Table of Contents

Table of figures

C++ Code:

```cpp
#include <iostream>
#include <string>
#include <iomanip>
#include <list>
#include <vector>
#include <algorithm>
using namespace std;

struct Student
{
    string name;
    int id;
    int grade;
    char category;
    double price;
};
struct Node
{
    Student data;
    Node* next;
};

class LinkedList
{
private:
    Node* head;

public:
    LinkedList() : head(NULL) {}

    void addStudent(const Student& newStudent)
    {
        Node* newNode = new Node;
        newNode->data = newStudent;
        newNode->next = NULL;

        if (head == NULL)
            head = newNode;
        else
        {
            Node* current = head;
            while (current->next != NULL)
                current = current->next;
            current->next = newNode;
        }
    }

    void display()
    {
        Node* current = head;
        cout << "-----------------------------------" << endl;
        cout << "|       Registered Students       |" << endl;
        cout << "-----------------------------------" << endl;
        cout << "| Name          | ID    | Grade  | Category | Price |" << endl;
        cout << "-----------------------------------" << endl;
        while (current != NULL)
        {
            Student student = current->data;
            cout << "| " << setw(13) << left << student.name << "| " << setw(6) << left << student.id
<< " | " << setw(5) << left << student.grade << " | " << setw(8) << left << student.category << " | "
<< setw(6) << left << student.price << " |" << endl;
            current = current->next;
        }
        cout << "-------------------------" << endl;
    }
};
struct PriorityQueueNode
{
    Student data;
    PriorityQueueNode* next;
};
```

```
class PriorityQueue
{
private:
    PriorityQueueNode* front;
    const int maxSize = 5;
    int size;

public:
    PriorityQueue() : front(NULL), size(0) {}

    void enqueue(const Student& newStudent)
    {
        if (size < maxSize)
        {
            PriorityQueueNode* newNode = new PriorityQueueNode;
            newNode->data = newStudent;
            newNode->next = NULL;

            if (front == NULL || newStudent.grade > front->data.grade)
            {
                newNode->next = front;
                front = newNode;
            }
            else
            {
                PriorityQueueNode* current = front;
                while (current->next != NULL && newStudent.grade <= current->next->data.grade)
                    current = current->next;
                newNode->next = current->next;
                current->next = newNode;
            }
            size++;
            cout << "Student " << newStudent.name << " added to the priority queue successfully." <<
endl;
        }
        else
            cout << "Priority queue is full. Cannot add more students." << endl;
    }

    bool isEmpty()
    {
        return size == 0;
    }

    Student frontStudent()
    {
        if (!isEmpty())
            return front->data;
        else
            throw "Priority queue is empty.";
    }

    void dequeue()
    {
        if (!isEmpty())
        {
            PriorityQueueNode* temp = front;
            front = front->next;
            delete temp;
            size--;
        }
    }

    void display()
    {
        PriorityQueueNode* current = front;
        cout << "-------------------------" << endl;
        cout << "| Priority Queue        |" << endl;
        cout << "-------------------------" << endl;
        cout << "| Name          | ID    | Grade |" << endl;
        cout << "-------------------------" << endl;
        while (current != NULL)
        {
            Student student = current->data;
            cout << "| " << setw(13) << left << student.name << "| " << setw(6) << left << student.id
<< " | " << setw(5) << left << student.grade << " | " << endl;
            current = current->next;
        }
        cout << "-------------------------" << endl;
    }
};
```

```
list<Student> registeredStudents;
int studentIDCounter = 0;
vector<string> registeredNames;

void Display_Registration()
{
    cout << "-----------------------------------" << endl;
    cout << "|       Registered Students       |" << endl;
    cout << "-----------------------------------" << endl;
    cout << "| Name          | ID    | Grade  | Category | Price |" << endl;
    cout << "-----------------------------------" << endl;
    for (const auto &student : registeredStudents)
    {
        cout << "| " << setw(13) << left << student.name << "| " << setw(6) << left << student.id << "
| " << setw(5) << left << student.grade << " | " << setw(8) << left << student.category << " | " <<
setw(6) << left << student.price << " |" << endl;
    }
    cout << "-------------------------" << endl;
}

void Print_System_Definition()
{
    cout << "         *******" << endl;
    cout << "*    Welcome to the Registration System!   *" << endl;
    cout << "         *******" << endl;
}

void assignCategory(Student& student)
{
    if (student.grade >= 290)
    {
        student.category = 'A';
        student.price = 22750.0;
    }
    else if (student.grade >= 190)
    {
        student.category = 'B';
        student.price = 25755.0;
    }
    else
    {
        student.category = 'C';
        student.price = 30750.0;
    }
}

void updateStudentGrade(list<Student>&students, const string& name, int newGrade)
{
    bool found = false;
    for (auto& student : students)
    {
        if (student.name == name)
        {
            student.grade = newGrade;
            if (student.grade >= 290)
