C++

Assignment – 9

Name – Kishan R Vaghamashi

Student ID – 202312014

1)

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
using namespace std;
template <typename T>
struct LinkedList
{
    struct Node
    {
        T data;
        unique_ptr<Node> next = nullptr;
        Node(T data)
        {
            this->data = data;
        }
        static auto get_new_node(T data)
        {
            return make_unique<Node>(data);
        }
    };
    unique_ptr<Node> head = nullptr;
    LinkedList() {}
    LinkedList(T data)
    {
        insert(data);
    }
    void print()
    {
        if (head == nullptr)
            return;
        auto traversal = head.get();
        while (true)
        {
            cout << traversal->data << " -> ";
            if (traversal->next != nullptr)
                traversal = traversal->next.get();
            else
                break;
        }
        cout << " NULL " << endl;
    }
    void push(T data)
    {
        auto new_node = Node::get_new_node(data);
```

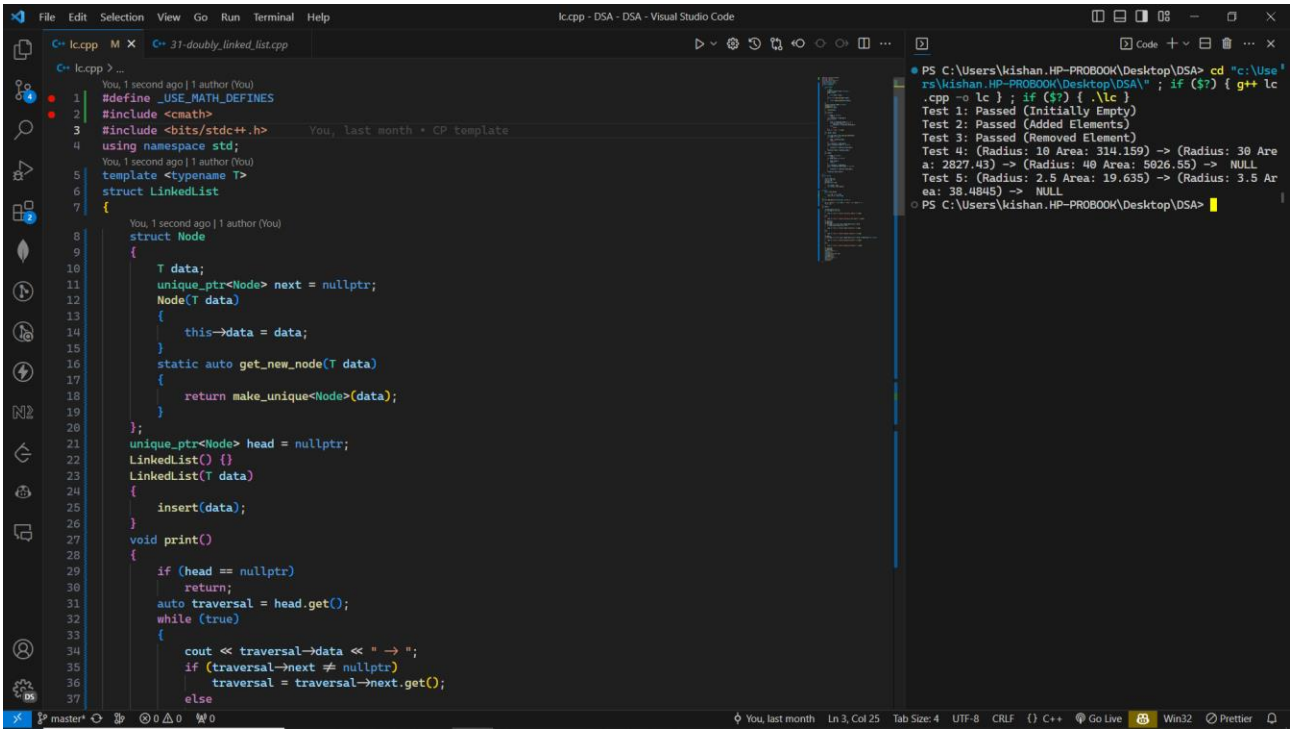
```
        if (head == nullptr)
        {
            head = move(new_node);
            return;
        }
        auto traversal = head.get();
        while (traversal->next != nullptr)
        {
            traversal = traversal->next.get();
        }
        traversal->next = move(new_node);
    }
    void pop()
    {
        if (head == nullptr)
            return;
        if (head->next == nullptr)
        {
            head.reset();
            return;
        }
        auto traversal = head.get();
        while (traversal->next->next != nullptr)
        {
            traversal = traversal->next.get();
        }
        traversal->next.reset();
    }
};
struct Circle
{
    double rad = 0;
    double area = 0;
    Circle() {}
    Circle(double rad)
    {
        this->rad = rad;
        this->area = calc_area();
    }

