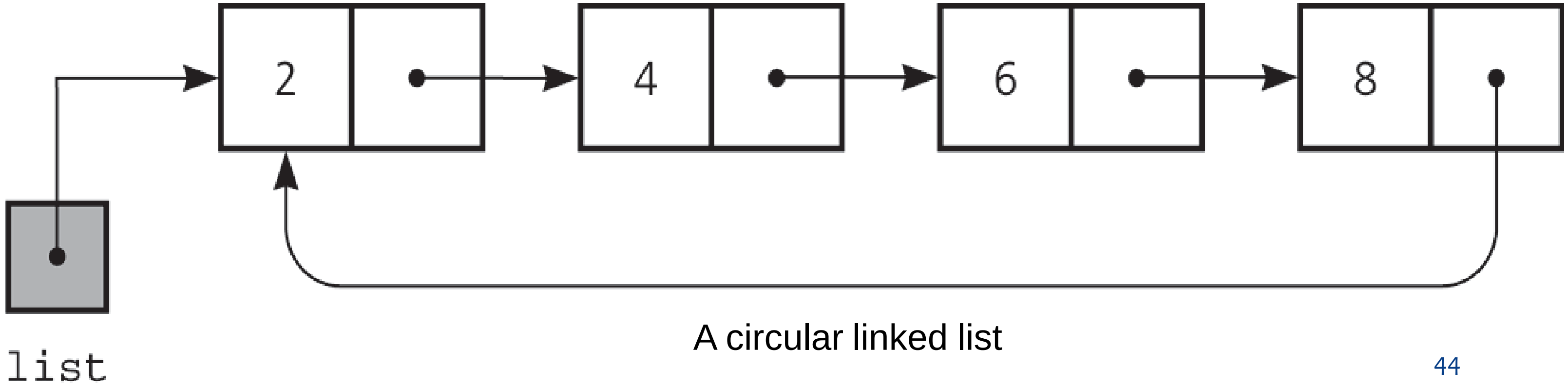
DLLs compared to SLLs

- Advantages:
  - Can be traversed in either direction (may be essential for some programs)
  - Some operations, such as deletion and inserting before a node, become easier

- Disadvantages:
  - Requires more space
  - List manipulations are slower (because more links must be changed)
  - Greater chance of having bugs (because more links must be manipulated)

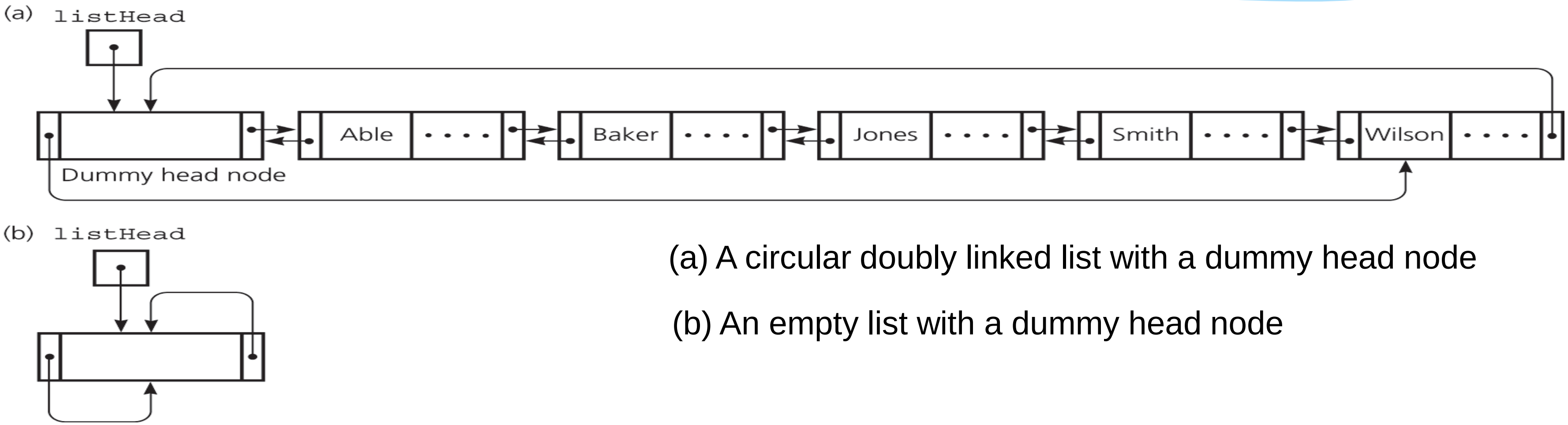
Circular Linked Lists

- Last node references the first node
- Every node has a successor
- No node in a circular linked list contains *NULL*



