Name- Abhijeet Sandeep Jadhav
Roll- 22mc3002

1)

main.cpp

Run

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

struct Student {
    int id;
    std::string name;
    std::string program;
    double grade;

Student(int _id, const std::string& _name, const std::string& _program, double
    _grade)
        : id(_id), name(_name), program(_program), grade(_grade) {}
};

class StudentManagementSystem {
private:
std::vector<Student> students;

public:
void addStudent(int id, const std::string& name, const std::string& program
    ,double grade) {
        students.push_back(Student(id, name, program, grade));
    }

```

main.cpp

Run

```cpp

Student* searchStudentByID(int id) {
    for (auto& student : students) {
        if (student.id == id) {
            return &student;
        }
    }
    return nullptr;
}


void deleteStudentByID(int id) {
    auto it = std::remove_if(students.begin(), students.end(), [id](const
        Student& student) {
        return student.id == id;
    });
    students.erase(it, students.end());
}

void displayTopPerformers(int numStudents) {
    std::sort(students.begin(), students.end(), [](const Student& a, const
        Student& b) {
        return a.grade > b.grade;
    });

```

main.cpp

Run

```
46
47          std::cout << "Top " << numStudents << " performers:\n";
48          for (int i = 0; i < numStudents && i < students.size(); ++i) {
49              std::cout << "Name: " << students[i].name << "\nProgram: " <<
                    students[i].program << "\nGrade: " << students[i].grade <<
                    "\n\n";
50          }
51      }
52
53      std::vector<Student> getStudentsByProgram(const std::string& program) {
54          std::vector<Student> programList;
55          for (auto& student : students) {
56              if (student.program == program) {
57                  programList.push_back(student);
58              }
59          }
60          return programList;
61      }
62
63      void displayAllStudents() {
64          for (const auto& student : students) {
65              std::cout << "ID: " << student.id << "\nName: " << student.name <<
                    "\nProgram: " << student.program << "\nGrade: " << student
                    .grade << "\n\n";
66          }
67      }
```

main.cpp

Run

```
69
70  int main() {
71      StudentManagementSystem sms;
72
73      sms.addStudent(3001, "Abhijeet Parashar", "Mathematics and Computing", 8.2
            );
74      sms.addStudent(3002, "Abhijeet Jadhav", "Mathematics and Computing", 7.3);
75      sms.addStudent(3003, "Adarsh Kumar", "Mathematics and Computing", 9.3);
76      sms.addStudent(3004, "Aman Barte", "Mathematics and Computing", 7.1);
77      sms.addStudent(3005, "Aman Kumar", "EV", 2.3);
78
79      std::cout << "All Students:\n";
80      sms.displayAllStudents();
81
82      int searchId = 2;
83      Student* foundStudent = sms.searchStudentByID(searchId);
84      if (foundStudent) {
85          std::cout << "Found student with ID " << searchId << ":\n";
86          std::cout << "Name: " << foundStudent->name << "\nProgram: " <<
                foundStudent->program << "\nGrade: " << foundStudent->grade <<
                "\n\n";
87      } else {
88          std::cout << "Student not found by ID " << searchId << ".\n";
89      }
90
91      int deleteId = 3;
92      sms.deleteStudentByID(deleteId);
```

main.cpp

Run

```
            foundStudent->program << "\nGrade: " << foundStudent->grade <<
            "\n\n";
    } else {
        std::cout << "Student not found by ID " << searchId << ".\n";
    }

    int deleteId = 3;
    sms.deleteStudentByID(deleteId);
    std::cout << "After deleting student with ID " << deleteId << ":\n";
    sms.displayAllStudents();

    int numTopPerformers = 3;
    sms.displayTopPerformers(numTopPerformers);

    std::string programToSearch = "Computer Science";
    std::vector<Student> programList = sms.getStudentsByProgram
        (programToSearch);
    std::cout << "Students in " << programToSearch << ":\n";
    for (const auto& student : programList) {
        std::cout << "Name: " << student.name << "\nProgram: " << student
            .program << "\nGrade: " << student.grade << "\n\n";
    }

    return 0;
}
```

OUTPUT:

Output

Clear

```
/tmp/ItSvCwODb7.o
All Students:
ID: 3001
Name: Abhijeet Parashar
Program: Mathematics and Computing
Grade: 8.2

ID: 3002
Name: Abhijeet Jadhav
Program: Mathematics and Computing
Grade: 7.3

ID: 3003Name: Adarsh Kumar
Program: Mathematics and Computing
Grade: 9.3

ID: 3004
Name: Aman Barte
Program: Mathematics and Computing
Grade: 7.1

ID: 3005
Name: Aman Kumar
Program: EV
Grade: 2.3
```

2)

main.cpp Run

```cpp
#include <iostream>
#include <string>
#include <vector>

struct Train {
    int trainNumber;
    std::string departure;
    std::string destination;
    int totalSeats;
    int availableSeats;
    std::vector<std::string> passengers;
    Train* next;

