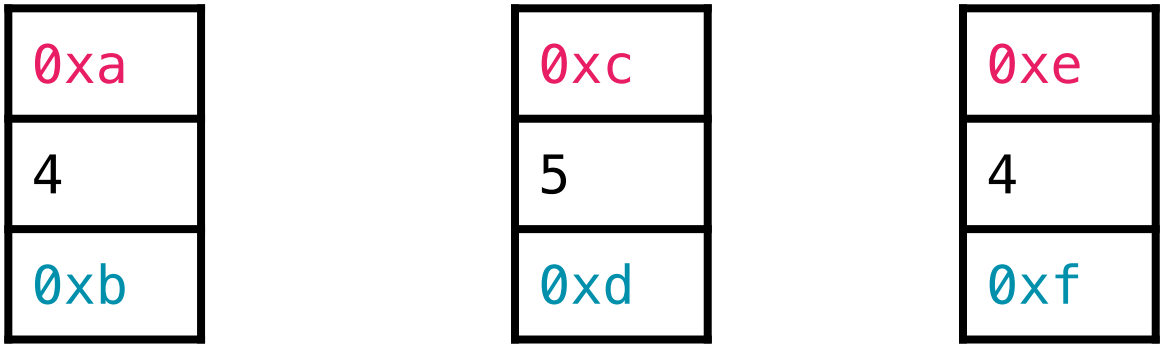
Lecture 7

Doubly Linked List

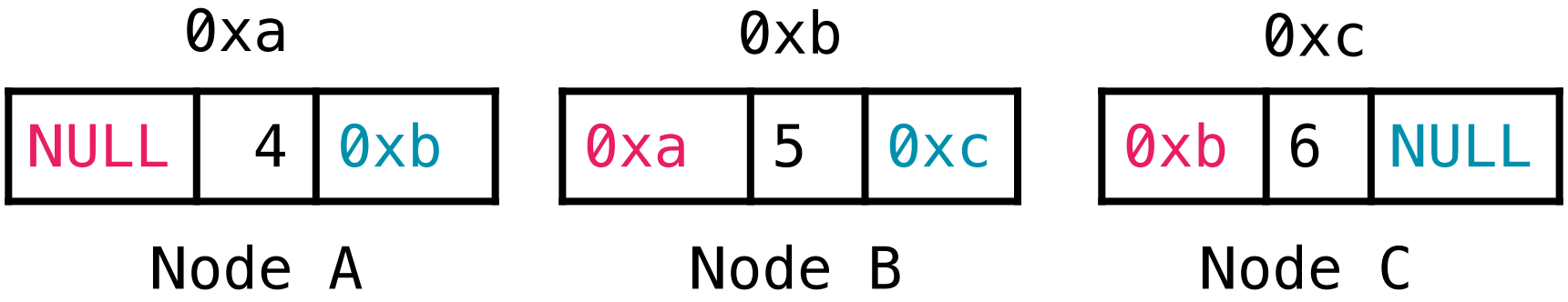
Doubly linked list

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

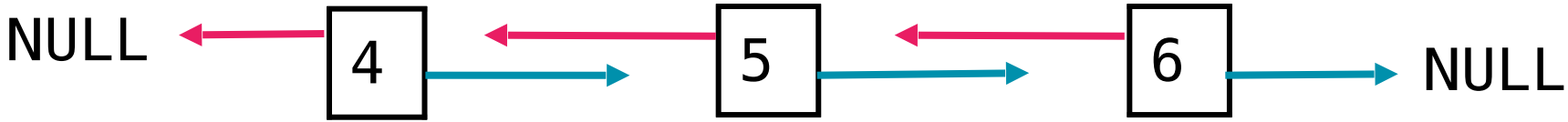
Doubly linked list



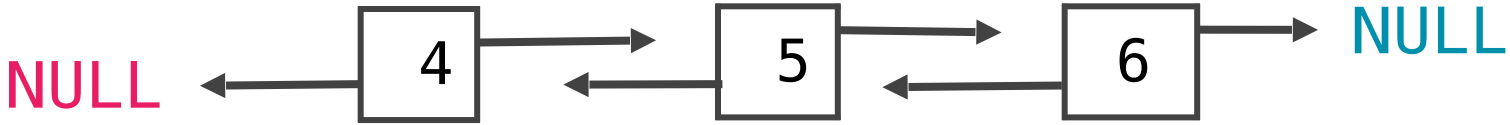
Doubly linked list

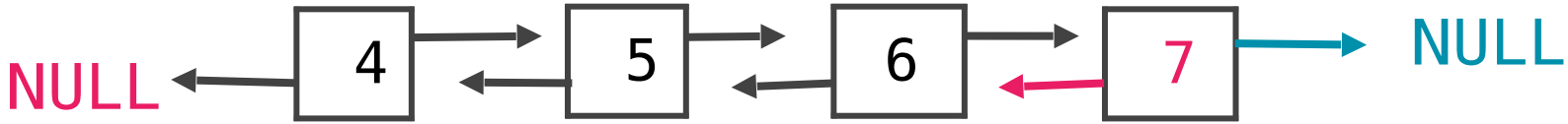


Doubly linked list



add at the end



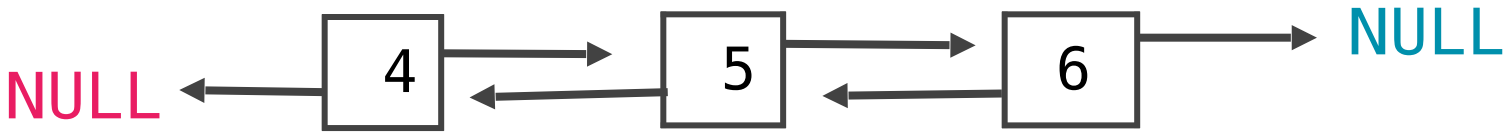


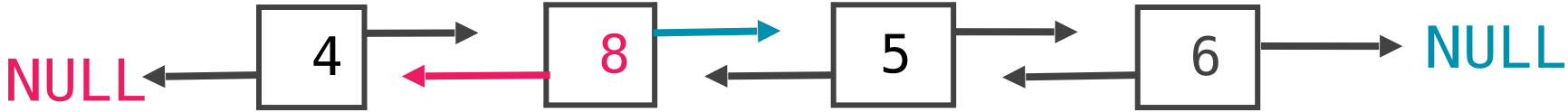
add at the end

```
void add(node* &dll, int data){
      if(dll == NULL){
            node *newNode = new node;
            newNode->data = data;
            newNode->next = dll;
            newNode->prev = NULL;
            dll = newNode;
      }
...
```

```
...
else{
        node *current = dll;
        while(current->next!=NULL){
                current = current->next;
        }
        node *newNode = new node;
        newNode->data = data;
        newNode->next = current->next;
        current->next = newNode;
        newNode->prev = current;
    }
}
```

add in the middle



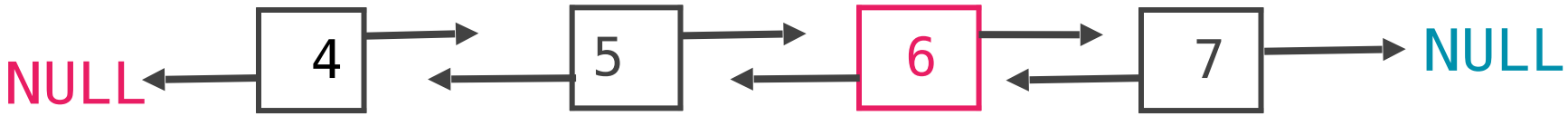


add in the middle

```
void insert(node *&dll, int index, int data){
        if(index == 0){
                node* newNode = new node;
        newNode->data = data;
        newNode->next = dll;
        newNode->prev = NULL;
