Spring 2019 (Course Number 009952)

Data Structures and Lab
(Lecture 03: Singly Linked List)

Prof. A. P. Shrestha, Ph.D.

Dept. of Computer Science and Engineering, Sejong University

Last Class

- Array
- Insertion and deletion in array
- Sorting in array

Today

- C++ Simple Program Exercises
  -Concepts of **class**, **object** and **constructors**
- Singly Linked List

Next class

- Operations in Linked List
- Doubly linked list

3.1.1 Introduction

Access specifier
(there is also another access specifier called “protected”)

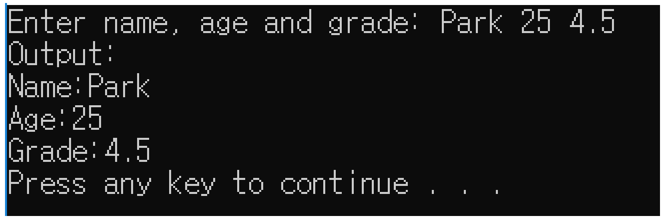
Do not forget semi-colon

```
#include<iostream>

using namespace std;

class className
{
   private:  //can access only within the class
        dataMembers;
   public:   //can access within the class or out-side class
        memberFunctions()
        {
             ……
        }

};

void main()
{
   className obj; //create object  named obj
   obj.memberFunctions(); //access member function using obj
}
```

3.1.2 Example 1-Simple Program in C++

```cpp
#include<iostream>
using namespace std;
class student
{
private:
        char name[20];
        int age;
        float grade;
public:
        void getInfo()
        {
                cout << "Enter name, age and grade: ";
                cin >> name >> age >> grade;
        }
        void putInfo()
        {
                cout << "Output:\n";
                cout << “Name:" << name << endl;
                cout << “Age:" << age << endl;
                cout << “Grade:" << grade << endl;
        }
};
```

```cpp
int main()
{
        student obj;
        obj.getInfo();
        obj.putInfo();
        system("pause");
}
```



```
Enter name, age and grade: Park 25 4.5
Output:
Name:Park
Age:25
Grade:4.5
Press any key to continue . . .
```

3.1.3 Example 2-Simple Program in C++

scope resolution operator

```cpp
#include<iostream>

using namespace std;

class student
{
private:
    char name[20];
    int age;
    float grade;
public:
    void getInfo();
    void putInfo();
};
```

```cpp
void student::getInfo()
{
  cout << "Enter name, age and grade:";
  cin >> name >> age >> grade;
}


void student::putInfo()
{
  cout << "Output:\n";
  cout << "Name:" << name << endl;
  cout << "Age:" << age << endl;
  cout << "Grade:" << grade << endl;
}
```

```cpp
int main()
{
       student obj;
       obj.getInfo();
       obj.putInfo();
       system("pause");
}
```

3.1.4 Constructors in C++

- Constructor is
  - a special member function of a class
  - has no return type (not even void!!)
  - *implicitly* invoked *when object is created*
  - *useful for initialization*
  - defined in public section of class by default

- The name of the constructor is *same* as the name of the class.

Q) Can a constructor be defined in private section of class ?

Ans: Yes, constructor can be defined in private section of class

Example: Using friend class (details are not covered in this course)

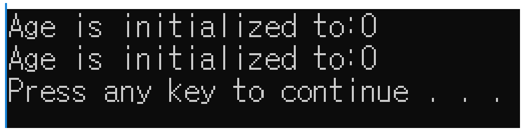
Note:

Self-study : Destructors (very easy!!)