A circular linked list

Circular Doubly Linked Lists



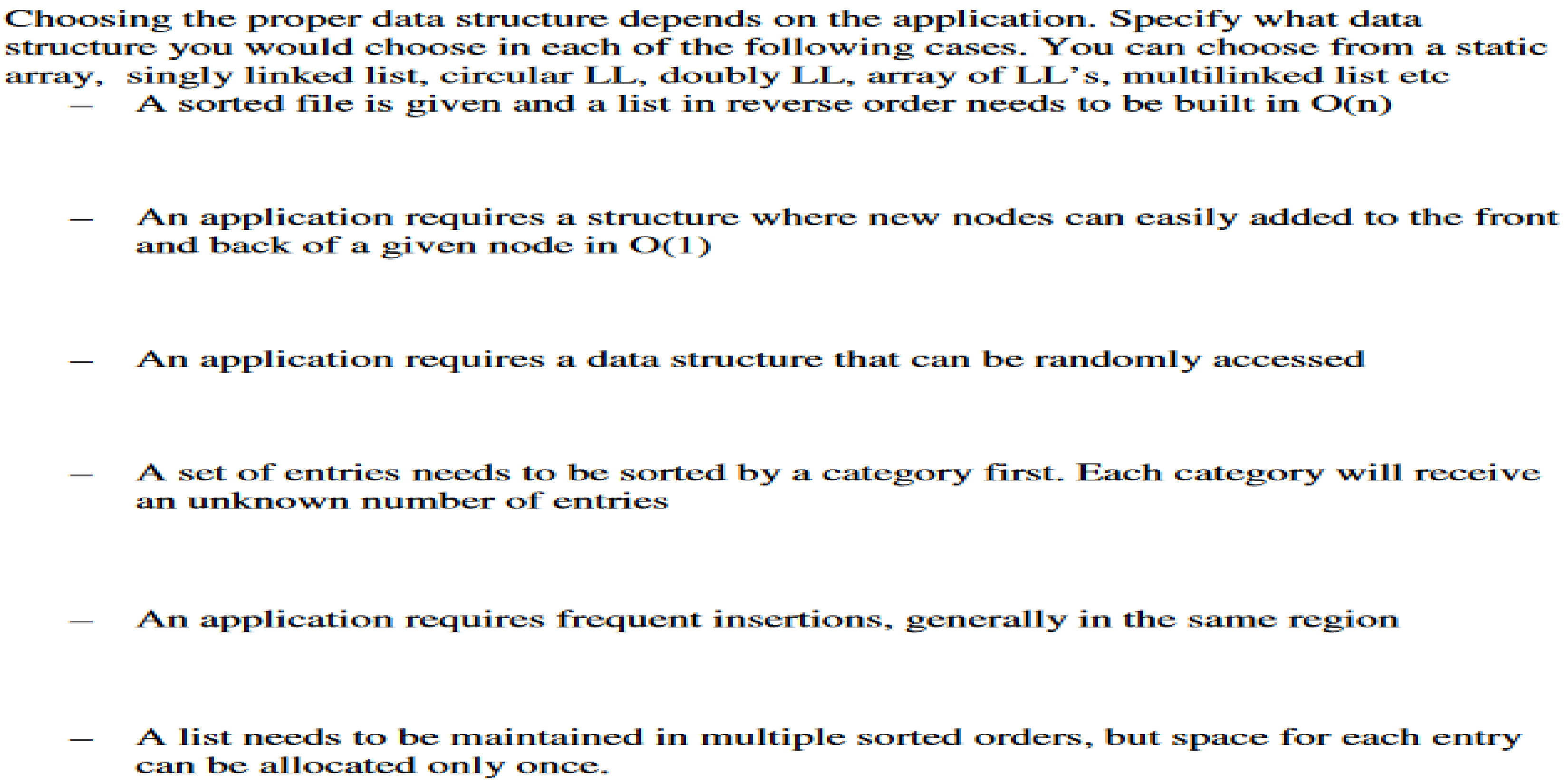
(a) A circular doubly linked list with a dummy head node

(b) An empty list with a dummy head node

Applications:

- Timesharing problem solved by the operating system
- For this application, there should be no NULL pointers unless there is absolutely no process requesting CPU time.