private:
    double calc_area()
    {
        auto rad = this->rad;
        return M_PI * rad * rad;
```

```
    }
};
ostream &operator<<(ostream &os, Circle c)
{
    os << "(Radius: " << c.rad << " Area: " << c.area << ")";
    return os;
}
int main()
{
    LinkedList<Circle> ll;
    if (ll.head == nullptr)
    {
        cout << "Test 1: Passed (Initially Empty)" << endl;
    }
    else
    {
        cout << "Test 1: Failed (Initially Not Empty)" << endl;
    }
    ll.push(10);
    ll.push(20);
    if (ll.head != nullptr && ll.head->data.rad == 10 &&
        ll.head->next->data.rad == 20)
    {
        cout << "Test 2: Passed (Added Elements)" << endl;
    }
    else
    {
        cout << "Test 2: Failed (Adding Elements)" << endl;
    }
    ll.pop();
    if (ll.head != nullptr && ll.head->data.rad == 10 && ll.head->next ==
nullptr)
    {
        cout << "Test 3: Passed (Removed Element)" << endl;
    }
    else
    {
        cout << "Test 3: Failed (Removing Element)" << endl;
    }
    ll.push(30);
    ll.push(40);
    cout << "Test 4: ";
    ll.print();
    LinkedList<Circle> ll2;
    ll2.push(2.5);
```