3.1.5 Example 2- Constructors in C++

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

class student
{
private:
        int age;
public:
        student()//constructor
        {
                age = 0;
        }
        void display()
        {
                cout << "Age is initialized to:" << age << endl;
        }
};
```

```cpp
int main()
{
        student obj[2]; // array of objects
        obj[0].display();
        obj[1].display();
        system("pause");
}
```

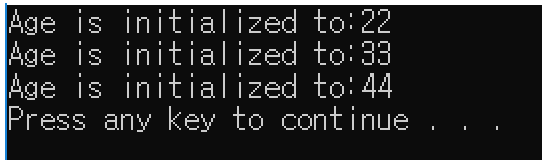


```
Age is initialized to:0
Age is initialized to:0
Press any key to continue . . .
```

3.1.6 Example 3-Constructors in C++

```cpp
#include<iostream>

using namespace std;

class student
{
private:
        int age;
public:
        student(int x)//Parameterized constructor
        {
                age = x;
        }
        void display()
        {
                cout << "Age is initialized to:" << age << endl;
        }
};
```

```cpp
int main()
{
    student obj_arr[2] = {22,33};
    student obj_var(44);
    obj_arr[0].display();
    obj_arr[1].display();
    obj_var.display();
    system("pause");
}
```



```
Age is initialized to:22
Age is initialized to:33
Age is initialized to:44
Press any key to continue . . .
```

3.2.1 List Data Structure

- The List is among the most generic of data structures.

- Real life examples:
    a. groceries list,
    b. list of people to invite to dinner
    c. list of presents to get

- A list is collection of items that are all of the same type (grocery items, integers, names)

- It is possible to insert new elements into various positions in the list and remove any element of the list

3.2.2 List Data Structure

- List is a set of elements in a linear order.