Questions



Choosing the proper data structure depends on the application. Specify what data structure you would choose in each of the following cases. You can choose from a static array, singly linked list, circular LL, doubly LL, array of LL's, multilinked list etc

- A sorted file is given and a list in reverse order needs to be built in O(n)

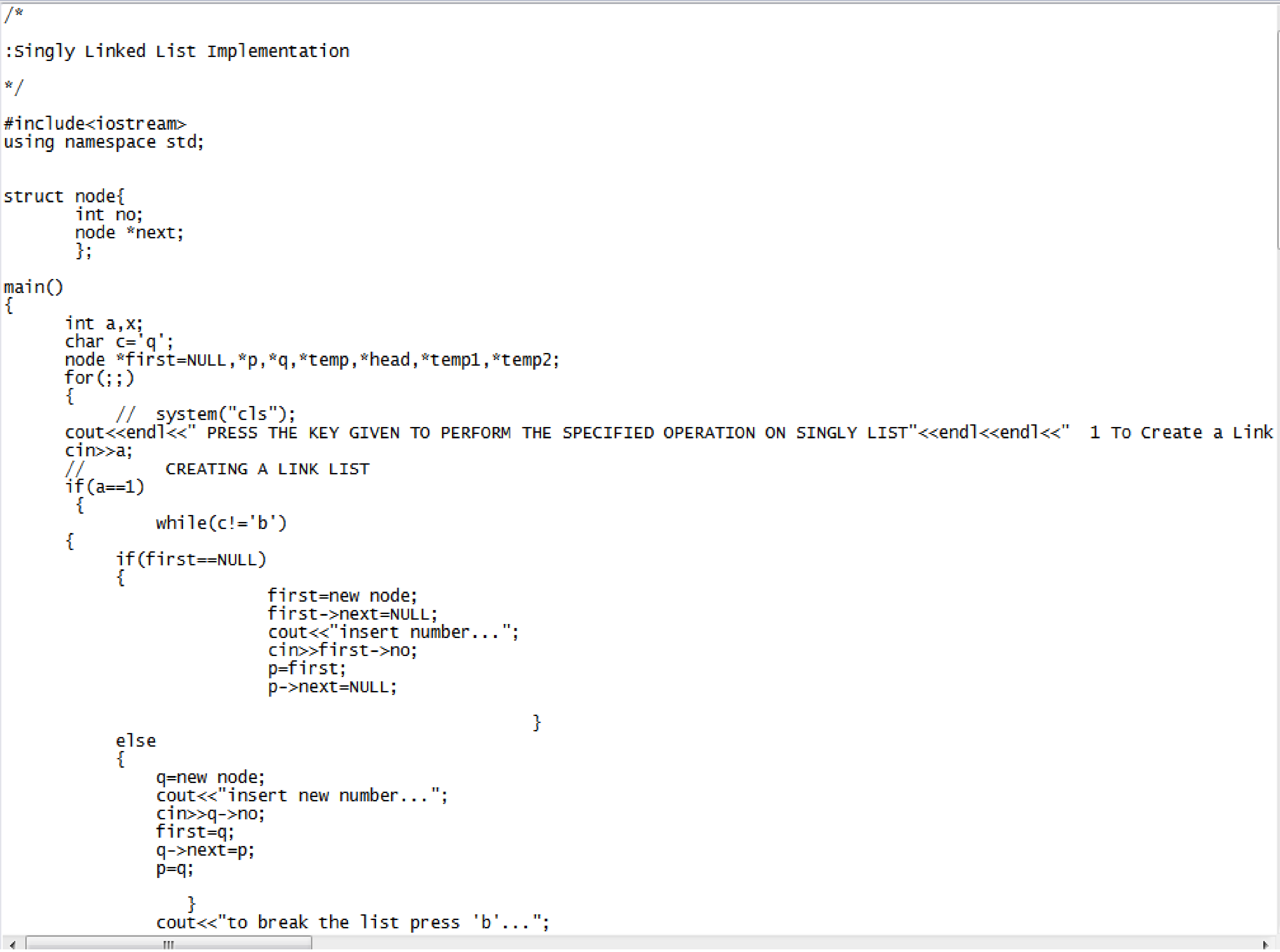
- An application requires a structure where new nodes can easily added to the front and back of a given node in O(1)