            {
                student.category = 'A';
                student.price = 22750.0;
            }
            else if (student.grade >= 190)
            {
                student.category = 'B';
                student.price = 25755.0;
            }
            else
            {
                student.category = 'C';
                student.price = 30750.0;
            }
            cout << "Grade of student " << name << " has been successfully updated." << endl;
            found = true;
            break;
        }
    }
    if (!found)
        cout << "Student " << name << " not found." << endl;
}
```

```
void deleteStudent(list<Student> &students, const string &name)
{
    for (auto it = students.begin(); it != students.end(); ++it)
    {
        if (it->name == name)
        {
            int deletedId = it->id;
            it = students.erase(it);
            int newId = 1;
            for (auto &student : students)
                student.id = newId++;
            studentIDCounter--;
            cout << "Student " << name << " has been successfully deleted." << endl;
            return;
        }
    }
    cout << "Student " << name << " not found." << endl;
}

void searchStudent(list<Student> &students, const string &name)
{
    for (const auto &student : students)
    {
        if (student.name == name)
        {
            cout << "Student " << name << " found with ID " << student.id << ", grade " <<
student.grade << ", category " << student.category << ", and price $" << student.price << endl;
            return;
        }
    }
    cout << "Student " << name << " not found." << endl;
}


////////////////////////////////////////////////////////////////////////////////////////////////////
void Process_Registration(PriorityQueue &priorityQueue)
{
    if (!priorityQueue.isEmpty())
    {
        Student newStudent = priorityQueue.frontStudent();
        bool isStudentRegistered = false;
        for (const auto& name : registeredNames) {
            if (name == newStudent.name) {
                isStudentRegistered = true;
                break;
            }
        }
        if (!isStudentRegistered)
        {
            registeredNames.push_back(newStudent.name);
            newStudent.id = ++studentIDCounter;
            assignCategory(newStudent);//call func assignCategory
            registeredStudents.push_back(newStudent);
            cout << "Student " << newStudent.name << " with ID " << newStudent.id << ", category " <<
newStudent.category << ", and price $" << newStudent.price << " registered successfully." << endl;
        }
        else
            cout << "Student " << newStudent.name << " has already been registered." << endl;
        priorityQueue.dequeue();
    }
    else
        cout << "Priority queue is empty." << endl;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void RunMain()
{
    int choice;
    Print_System_Definition();
    PriorityQueue priorityQueue;
    do
    {
        cout << "Choose an option:" << endl;
        cout << "1. Add Student" << endl;
        cout << "2. Process Registration" << endl;
        cout << "3. Display Registered Students" << endl;
        cout << "4. Display Priority Queue" << endl;
        cout << "5. Delete student" << endl;
        cout << "6. Update student" << endl;
        cout << "7. Search student" << endl;
        cout << "8. Exit" << endl;
        cout << "Enter choice: ";
        cin >> choice;
        system("cls");
        switch (choice)
        {

        case 1:
        {
            Student newStudent;
            cout << "Enter student name: ";
            cin.ignore();
            getline(cin, newStudent.name);
            cout << "Enter student grade: ";
            cin >> newStudent.grade;
            if (newStudent.grade >= 200)
                priorityQueue.enqueue(newStudent);
            else
            {
                cout << "Student " << newStudent.name << " cannot be registered with grade less than
200." << endl;
            }
            break;
        }
        case 2:
        {
            cout << "Processing registration..." << endl;
            Process_Registration(priorityQueue);
            break;
        }
        case 3:
        {
            Display_Registration();
            break;
        }
        case 4:
        {
            priorityQueue.display();
            break;
        }
        case 5:
        {
            string nameToDelete;
            cout << "Enter student name to delete: ";
            cin.ignore();
            getline(cin, nameToDelete);
            deleteStudent(registeredStudents, nameToDelete);
            break;
        }
        case 6:
        {
            string nameToUpdate;
            int newGrade;
            cout << "Enter student name to update grade: ";
            cin.ignore();
            getline(cin, nameToUpdate);
            cout << "Enter new grade: ";
            cin >> newGrade;
            updateStudentGrade(registeredStudents, nameToUpdate, newGrade);
            break;
        }
        case 7:
        {
            string nameToSearch;
            cout << "Enter student name to search: ";
            cin.ignore();
            getline(cin, nameToSearch);
            searchStudent(registeredStudents, nameToSearch);
            break;
        }
        case 8:
        {
            cout << "Exiting program..." << endl;
            break;
        }
        default:
        {
            cout << "Invalid choice. Please try again." << endl;
        }
        }
    }
    while (choice != 8);
}

int main()
{
    RunMain();
    return 0;
}
```

