មហាវិទ្យាល័យវិស្វកម្ម
FACULTY OF ENGINEERING

# Data Structure & Algorithm

## Lecture 9
## Abstract Data Types: Single Linked Lists

Chhoeum Vantha, Ph.D.
Telecom & Electronic Engineering

# Content

- Abstract Data Types
  - Stacks
  - Queues and Priority Queues
  - Linked Lists: Single Linked Lists
  - Abstract Data Types
  - Specialized Lists

## Check the video Before the lecture to get some ideas related linked list:

1. Introduction to Linked List

   https://www.youtube.com/watch?v=R9PTBwOzceo

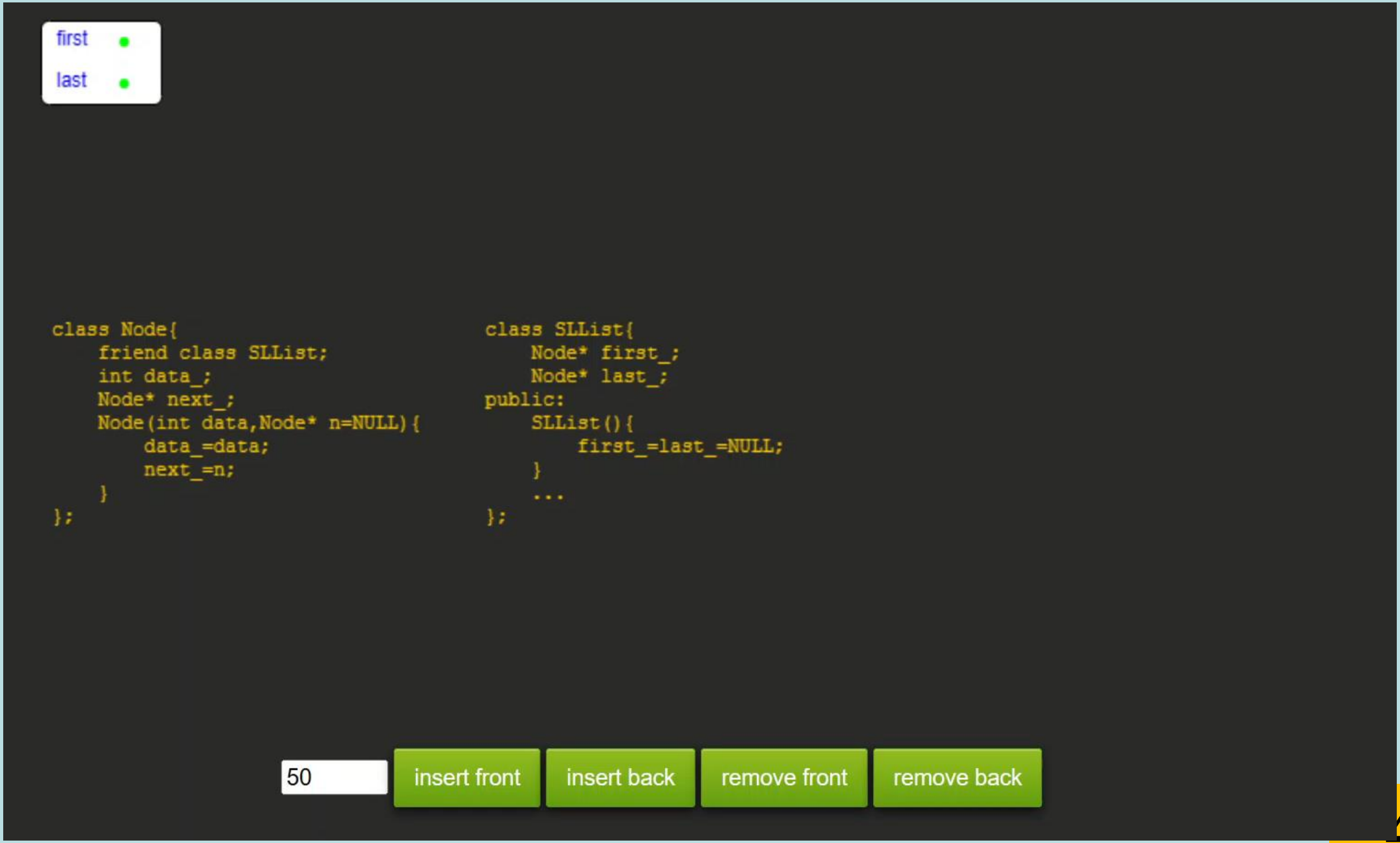
2. Creating the Node of a Single Linked List:

   https://www.youtube.com/watch?v=DneLxrPmmsw

3. Array vs. Single Linked List

   https://www.youtube.com/watch?v=b5QR4AmrspU

# What is the video all about?



```
class Node{
    friend class SLList;
    int data_;
    Node* next_;
    Node(int data,Node* n=NULL){
        data_=data;
        next_=n;
    }
};
```