    Train(int _trainNumber, const std::string& _departure, const std::string&
        _destination, int _totalSeats)
        : trainNumber(_trainNumber), departure(_departure), destination
            (_destination), totalSeats(_totalSeats),
          availableSeats(_totalSeats), next(nullptr) {}
};

class TrainBookingSystem {
private:
    Train* head;

public:
    TrainBookingSystem() : head(nullptr) {}
```

```cpp
public:
    TrainBookingSystem() : head(nullptr) {}

    void addTrain(int trainNumber, const std::string& departure, const std
        ::string& destination, int totalSeats) {
        Train* newTrain = new Train(trainNumber, departure, destination,
            totalSeats);
        if (!head) {
            head = newTrain;
        } else {
            Train* current = head;
            while (current->next) {
                current = current->next;
            }
            current->next = newTrain;
        }
    }

    void reserveSeat(int trainNumber, const std::string& passengerName) {
        Train* current = head;
        while (current) {
            if (current->trainNumber == trainNumber) {
                if (current->availableSeats > 0) {
                    current->passengers.push_back(passengerName);
                    current->availableSeats--;
                    std::cout << "Seat reserved for " << passengerName << " on
```

main.cpp

Run

```
                std::cout << "Seat reserved for " << passengerName << " on
                    Train " << trainNumber << ".\n";
            } else {
                std::cout << "Train " << trainNumber << " is fully booked
                    .\n";
            }
            return;
        }
        current = current->next;
    }
    std::cout << "Train " << trainNumber << " not found.\n";
}

void cancelReservation(int trainNumber, const std::string& passengerName)
    {
    Train* current = head;
    while (current) {
        if (current->trainNumber == trainNumber) {
            for (auto it = current->passengers.begin(); it != current
                ->passengers.end(); ++it) {
                if (*it == passengerName) {
                    current->passengers.erase(it);
                    current->availableSeats++;
                    std::cout << "Reservation canceled for " <<
                        passengerName << " on Train " << trainNumber << "
                        .\n";
```

main.cpp

Run

```
                        passengerName << " on Train " << trainNumber << "
                        .\n";
                    return;
                }
            }
            std::cout << "Passenger " << passengerName << " not found on
                Train " << trainNumber << ".\n";
            return;
        }
        current = current->next;
    }
    std::cout << "Train " << trainNumber << " not found.\n";
}

void displayTrainInformation(int trainNumber) {
    Train* current = head;
    while (current) {
        if (current->trainNumber == trainNumber) {
            std::cout << "Train Number: " << current->trainNumber << "\n";
            std::cout << "Departure: " << current->departure << "\n";
            std::cout << "Destination: " << current->destination << "\n";
            std::cout << "Total Seats: " << current->totalSeats << "\n";
            std::cout << "Available Seats: " << current->availableSeats <<
                "\n";
            std::cout << "Passenger List:\n";
            for (const std::string& passenger : current->passengers) {
```

main.cpp

Run

```
86                  std::cout << "Passenger List:\n";
87                  for (const std::string& passenger : current->passengers) {
88                      std::cout << "Name: " << passenger << "\n";
89                  }
90                  return;
91              }
92              current = current->next;
93          }
94          std::cout << "Train " << trainNumber << " not found.\n";
95      }
96  };
97
98  int main() {
99      TrainBookingSystem tbs;
100
101     tbs.addTrain(101, "City A", "City B", 50);
102     tbs.addTrain(102, "City B", "City C", 40);
103
104     tbs.reserveSeat(101, "Passenger 1");
105     tbs.reserveSeat(101, "Passenger 2");
106
107     tbs.displayTrainInformation(101);
108
109     tbs.cancelReservation(101, "Passenger 1");
110
111     tbs.displayTrainInformation(101);
```

Output:

Output

Clear

```
/tmp/ItSvCw0Db7.o
Seat reserved for Passenger 1 on Train 101.
Seat reserved for Passenger 2 on Train 101.
Train Number: 101
Departure: City A
Destination: City B
Total Seats: 50
Available Seats: 48
Passenger List:
Name: Passenger 1
Name: Passenger 2
Reservation canceled for Passenger 1 on Train 101.
Train Number: 101
Departure: City A
Destination: City B
Total Seats: 50
Available Seats: 49
Passenger List:
Name: Passenger 2
```

3)

main.cpp Run

```cpp
#include <iostream>

struct MemoryBlock {
    int size;
    bool isAllocated;
    MemoryBlock* next;

    MemoryBlock(int _size) : size(_size), isAllocated(false), next(nullptr) {}
};

class MemoryManager {
private:
    MemoryBlock* head;

public:
    MemoryManager(int initialSize) : head(new MemoryBlock(initialSize)) {}

    void allocateMemory(int size) {
        MemoryBlock* current = head;
        while (current) {
            if (!current->isAllocated && current->size >= size) {
                if (current->size > size)
                {