- An application requires a data structure that can be randomly accessed

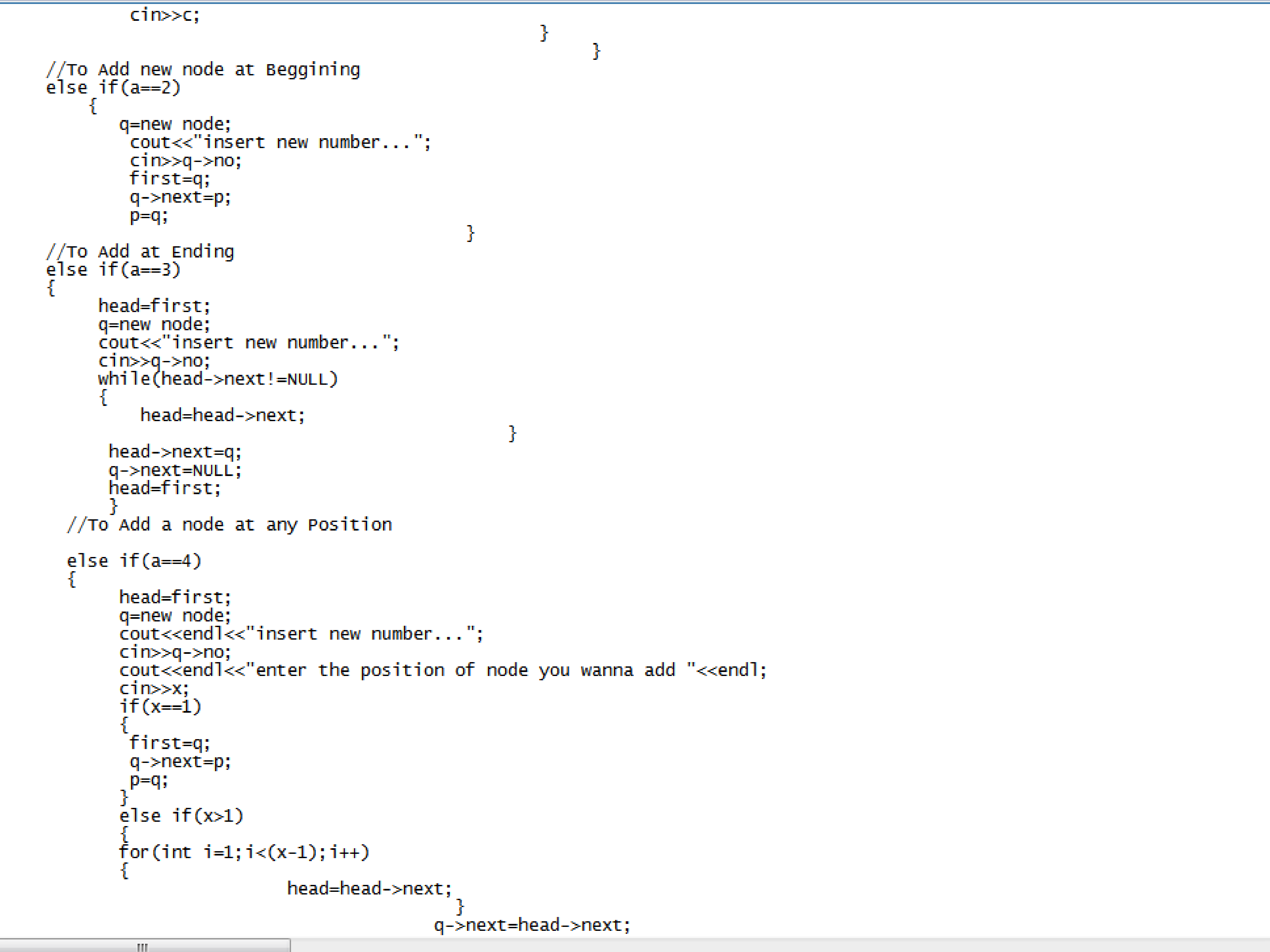
- A set of entries needs to be sorted by a category first. Each category will receive an unknown number of entries