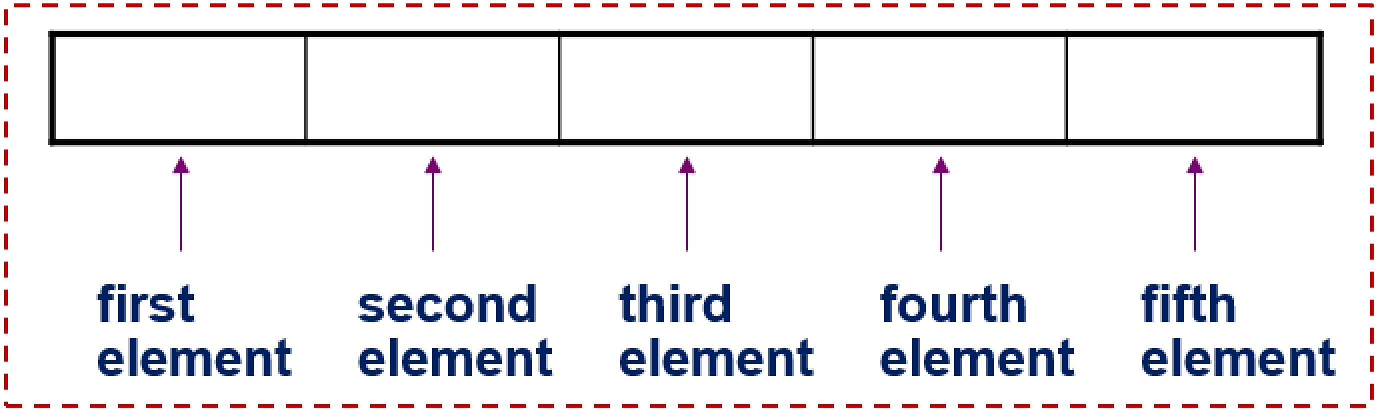
```
class SLList{
    Node* first_;
    Node* last_;
public:
    SLList(){
        first_=last_=NULL;
    }
    ...
};
```

# Lists

- Arrays
- Linked Lists
    - Singly Linked List
    - Doubly Linked List

# Arrays (Storage in Memory)

- In the definition:
- int [ ] tests;
- tests = new int[SIZE]; // **SIZE is 5**
- allocates the following memory



# An Arrays structure

- Storing data items in arrays has at least two limitations
  - The array size is fixed once it is created: Changing the size of the array requires creating a new array ad then copying all data from the old array to the new array
  - The data items in the array are next to each other in memory: Inserting an item inside the array requires shifting other items

# A linked structure

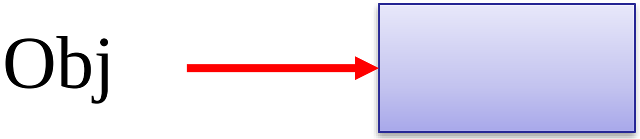
- A linked structure is introduced to overcome the limitations of arrays and allow easy insertion and deletion

# A linked structure

- A collection of nodes storing data items and links to other nodes
- If each node has a data field and a reference field to another node called next or successor, the sequence of nodes is referred to as a singly linked list
- Nodes can be located anywhere in the memory
- The first node is called the head and the last node is called tail

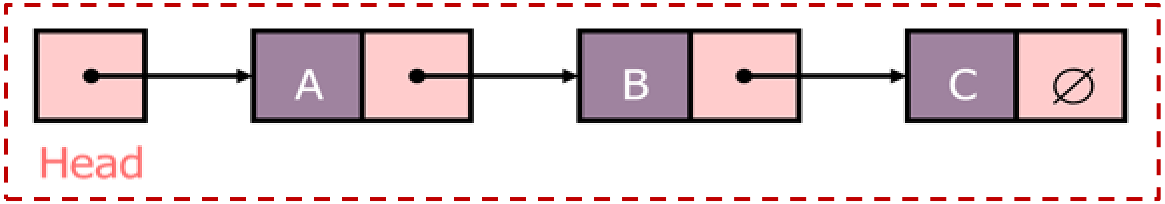
# A linked structure

- A linked structure uses object references to create links between objects
- Recall that an object reference variable holds the address of an object



# A linked structure

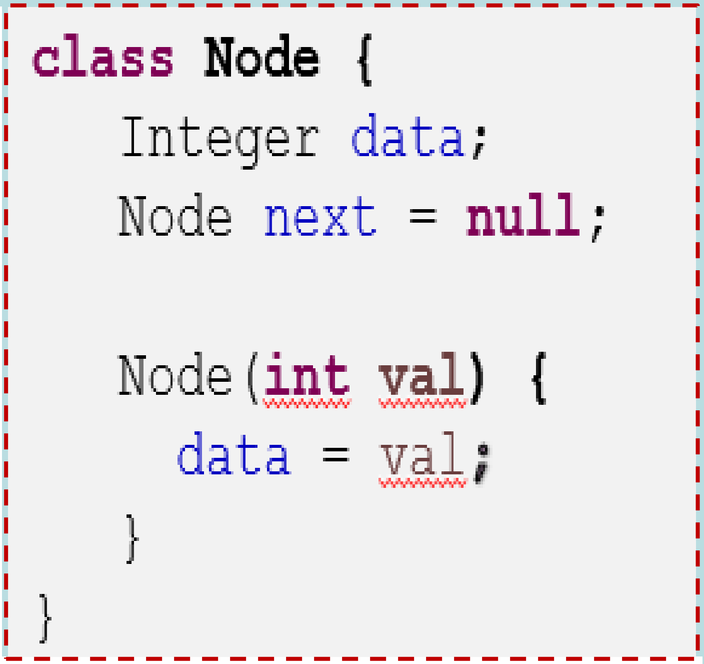
- Linked Lists are ***dynamic*** data structures that grow and shrink one element at a time, normally without some of the inefficiencies of arrays.
- A linked list is a series of connected ***nodes***



- We create a new node every time we add something to the List and we remove nodes when item removed from list and reclaim memory