```
    ll2.push(3.5);
    cout << "Test 5: ";
    ll2.print();
    return 0;
}
```

Output:



2)

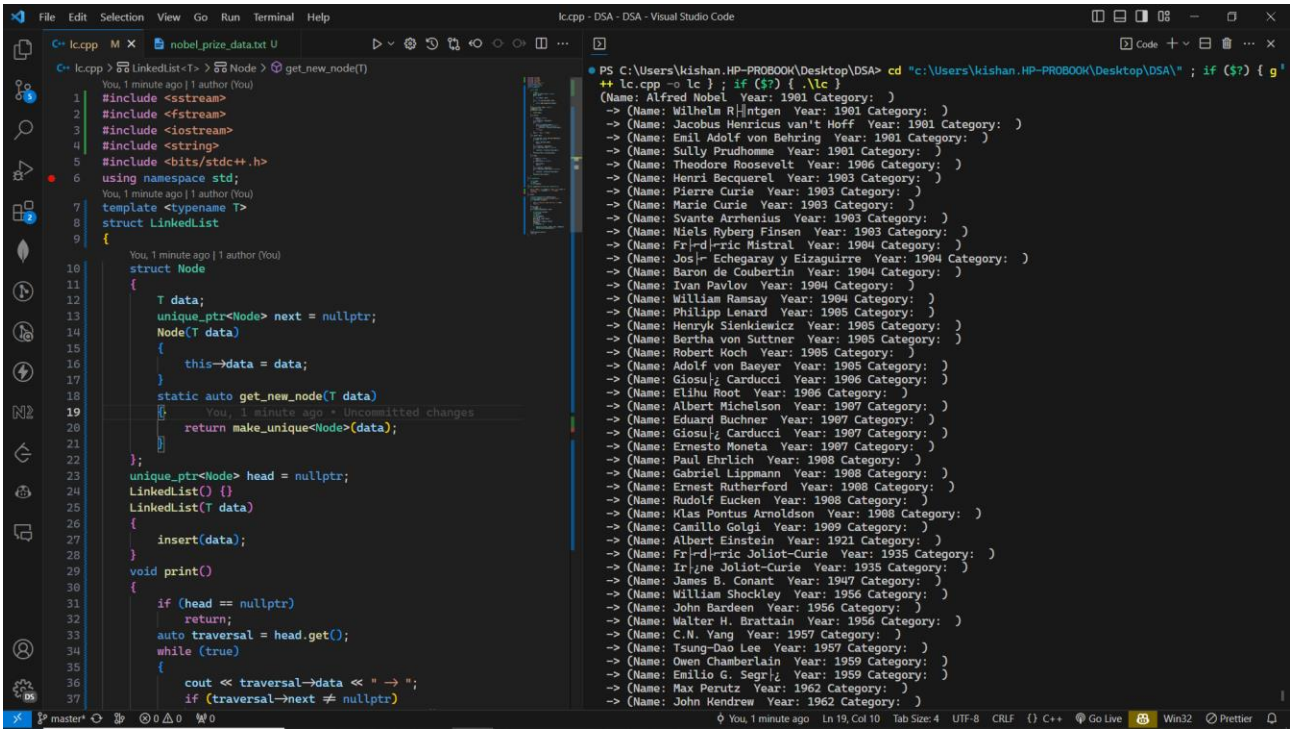
```
#include <sstream>
#include <fstream>
#include <iostream>
#include <string>
#include <bits/stdc++.h>
using namespace std;
template <typename T>
struct LinkedList
{
    struct Node
    {
        T data;
        unique_ptr<Node> next = nullptr;
        Node(T data)
        {
            this->data = data;
        }
```

```
        static auto get_new_node(T data)
        {
            return make_unique<Node>(data);
        }
    };
    unique_ptr<Node> head = nullptr;
    LinkedList() {}
    LinkedList(T data)
    {
        insert(data);
    }
    void print()
    {
        if (head == nullptr)
            return;
        auto traversal = head.get();
        while (true)
        {
            cout << traversal->data << " -> ";
            if (traversal->next != nullptr)
                traversal = traversal->next.get();
            else
                break;
        }
        cout << " NULL " << endl;
    }
    void push(T data)
    {
        auto new_node = Node::get_new_node(data);
        if (head == nullptr)
        {
            head = move(new_node);
            return;
        }
        auto traversal = head.get();
        while (traversal->next != nullptr)
        {
            traversal = traversal->next.get();
        }
        traversal->next = move(new_node);
    }
    void pop()
    {
        if (head == nullptr)
            return;
```

```
        if (head->next == nullptr)
        {
            head.reset();
            return;
        }
        auto traversal = head.get();
        while (traversal->next->next != nullptr)
        {
            traversal = traversal->next.get();
        }
        traversal->next.reset();
    }
};
struct NobelPrize
{
    string name;
    int year;
    string category;
};
ostream &operator<<(ostream &os, NobelPrize n)
{
    os << "(Name: " << n.name << " Year: " << n.year << " Category: " <<
n.category << ")" << endl;
    return os;
}
int main()
{
    LinkedList<NobelPrize> nobelPrizeList;
    ifstream inputFile("nobel_prize_data.txt");
    if (!inputFile.is_open())
    {
        cerr << "Failed to open the file." << endl;
        return 1;
    }
    string line;
    int lineNum = 0;
    while (getline(inputFile, line))
    {
        stringstream ss(line);
        string name;
        int year;
        string category;
        char delim = '|';
        getline(ss, name, delim);
        ss >> year;
```

```
        getline(ss, category, delim);
        lineNum++;
        if (lineNum > 2)
        {
            NobelPrize prize = {name, year, category};
            nobelPrizeList.push(prize);
        }
    }
    nobelPrizeList.print();
    return 0;
}
```

Output:



3)