Example:

data values a_1, a_2, a_3, a_4 can be arranged in a list:

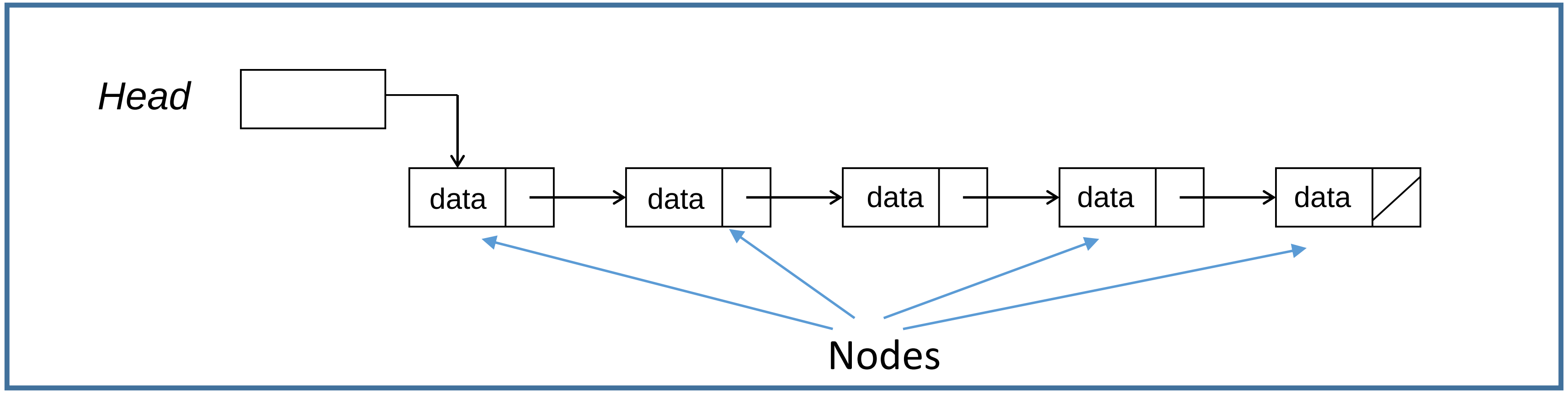
$$(a_3, a_1, a_2, a_4)$$

In this list, a_3, is the first element, a_1 is the second element, and so on.

- The order is important here; this is not just a random collection of elements, it is an ordered collection

3.2.3 Singly Linked List

- Allocate memory for each element separately and only when necessary.
- Appropriate when the number of data elements is unpredictable.
- Each node does not necessarily follow the previous one physically in the memory.
- Each node in a list consists of at least two parts:
  i) data
  ii) pointer to the next node



3.2.4 Linked List vs. Array

- **Advantages over arrays**

I) Dynamic size: length of a list can be increased or decreased

II) Ease of insertion/deletion: Linked lists can be maintained in sorted order by inserting or deleting an element at the proper point in the list.

- **Drawbacks**

I) Random access is not possible.

- We have to access elements sequentially starting from the first node.
- binary search with linked lists is not possible *(binary search will be covered on week 7).*

II) With each element of the list, extra memory space for a pointer is required .

3.3.1 List- Some Useful Operations

Operations	**Meaning**
createList():	create a new list (presumably empty)
copy():	set one list to be a copy of another
clear():	clear a list (remove all elements)
insert(X, ?):	insert element X at a particular position in the list
remove(?):	remove element at a particular position in the list
get(?):	get element at a particular position
update(X, ?):	replace the element at a particular position with X