- An application requires frequent insertions, generally in the same region

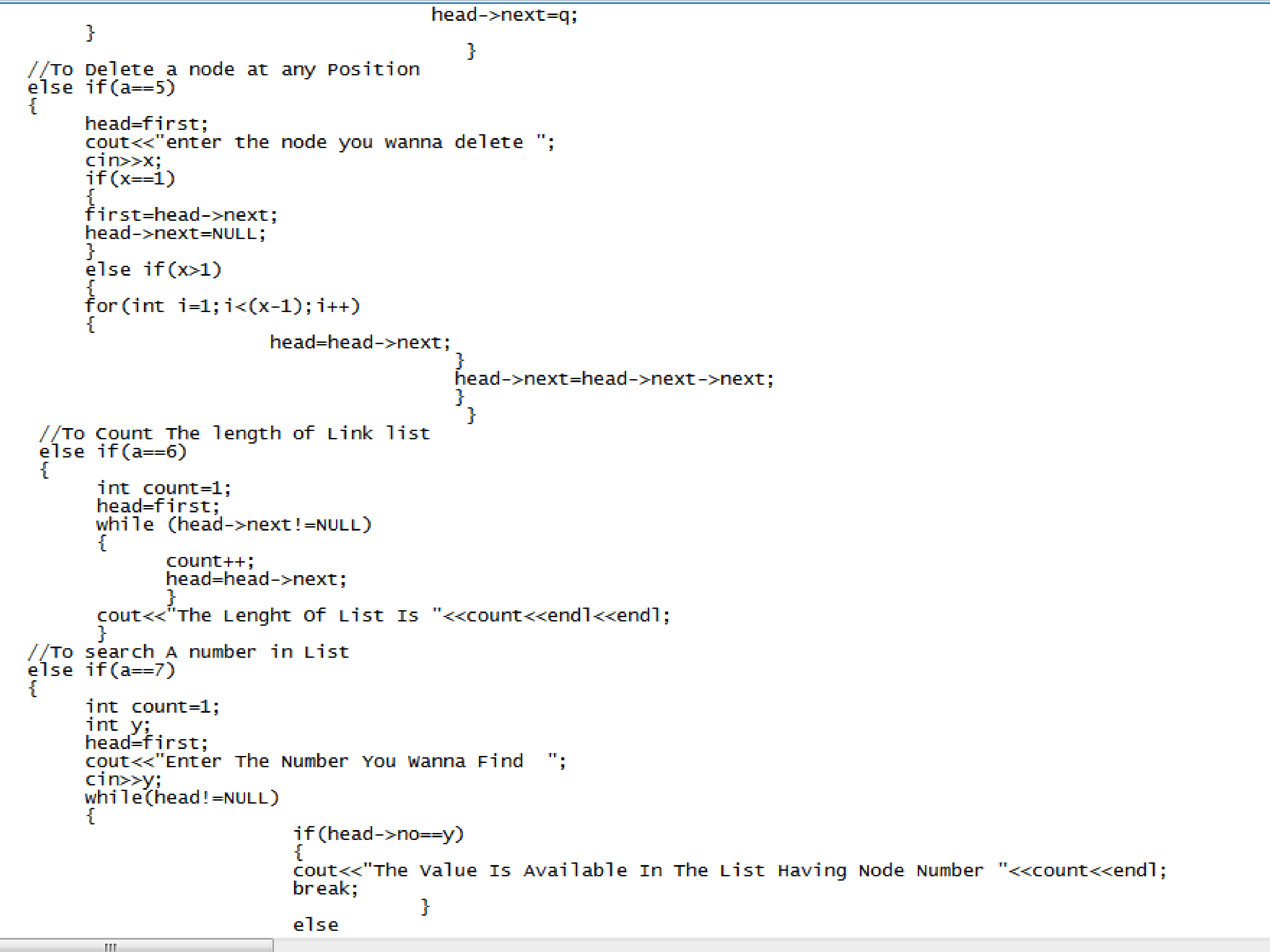
- A list needs to be maintained in multiple sorted orders, but space for each entry can be allocated only once.



```
/*

:Singly Linked List Implementation

*/

#include<iostream>
using namespace std;


struct node{
       int no;
       node *next;
       };

main()
{
      int a,x;
      char c='q';
      node *first=NULL,*p,*q,*temp,*head,*temp1,*temp2;
      for(;;)
      {
           //  system("cls");
      cout<<endl<<" PRESS THE KEY GIVEN TO PERFORM THE SPECIFIED OPERATION ON SINGLY LIST"<<endl<<endl<<"  1 To Create a Link
      cin>>a;
      //         CREATING A LINK LIST
      if(a==1)
       {
               while(c!='b')
      {
           if(first==NULL)
           {
                          first=new node;
                          first->next=NULL;
                          cout<<"insert number...";
                          cin>>first->no;
                          p=first;
                          p->next=NULL;

                                                    }
           else
           {
               q=new node;
               cout<<"insert new number...";
               cin>>q->no;
               first=q;
               q->next=p;
               p=q;

                  }
               cout<<"to break the list press 'b'...";
```



```
            cin>>c;
                                                   }
                                                        }

    //To Add new node at Beggining
    else if(a==2)
        {
           q=new node;
            cout<<"insert new number...";
            cin>>q->no;
            first=q;
            q->next=p;
            p=q;
                                            }
    //To Add at Ending
    else if(a==3)
    {
         head=first;
         q=new node;
         cout<<"insert new number...";
         cin>>q->no;
         while(head->next!=NULL)
         {
             head=head->next;
                                                }

          head->next=q;
          q->next=NULL;
          head=first;
          }
      //To Add a node at any Position

      else if(a==4)
      {
           head=first;
           q=new node;
           cout<<endl<<"insert new number...";
           cin>>q->no;
           cout<<endl<<"enter the position of node you wanna add "<<endl;
           cin>>x;
           if(x==1)
           {
            first=q;
            q->next=p;
            p=q;
           }
           else if(x>1)
           {
           for(int i=1;i<(x-1);i++)