                if((dll)!= NULL){(dll)->prev = newNode;}
                dll = newNode;
        }
...
```

```
...
else{
        node* current = dll;
        for(int i =0; i<index-1; ++i){
                current = current->next;
        }
        node* newNode = new node;
        newNode->data = data;
        newNode->next = current->next;
        if(current->next!=NULL){
                current->next->previous = newNode;
        }
                current->next = newNode;
                newNode->prev = current;
    }
}
```

remove from the middle

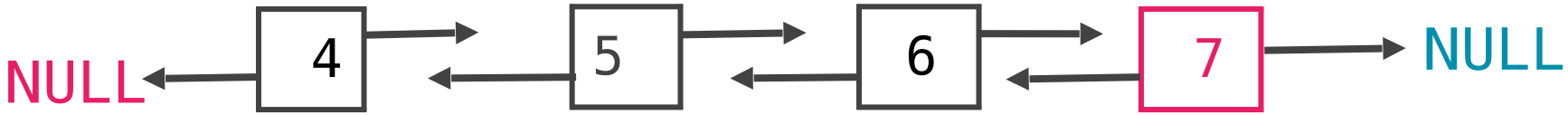


remove( )

```
void removeElement(node *&dll, int index){
        if(index ==0){
                node* junk = dll;
                dll = dll->next;
                if(dll!=NULL)
                        {dll->prev = NULL;}
                delete junk;
        }
...
```

```
...
        else{
                node* current = dll;
                for(int i =0; i<index-1; ++i){
                        current = current->next;
                }
                node *junk = current->next;
                current->next = current->next->next;

                if(current->next!=NULL)
                        current->next->prev= current;

                delete junk;
        }
}
```

remove from the end



display( )