Pseudo Code:

Create structure named Student with fields for the student's name, ID, grade,price ,and category

Create class priority queue to store students based on their grades, ordered from highest to lowest

Checks if the queue is empty

Adds a new student to the queue if the maximum capacity has not been exceeded.

Removes the top student from the queue

Retrieves the top student in the queue

Shows the contents of the queue

Create a list to store registered students, using a linked list instead of a vector

Create function named ShowRegisteredStudents to display all registered students

PrintSystemDefinition print the system definition

UpdateStudentGrade update a student's grade

DeleteStudent remove a student from the registered students list

SearchStudent search for a student in the registered students list

RegistrationProcess that handles the process of registering students from the priority queue to the registered students list

We have a main function that starts the program by calling RunMain

the user will choice one option from 8 (Add Student,Process Registration,Display

Registred Students,Display Priority Queue,Delete student,Update student,Search student,Exit)

end program

Flow Chart:

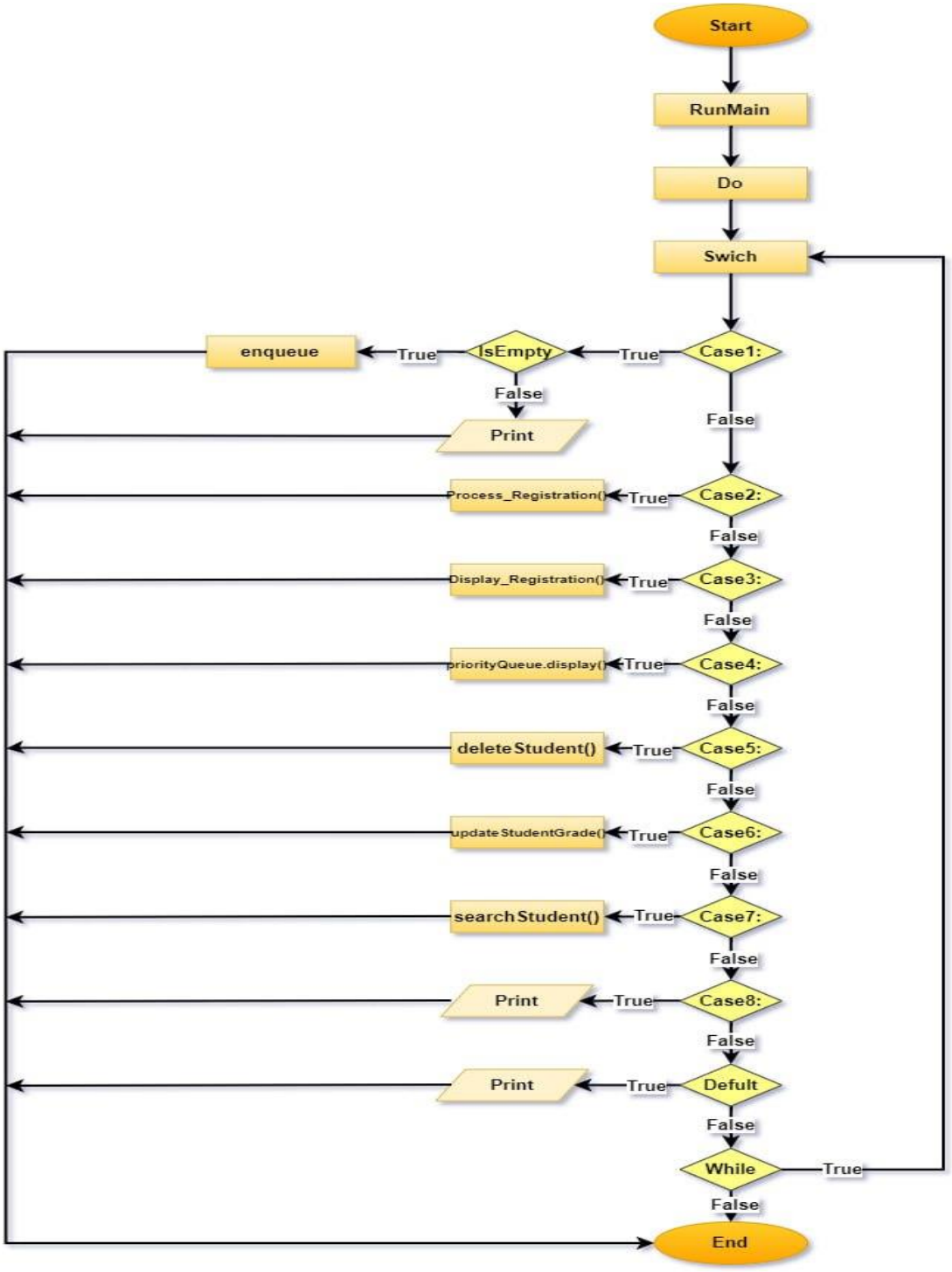

***Figure 1**:flow chart*

GUI:

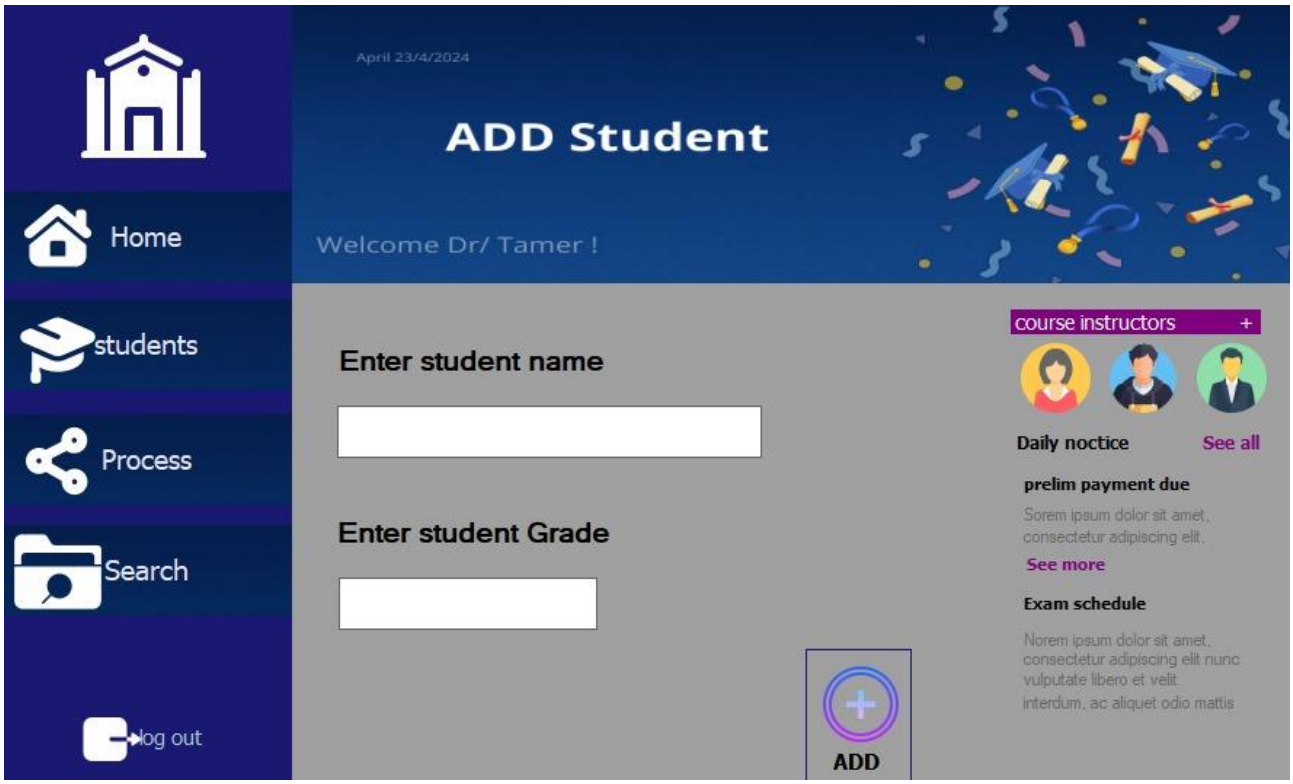

Figure2:GUI