find(X):	determine if the element X is in the list
length():	return the length of the list

3.3.2 List-Useful Operations

- What is meant by “a particular position”

- There are two possibilities:

1. Use the actual index of element: insert after element 3, get element number 6. This approach is taken by arrays. (Is it possible in linked list?)

2. Use a “current” marker or pointer to refer to a particular position in the list.

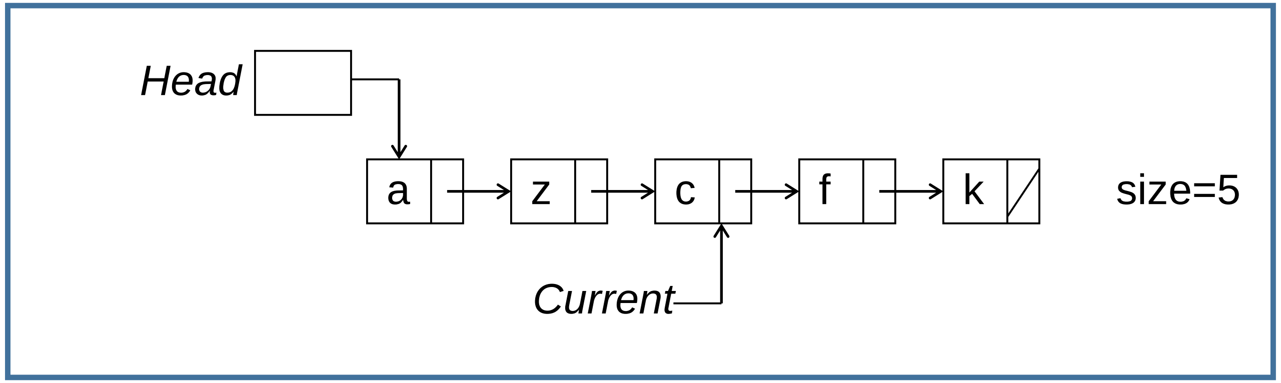
3.3.3 List-Useful Operations

- If we use the “current” marker, the following four methods (i.e. member function of a class) would be useful:

  1. **start()**: moves to “current” pointer to the very first element.
  2. **tail()**: moves to “current” pointer to the very last element.
  3. **next**(): move the current position forward one element
  4. **back**(): move the current position backward one element

3.3.4 Singly Linked List

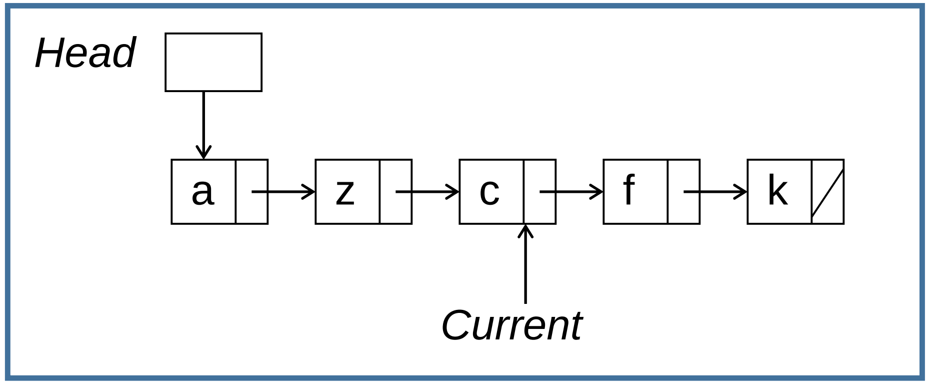
- A list (a, z, c, f, k) stored as a singly linked list:

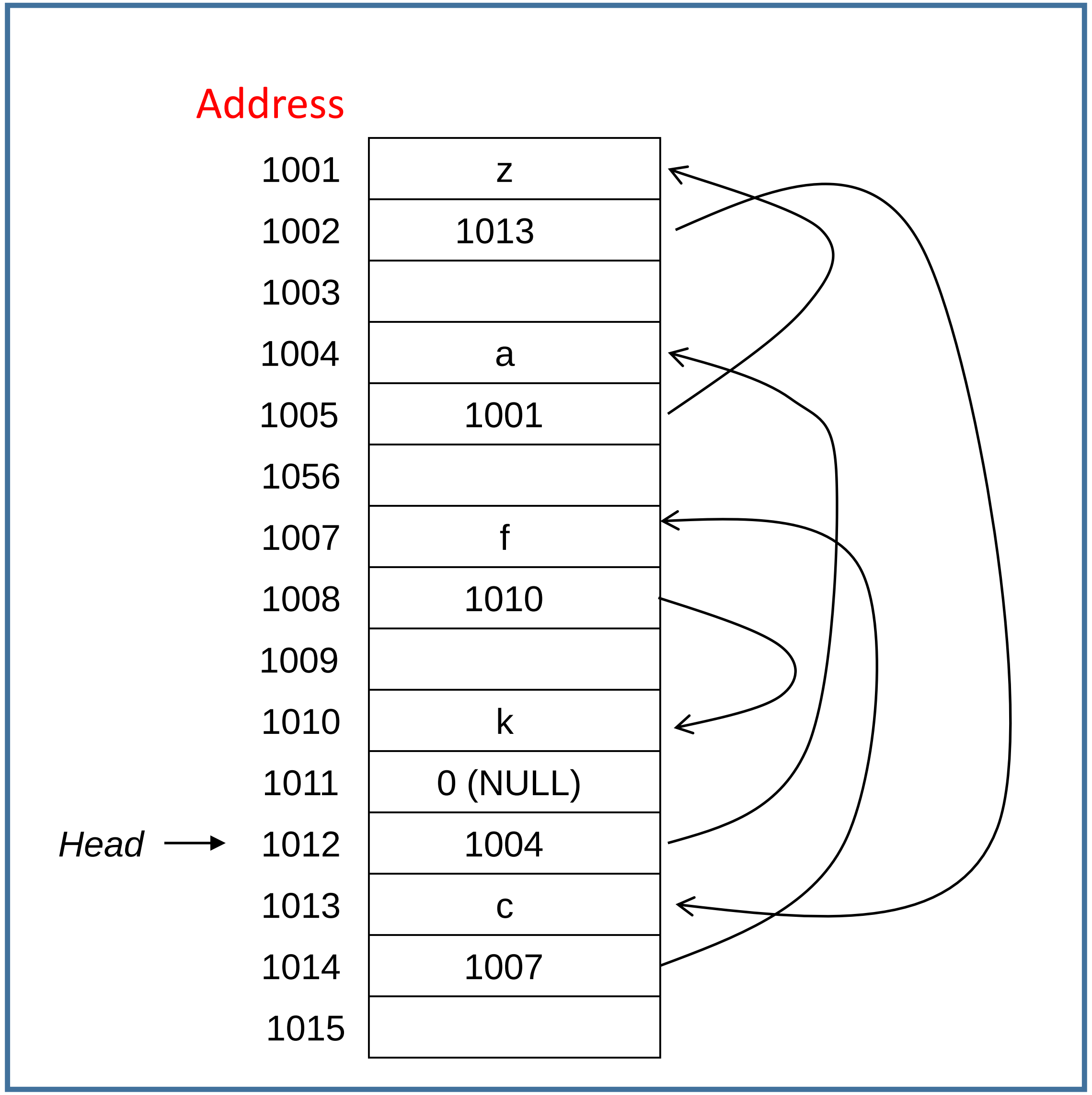


Note!!

- Need a *head* to point to the first node of the list. Otherwise, we won't know where the start of the list is.
- The *current* here is a pointer, not an index
- The next field in the last node points to *nothing*. We will place the memory address NULL which is guaranteed to be inaccessible

3.3.5 Singly Linked List –Memory Diagram





Address	
1001	z
1002	1013
1003	
1004	a
1005	1001
1056	
1007	f
1008	1010
1009	
1010	k
1011	0 (NULL)
Head → 1012	1004
1013	c
1014	1007
1015	

3.4.1 Singly Linked List-Implementation

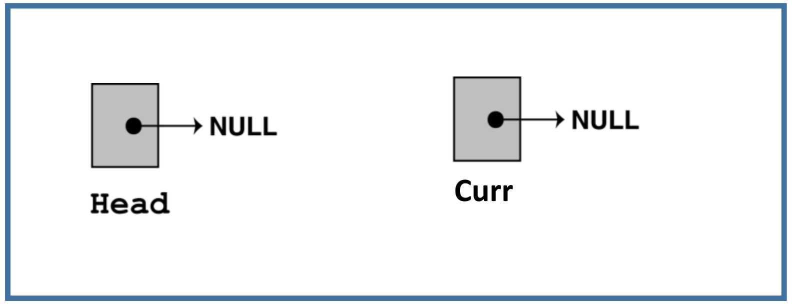
```
class List
{
private:
        struct node
        {
                int data;
                node *next;
        };