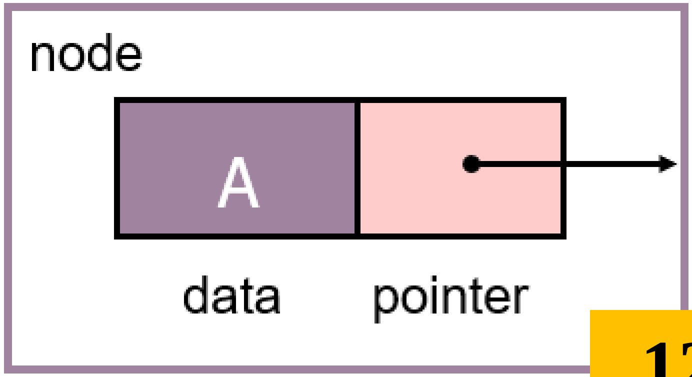
# A linked structure

- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- head : pointer to the first node
  - Sometimes called front, first
- The last node points to NULL



```
class Node {
    Integer data;
    Node next = null;

    Node(int val) {
      data = val;
    }
}
```

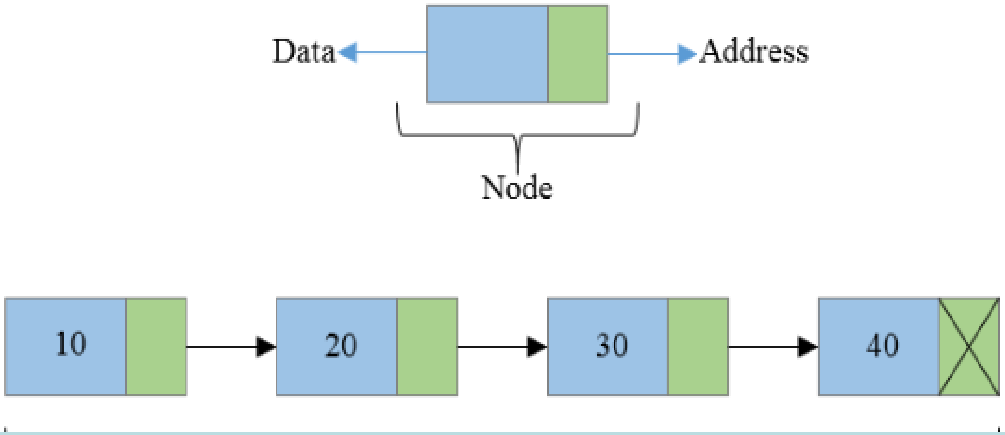


# Variations of Linked Lists

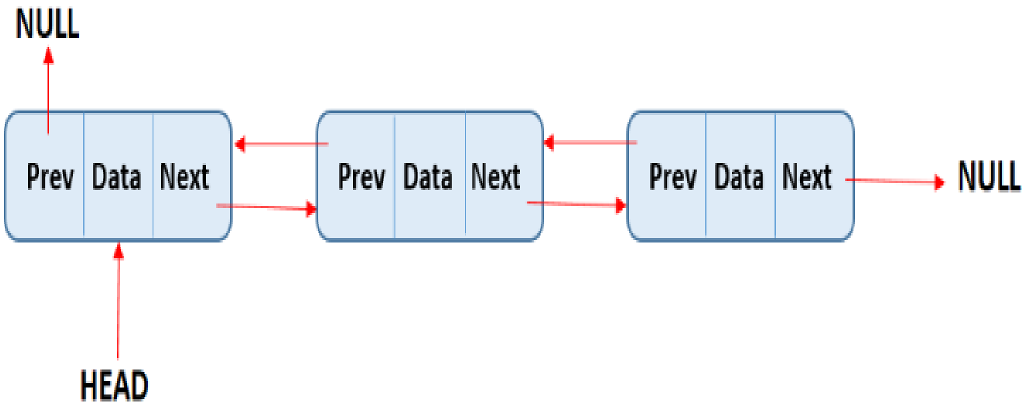
- Linked list can be
  - Singly-Linked
  - Doubly-Linked
  - Circular linked lists

# Variations of Linked Lists

- A single linked list contains only a next member.

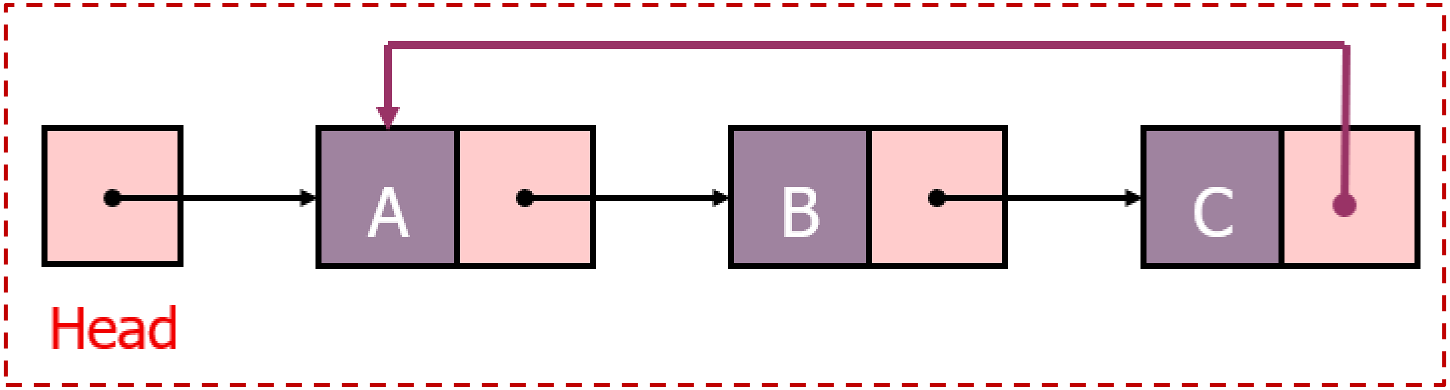


- A doubly linked list contains next and previous members



# Variations of Linked Lists

- **Circular linked lists**
  - The last node points to the first node of the list

