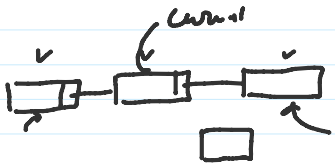
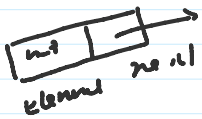
# Class on 4-2-2021

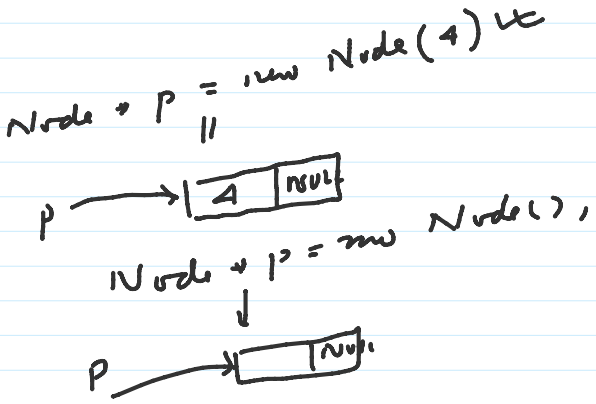
04 February 2021 12:50 PM



```
typedef int E;
class List {          // List ADT

  public:
    List() {}                          // Default constructor
    virtual ~List() {}                 // Base destructor
    virtual void clear() = 0;          // Clear contents from the list, to make it empty.
    virtual void insert(const E& item) = 0;  // Insert an element at the current location.
    virtual void append(const E& item) = 0;  // Append an element at the end of the list.
    virtual E remove() = 0;            // Remove and return the current element.
    virtual void moveToStart() = 0;    // Set the current position to the start of the list
    virtual void moveToEnd() = 0;      // Set the current position to the end of the list
    virtual void prev() = 0;           // Move the current position one step left. No change if already at beginning.
    virtual void next() = 0;           // Move the current position one step right. No change if already at end.
    virtual int length() const = 0;    // Return: The number of elements in the list.
    virtual int currPos() const = 0;   // Return: The position of the current element.
    virtual void moveToPos(int pos) = 0;  // Set current position.
    virtual E& getValue() const = 0;   // Return: The current element.
};
```



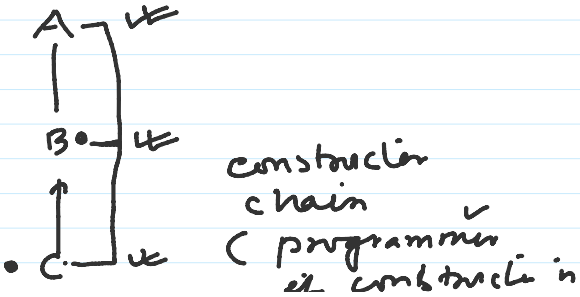
```
#include <cstddef>
using namespace std;

typedef int E;
class Node {
  public:
    E element;          // Value for this
    Node *next;         // Pointer to next  in list
                        // Constructors
    Node(const E& elemval, Node* nextval =NULL){
      element = elemval;  next = nextval;
    }
    Node(Node* nextval = NULL)
    {
      next = nextval;
    }
};
```



```
#include "ListADT.h"
#include "Node.h"

class LinkedList: public List {
  private:
    Node* head;                 // Pointer to list header
    Node* tail;                 // Pointer to last element
    Node* curr;                 // Access to current element
    int cnt;                    // Size of list

    void init() {               // Intialization helper method
      curr = tail = head = new Node();
      cnt = 0;
    }

    void removeall() {          // Return Node s to free store
      while(head != NULL) {
        curr = head;
        head = head->next;
        delete curr;
      }
    }

  public:
    LinkedList(int size=0) { init(); }          // Constructor
    ~LinkedList() { removeall(); }              // Destructor
    void print() const;                         // Print list contents
    void clear() { removeall(); init(); }       // Clear list


    void insert(const E& it) {                  // Insert "it" at current position
```

```
  curr->next = new Node(it, curr->next);
  if (tail == curr) tail = curr->next;  // New tail
  cnt++;
 }

 void append(const E& it) {                    // Append "it" to list
  tail = tail->next = new Node(it, NULL);
  cnt++;
 }


 E remove() {                                  // Remove and return current element

  E it = curr->next->element;                  // Remember value
  Node* ltemp = curr->next;                    // Remember Node
  if (tail == curr->next) tail = curr;         // Reset tail
  curr->next = curr->next->next;               // Remove from list
  delete ltemp;                                // Reclaim space
  cnt--;                                       // Decrement the count
  return it;
 }

 void moveToStart()                            // Place curr at list start
  { curr = head; }

 void moveToEnd()                              // Place curr at list end
  { curr = tail; }


 void prev() {                                 // Move curr one step left; no change if already at front
  if (curr == head) return;                    // No previous element
  Node* temp = head;
                                               // March down list until we find the previous element
  while (temp->next!=curr) temp=temp->next;
  curr = temp;
 }


 void next()                                   // Move curr one step right; no change if already at end
  { if (curr != tail) curr = curr->next; }

 int length() const  { return cnt; } // Return length


 int currPos() const {                         // Return the position of the current element
  Node* temp = head;
  int i;
  for (i=0; curr != temp; i++)
    temp = temp->next;
  return i;
 }


 void moveToPos(int pos) {                     // Move down list to "pos" position

  curr = head;
  for(int i=0; i<pos; i++) curr = curr->next;
 }

 E& getValue() const {                         // Return current element

  return curr->next->element;
 }
};
```

```
#include <iostream>
using namespace std;
#include "LList.h"
typedef int E;
class Queue :private LinkedList{
 public:
   Queue( int size =0);
   ~Queue(){};
   E & front() ;
   E & back() ;
   int size() ;
   void insert( const E & thisElement);
   E remove();
   void  clearALL();
   void display();
```



```
    void  clearALL();
    void display();

};
Queue::Queue( int size):LinkedList(0){ cout << "Queue is created"<<endl;}

E & Queue::front()
{
  LinkedList::moveToStart();
  return LinkedList::getValue();

}
E & Queue::back(){
  LinkedList::moveToEnd();
  return LinkedList::getValue();

}
int Queue::size(){

  return LinkedList::length();
}
void Queue::insert(const E & thisElement){
  LinkedList::append(thisElement);
}
E Queue::remove(){
  LinkedList::moveToStart();
  return LinkedList::remove();

}
void Queue::display(){
  cout<<"The elements in the queue are :";
  int length=size();
  LinkedList::moveToStart();
  for(int i= 1; i<= length; i++){

    cout<<LinkedList::getValue()<<" ";
    LinkedList::next();

  }
  cout<<endl;


}
void  Queue::clearALL(){

   LinkedList::clear();

}
```

chain (programmer if constructor in parameters used)

Const

assignment  initialization

moveToStart();

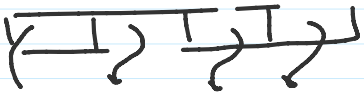
delegation

construct ( ) : List() {

" assign.

}

Len: 4  i: 1, 2, 3, 4



OO principle: Inheritance.

public  protected  private

Queue

LinkedList

array

previous lecture

ListADT <<Abstract>>

↑ public

LinkedList

↑ private

Queue

private:

→ all public methods of LinkedList

public:

- Queue operation

back  size
front  empty
insert  clearALL
remove  display

Super Class
private members
protected "
public "

Child Class (private)
private memb.
protected "
public "

Queue q;
q.front();
q.back();
q.insert(2);
q.remove()
Member fun nesting