        typedef node* nodePtr;
        nodePtr head, curr, temp;

public:
        List(); // constructor
        void AddNode(int addData);// adds a new node at end of list with key addData
        void DeleteNode(int delData); // deletes a node with given key delData
        void PrintList(); // prints all the elements in the list
};
```

Check https://www.youtube.com/watch?v=H5lkmKkfjD0

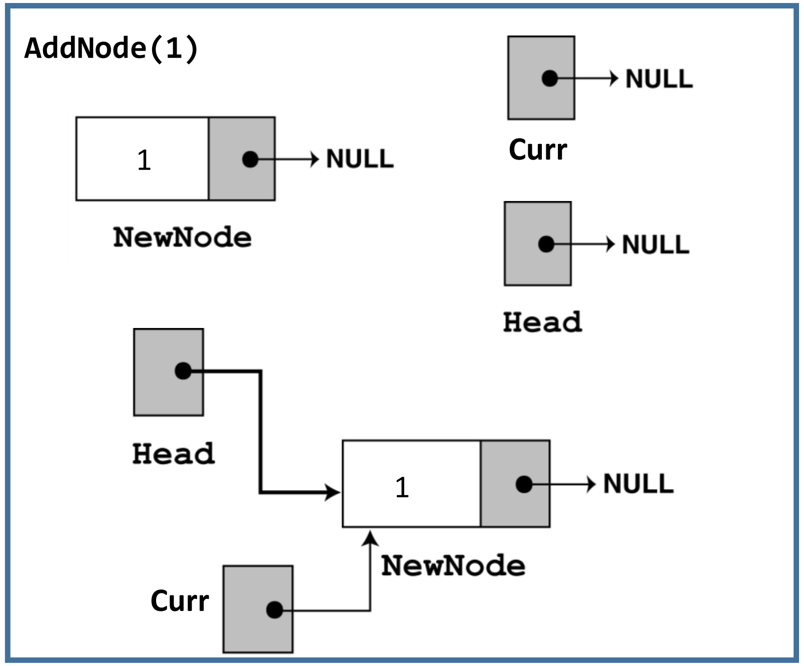
3.4.2 Singly Linked List-Constructor