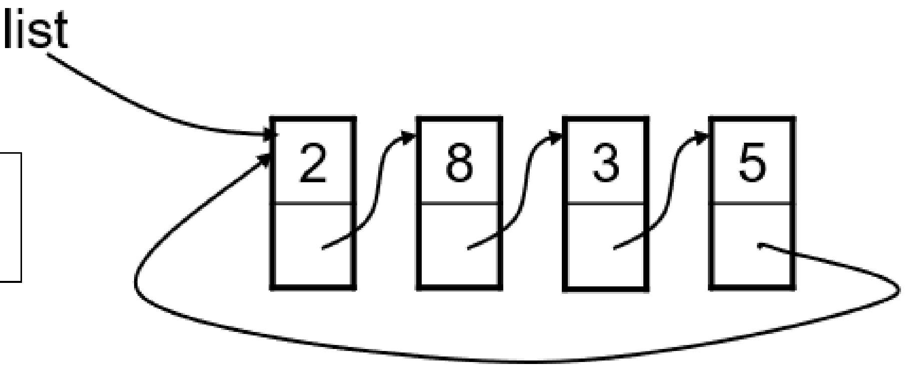


  - How do we know when we have finished traversing the list? (Tip: check if the pointer of the current node is equal to the head.)

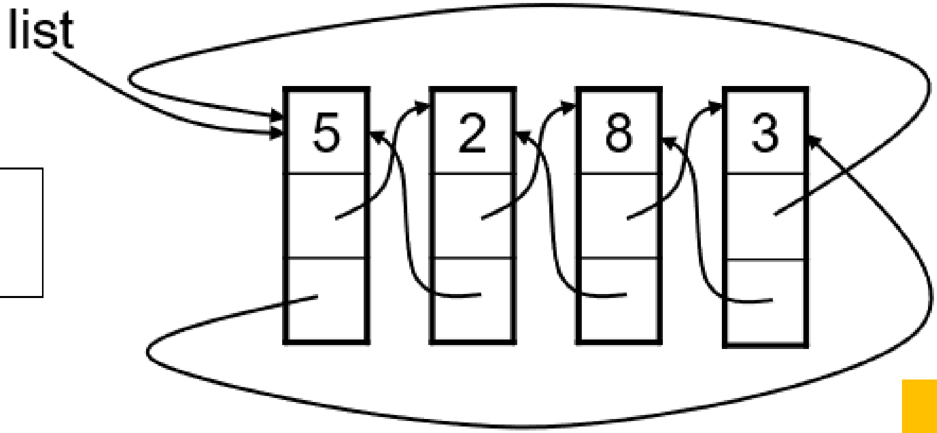
# Variations of Linked Lists

- **Circular linked lists**

Circular singly linked list

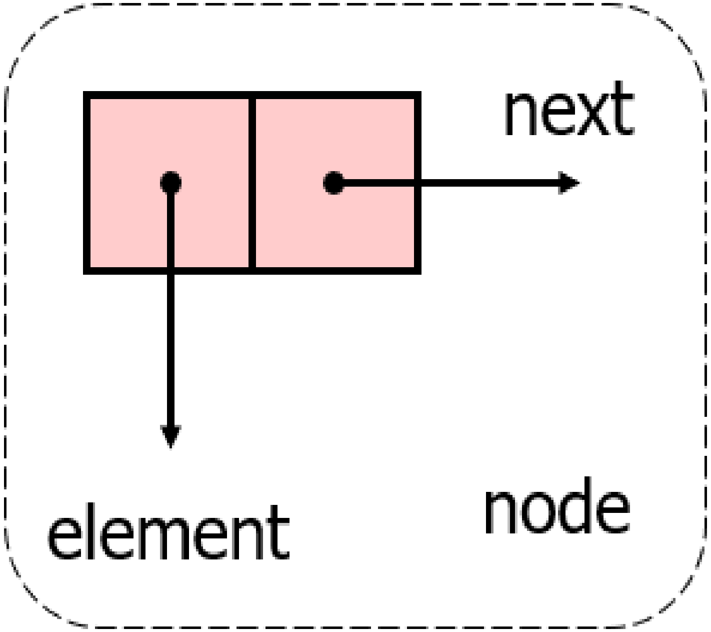


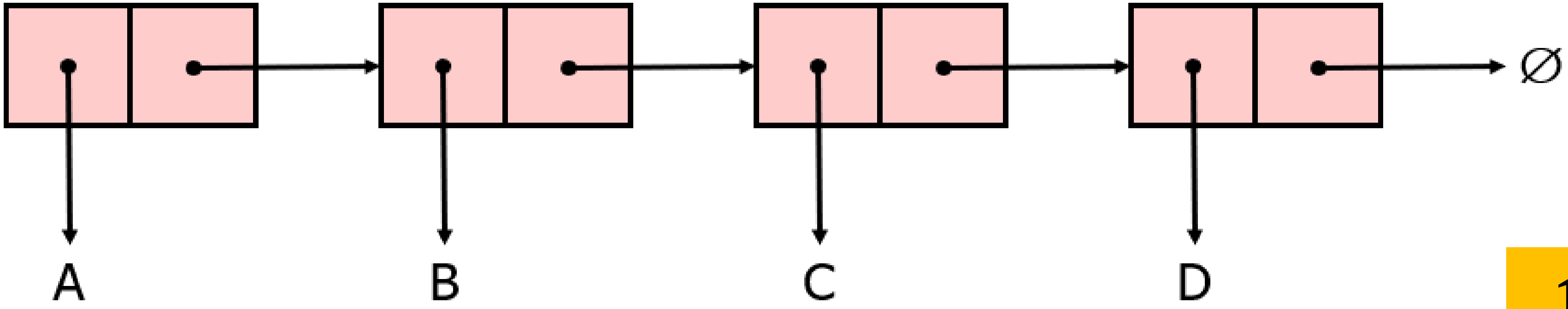
Circular doubly linked list



# Singly Linked Lists

- A singly linked list is a concrete data structure consisting of a sequence of nodes
- Each node stores
  - element
  - link to the next node





