Estructuras de Datos

Listas Simplemente Enlazadas

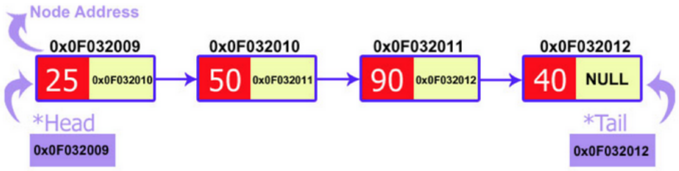
Prof. Luis Garreta

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

2 de noviembre de 2017

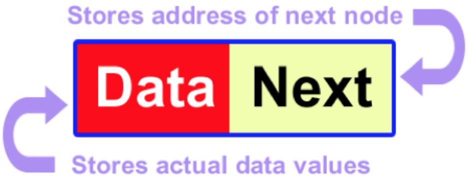
Linked Lists

- A linked list is a data structure that can store an indefinite amount of items.
- These items are connected using pointers in a sequential manner.
- There are two types of linked list;
  - singly-linked list, and
  - doubly-linked list.
- In a singly-linked list, every element contains some data and a link to the next element.
- In a doubly-linked list, every node in a doubly-linked list contains some data, a link to the next node and a link to the previous node.



Nodes

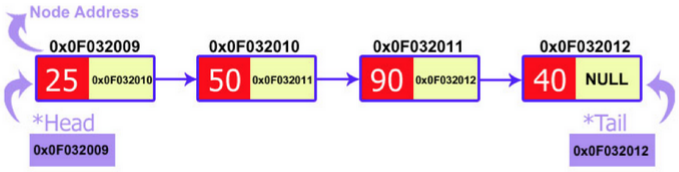
- The elements of a linked list are called the nodes.
- A node has two fields: data and next:
- **The data field:**
  - It contains the data being stored in that specific node.
  - It cannot just be a single variable.
  - There may be many variables presenting the data section of a node.
- **The next field:**
  - It contains the address of the next node.
  - So this is the place where the link between nodes is established.



Head and Tail

- No matter how many nodes are present in the linked list:
  - the very first node is called **head** and
  - the last node is called the **tail**.
- If there is just one node created then it is called both head and tail.

Sep



Implementation of Linked List Using C++

- As linked list consists of nodes, we need to declare a structure which defines a single node.
- Our structure should have at least one variable for data section and a pointer for the next node.
- In C++, our code would look like this:

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

Class List

- Now, we need a class which will contain the functions to handle the nodes.
- This class should have two important pointers, i.e.:
  - head and tail.
- The constructer will make them NULL to avoid any garbage value:

```
class list  {
  private:
    node *head, *tail;
  public:
    list() {
      head=NULL;
      tail=NULL;
    }
};
```

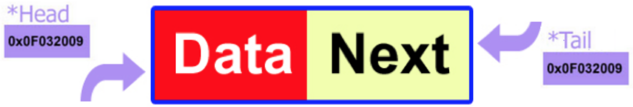
Node Creation

- Now, we will write a function for the node creation.
- The process of creating node is very simple:
  - We need a pointer of a node type (which we defined) and
  - we will insert the value in its data field.
  - The next field of node would be declared as NULL as it would be the last node of linked list.

```
void createnode(int value)
  {
    node *temp=new node;
    temp->data=value;
    temp->next=NULL;

  //...
  }
```

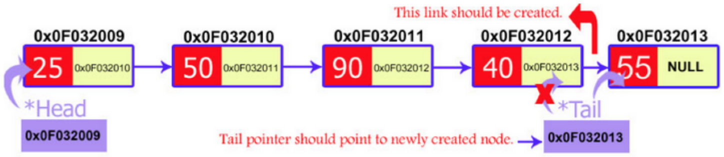
Connecting the new node to the Linking List

- What would happen if the linked list is still empty?
    - We will have to check it.
    - Do you remember that the head points to the first node?
    - It means if the head is equal to NULL then we can conclude that the linked list is empty.
- What would happen if there is just one node (which we are going to create) in linked lists,
    - then it is called both head and tail.
- And if a linked list is created already?



