CSCI 2270 Data Structures Recitation 5
Instructors:
Doubly Linked List

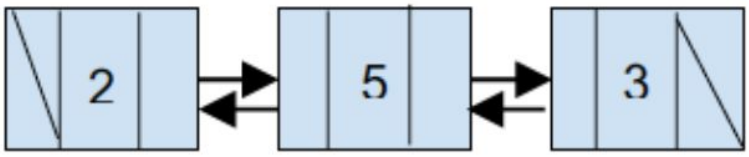
Learning Objectives:

- Understanding the structures of doubly linked lists;
- Work with a tail pointer;
- Delete a node from a doubly linked list.

---

1. Doubly Linked List

In a doubly linked list, each node in the list contains the node data, a pointer to the next node in the list, and a pointer to the previous node in the list. In the figure below, each node has three properties: an integer key, a pointer to the next node in the list, and a pointer to the previous node in the list. For the first node in the list, the previous pointer is set to NULL, and for the last node in the list, the next pointer is set to NULL. Nodes in a linked list can also be much more complex than these simple examples.



The node at the beginning of the list is called the head of the list. When implementing a linked list, a separate pointer should be stored to this node as it is the only entrance to the list. The last node in the list is called the tail of the list. All variables stored in memory have a memory location that can be accessed using a pointer variable. A linked list node can be implemented in C++ using a class or a struct and the next and previous pointers in the node reference another instance of the node.

The basic structure for each node in a doubly linked list looks like this:

```
//node implementation for doubly linked list
struct Node {
    int     key;
    Node*    previous;
    Node*    next;
};
```

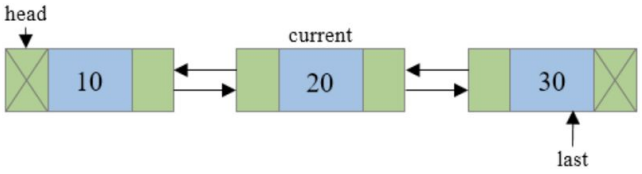
2. Deleting a node from linked lists

Deleting a node in the middle of a linked list requires traversing the list and making sure that we link the remaining nodes. If we find the node we want to delete and we delete it, without linking the remaining nodes together, we create a memory leak. It is important to link the preceding and following node with one another before deleting the desired node.

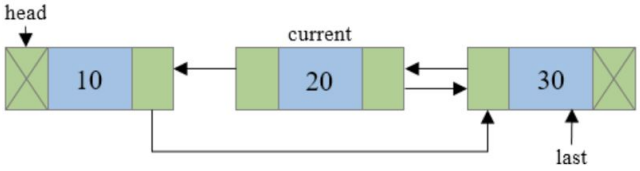
Example:

Let us suppose that we want to delete node from 2nd position.

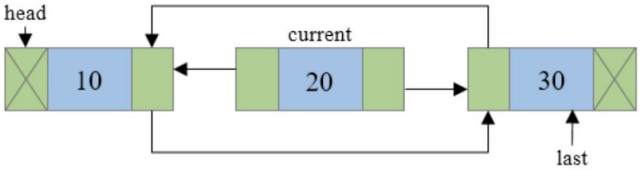
1. Traverse to `N^th` node of the linked list, lets say a pointer `current` points to `N^th` node in our case 2 node.



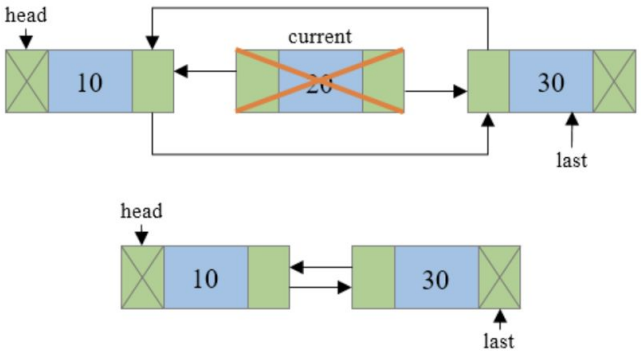
2. Link the node behind `current` node with the node ahead of `current` node, which means now the `N-1^th` node will point to `N+1^th` node of the list. Which can be implemented as `current->prev->next = current->next` .



3. If `N+1^th` node is not `NULL` then link the `N+1^th` node with `N-1^th` node i.e. now the previous address field of `N+1^th` node will point to `N-1^th` node. Which can be implemented as `current->next->prev = current->prev` .



4. Finally delete the `current` node from memory and you are done.



3. Including multiple files

In practice, you might want to include multiple source files. This way, your code will be more organized. Typically, we put the interface, or the class and function declarations in a header file, as shown below.

doublyLL.h

```
class doublyLL {
    private:
      Node* createNode(int,Node*,Node*);
      Node* head = nullptr;
      Node* tail = nullptr;
    public:
      bool insertNodeAtEnd(int);
      bool deleteNode(int);
      void print_list();
      doublyLL();
      doublyLL(int);
      ~doublyLL();    //destructor
};
```

The implementation of these functions will be written in another file with same name but an extension of ".cpp". The file needs to use "include" keyword to include the header file.

doublyLL.cpp

```
#include “doublyLL.h”

bool doublyLL::insertNodeAtEnd(int a) {
    // Your code here
}
bool doublyLL::deleteNode(int) {
    // Your code here
}

void doublyLL::print_list() {
    // Your code here
}

doublyLL::doublyLL() {
    // Your code here
}

doublyLL::doublyLL(int) {
    // Your code here
}

doublyLL::~doublyLL() {
    // Your code here
}
```

In your main function, then, you can include the cpp file like this

```
#include “doublyLL.cpp”
```

and use the functions you implemented as usual.

4. Programming exercise

In this recitation, you need to create a simple doubly linked (with at least 20 elements, out of which at least 5 elements have the same value, say ‘4’) first with the

node structure shown in section 1. Then, you need to delete all the nodes that have the same value in the list (like '4'), and print out the list.

You also need to create a class for this doubly linked list. For example,

```
class doublyLL {
    private:
      Node* createNode(int,Node*,Node*);
      Node* head = nullptr;
      Node* tail = nullptr;
    public:
      bool insertNodeAtEnd(int);
      bool deleteNode(int);
      void print_list();
      doublyLL();    //default constructor
      doublyLL(int);    //overloaded constructor
      ~doublyLL();    //destructor
};
```

Thus, you will need to implement the constructors and destructors, and all the other functions in this class.

Note that you need to write a header file which contains all the definitions of the node structs and linked list class; the implementation should be in another cpp file.