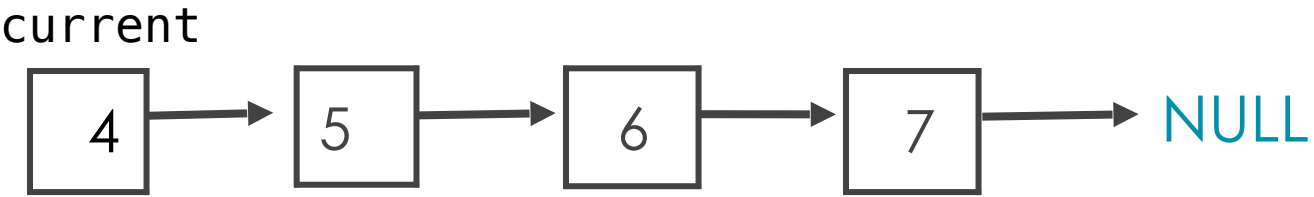
```
void display(node *&dll) {

      node* current = dll;
      while(current->next!=NULL){
            cout<<" "<< current->data<<" ";
            current = current->next;
      }

      cout<<" "<< current->data<<" ";
...
```

```
...
      cout<<"\Display in Reverse"<<endl;

      while(current->prev!=NULL){
            cout<<" "<< current->data<<" ";
            current = current->prev;
      }
      cout<<" "<< current->data<<" ";

      cout<<endl;
}
```

Practice Problems

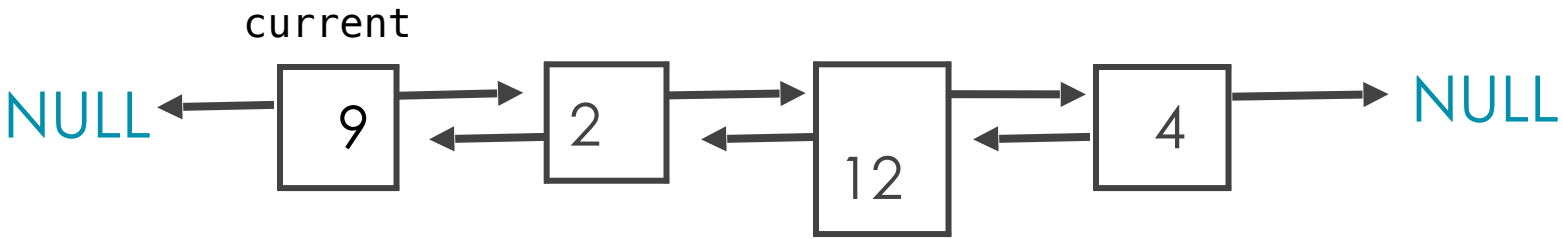
1 . Given a node defined as :

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



The nodes are connected using singly linked list. Current pointer defined as node* **current** points to the element **4.** Find the average of the elements present from **4** till the end of the list.

2.Assume you already have a Doubly linked list where current in **node* current** points to the beginning of the list. Write a function printLN() to print all the numbers that are divisible by **2** from current node to the end of the list. Then print all numbers that are divisible by **3** from the end of the list to the beginning of the list.



For above test case, output would be:

2 12 4

12 9

The struct node is defined as struct **node**{

int data;

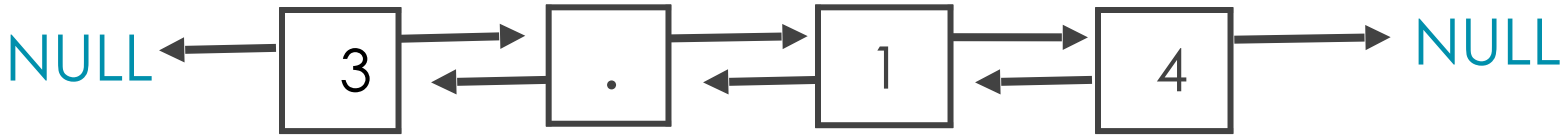
node* **next;**

node* **prev;**

}

3. Write a function for doubly linked list:

```
int printReverse(node *&list){


}
```



The function will take the list as input and return the number, that will be obtained by reversing the list.Take care of the position of decimal.

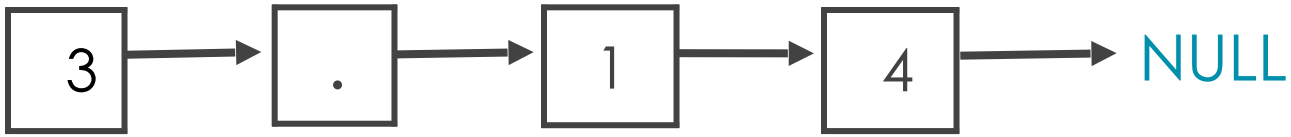
Sample output from the above list: **41.3**

The struct node is defined as

```
struct node{
        char data;
        node* next;
        node* prev;
}
```

4. Write a function for singly linked list:

```
int printReverse(node *&list){

}
```



The function will take the list as input and return the number, that will be obtained by reversing the list.Take care of the position of decimal.

Sample output from the above list: **41.3**

The struct node is defined as

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