```
List::List()
{
        head=NULL; // beginning of List
        curr=NULL; // “current” marker/pointer to refer to a particular position in list
        temp=NULL;
}
```

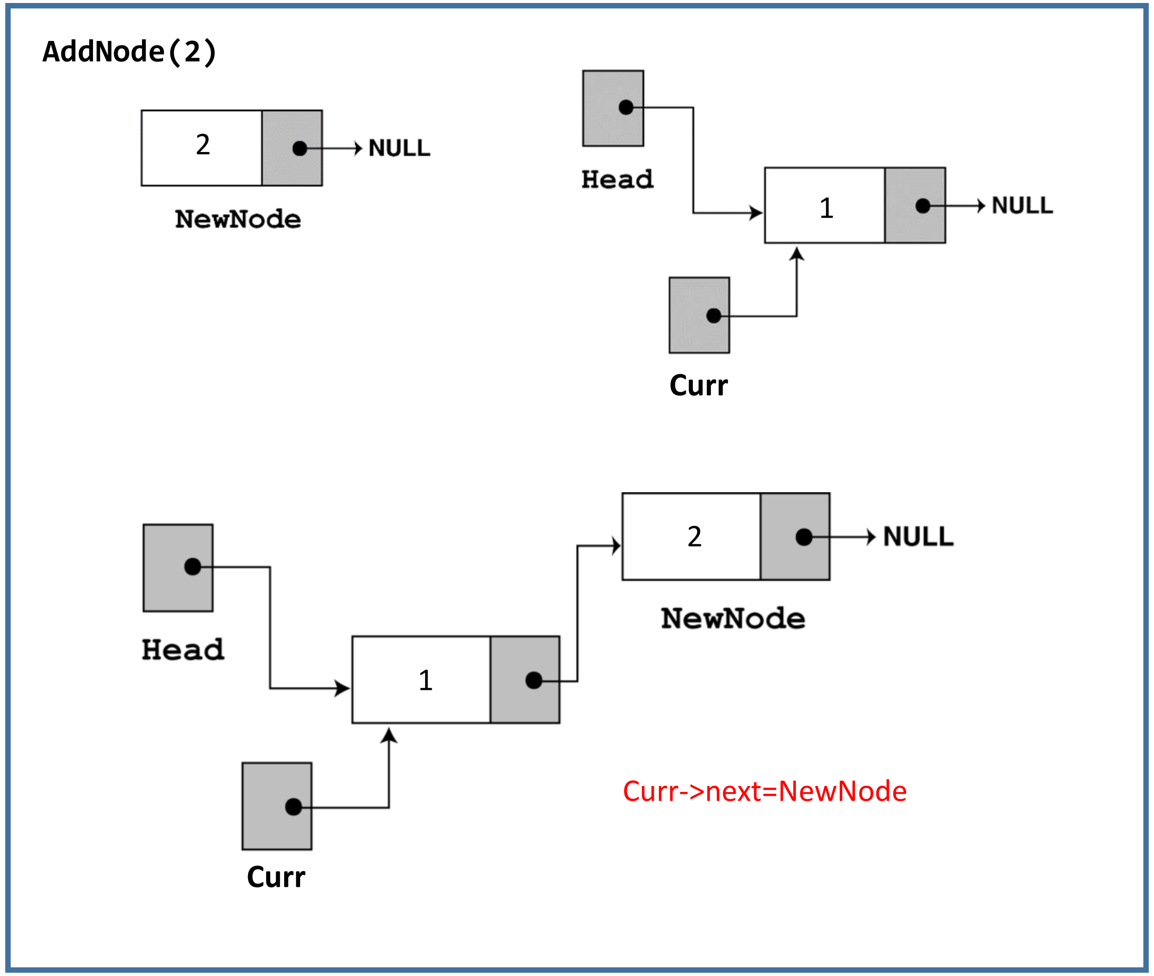
3.4.3 Singly Linked List-Add Node at End