Adding the node to a created list

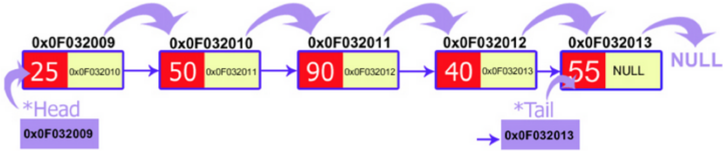
- then we would insert this node at the end of the linked list.
- We know that the last node is called a tail.
- So we are going to create this newly created node next to a tail node.
- The creation of a new node at the end of linked list has 2 steps:
  - Linking the newly created node with tail node.
  - Means passing the address of a new node to the next pointer of a tail node.
  - The tail pointer should always point to the last node.
  - So we will make our tail pointer equal to a new node.



The C++ code for the creation of new a node

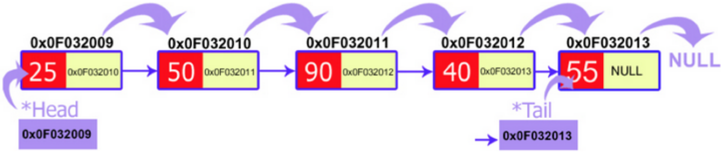
```
void createnode(int value) {
    node *temp=new node;
    temp->data=value;
    temp->next=NULL;

    if(head==NULL)  {
      head=temp;
      tail=temp;
      temp=NULL;
    } else {
      tail->next=temp;
      tail=temp;
    }
  }
```

Displaying Linked List Using C++



- Now we have a working linked list which allows creating nodes.
- If we want to see that what is placed in our linked list then we will have to make a display function.
- The logic behind this function is that we make a temporary node and pass the address of the head node to it.
- Now we want to print all the nodes on the screen.
- So we need a loop which runs as many times as nodes exist.
- Every node contains the address of the next node so the temporary node walks through the whole linked list.
- If the temporary node becomes equal to NULL then the loop would be terminated.

The code for displaying nodes of linked list

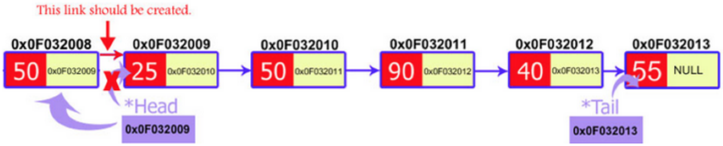


```
void display() {
  node *temp=new node;
  temp=head;

  while(temp!=NULL)   {
    cout<<temp->data<<"\t";
    temp=temp->next;
  }
}
```

Other Operations

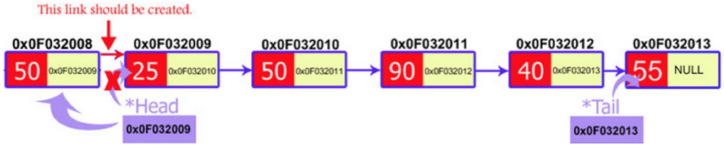
- The basic framework of a singly-linked list is ready.
- Now it is the time to perform some other operations on the list.
- Basically, two operations are performed on linked lists:
    - Insertion
    - Deletion

Insertion at the start



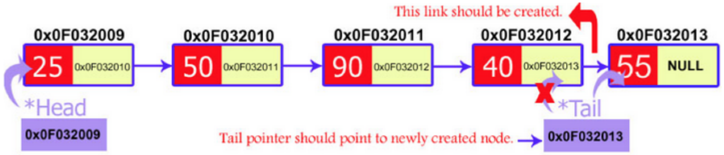
- Insertion of a new node is quite simple.
- It is just a 2-step algorithm which is performed to insert a node at the start of a singly linked list.
    - New node should be connected to the first node, which means the head. This can be achieved by putting the address of the head in the next field of the new node.
    - New node should be considered as a head. It can be achieved by declaring head equals to a new node.
- The diagrammatic demonstration of this process is given below:

Code for the insertion at the start process



```
void insert_start(int value) {
  node *temp=new node;
  temp->data=value;
  temp->next=head;
  head=temp;
}
```

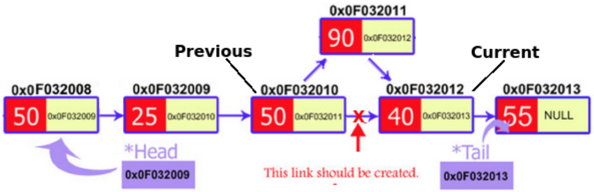
Insertion at the End



- The insertion of a node at the end of a linked list is the same as we have done in node creation function.
- If you noticed then, we inserted the newly created node at the end of the linked list.
- So this process is the same.

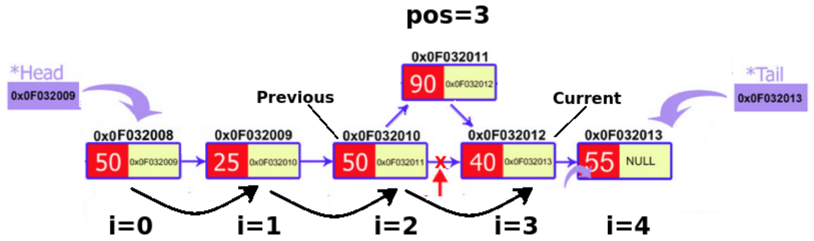
Insertion at Particular Position

- The insertion of a new node at a particular position is a bit difficult to understand.
- In this case, we don't disturb the head and tail nodes.
- Rather, a new node is inserted between two consecutive nodes.
- So, these two nodes should be accessible by our code.
- We call one node as current and the other as previous, and the new node is placed between them.



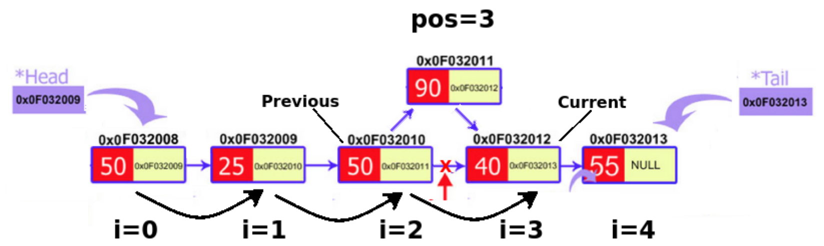
insertion: Connecting the nodes

- Now the new node can be inserted between the previous and current node by just performing two steps:
  - Pass the address of the new node in the next field of the previous node.
  - Pass the address of the current node in the next field of the new node.



- We will access these nodes by asking the user at what position he wants to insert the new node.
- Now, we will start a loop to reach those specific nodes.
- We initialized our current node by the head and move through the linked list.
- At the end, we would find two consecutive nodes.

C++ code for insertion of node



```
void insert_position(int pos, int value) {
  node *pre=new node;
  node *cur=new node;
  node *temp=new node;
  cur=head;

  for(int i=1;i<pos;i++) {
    pre=cur;
    cur=cur->next;
  }

  temp->data=value;
  pre->next=temp;
  temp->next=cur;
}
```

Deletion of a node

- So, you have become familiar with linked list creation.
- Now, it's time to do some manipulation on the linked list created.
- Linked lists provide us the great feature of deleting a node.
- The process of deletion is also easy to implement.
- The basic structure is to declare a temporary pointer which points the node to be deleted.
- Then a little bit of working on links of nodes.
- There are also three cases in which a node can be deleted:
    - Deletion at the start
    - Deletion at the end
    - Deletion at a particular position