           {
                           head=head->next;
                                           }
                                         q->next=head->next;
```



```
                                   head->next=q;
     }
                                      }
//To Delete a node at any Position
else if(a==5)
{
     head=first;
     cout<<"enter the node you wanna delete ";
     cin>>x;
     if(x==1)
     {
     first=head->next;
     head->next=NULL;
     }
     else if(x>1)
     {
     for(int i=1;i<(x-1);i++)
     {
                     head=head->next;
                                     }
                                     head->next=head->next->next;
                                     }
                                      }
 //To Count The length of Link list
 else if(a==6)
 {
      int count=1;
      head=first;
      while (head->next!=NULL)
      {
            count++;
            head=head->next;
            }
      cout<<"The Lenght Of List Is "<<count<<endl<<endl;
      }
//To search A number in List
else if(a==7)
{
     int count=1;
     int y;
     head=first;
     cout<<"Enter The Number You wanna Find  ";
     cin>>y;
     while(head!=NULL)
     {
                       if(head->no==y)
                       {
                       cout<<"The Value Is Available In The List Having Node Number "<<count<<endl;
                       break;
                                  }
                       else
```

```
                          else
                          {
                          count++;
                          head=head->next;
                          }
                          }
                          if(head==NULL)
                          {
                          cout<<"The Value Is Not Available In The List "<<endl;
                          }
        }
 //To Print The List
 else if(a==8)
    {
          head=first;
      while(head!=NULL)
      {
                             cout<<head->no<<"->";
                             head=head->next;
                                               }

                             cout<<endl<<endl;
                                                 }
 //To Revrse the List
 else if(a==9)
 {
      temp=first;
      temp1=first->next;
      temp2=temp1->next;
      temp->next=NULL;
      temp1->next=temp;
      while(temp2!=NULL)
      {
      temp=temp1;
      temp1=temp2;
      temp2=temp2->next;
      temp1->next=temp;
      }
      first=temp1;
      }
 // To Exit
 else if(a==10)
 {
      exit(1);
      }
          }
system("pause");
return 0;
}
```