```
#include <sstream>
#include <fstream>
#include <bits/stdc++.h>
#include <string>
#include <functional>
using namespace std;
template <typename T>
struct LinkedList
{
    struct Node
    {
```

```
        T data;
        unique_ptr<Node> next = nullptr;
        Node(T data)
        {
            this->data = data;
        }
        static auto get_new_node(T data)
        {
            return make_unique<Node>(data);
        }
    };
    unique_ptr<Node> head = nullptr;
    LinkedList() {}
    LinkedList(T data)
    {
        insert(data);
    }
    void print()
    {
        if (head == nullptr)
            return;
        auto traversal = head.get();
        while (true)
        {
            cout << traversal->data << " -> ";
            if (traversal->next != nullptr)
                traversal = traversal->next.get();
            else
                break;
        }
        cout << " NULL " << endl;
    }
    void push(T data)
    {
        auto new_node = Node::get_new_node(data);
        if (head == nullptr)
        {
            head = move(new_node);
            return;
        }
        auto traversal = head.get();
        while (traversal->next != nullptr)
        {
            traversal = traversal->next.get();
        }
```

```
        traversal->next = move(new_node);
    }
    void pop()
    {
        if (head == nullptr)
            return;
        if (head->next == nullptr)
        {
            head.reset();
            return;
        }
        auto traversal = head.get();
        while (traversal->next->next != nullptr)
        {
            traversal = traversal->next.get();
        }
        traversal->next.reset();
    }
};
struct NobelPrize
{
    string name;
    int year;
    string category;
};
bool CompareNobelPrizesByYear(const NobelPrize &a, const NobelPrize &b)
{
    return a.year >= b.year;
}
ostream &operator<<(ostream &os, NobelPrize n)
{
    os << "(Name: " << n.name << " Year: " << n.year << " Category: " <<
n.category << ")" << endl;
    return os;
}
void insertIntoSorted(LinkedList<NobelPrize> &list, const NobelPrize &prize,
                      std::function<bool(const NobelPrize &, const NobelPrize
&)> compareFunction)
{
    auto new_node = LinkedList<NobelPrize>::Node::get_new_node(prize);
    if (list.head == nullptr || compareFunction(prize, list.head->data))
    {
        new_node->next = move(list.head);
        list.head = move(new_node);
    }
```

```
    else
    {
        auto prevSorted = list.head.get();
        auto current = list.head.get();
        while (current != nullptr && compareFunction(current->data,
                                                     prize))
        {
            prevSorted = current;
            current = current->next.get();
        }
        new_node->next = move(prevSorted->next);
        prevSorted->next = move(new_node);
    }
}
int main()
{
    LinkedList<NobelPrize> nobelPrizeList;
    LinkedList<NobelPrize> sortedNobelPrizeList;
    auto SortNobelPrizes = [&](std::function<bool(const NobelPrize &, const
NobelPrize &)> compareFunction)
    {
        sortedNobelPrizeList.head.reset();
        auto current = nobelPrizeList.head.get();
        while (current)
        {
            auto next = current->next.get();
            insertIntoSorted(sortedNobelPrizeList, current->data,
                             compareFunction);
            current = next;
        }
    };
    ifstream inputFile("nobel_prize_data.txt");
    if (!inputFile.is_open())
    {
        cerr << "Failed to open the file." << endl;
        return 1;
    }
    string line;
    int lineNum = 0;
    while (getline(inputFile, line))
    {
        stringstream ss(line);
        string name;
        int year;
        string category;
```

```
        char delim = '|';
        getline(ss, name, delim);
        ss >> year;
        getline(ss, category, delim);
        lineNum++;
        if (lineNum > 2)
        {
            NobelPrize prize = {name, year, category};
            nobelPrizeList.push(prize);
        }
    }
    SortNobelPrizes(CompareNobelPrizesByYear);
    sortedNobelPrizeList.print();
    return 0;
}
```

Output: