Lab 2

Session: 2021 – 2024

Submitted by:

Uswa Arif

2021-CS-77

Submitted To:

Laeeq Khan Niazi

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

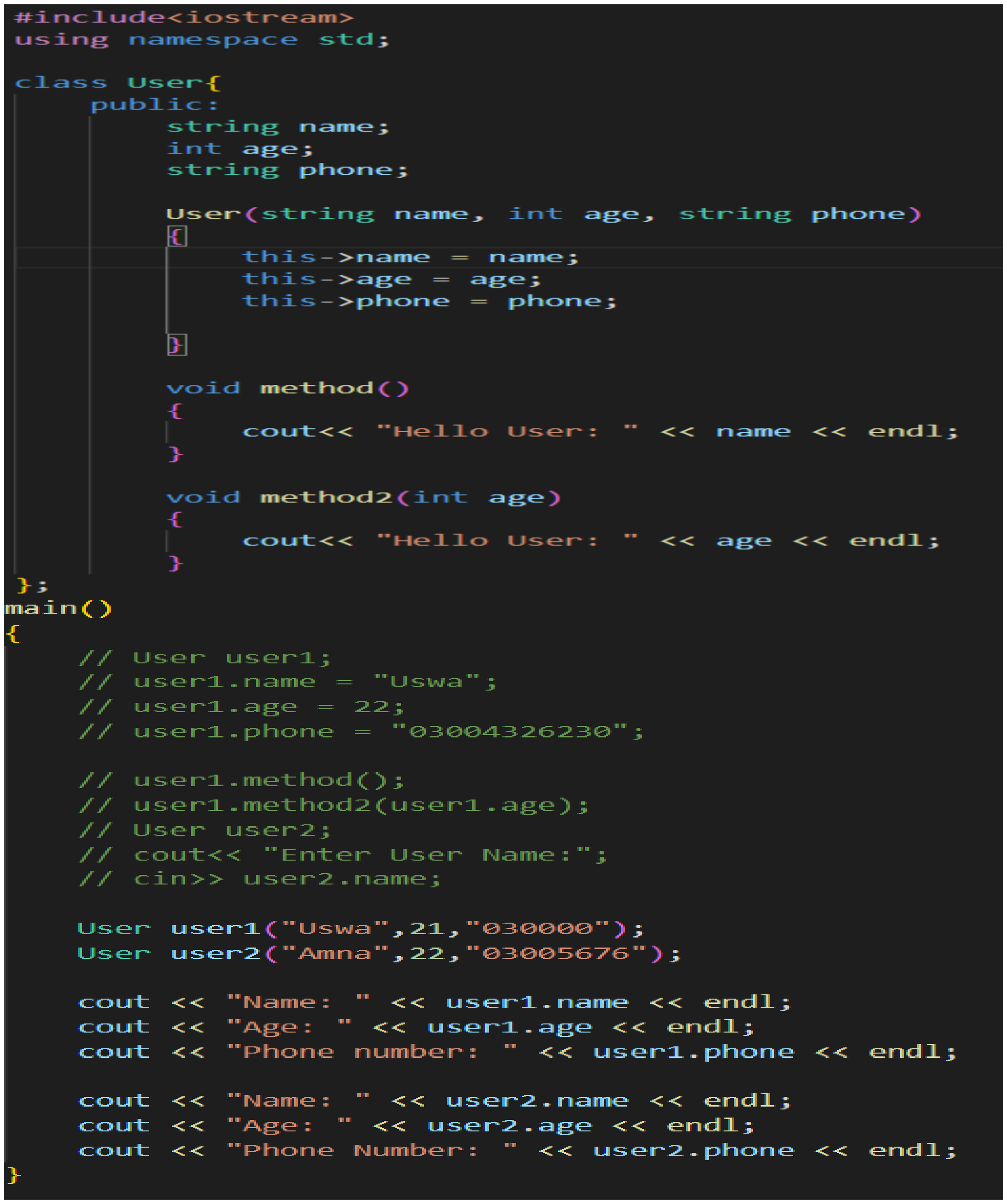
Classes:

C++ is an object-oriented programming language means that classes and objects methods can be defined in c++. In classes, attributes and methods can be defined using the syntax that language includes. Attributes are like data-type variables and Methods are like functions defined in C++.

For Example, Laptop is an Object and laptop has attributes color, model and methods can be features.

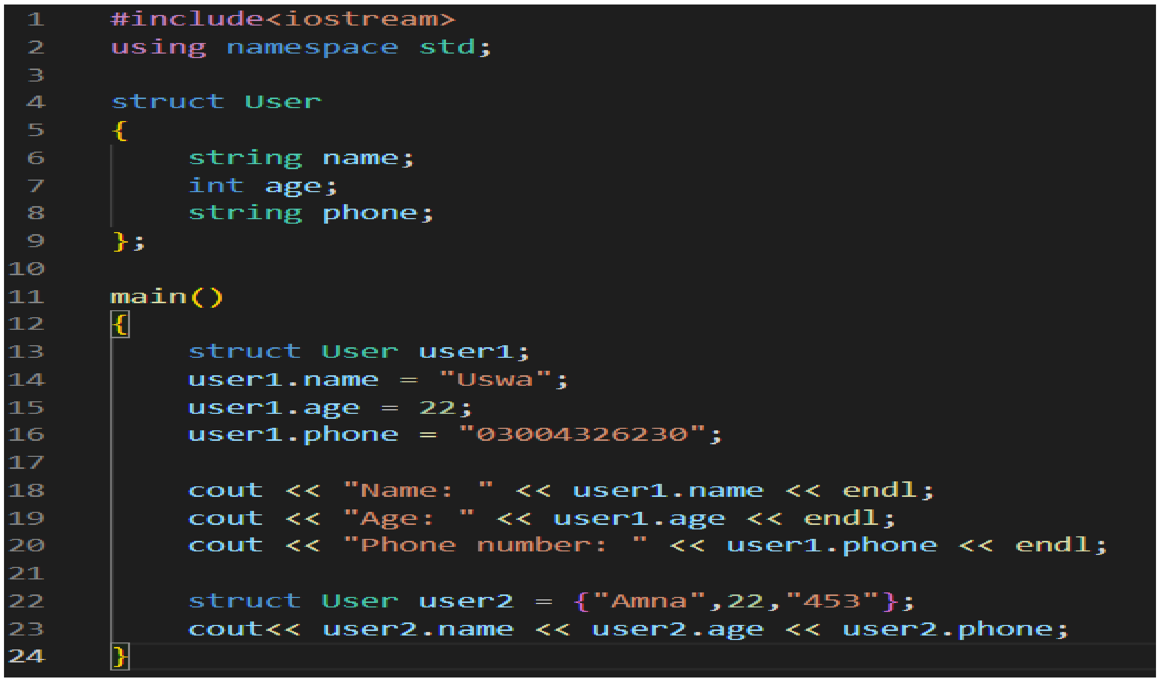
Implementation:

Method to create the class in C++ is:



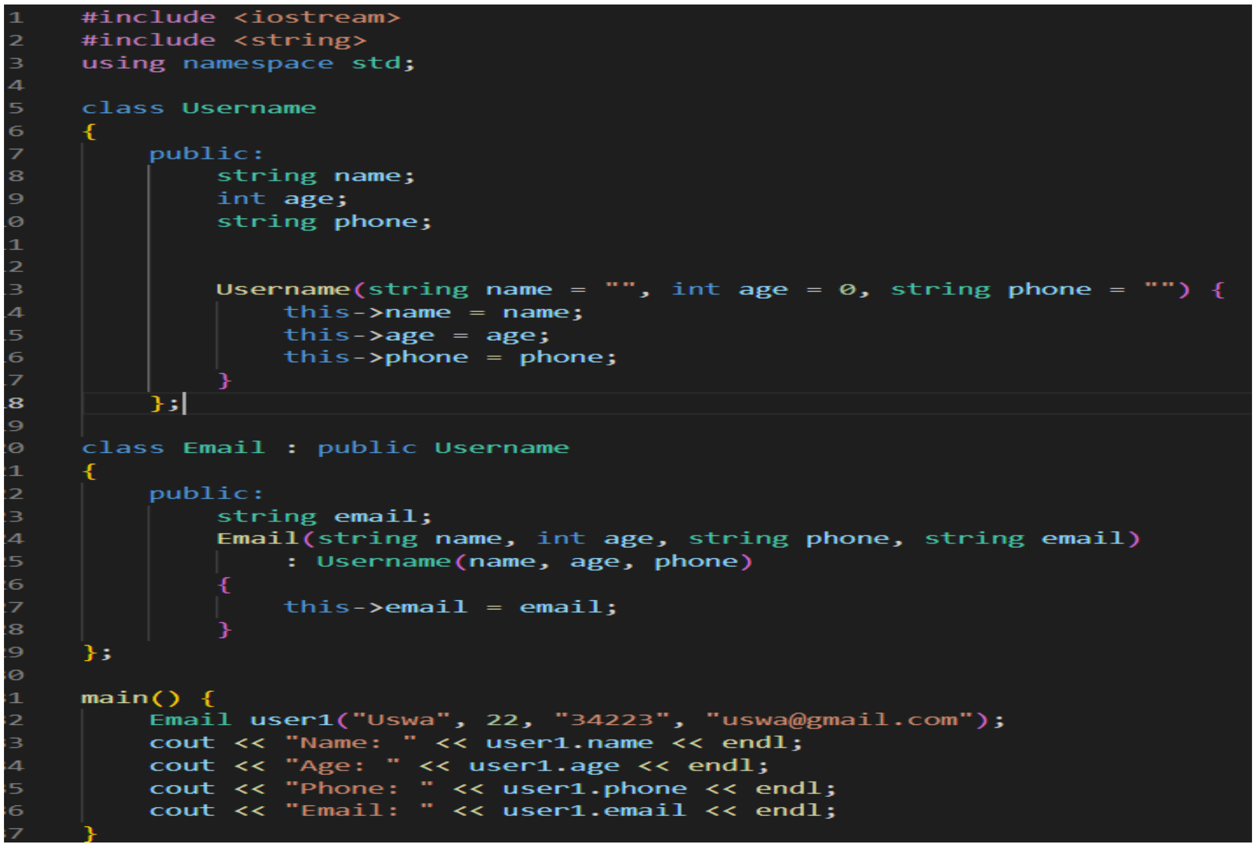
```
#include<iostream>
using namespace std;

class User{
    public:
        string name;
        int age;
        string phone;

        User(string name, int age, string phone)
        {
            this->name = name;
            this->age = age;
            this->phone = phone;

        }

        void method()
        {
            cout<< "Hello User: " << name << endl;
        }

        void method2(int age)
        {
            cout<< "Hello User: " << age << endl;
        }
};
main()
{
    // User user1;
    // user1.name = "Uswa";
    // user1.age = 22;
    // user1.phone = "03004326230";

    // user1.method();
    // user1.method2(user1.age);
    // User user2;
    // cout<< "Enter User Name:";
    // cin>> user2.name;

    User user1("Uswa",21,"030000");
    User user2("Amna",22,"03005676");

    cout << "Name: " << user1.name << endl;
    cout << "Age: " << user1.age << endl;
    cout << "Phone number: " << user1.phone << endl;

    cout << "Name: " << user2.name << endl;
    cout << "Age: " << user2.age << endl;
    cout << "Phone Number: " << user2.phone << endl;
}
```

Structure:



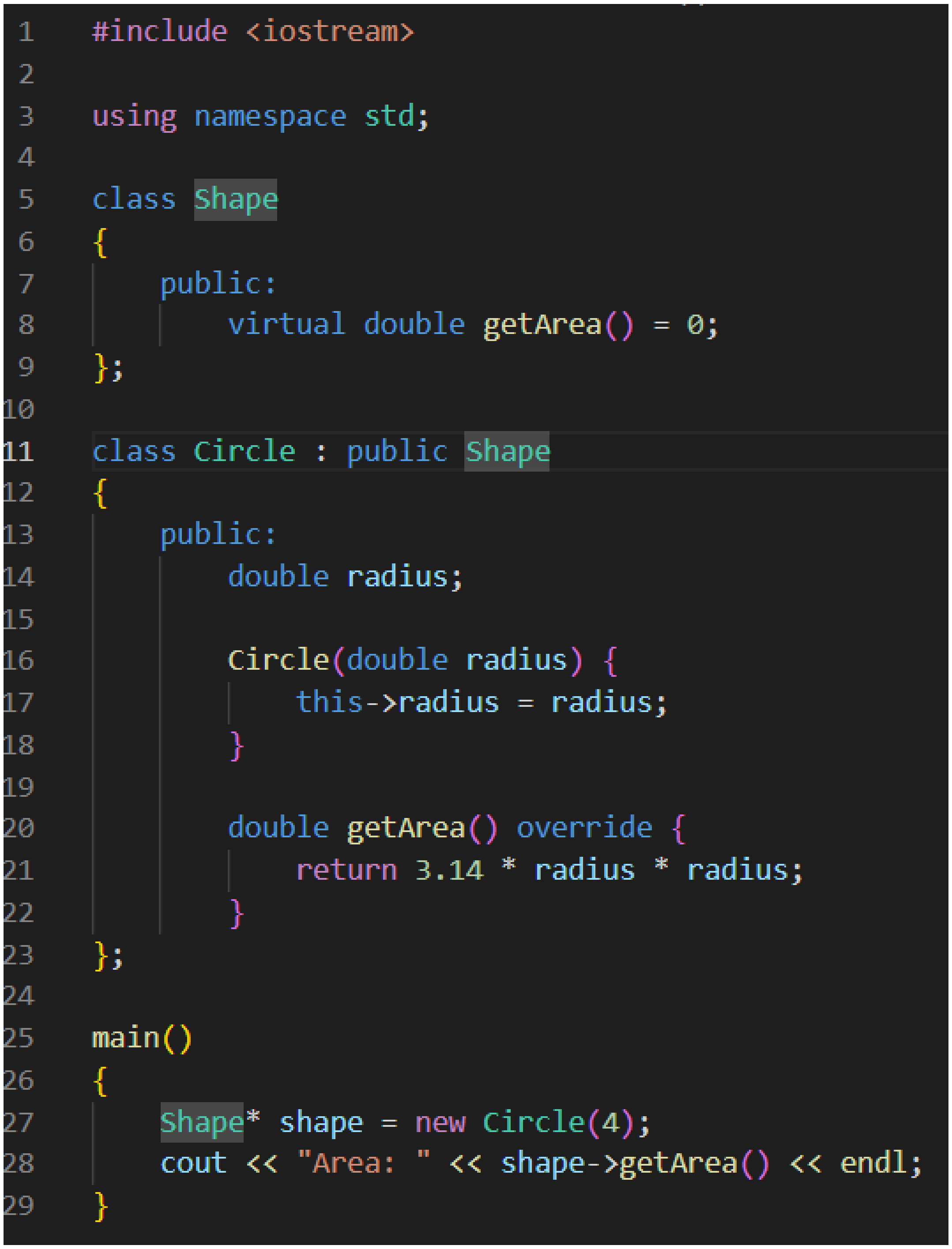
```
#include<iostream>
using namespace std;

struct User
{
    string name;
    int age;
    string phone;
};

main()
{
    struct User user1;
    user1.name = "Uswa";
    user1.age = 22;
    user1.phone = "03004326230";

    cout << "Name: " << user1.name << endl;
    cout << "Age: " << user1.age << endl;
    cout << "Phone number: " << user1.phone << endl;

    struct User user2 = {"Amna",22,"453"};
    cout<< user2.name << user2.age << user2.phone;
}
```

Inheritance:



```
#include <iostream>
#include <string>
using namespace std;

class Username
{
    public:
        string name;
        int age;
        string phone;


        Username(string name = "", int age = 0, string phone = "") {
            this->name = name;
            this->age = age;
            this->phone = phone;
        }
    };

class Email : public Username
{
    public:
        string email;
        Email(string name, int age, string phone, string email)
            : Username(name, age, phone)
        {
            this->email = email;
        }
};

main() {
    Email user1("Uswa", 22, "34223", "uswa@gmail.com");
    cout << "Name: " << user1.name << endl;
    cout << "Age: " << user1.age << endl;
    cout << "Phone: " << user1.phone << endl;
    cout << "Email: " << user1.email << endl;
}
```

Virtual Function:



```cpp
#include <iostream>

using namespace std;

class Shape
{
    public:
        virtual double getArea() = 0;
};

class Circle : public Shape
{
    public:
        double radius;

        Circle(double radius) {
            this->radius = radius;
        }

        double getArea() override {
            return 3.14 * radius * radius;
        }
};

main()
{
    Shape* shape = new Circle(4);
    cout << "Area: " << shape->getArea() << endl;
}
```

STL Container:

1- Vectors:

Vectors are the resizable array. The size of an array cannot be increased or decreased. A fixed size is given to the array. Vectors are similar to arrays but we can resize the arrays in vectors. Vectors have dynamic sizes. Vectors store same data type of elements throughout the array. Vectors use less memory.
Vector is included in header file as #include<vector>;

Time Complexity:

Insertion:

Worst time: 0(n)
Best time(at the end): 0(1)

Deletion:

Worst time: 0(n)
Best time(at the end): 0(1)

Vector Declaration:

Vector<datatype> array_name = {} ;

Vector Important Function:

- Push_back: This function inserts a variable in the end of the array. And hence time complexity of best time is 0(1).
- Pop_back: This function deletes a variable in the end of the array. And hence time complexity of best time is 0(1).

2- Lists:

Lists can store elements of the same datatype and the size of list can also grow in size. List can also store elements on different location of memory but relate each elements but using pointers like prev, next. The previous pointer has memory location of the previous node and next pointer has memory location of the next Node and hence, nodes can relate to each other present on different location of memory. Memory is non-contiguous.
List is included in header file as #include<list >;

Time Complexity:

Insertion:

Worst time: 0(n)
Best time(at the start, end): 0(1)

Deletion:

Worst time: 0(n)
Best time(at the start, end): 0(1)

List Declaration:

list<datatype> list_name = {value1, value2} ;

List Important Function:

Push_front, pop_front, push_back, pop_back.

Difference Between List and Vector:

List can apply insertion and deletion at beginning and end but vector can only apply at end. List cannot have random access of index of element but vectors can access through the index.

3- Deque:

Deque are also called Double Ended Queue. They are similar to queue data structure. But in queue, elements are removed from the front but added at the back. In deque, elements can be added at the front and back, Similarly, elements can be deleted at the front and back. Memory is non-contiguous.
Deque is included in header file as #include<deque >;
Time Complexity:
Insertion:
Worst time: 0(n)
Best time(at the start, end): 0(1)

Deletion:

Worst time: 0(n)
Best time(at the start, end): 0(1)

List Declaration:

deque<datatype> name ;

Difference Between Deque, List and Vector:

Deque can apply random access and list cannot have random access. Vectors have contiguous memory and deque has noncontiguous memory.

4- Stack:

Stack is the data structure that evolve the order of LIFO (Last in First out). Elements are inserted in the last (push) and delete from the front (pop). Elements are not accessed by index. Memory is contiguous in stack if we use array implementation and memory is non-contiguous if we use list implementation.
Stack is included in header file as #include<stack >;
Time Complexity:
Insertion:
Best time: 0(1)

Deletion:

Best time: 0(1)

List Declaration:

stack<datatype> name ;

Difference Between Stack, Deque, List and Vector:

Deque, vectors can apply random access and stack cannot have random access. Stack is faster than others.

5- Queue:

Queue is the data structure that operates on order of FIFO (First in First out). In queue, the elements that are added first will be removed first. The methods that are used in queue are enqueue and dequeue. The elements will be added at one end and removed from the other end. Memory is contiguous if we use array implementation and memory is non-contiguous if we use list implementation.
Queue is included in header file as #include<queue >;
Time Complexity:
Insertion:
Best time: 0(1)

Deletion:

Best time: 0(1)

List Declaration:

queue<datatype> name ;

Difference Between Queue, Stack, Deque, List and Vector:

Queue works on FIFO and stack on LIFO. Deque apply insertion and deletion from both end that is the difference from queue. Queue differs from list and vector due to FIFO operation.

6- Priority Queue:

Priority Queue is the data structure that works on the queue operation FIFO but it also applies a priority on each element. The priority is like that element will be in ascending order or descending order. By default, priority queue uses descending order. Like if there is highest value in elements then it will be pop first. Memory is contiguous if we use array implementation and memory is non-contiguous if we use list implementation.
Priority Queue is included in header file as #include<queue >;
Time Complexity:
Insertion:
Best time: 0(log n)

Deletion:

Best time: 0(log n)

List Declaration:

Priority_queue<datatype> name ;

7- Set:

Sets are the unique elements that have same data-types and are in sorted manner. In sets, elements can be added or removed but we cannot change the values of elements. Memory in sets can be contiguous if implemented with arrays and non-contiguous if implemented with trees.
Sets is included in header file as #include<set >;

Time Complexity:

Insertion:

Best time: 0(log n)

Deletion:

Best time: 0(log n)

List Declaration:

set <datatype> name = {key1, key2, …} ;

Difference Between Set, Queue, Stack, Deque, List and Vector:

Set do not allow duplicates while others allows duplicates. Elements are added and deleted from anywhere.

8- Map:

Maps are the STL containers that use key-value pairs method. In this method, keys are the unique values while values are not to be supposed unique. Keys are in sorted order. Memory in maps may be contiguous or non-contiguous.
Maps is included in header file as #include<map >;

Time Complexity:

Insertion:

Best time: 0(log n)

Deletion:

Best time: 0(log n)

List Declaration:

map<,keytype, valuetype> name = {{key1, value1}, {key2, value2}, ….} ;

Difference Between Set, Queue, Stack, Deque, List and Vector:

Difference with other structures is the key value pair.

9- Unordered Map/Set:

Unordered maps are like the dictionary data structure that allows unordered key values. Memory in maps may be contiguous or non-contiguous.

Unordered Maps is included in header file as #include<unordered_map >;
Time Complexity:
Insertion:

Average time: 0(1)
Worst time: 0(n)

Deletion:

Average time: 0(1)
Worst time: 0(n)

List Declaration:

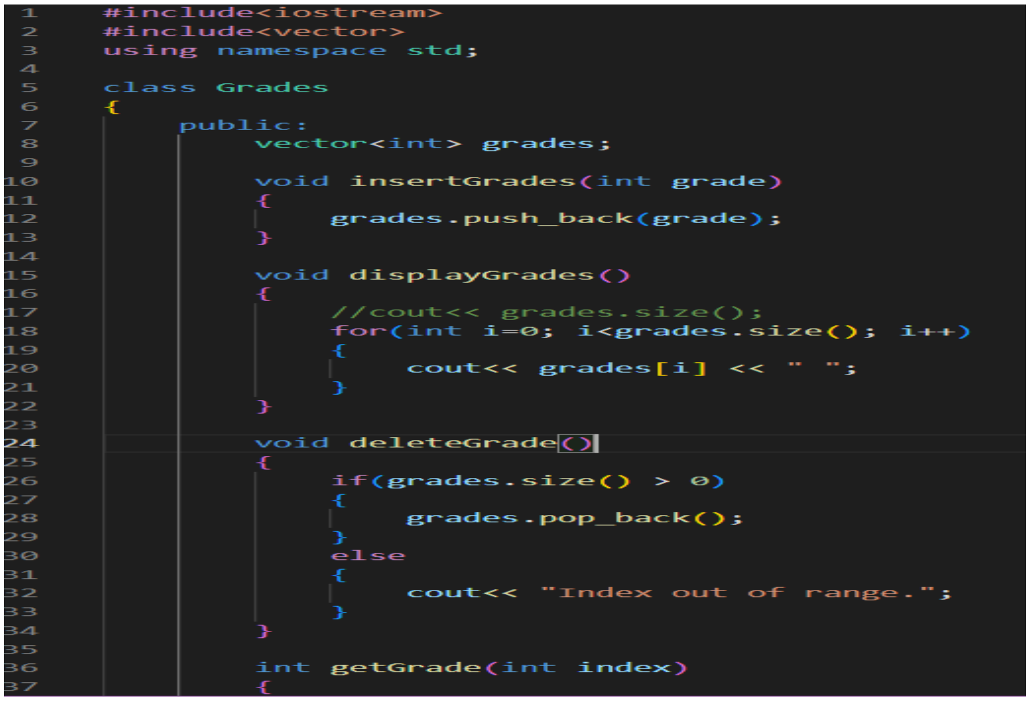
unordered_map<,keytype, valuetype> name;

Difference Between Set, Queue, Stack, Deque, List and Vector:

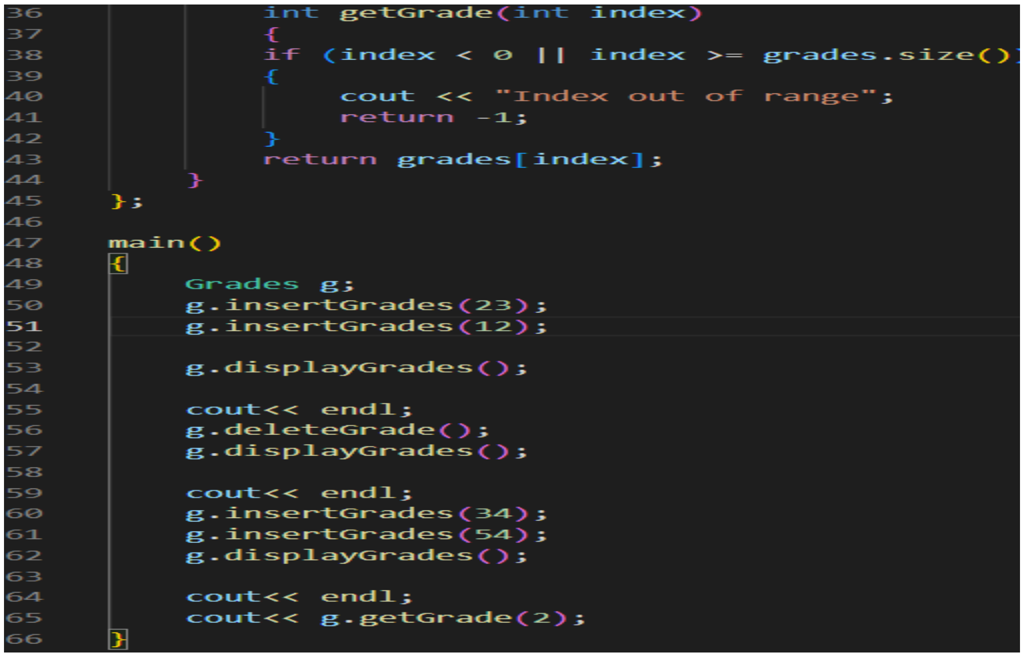
Difference with other structures is the key value pair.

Tasks

Task 1:

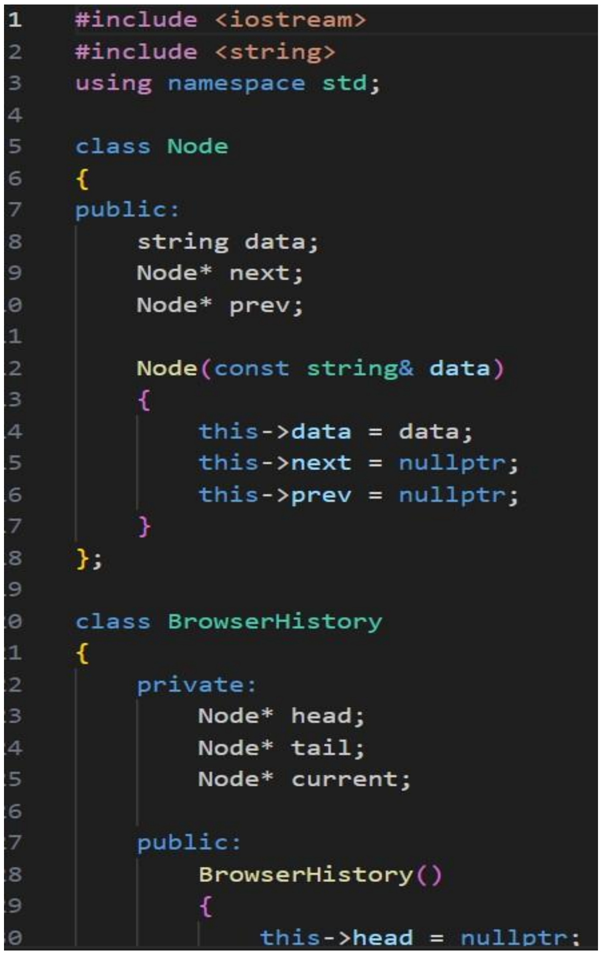


```
#include<iostream>
#include<vector>
using namespace std;

class Grades
{
    public:
        vector<int> grades;

        void insertGrades(int grade)
        {
            grades.push_back(grade);
        }

        void displayGrades()
        {
            //cout<< grades.size();
            for(int i=0; i<grades.size(); i++)
            {
                cout<< grades[i] << " ";
            }
        }

        void deleteGrade()
        {
            if(grades.size() > 0)
            {
                grades.pop_back();
            }
            else
            {
                cout<< "Index out of range.";
            }
        }

        int getGrade(int index)
        {
```

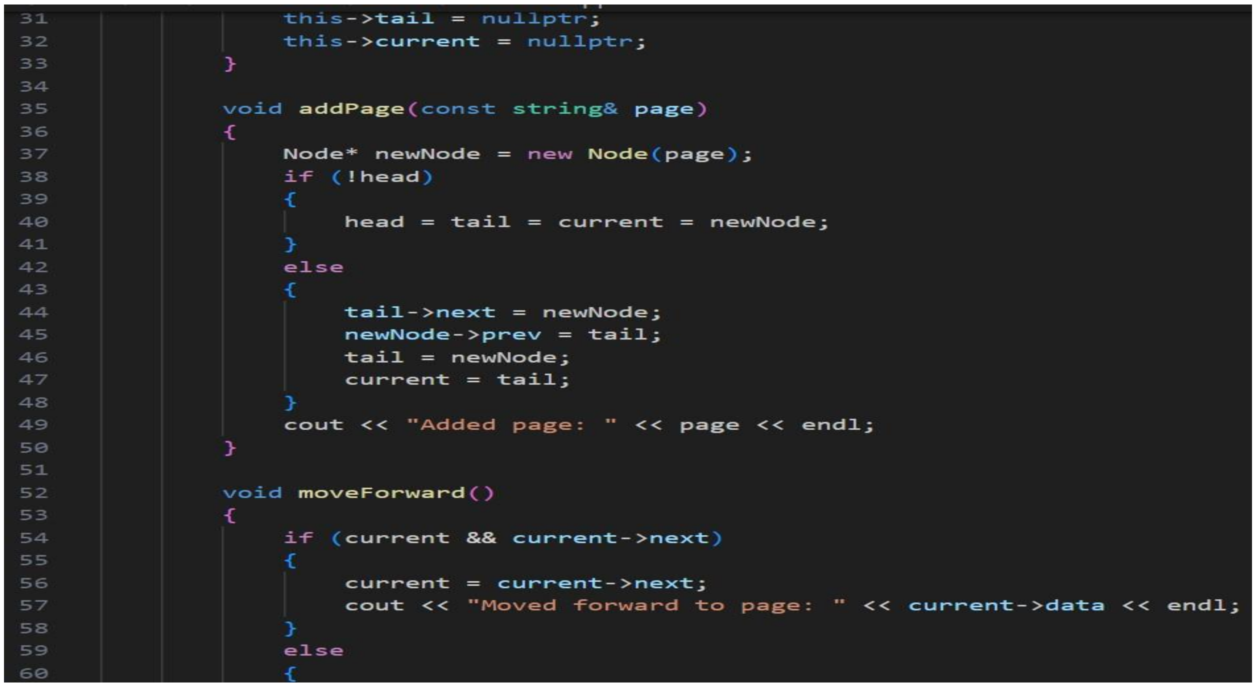


```
            int getGrade(int index)
            {
            if (index < 0 || index >= grades.size()
            {
                cout << "Index out of range";
                return -1;
            }
            return grades[index];
        }
    };

    main()
    {
        Grades g;
        g.insertGrades(23);
        g.insertGrades(12);

        g.displayGrades();

        cout<< endl;
        g.deleteGrade();
        g.displayGrades();

        cout<< endl;
        g.insertGrades(34);
        g.insertGrades(54);
        g.displayGrades();

        cout<< endl;
        cout<< g.getGrade(2);
    }
```

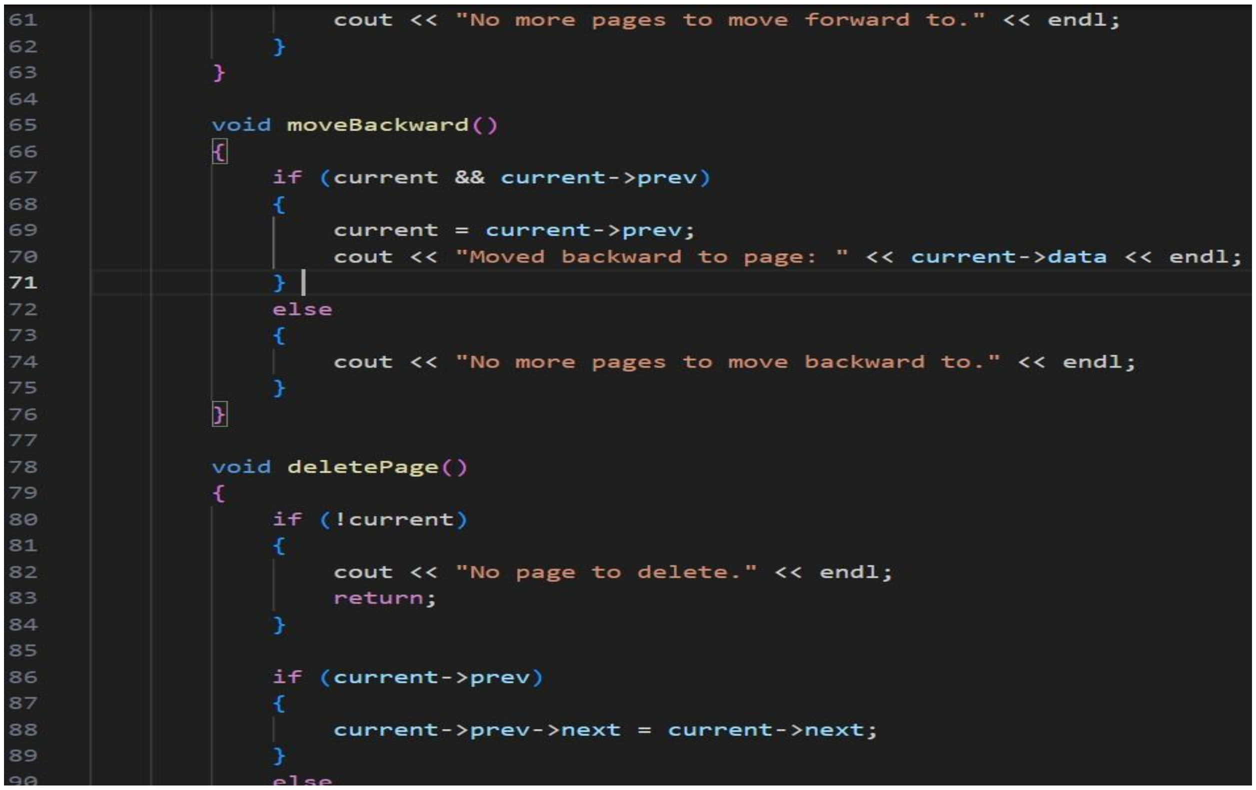
Task 2:



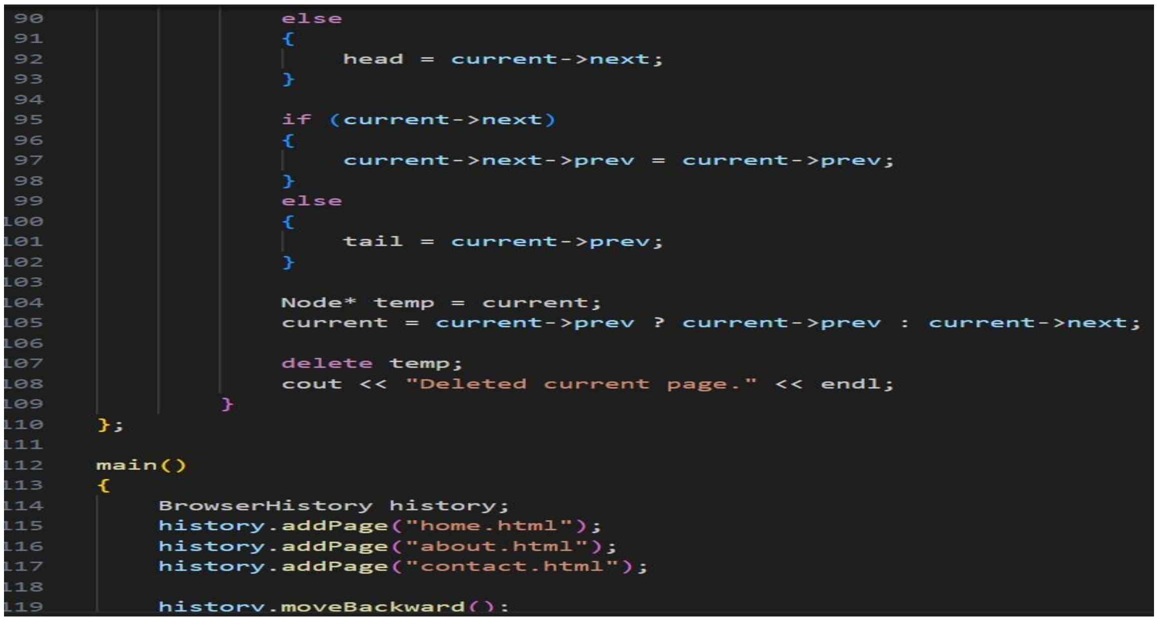
```
#include <iostream>
#include <string>
using namespace std;

class Node
{
public:
    string data;
    Node* next;
    Node* prev;

    Node(const string& data)
    {
        this->data = data;
        this->next = nullptr;
        this->prev = nullptr;
    }
};

class BrowserHistory
{
    private:
        Node* head;
        Node* tail;
        Node* current;

    public:
        BrowserHistory()
        {
            this->head = nullptr;
```



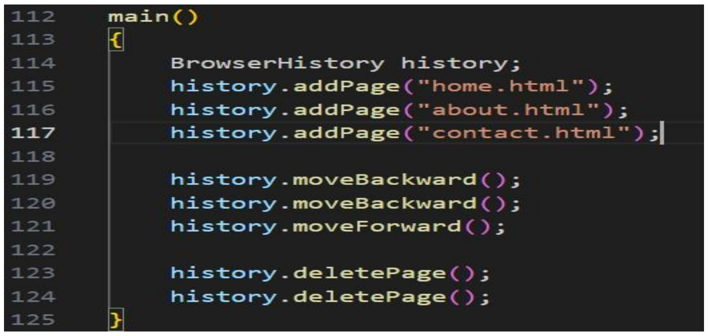
```
            this->tail = nullptr;
            this->current = nullptr;
        }

        void addPage(const string& page)
        {
            Node* newNode = new Node(page);
            if (!head)
            {
                head = tail = current = newNode;
            }
            else
            {
                tail->next = newNode;
                newNode->prev = tail;
                tail = newNode;
                current = tail;
            }
            cout << "Added page: " << page << endl;
        }

        void moveForward()
        {
            if (current && current->next)
            {
                current = current->next;
                cout << "Moved forward to page: " << current->data << endl;
            }
            else
            {
```



```
                cout << "No more pages to move forward to." << endl;
            }
        }

        void moveBackward()
        {
            if (current && current->prev)
            {
                current = current->prev;
                cout << "Moved backward to page: " << current->data << endl;
            }
            else
            {
                cout << "No more pages to move backward to." << endl;
            }
        }

        void deletePage()
        {
            if (!current)
            {
                cout << "No page to delete." << endl;
                return;
            }

            if (current->prev)
            {
                current->prev->next = current->next;
            }
            else
```

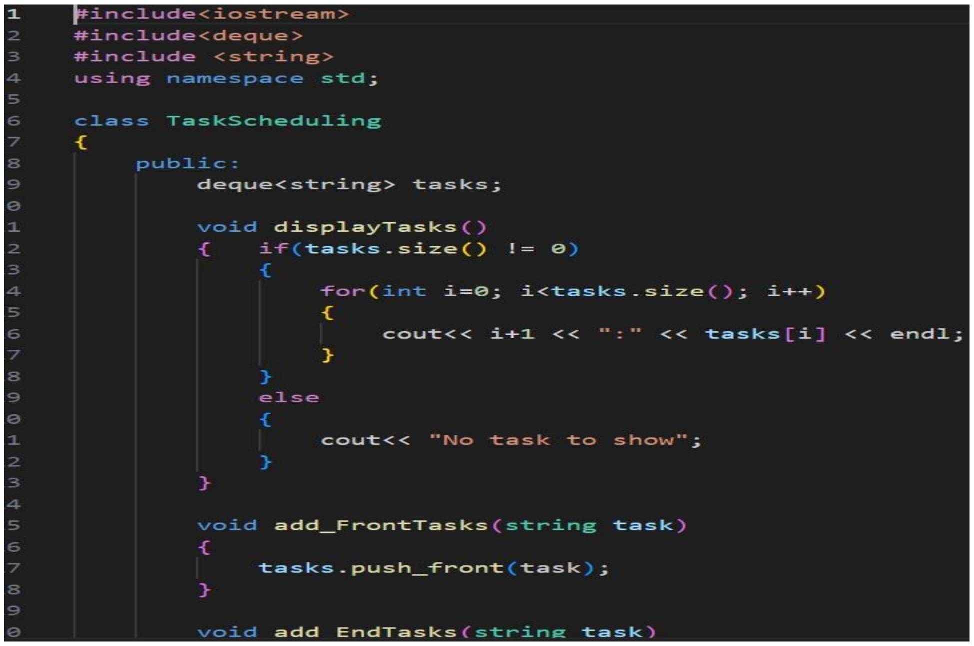


```
                else
                {
                    head = current->next;
                }

                if (current->next)
                {
                    current->next->prev = current->prev;
                }
                else
                {
                    tail = current->prev;
                }

                Node* temp = current;
                current = current->prev ? current->prev : current->next;

                delete temp;
                cout << "Deleted current page." << endl;
            }
};

main()
{
    BrowserHistory history;
    history.addPage("home.html");
    history.addPage("about.html");
    history.addPage("contact.html");

    history.moveBackward();
```

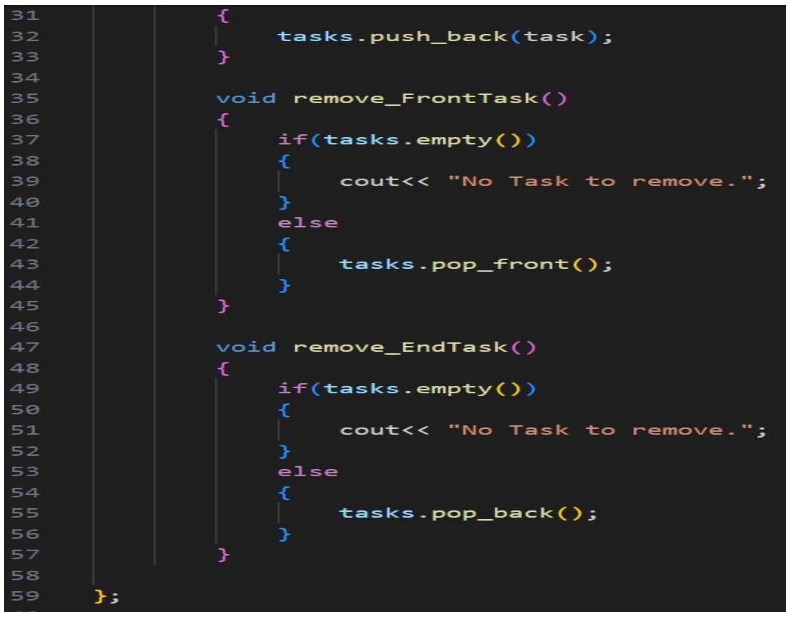


```
main()
{
    BrowserHistory history;
    history.addPage("home.html");
    history.addPage("about.html");
    history.addPage("contact.html");

    history.moveBackward();
    history.moveBackward();
    history.moveForward();

    history.deletePage();
    history.deletePage();
}
```

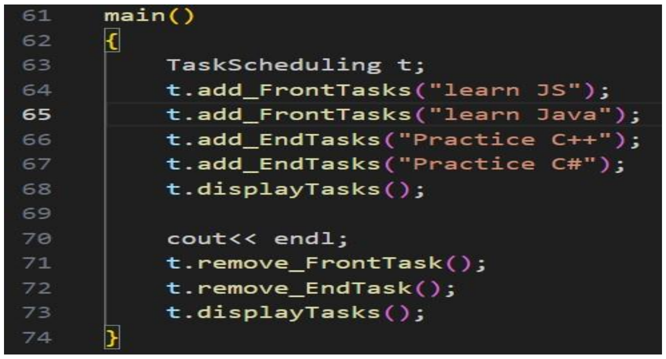
Task 3:



```
#include<iostream>
#include<deque>
#include <string>
using namespace std;

class TaskScheduling
{
    public:
        deque<string> tasks;

        void displayTasks()
        {   if(tasks.size() != 0)
            {
                for(int i=0; i<tasks.size(); i++)
                {
                    cout<< i+1 << ":" << tasks[i] << endl;
                }
            }
            else
            {
                cout<< "No task to show";
            }
        }

        void add_FrontTasks(string task)
        {
            tasks.push_front(task);
        }

        void add_EndTasks(string task)
```

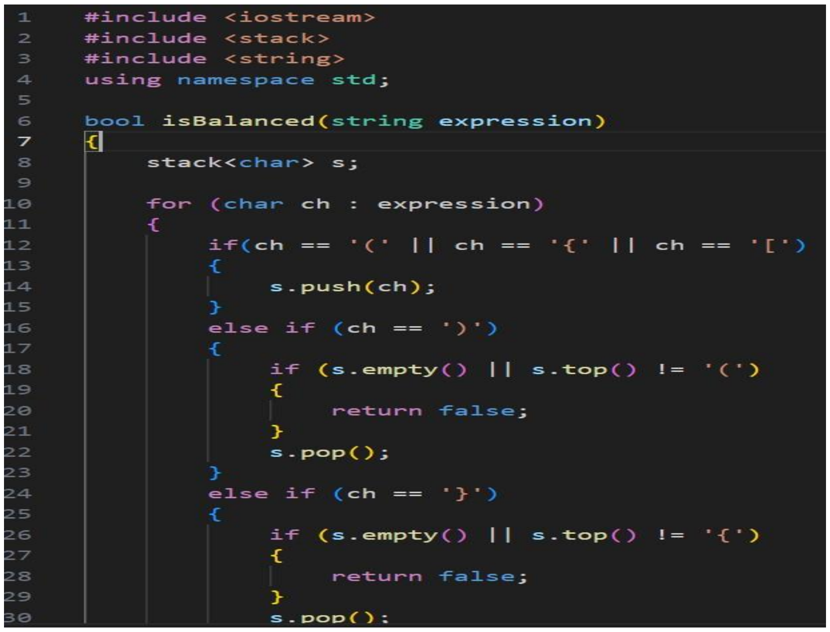


```
        {
            tasks.push_back(task);
        }

        void remove_FrontTask()
        {
            if(tasks.empty())
            {
                cout<< "No Task to remove.";
            }
            else
            {
                tasks.pop_front();
            }
        }

        void remove_EndTask()
        {
            if(tasks.empty())
            {
                cout<< "No Task to remove.";
            }
            else
            {
                tasks.pop_back();
            }
        }

};
```

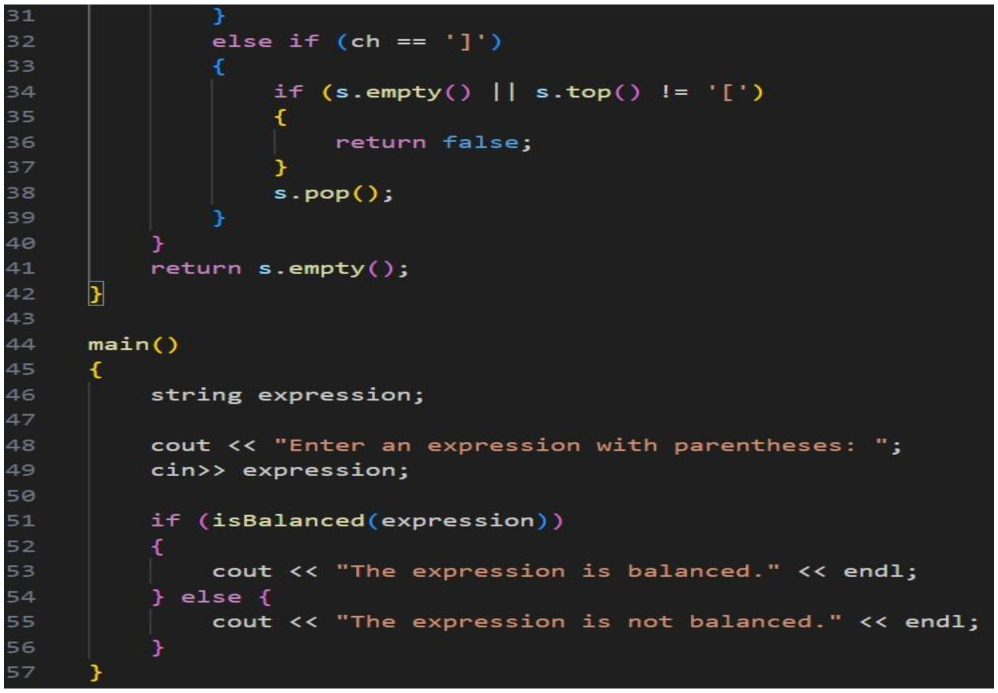


```
main()
{
    TaskScheduling t;
    t.add_FrontTasks("learn JS");
    t.add_FrontTasks("learn Java");
    t.add_EndTasks("Practice C++");
    t.add_EndTasks("Practice C#");
    t.displayTasks();

    cout<< endl;
    t.remove_FrontTask();
    t.remove_EndTask();
    t.displayTasks();
}
```

Task 4:

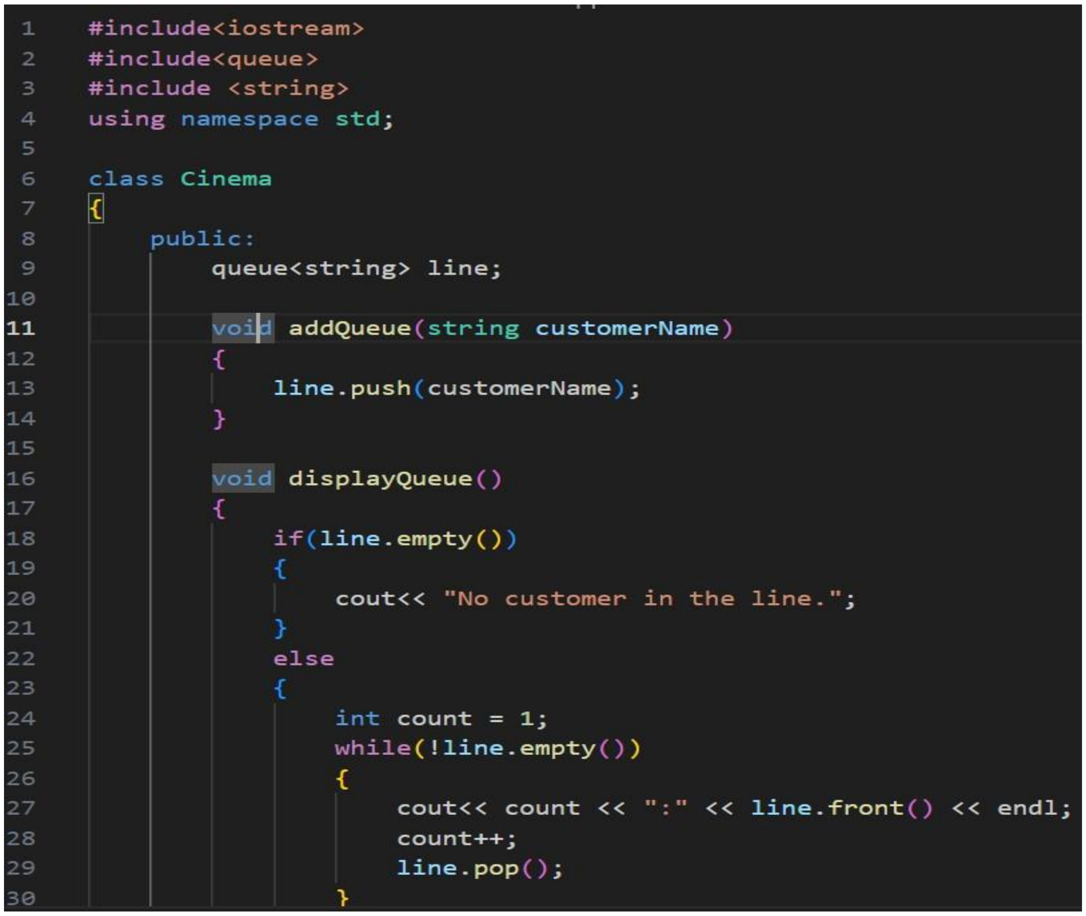


```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

bool isBalanced(string expression)
{
    stack<char> s;

    for (char ch : expression)
    {
        if(ch == '(' || ch == '{' || ch == '[')
        {
            s.push(ch);
        }
        else if (ch == ')')
        {
            if (s.empty() || s.top() != '(')
            {
                return false;
            }
            s.pop();
        }
        else if (ch == '}')
        {
            if (s.empty() || s.top() != '{')
            {
                return false;
            }
            s.pop();
```

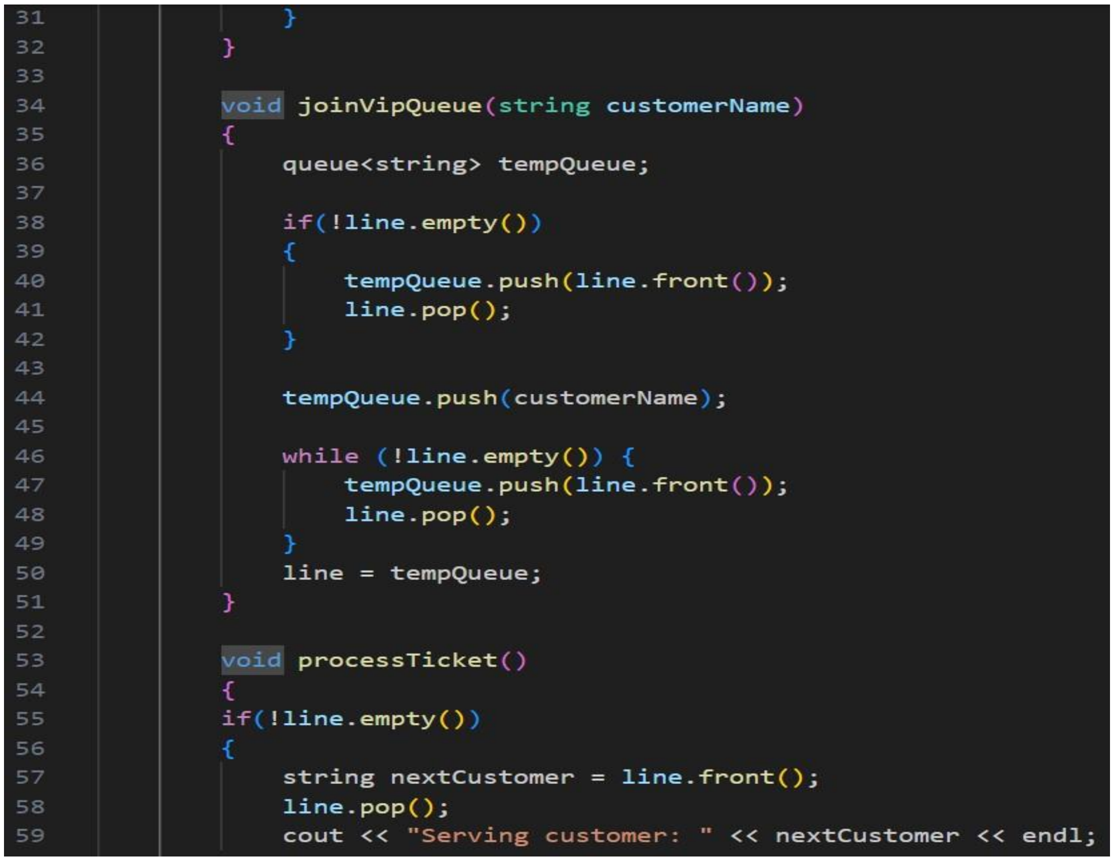


```
            }
            else if (ch == ']')
            {
                if (s.empty() || s.top() != '[')
                {
                    return false;
                }
                s.pop();
            }
        }
        return s.empty();
}

main()
{
        string expression;

        cout << "Enter an expression with parentheses: ";
        cin>> expression;

        if (isBalanced(expression))
        {
            cout << "The expression is balanced." << endl;
        } else {
            cout << "The expression is not balanced." << endl;
        }
}
```

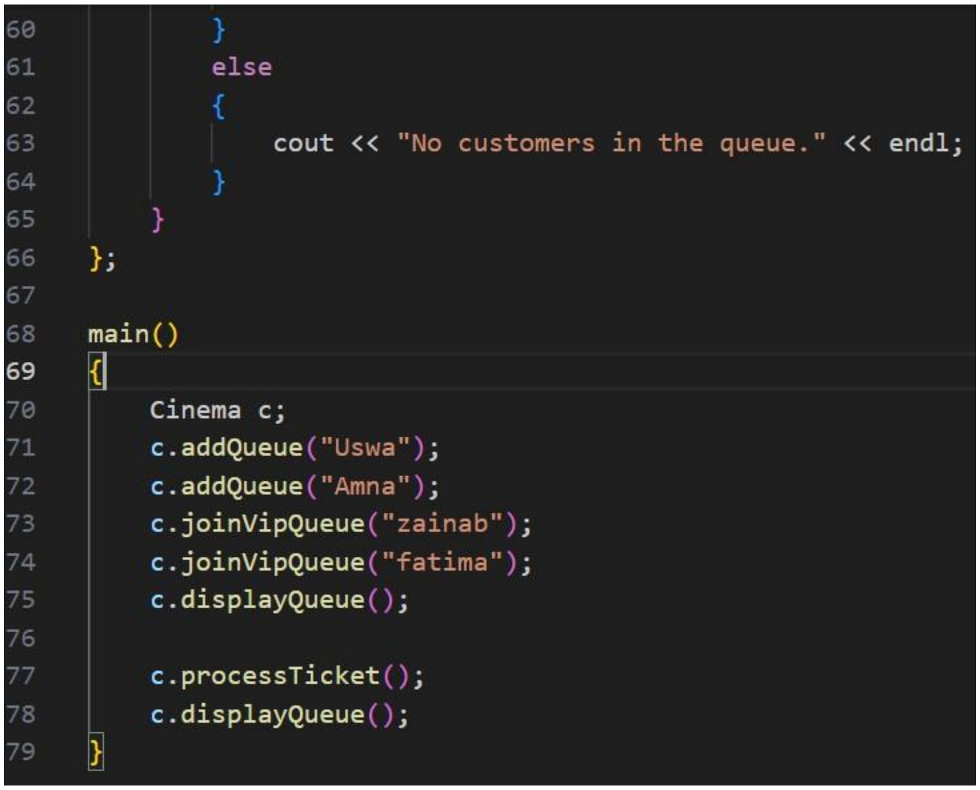
Task 5:



```
#include<iostream>
#include<queue>
#include <string>
using namespace std;

class Cinema
{
    public:
        queue<string> line;

        void addQueue(string customerName)
        {
            line.push(customerName);
        }

        void displayQueue()
        {
            if(line.empty())
            {
                cout<< "No customer in the line.";
            }
            else
            {
                int count = 1;
                while(!line.empty())
                {
                    cout<< count << ":" << line.front() << endl;
                    count++;
                    line.pop();
                }
```

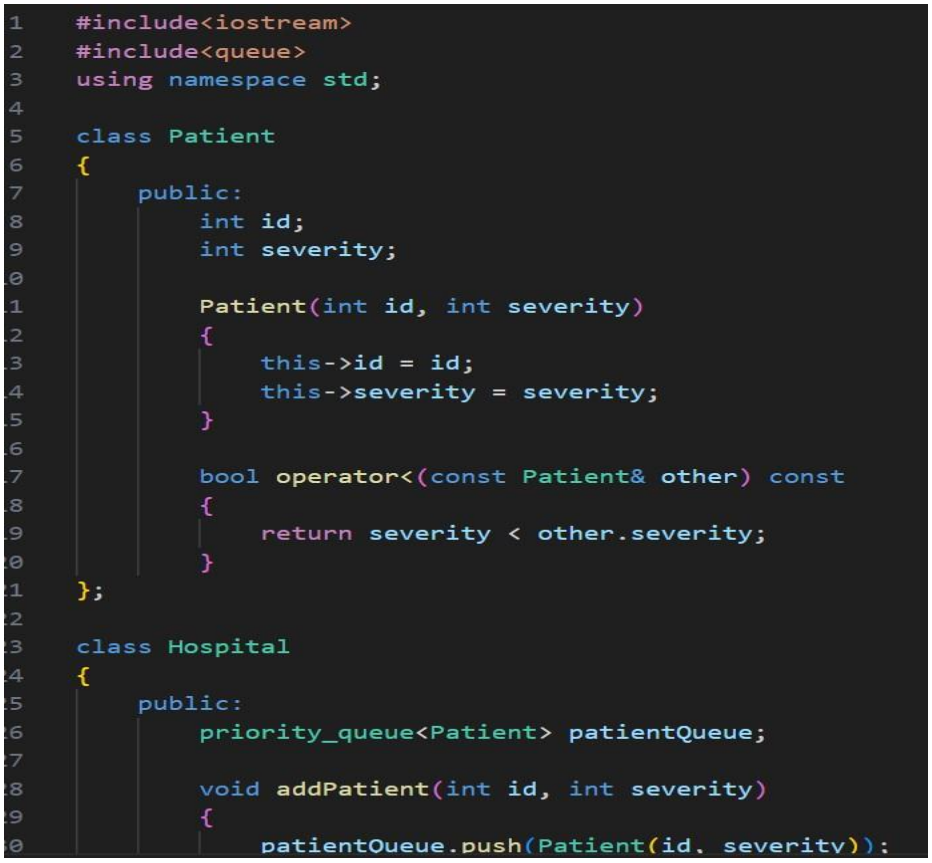


```
            }
        }

        void joinVipQueue(string customerName)
        {
            queue<string> tempQueue;

            if(!line.empty())
            {
                tempQueue.push(line.front());
                line.pop();
            }

            tempQueue.push(customerName);

            while (!line.empty()) {
                tempQueue.push(line.front());
                line.pop();
            }
            line = tempQueue;
        }

        void processTicket()
        {
        if(!line.empty())
        {
            string nextCustomer = line.front();
            line.pop();
            cout << "Serving customer: " << nextCustomer << endl;
```

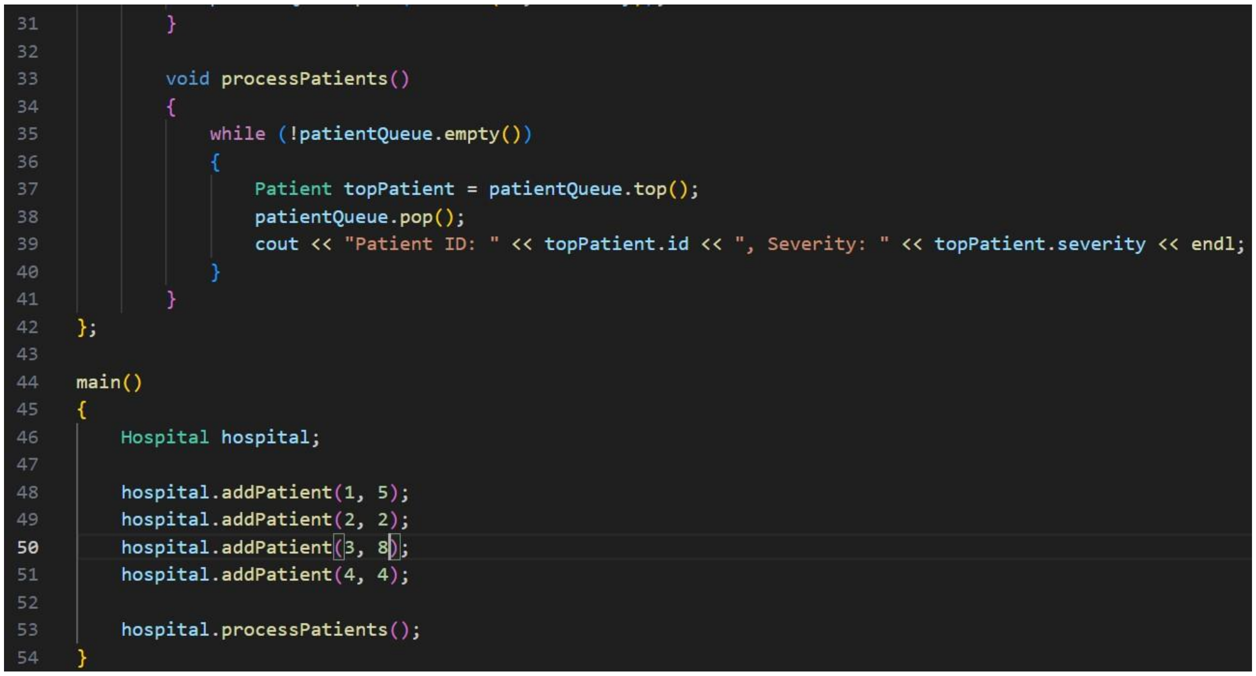


```
        }
        else
        {
            cout << "No customers in the queue." << endl;
        }
    }
};

main()
{
    Cinema c;
    c.addQueue("Uswa");
    c.addQueue("Amna");
    c.joinVipQueue("zainab");
    c.joinVipQueue("fatima");
    c.displayQueue();

    c.processTicket();
    c.displayQueue();
}
```

Task 6:

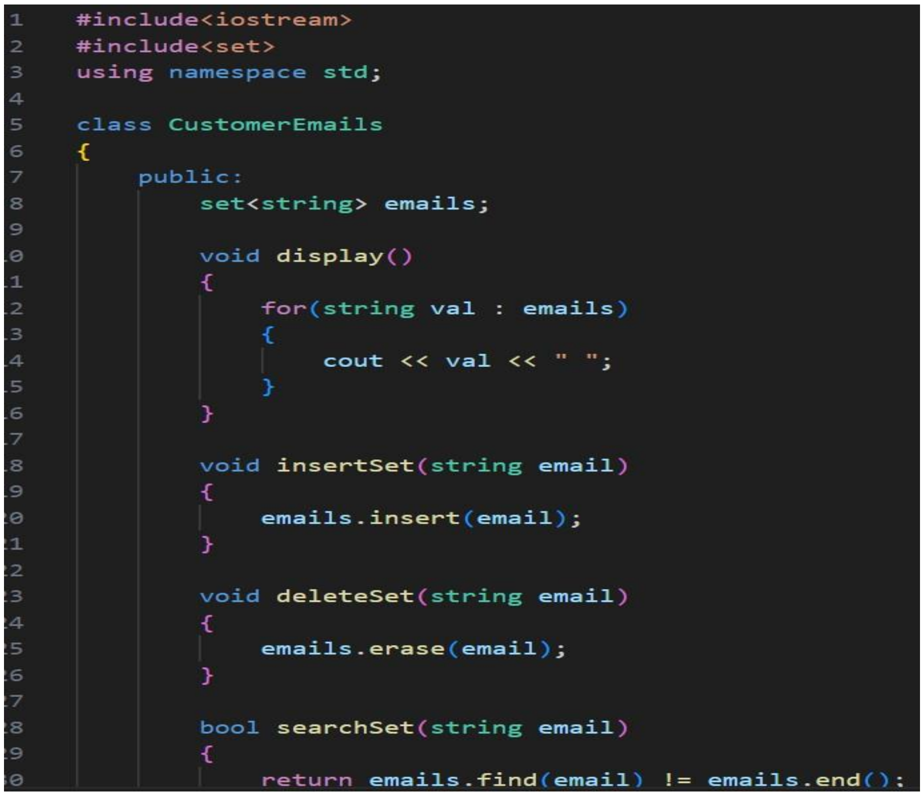


```
#include<iostream>
#include<queue>
using namespace std;

class Patient
{
    public:
        int id;
        int severity;

        Patient(int id, int severity)
        {
            this->id = id;
            this->severity = severity;
        }

        bool operator<(const Patient& other) const
        {
            return severity < other.severity;
        }
};

class Hospital
{
    public:
        priority_queue<Patient> patientQueue;

        void addPatient(int id, int severity)
        {
            patientQueue.push(Patient(id, severity));
```

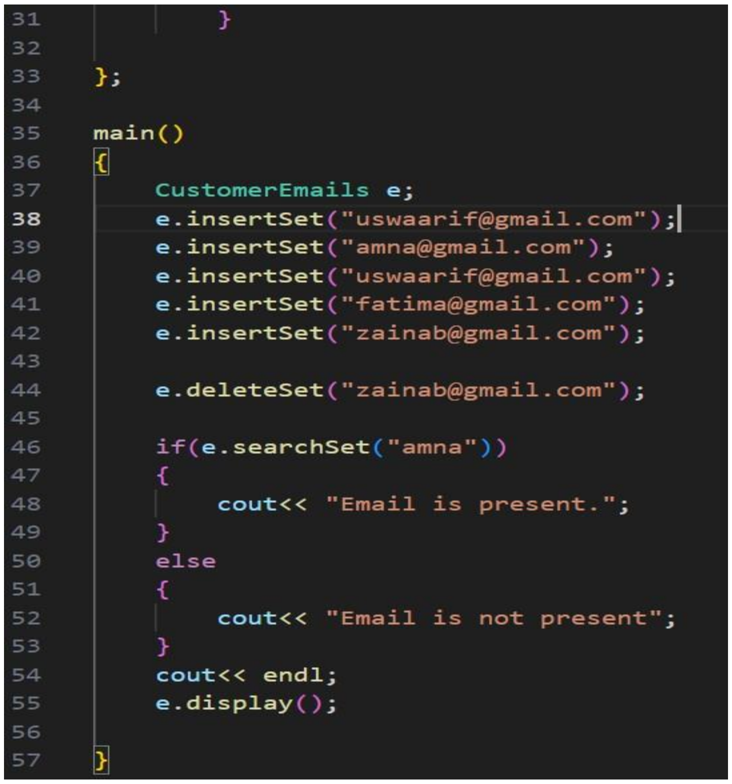


```
        }

        void processPatients()
        {
            while (!patientQueue.empty())
            {
                Patient topPatient = patientQueue.top();
                patientQueue.pop();
                cout << "Patient ID: " << topPatient.id << ", Severity: " << topPatient.severity << endl;
            }
        }
};

main()
{
    Hospital hospital;

    hospital.addPatient(1, 5);
    hospital.addPatient(2, 2);
    hospital.addPatient(3, 8);
    hospital.addPatient(4, 4);

    hospital.processPatients();
}
```

Task 7:

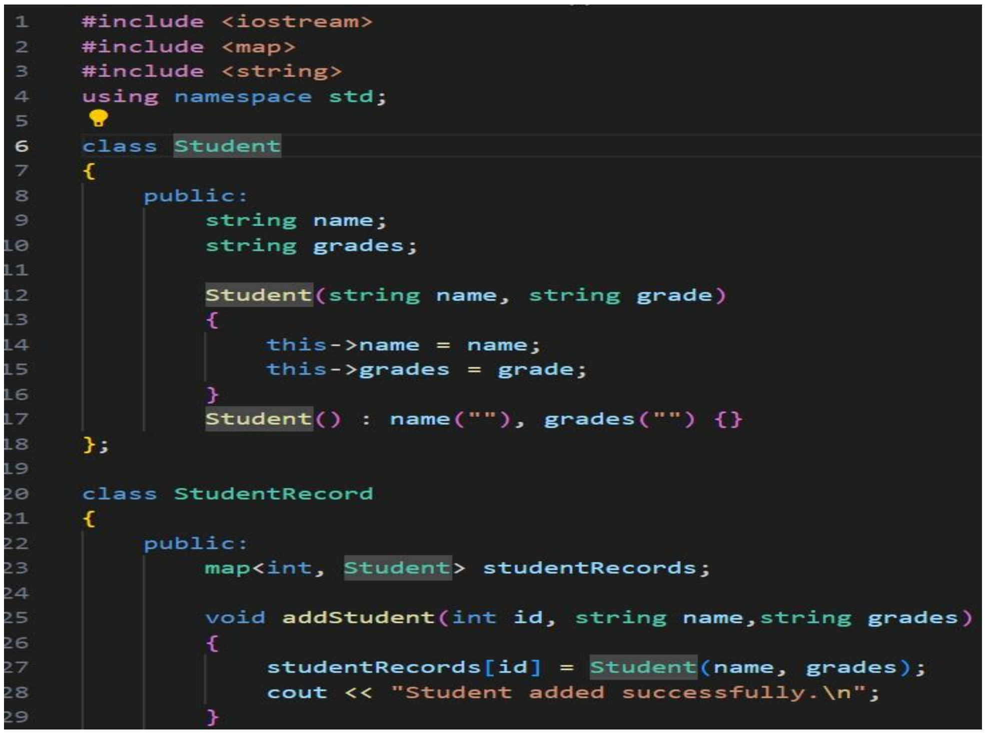


```cpp
#include<iostream>
#include<set>
using namespace std;

class CustomerEmails
{
    public:
        set<string> emails;

        void display()
        {
            for(string val : emails)
            {
                cout << val << " ";
            }
        }

        void insertSet(string email)
        {
            emails.insert(email);
        }

        void deleteSet(string email)
        {
            emails.erase(email);
        }

        bool searchSet(string email)
        {
            return emails.find(email) != emails.end();
```

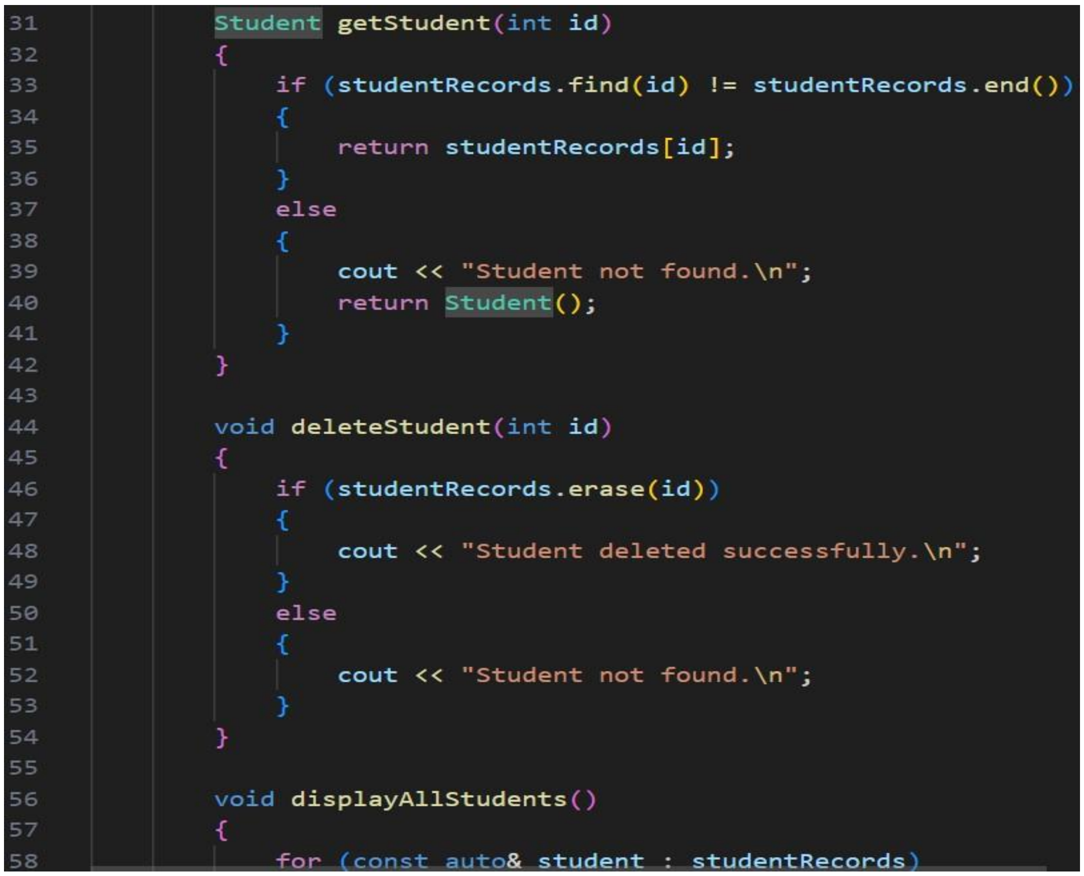


```cpp
        }

};

main()
{
    CustomerEmails e;
    e.insertSet("uswaarif@gmail.com");
    e.insertSet("amna@gmail.com");
    e.insertSet("uswaarif@gmail.com");
    e.insertSet("fatima@gmail.com");
    e.insertSet("zainab@gmail.com");

    e.deleteSet("zainab@gmail.com");

    if(e.searchSet("amna"))
    {
        cout<< "Email is present.";
    }
    else
    {
        cout<< "Email is not present";
    }
    cout<< endl;
    e.display();

}
```

