

LAB 07

Miss Atika Aslam

Muhammad Abdullah

53457

### Lab Task 1:

Assume that a singly linked list is implemented with a header node, but no tail node, and that it maintains only a pointer to the header node.

Write a class that includes methods to

* Return the size of the linked list (total number of nodes in list)
* Print the linked list
* Test if a value x is contained in the linked list

Add a value x if it is not already contained in the linked list e. remove a value x if it is contained in the linked list

**Code:**

#*include* <iostream>

using namespace std;

class SinglyLinkedList {

private:

    struct Node {

        int data;

        Node\* next;

*Node*(int value = 0, Node*\** nextNode = nullptr) : *data*(value), *next*(nextNode) {}

    };

    Node\* header;// *Header node*

public:

*SinglyLinkedList*() {

        header = new *Node*();// *Initialize header node*

    }

    int *size*()  {

        int count = 0;

        Node\* current = header->next;

*while* (current != nullptr) {

            count++;

            current = current->next;

        }

*return* count;

    }

    void *print*() {

        Node\* current = header->next;

*while* (current != nullptr) {

            cout *<<* current->data *<<* " ";

            current = current->next;

        }

        cout *<<* *endl*;

    }

    bool *contains*(int x)  {

        Node\* current = header->next;

*while* (current != nullptr) {

*if* (current->data == x) {

*return* true;

            }

            current = current->next;

        }

*return* false;

    }

    void *add*(int x) {

            Node\* newNode = new *Node*(x, header->next);

            header->next = newNode;

    }

    void *remove*(int x) {

        Node\* current = header;

*while* (current->next != nullptr) {

*if* (current->*next*->data == x) {

                Node\* toDelete = current->next;

                current->next = toDelete->next;

                delete toDelete;

*return*;

            }

            current = current->next;

        }

    }

};

// *Main function for testing*

int *main*() {

    SinglyLinkedList list;

// *Adding elements*

    list.*add*(10);

    list.*add*(20);

    list.*add*(30);

// *Print list*

    cout *<<* "List: ";

    list.*print*();

// *Size of the list*

    cout *<<* "Size: " *<<* list.*size*() *<<* *endl*;

// *Check if an element exists*

    cout *<<* "Contains 20: " *<<* (list.*contains*(20) ? "Yes" : "No") *<<* *endl*;

// *Remove an element*

    list.*remove*(20);

    cout *<<* "List after removing 20: ";

    list.*print*();

// *Add an existing element*

    list.*add*(100);

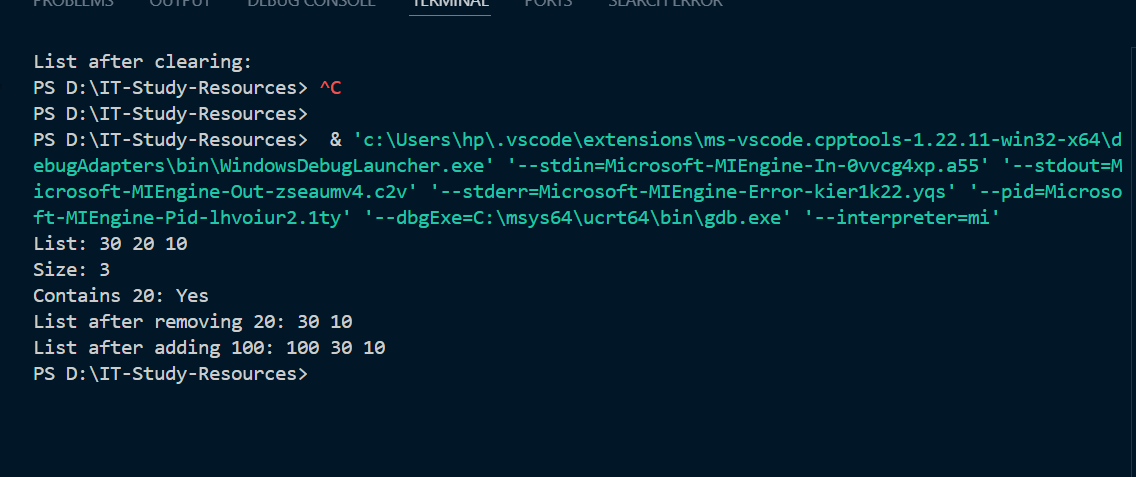
    cout *<<* "List after adding 100: ";

    list.*print*();

*return* 0;

}

## Output



**Lab Task 2:**

Write a class named Mobile that holds data about a mobile in a retail store. The class should have the following member variables:

• Name/brand. A string that holds name of the brand of phone.

• unitsOnHand. An int that holds the number of units currently in inventory.

• price. A double that holds the retail price.

Write a constructor that accepts arguments for each member variable, appropriate functions that store values in these member variables, and accessor functions that return the values in these member variables.

Once you have written the class,

Write another class Store (link list) having Mobile as a node then provide function to insert new mobile in list and to Delete a mobile and to display all mobiles.

Write a main program that shows all the functionality.

Note: you have to implement all scenarios (insert and delete at start, end, in between).

**Code:**

#*include* <iostream>

#*include* <string>

using namespace std;

// *Mobile class*

class Mobile {

private:

    string brand;// *Brand name*

    int unitsOnHand;// *Units in inventory*

    double price;// *Retail price*

public:

// *Constructor*

*Mobile*(string b = "", int u = 0, double p = 0.0) : *brand*(b), *unitsOnHand*(u), *price*(p) {}

    void *setBrand*(string b) { brand *=* b; }

    void *setUnitsOnHand*(int u) { unitsOnHand = u; }

    void *setPrice*(double p) { price = p; }

    string *getBrand*()  { *return* brand; }

    int *getUnitsOnHand*()  { *return* unitsOnHand; }

    double *getPrice*()  { *return* price; }

    void *display*()  {

        cout *<<* "Brand: " *<<* brand *<<endl*;

        cout*<<*"Units: " *<<* unitsOnHand *<<* *endl*;

        cout*<<*"Price: $" *<<* price *<<* *endl*;

    }

};

class Store {

private:

    struct Node {

        Mobile data;// *Mobile object*

        Node\* next;// *Pointer to next node*

*Node*(Mobile m, Node*\** n = nullptr) : *data*(m), *next*(n) {}

    };

    Node\* head;// *Pointer to the start of the list*

public:

*Store*() : *head*(nullptr) {}

*~Store*() {

*clear*();

    }

// *Insert a new mobile at the start*

    void *insertAtStart*(*const* Mobile*&* mobile) {

        head = new *Node*(mobile, head);

    }

// *Insert a new mobile at the end*

    void *insertAtEnd*(*const* Mobile*&* mobile) {

*if* (!head) {

            head = new *Node*(mobile);

        } *else* {

            Node\* temp = head;

*while* (temp->next) {

                temp = temp->next;

            }

            temp->next = new *Node*(mobile);

        }

    }

    void *insertAfter*(int position,Mobile*&* mobile) {

*if* (!head || position < 0) {

            cout *<<* "Invalid position!" *<<* *endl*;

*return*;

        }

        Node\* temp = head;

*for* (int i = 0; i < position; i++) {

*if* (!temp->next) {

                cout *<<* "Position out of range!" *<<* *endl*;

*return*;

            }

            temp = temp->next;

        }

        temp->next = new *Node*(mobile, temp->next);

    }

    void *deleteFromStart*() {

*if* (!head) {

            cout *<<* "List is empty!" *<<* *endl*;

*return*;

        }

        Node\* temp = head;

        head = head->next;

        delete temp;

    }

    void *deleteFromEnd*() {

*if* (!head) {

            cout *<<* "List is empty!" *<<* *endl*;

*return*;

        }

*if* (!head->next) {

            delete head;

            head = nullptr;

*return*;

        }

        Node\* temp = head;

*while* (temp->*next*->next) {

            temp = temp->next;

        }

        delete temp->next;

        temp->next = nullptr;

    }

    void *deleteFromPosition*(int position) {

*if* (!head || position < 0) {

            cout *<<* "Invalid position!" *<<* *endl*;

*return*;

        }

*if* (position == 0) {

*deleteFromStart*();

*return*;

        }

        Node\* temp = head;

*for* (int i = 0; i < position - 1; i++) {

*if* (!temp->next || !temp->*next*->next) {

                cout *<<* "Position out of range!" *<<* *endl*;

*return*;

            }

            temp = temp->next;

        }

        Node\* toDelete = temp->next;

        temp->next = toDelete->next;

        delete toDelete;

    }

    void *display*() *const* {

*if* (!head) {

            cout *<<* "Store is empty!" *<<* *endl*;

*return*;

        }

        Node\* temp = head;

*while* (temp) {

            temp->*data*.*display*();

            temp = temp->next;

        }

    }

    void *clear*() {

*while* (head) {

*deleteFromStart*();

        }

    }

};

int *main*() {

    Store store;

    Mobile *m1*("Samsung", 50, 799.99);

    Mobile *m2*("Apple", 30, 999.99);

    Mobile *m3*("OnePlus", 40, 699.99);

    store.*insertAtStart*(m1);

    cout *<<* "After inserting at start:" *<<* *endl*;

    store.*display*();

// *Insert at end*

    store.*insertAtEnd*(m2);

    cout *<<* "\nAfter inserting at end:" *<<* *endl*;

    store.*display*();

// *Insert after position 0*

    store.*insertAfter*(0, m3);

    cout *<<* "\nAfter inserting after position 0:" *<<* *endl*;

    store.*display*();

    store.*deleteFromStart*();

    cout *<<* "\nAfter deleting from start:" *<<* *endl*;

    store.*display*();

    store.*deleteFromEnd*();

    cout *<<* "\nAfter deleting from end:" *<<* *endl*;

    store.*display*();

    store.*insertAtEnd*(m1);

    store.*insertAtEnd*(m3);

    cout *<<* "\nAfter Inserting At end again:" *<<* *endl*;

    store.*display*();

    store.*deleteFromPosition*(1);

    cout *<<* "\nAfter deleting from position 1:" *<<* *endl*;

    store.*display*();

*return* 0;

}

***OUTPUT***