                    MemoryBlock* newFreeBlock = new MemoryBlock(current->size -
                        size);
```

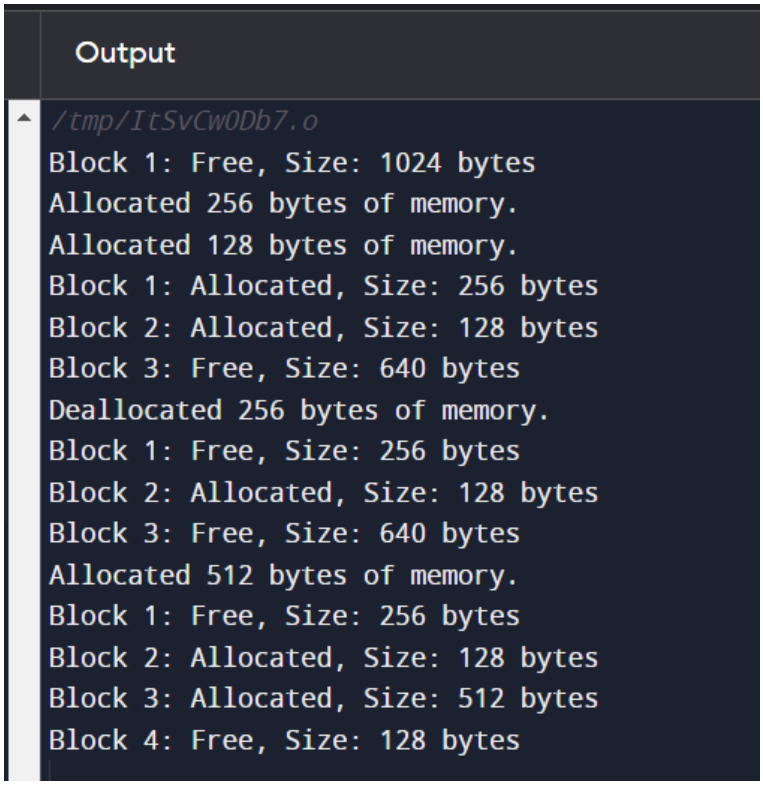
main.cpp Run

```cpp
                    MemoryBlock* newFreeBlock = new MemoryBlock(current->size -
                        size);
                    newFreeBlock->next = current->next;
                    current->next = newFreeBlock;
                }
                current->size = size;
                current->isAllocated = true;
                std::cout << "Allocated " << size << " bytes of memory.\n";
                return;
            }
            current = current->next;
        }
        std::cout << "Memory allocation failed for " << size << " bytes.\n";
    }

    void deallocateMemory(int size) {
        MemoryBlock* current = head;
        while (current) {
            if (current->isAllocated && current->size == size) {
                current->isAllocated = false;
                std::cout << "Deallocated " << size << " bytes of memory.\n";

                MemoryBlock* nextBlock = current->next;
                while (nextBlock && !nextBlock->isAllocated) {
                    current->size += nextBlock->size;
                    current->next = nextBlock->next;
```

```
                    current->size += nextBlock->size;
                    current->next = nextBlock->next;
                    delete nextBlock;
                    nextBlock = current->next;
                }
                return;
            }
            current = current->next;
        }
        std::cout << "Memory deallocation failed for " << size << " bytes.\n";
    }

    void displayMemoryStatus() {
        MemoryBlock* current = head;
        int blockNumber = 1;
        while (current) {
            std::string status = current->isAllocated ? "Allocated" : "Free";
            std::cout << "Block " << blockNumber << ": " << status << ", Size:
                " << current->size << " bytes\n";
            current = current->next;
            blockNumber++;
        }
    }
};

int main() {
```

```
            std::cout << "Block " << blockNumber << ": " << status << ", Size:
                " << current->size << " bytes\n";
            current = current->next;
            blockNumber++;
        }
    }
};

int main() {
    MemoryManager memoryManager(1024);

    memoryManager.displayMemoryStatus();

    memoryManager.allocateMemory(256);
    memoryManager.allocateMemory(128);
    memoryManager.displayMemoryStatus();

    memoryManager.deallocateMemory(256);
    memoryManager.displayMemoryStatus();

    memoryManager.allocateMemory(512);
    memoryManager.displayMemoryStatus();

    return 0;
}
```

OutPut:


Output
/tmp/ItSvCwODb7.o
Block 1: Free, Size: 1024 bytes
Allocated 256 bytes of memory.
Allocated 128 bytes of memory.
Block 1: Allocated, Size: 256 bytes
Block 2: Allocated, Size: 128 bytes
Block 3: Free, Size: 640 bytes
Deallocated 256 bytes of memory.
Block 1: Free, Size: 256 bytes
Block 2: Allocated, Size: 128 bytes
Block 3: Free, Size: 640 bytes
Allocated 512 bytes of memory.
Block 1: Free, Size: 256 bytes
Block 2: Allocated, Size: 128 bytes
Block 3: Allocated, Size: 512 bytes
Block 4: Free, Size: 128 bytes