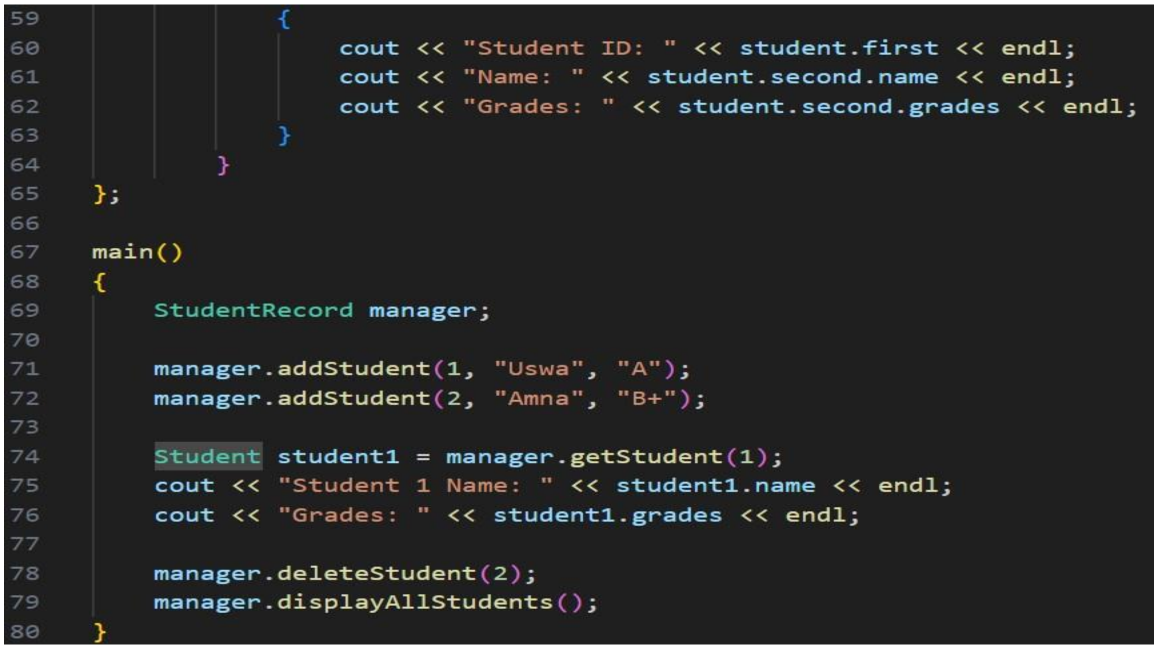
Task 8:



```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

class Student
{
    public:
        string name;
        string grades;

        Student(string name, string grade)
        {
            this->name = name;
            this->grades = grade;
        }
        Student() : name(""), grades("") {}
};

class StudentRecord
{
    public:
        map<int, Student> studentRecords;

        void addStudent(int id, string name,string grades)
        {
            studentRecords[id] = Student(name, grades);
            cout << "Student added successfully.\n";
        }
```

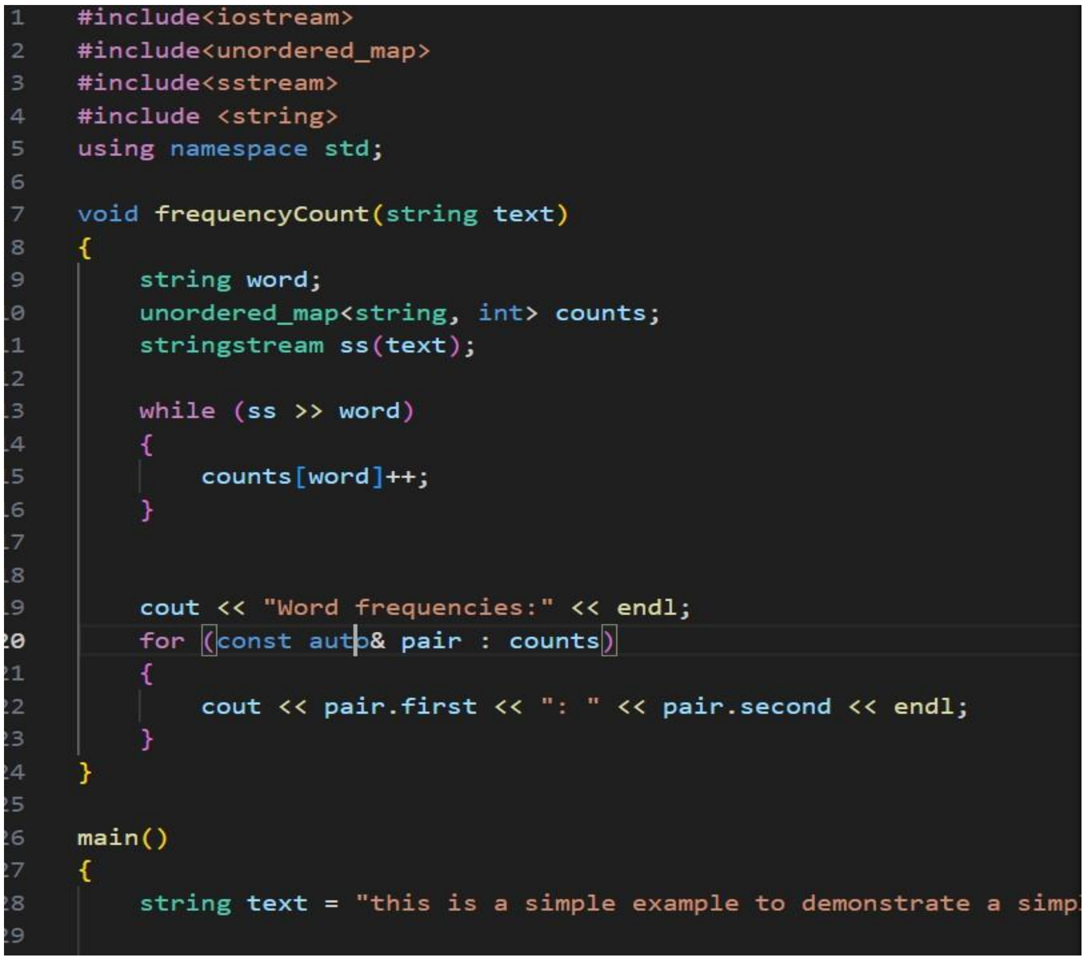


```cpp
        Student getStudent(int id)
        {
            if (studentRecords.find(id) != studentRecords.end())
            {
                return studentRecords[id];
            }
            else
            {
                cout << "Student not found.\n";
                return Student();
            }
        }

        void deleteStudent(int id)
        {
            if (studentRecords.erase(id))
            {
                cout << "Student deleted successfully.\n";
            }
            else
            {
                cout << "Student not found.\n";
            }
        }

        void displayAllStudents()
        {
            for (const auto& student : studentRecords)
```

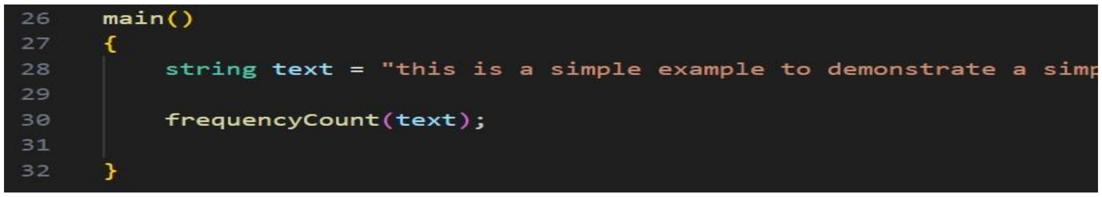


```
                {
                    cout << "Student ID: " << student.first << endl;
                    cout << "Name: " << student.second.name << endl;
                    cout << "Grades: " << student.second.grades << endl;
                }
            }
};

main()
{
    StudentRecord manager;

    manager.addStudent(1, "Uswa", "A");
    manager.addStudent(2, "Amna", "B+");

    Student student1 = manager.getStudent(1);
    cout << "Student 1 Name: " << student1.name << endl;
    cout << "Grades: " << student1.grades << endl;

    manager.deleteStudent(2);
    manager.displayAllStudents();
}
```

Task 9:



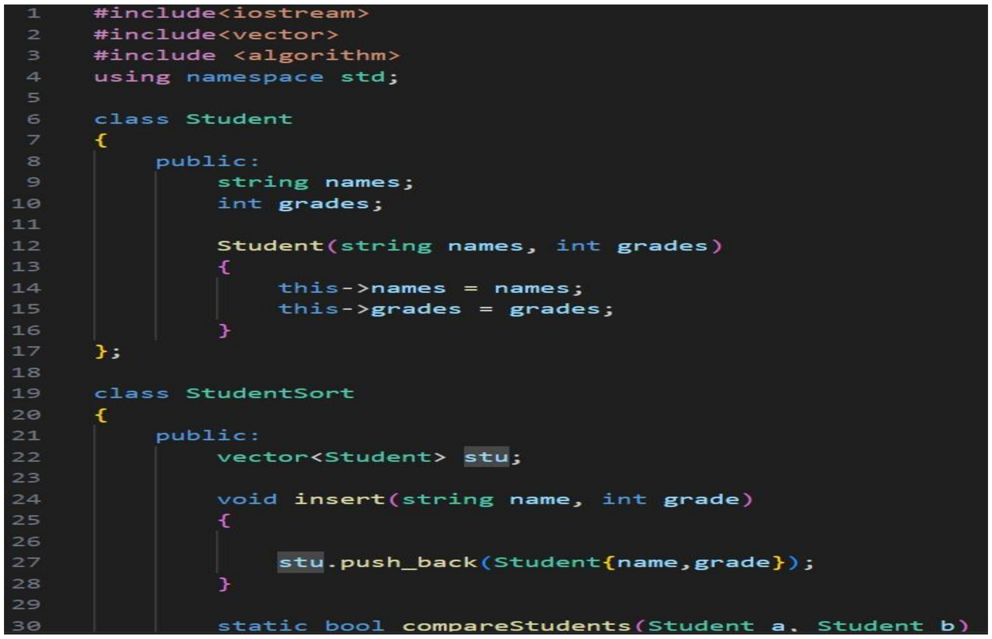
```
#include<iostream>
#include<unordered_map>
#include<sstream>
#include <string>
using namespace std;

void frequencyCount(string text)
{
    string word;
    unordered_map<string, int> counts;
    stringstream ss(text);

    while (ss >> word)
    {
        counts[word]++;
    }


    cout << "Word frequencies:" << endl;
    for (const auto& pair : counts)
    {
        cout << pair.first << ": " << pair.second << endl;
    }
}

main()
{
    string text = "this is a simple example to demonstrate a simp
```

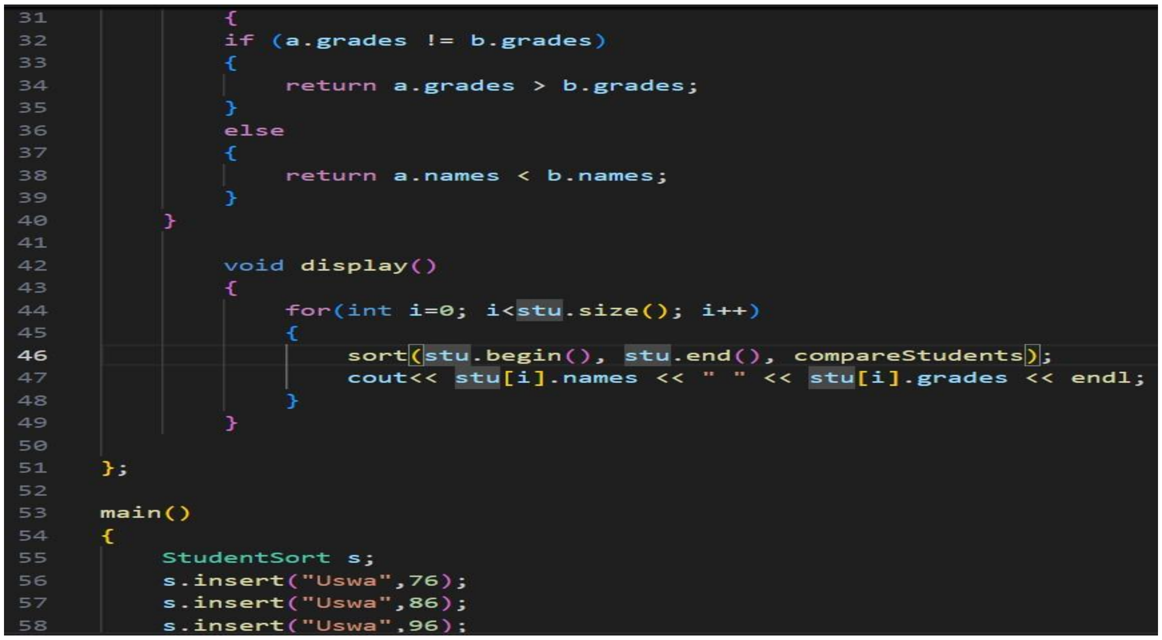


```
main()
{
    string text = "this is a simple example to demonstrate a simp

    frequencyCount(text);

}
```