AddNode(1)

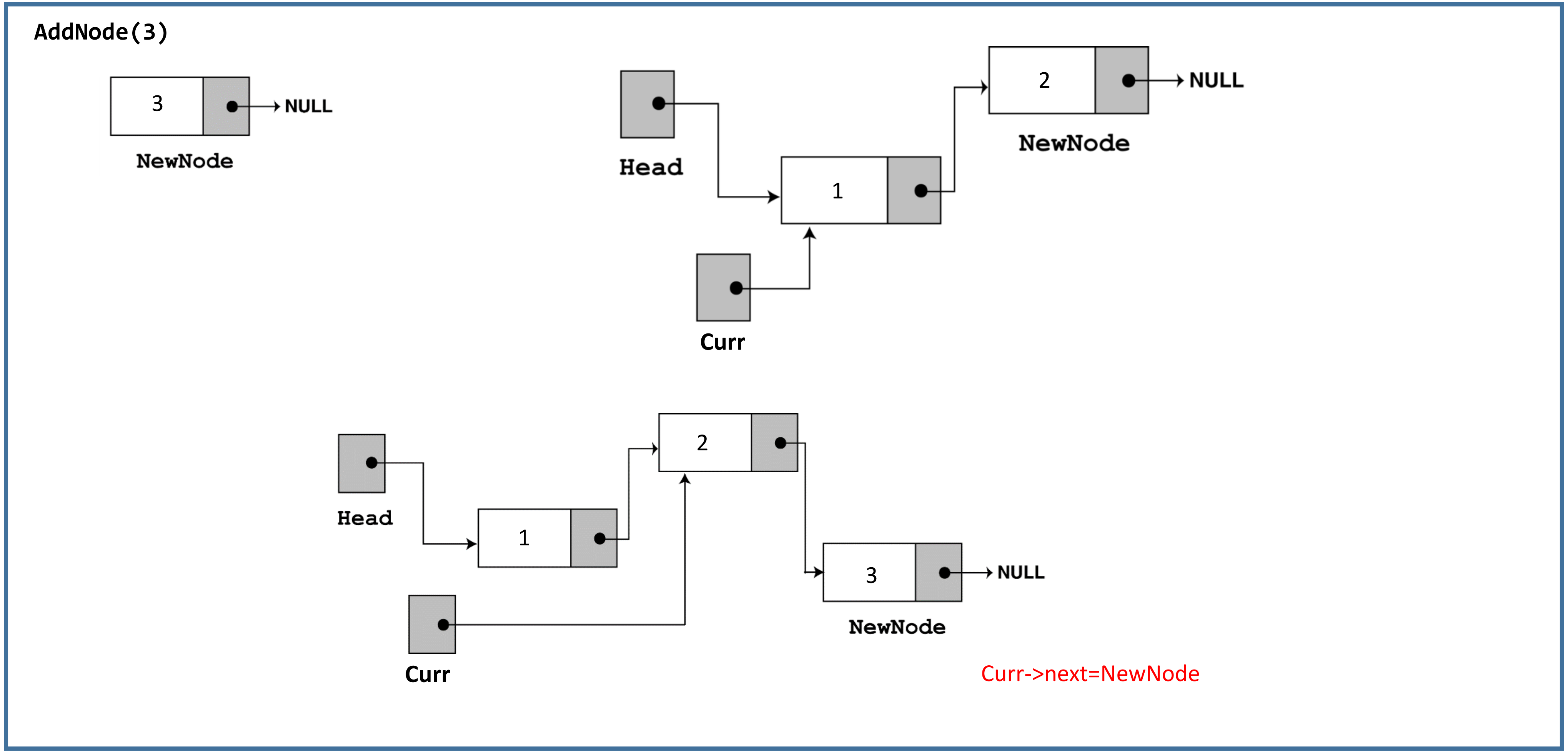


AddNode(2)



Curr->next=NewNode

3.4.4 Singly Linked List-Add Node at End



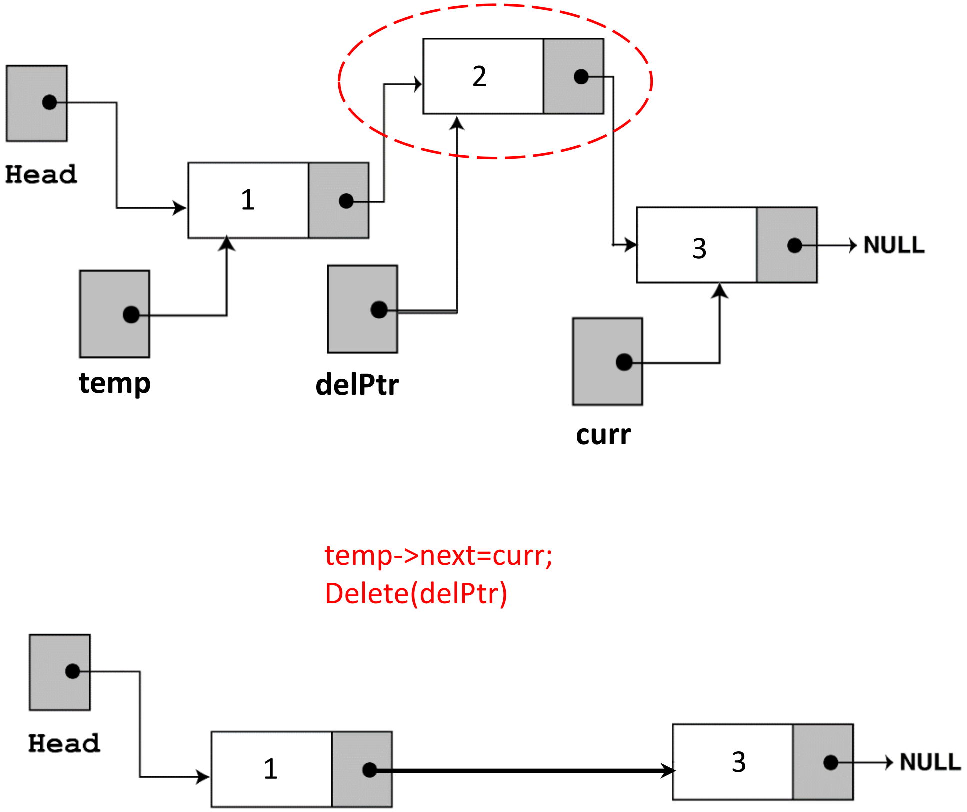
Curr->next=NewNode

3.4.5 Singly Linked List-AddNode Function

```
void List::AddNode(int addData)
{
        nodePtr n= new node; //Dynamic memory allocation
        n->next= NULL;
        n->data=addData;

        if(head!=NULL)//List is already set up
        {
                curr = head;
                while(curr->next!=NULL){
                        curr = curr->next;
                }
                curr->next=n;
        }
        else //List is empty or there is no node in List
        {
                head = n;
        }
}
```

3.4.6 Singly Linked List-Delete Node with a Given Key

```
DeleteNode(2)
```



```
temp->next=curr;
Delete(delPtr)
```

3.4.7 Singly Linked List-AddNode Function

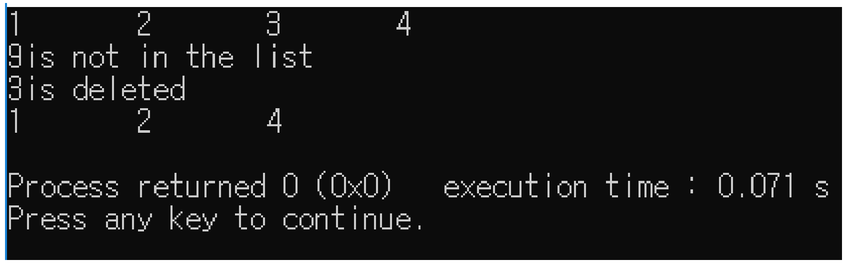
```
void List:: DeleteNode(int delData)
{
    nodePtr delPtr =  NULL;
    temp = head;
    curr = head;
    while (curr!=NULL && curr->data!=delData){
        temp = curr;
        curr = curr->next;
    }
    if (curr == NULL){
        cout<< delData <<"is not in the list\n";
        delete delPtr;
    }
```

```
else{
        delPtr = curr;
        curr= curr->next;
        temp->next=curr;
        if(delPtr==head){
            head = head->next;
            temp = NULL;
        }
        delete delPtr;
        cout<< delData<<"is deleted\n";
}

}//end of function
```

//search delData until found to the end of list

3.4.8 Singly Linked List-Test

```
int main()
{
        List l1;
        l1.AddNode(1);
        l1.AddNode(2);
        l1.AddNode(3);
        l1.AddNode(4);
        l1.PrintList();
        l1.DeleteNode(9);
        l1.DeleteNode(3);
        l1.PrintList();

        return 0;
}
```



```
1       2       3       4
9is not in the list
3is deleted
1       2       4

Process returned 0 (0x0)   execution time : 0.071 s
Press any key to continue.
```

Note:

When you code, use switch statement and loop to create MENU

Q & A ?