# Singly Linked Lists - Operations

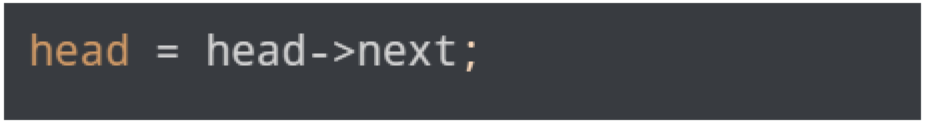
- **Traversal -** access each element of the linked list
- **Insertion** - adds a new element to the linked list
- **Deletion** - removes the existing elements
- **Search** - find a node in the linked list
- **Sort** - sort the nodes of the linked list

# Singly Linked Lists - Insertion

- We can add a node anywhere into the list:
  - Empty List
  - Before head
  - After tail
  - In between

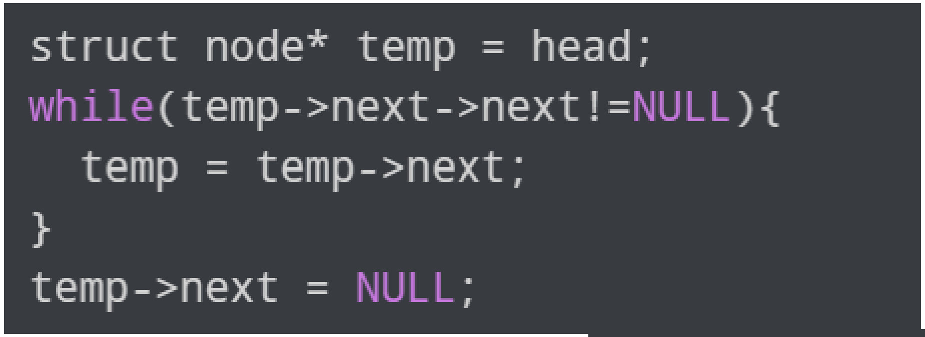
# Singly Linked Lists - Delete

- from beginning



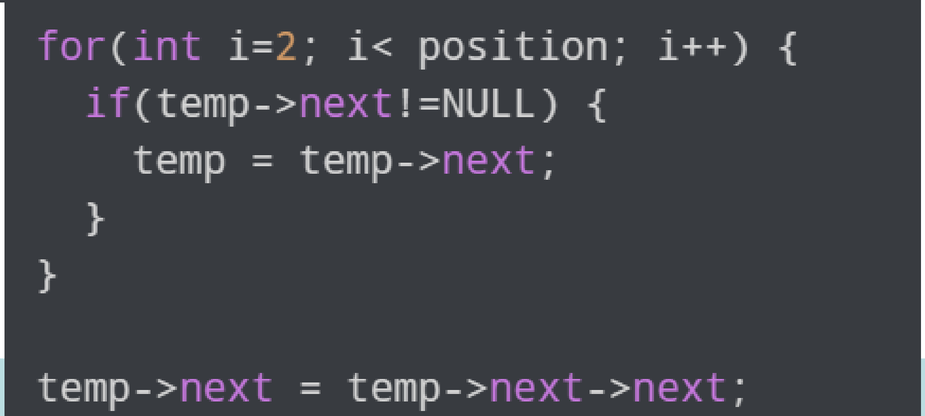
```
head = head->next;
```

- from end



```
struct node* temp = head;
while(temp->next->next!=NULL){
  temp = temp->next;
}
temp->next = NULL;
```

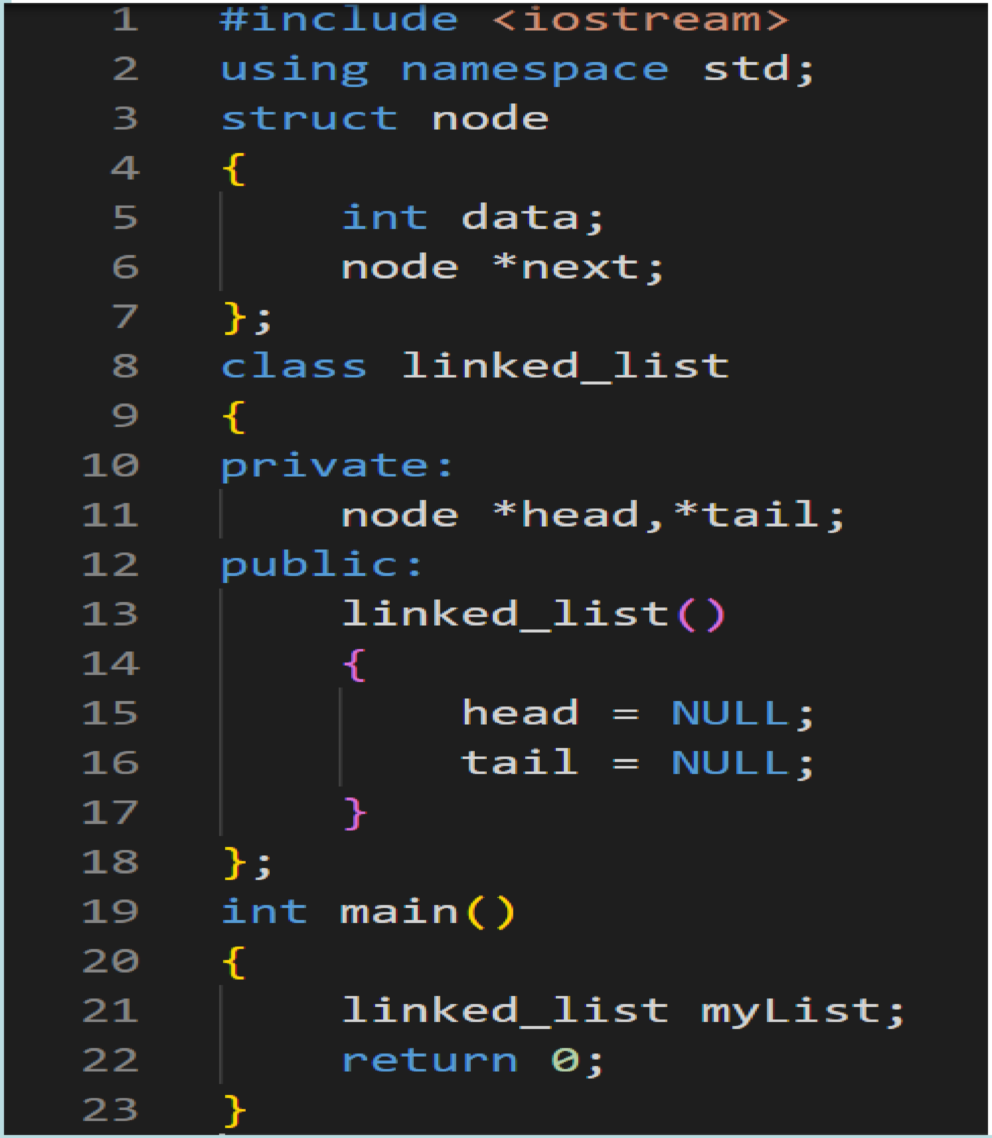
- from middle



```
for(int i=2; i< position; i++) {
  if(temp->next!=NULL) {
    temp = temp->next;
  }
}

temp->next = temp->next->next;
```

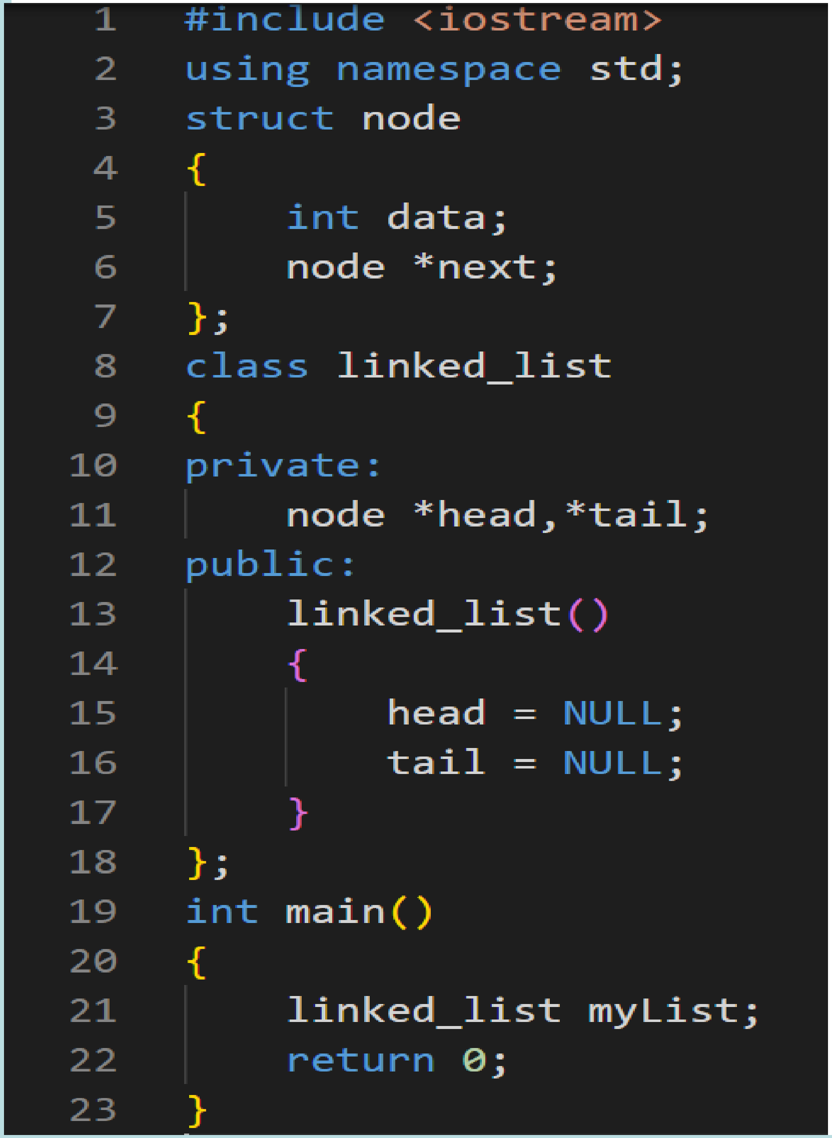
# Singly Linked Lists – C++



```
#include <iostream>
using namespace std;
struct node
{
    int data;
    node *next;
};
class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }
};
int main()
{
    linked_list myList;
    return 0;
}
```

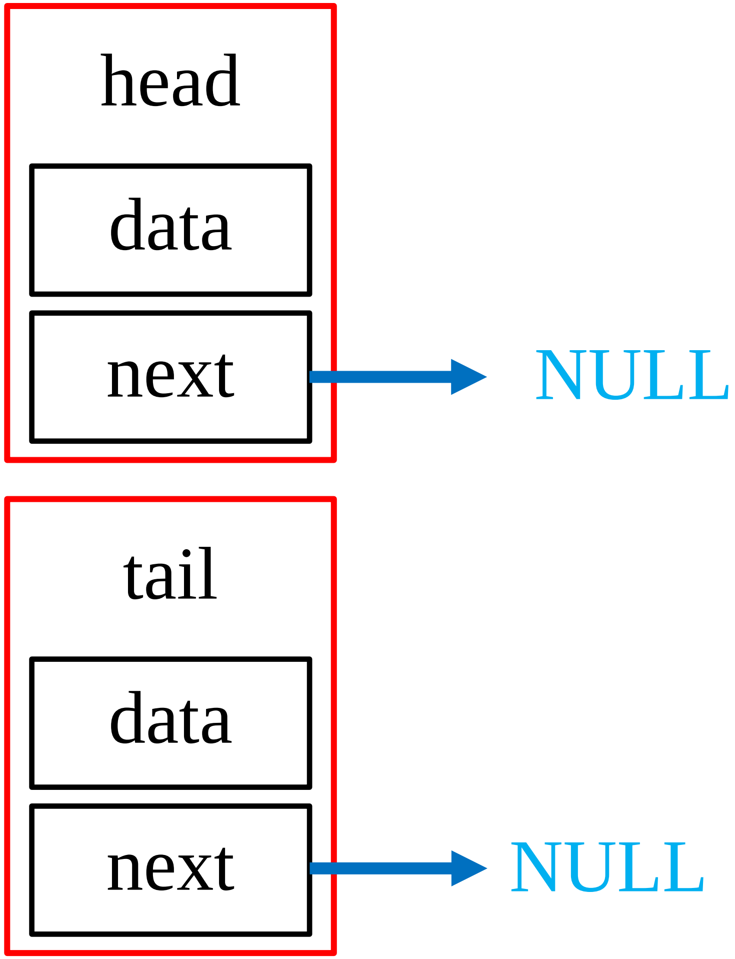
- What is this code representation?

# Singly Linked Lists – C++



```cpp
#include <iostream>
using namespace std;
struct node
{
    int data;
    node *next;
};
class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }
};
int main()
{
    linked_list myList;
    return 0;
}
```

- myList

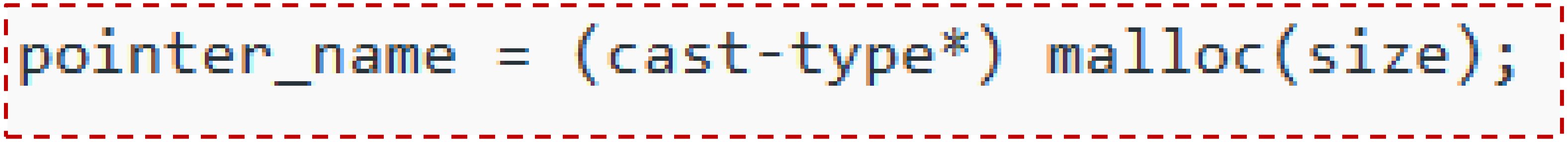


# Singly Linked Lists – malloc()

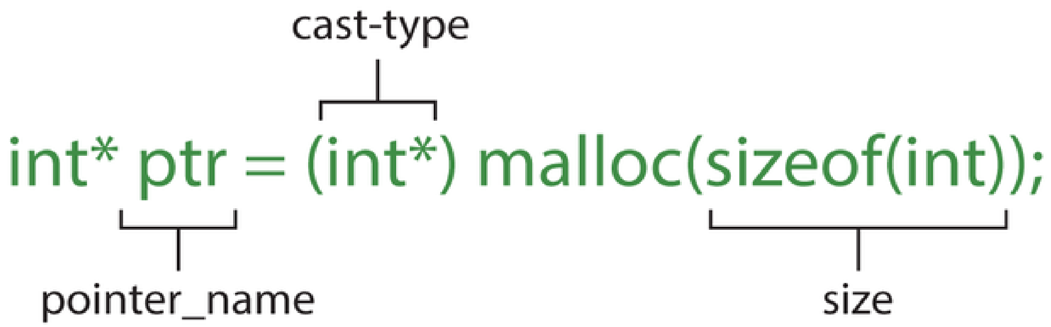
- The **function malloc()** in C++ is used to allocate the requested size of bytes and it returns a pointer to the first byte of allocated memory.
- A malloc() in C++ is a function that allocates memory at the runtime
  - **malloc()** is a dynamic memory allocation technique

# Singly Linked Lists – malloc()

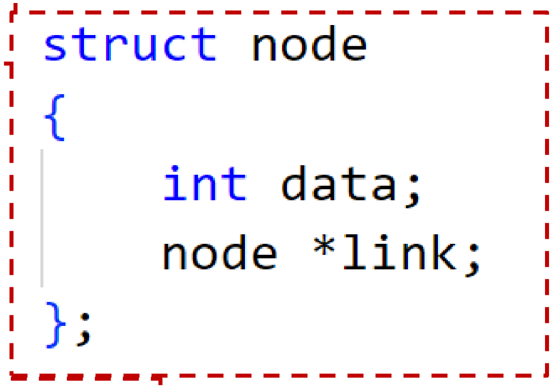
- **Syntax:**



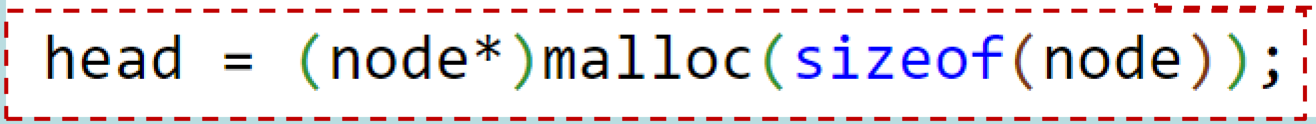
```
pointer_name = (cast-type*) malloc(size);
```



```
int* ptr = (int*) malloc(sizeof(int));
```



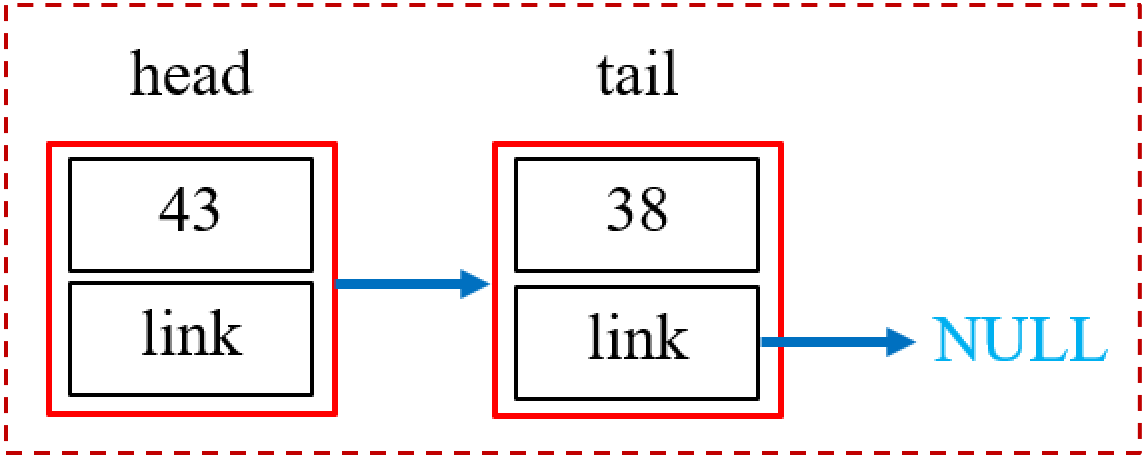
```
struct node
{
    int data;
    node *link;
};
```



```
head = (node*)malloc(sizeof(node));
```

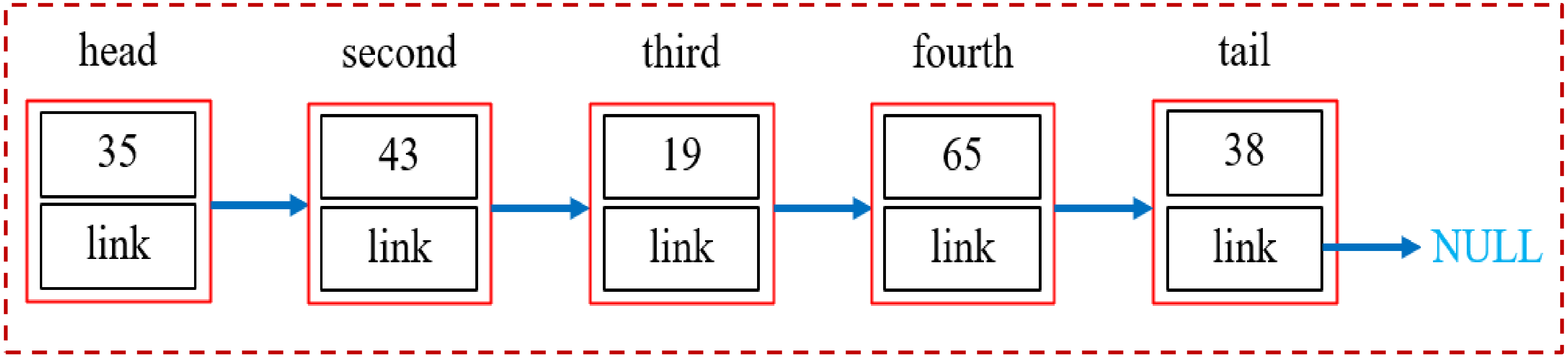
# Singly Linked Lists – head/tail

- Write in C++ to link head and tail node
- Print linked list



# Singly Linked Lists – link many nodes

- Write in C++ to link multiple nodes as show below:
- Print linked list



# Singly Linked Lists – insertBegine

- Write in C++ to insert element to begine of node
- Print linked list

# W9-10 – Lab 9-10

# Exercise- check at the End of lab

- Create a linked list class with the following functions:

```
// Function to insert a node at the end of the linked list.
        void insertNode(int);
// Function to print the linked list.
        void printList();
// Function to delete the node at given position
        void deleteNode(int);
```

# Exercise- extend at home

- Create a linked list class with the following functions:

```
// Search a node
        bool searchNode(p1, p2)
// Sort the linked list: Bobble sort
        void sortLinkedList()
```

Thanks!