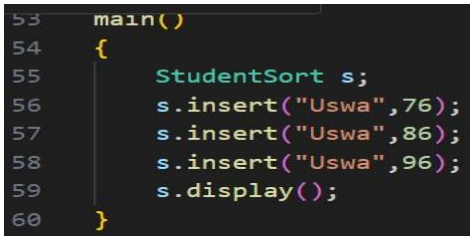
Algorithms

Sort Method:



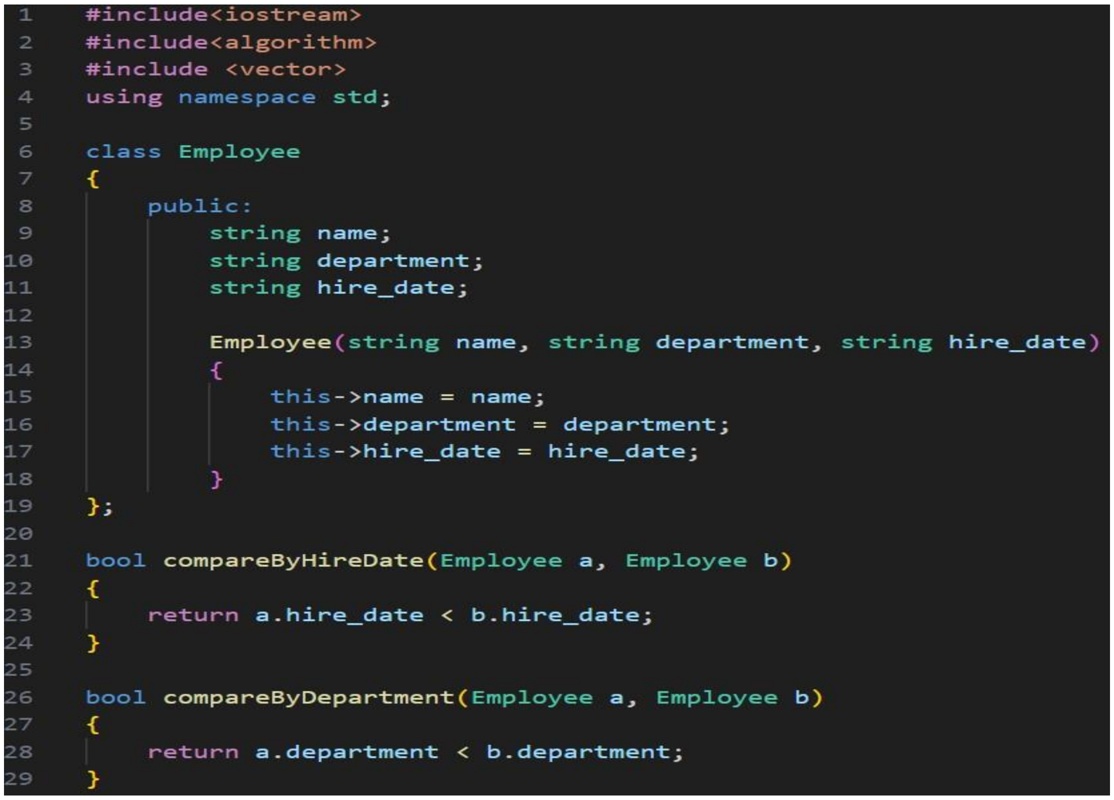


```
#include<iostream>
#include<vector>
#include <algorithm>
using namespace std;

class Student
{
    public:
        string names;
        int grades;

        Student(string names, int grades)
        {
            this->names = names;
            this->grades = grades;
        }
};

class StudentSort
{
    public:
        vector<Student> stu;

        void insert(string name, int grade)
        {

            stu.push_back(Student{name,grade});
        }

        static bool compareStudents(Student a, Student b)
        {
        if (a.grades != b.grades)
        {
            return a.grades > b.grades;
        }
        else
        {
            return a.names < b.names;
        }
    }

        void display()
        {
            for(int i=0; i<stu.size(); i++)
            {
                sort(stu.begin(), stu.end(), compareStudents);
                cout<< stu[i].names << " " << stu[i].grades << endl;
            }
        }

};

main()
{
    StudentSort s;
    s.insert("Uswa",76);
    s.insert("Uswa",86);
    s.insert("Uswa",96);
```

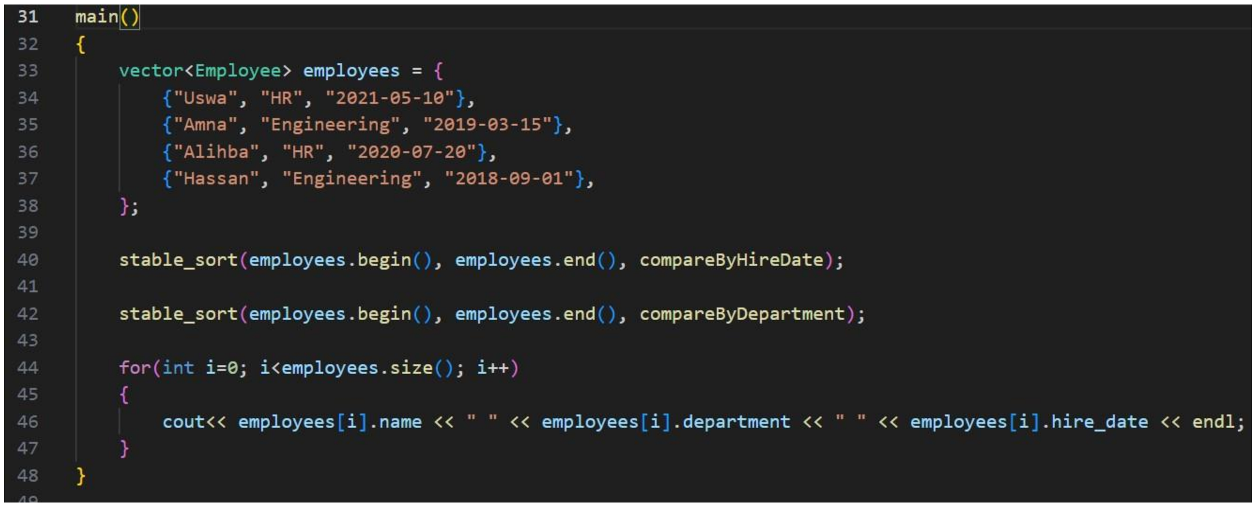


```
main()
{
    StudentSort s;
    s.insert("Uswa",76);
    s.insert("Uswa",86);
    s.insert("Uswa",96);
    s.display();
}
```

Stable Sort Method:



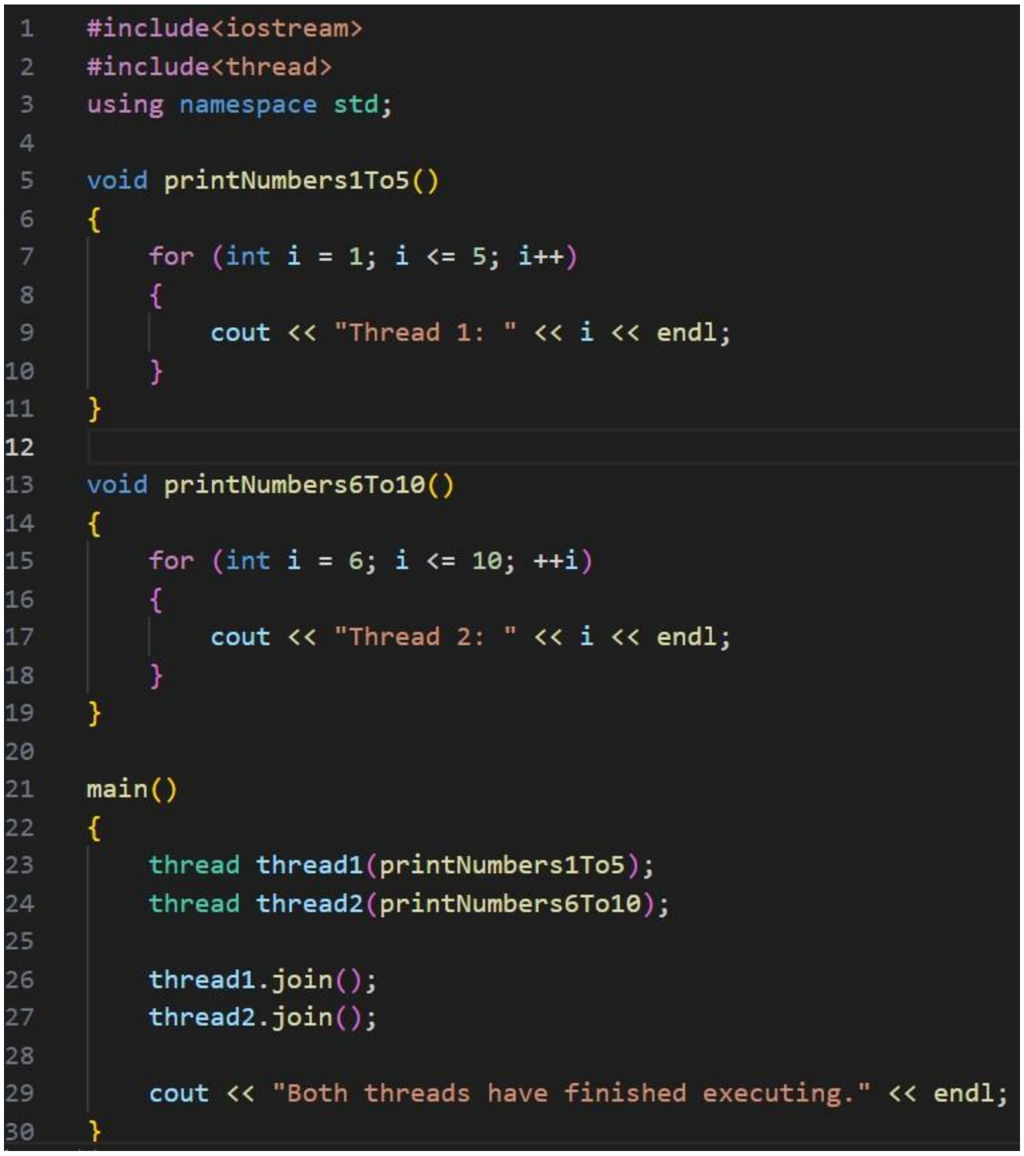
```
#include<iostream>
#include<algorithm>
#include <vector>
using namespace std;

class Employee
{
    public:
        string name;
        string department;
        string hire_date;

        Employee(string name, string department, string hire_date)
        {
            this->name = name;
            this->department = department;
            this->hire_date = hire_date;
        }
};

bool compareByHireDate(Employee a, Employee b)
{
    return a.hire_date < b.hire_date;
}

bool compareByDepartment(Employee a, Employee b)
{
    return a.department < b.department;
}
```



```
main()
{
    vector<Employee> employees = {
        {"Uswa", "HR", "2021-05-10"},
        {"Amna", "Engineering", "2019-03-15"},
        {"Alihba", "HR", "2020-07-20"},
        {"Hassan", "Engineering", "2018-09-01"},
    };

    stable_sort(employees.begin(), employees.end(), compareByHireDate);

    stable_sort(employees.begin(), employees.end(), compareByDepartment);

    for(int i=0; i<employees.size(); i++)
    {
        cout<< employees[i].name << " " << employees[i].department << " " << employees[i].hire_date << endl;
    }
}
```

Concurrency

Threads:



```cpp
#include<iostream>
#include<thread>
using namespace std;

void printNumbers1To5()
{
    for (int i = 1; i <= 5; i++)
    {
        cout << "Thread 1: " << i << endl;
    }
}

void printNumbers6To10()
{
    for (int i = 6; i <= 10; ++i)
    {
        cout << "Thread 2: " << i << endl;
    }
}

main()
{
    thread thread1(printNumbers1To5);
    thread thread2(printNumbers6To10);

    thread1.join();
    thread2.join();

    cout << "Both threads have finished executing." << endl;
}
```