UTM UNIVERSITI TEKNOLOGI MALAYSIA **FACULTY OF COMPUTING** UTM Johor Bahru

Semester 2 2023/2024

Subject : Programming Technique II

Section : 08

Task : Project Final

Group Member

	Name	**Matric Number**
1	YONG JING WEN	A23CS0202
2	SIOH YING YI	A23CS0180
3	YAP KAR YING	A23CS0200

Section A: Project Description

1. General Idea

The scholarship application system typically involves students who need financial help to support their education. It is also used to award the students who get a good result. Applicants usually need to submit their application form, the evidence of academic results and the type of scholarship they want to apply. For merit based scholarship, the selection between the applicants will be chosen based on the academic performance while for the need-based scholarship academic performance and family income will be used to determine whether they qualify for this scholarship or not.The system aims to provide the reward for good performance students and also give opportunity for students to complete their studies regardless of financial constraints .

2. Objectives

- Assigns scholarship that fulfill the requirements based on student's information (based on faculty/ course/ family income group)
- To encourage students to pursue higher education by providing financial support.
- To give awards for students who get a good result.

3. How to use the system

a. User registration / login

New users need to sign in by using the basic information such as the name,age, email, student ID, contact number,family income, CGPA, course fee, year of study in the scholarship application system to create an account.

b. Home page

After log in to the scholarship application system, students will come to the home page that include 2 options which are merit based scholarship and need based scholarship. Students are required to choose which scholarship they wish to apply for.

- Merit based scholarship

  Students are able to see the requirement of the merit based scholarship. The scholarship will be awarded based on the CGPA we achieve at the university .We have opened three categories of the merit-based scholarship. Different CGPA will qualify for different amounts of the scholarship.The categories are as shown below:

  CGPA = 4.0

  Total amount of scholarship received by the applicant is equal to the applicant’s course fee.

CGPA >= 3.76
Total amount of scholarship received by the applicant is 75 % from the applicant's course fee.

CGPA >= 3.0
Total amount of scholarship received by the applicant is 50 % from the applicant's course fee.

- Need based scholarship
  Students are able to see the requirement of need based scholarship. Need Based scholarship will be given based on the current CGPA and family income.

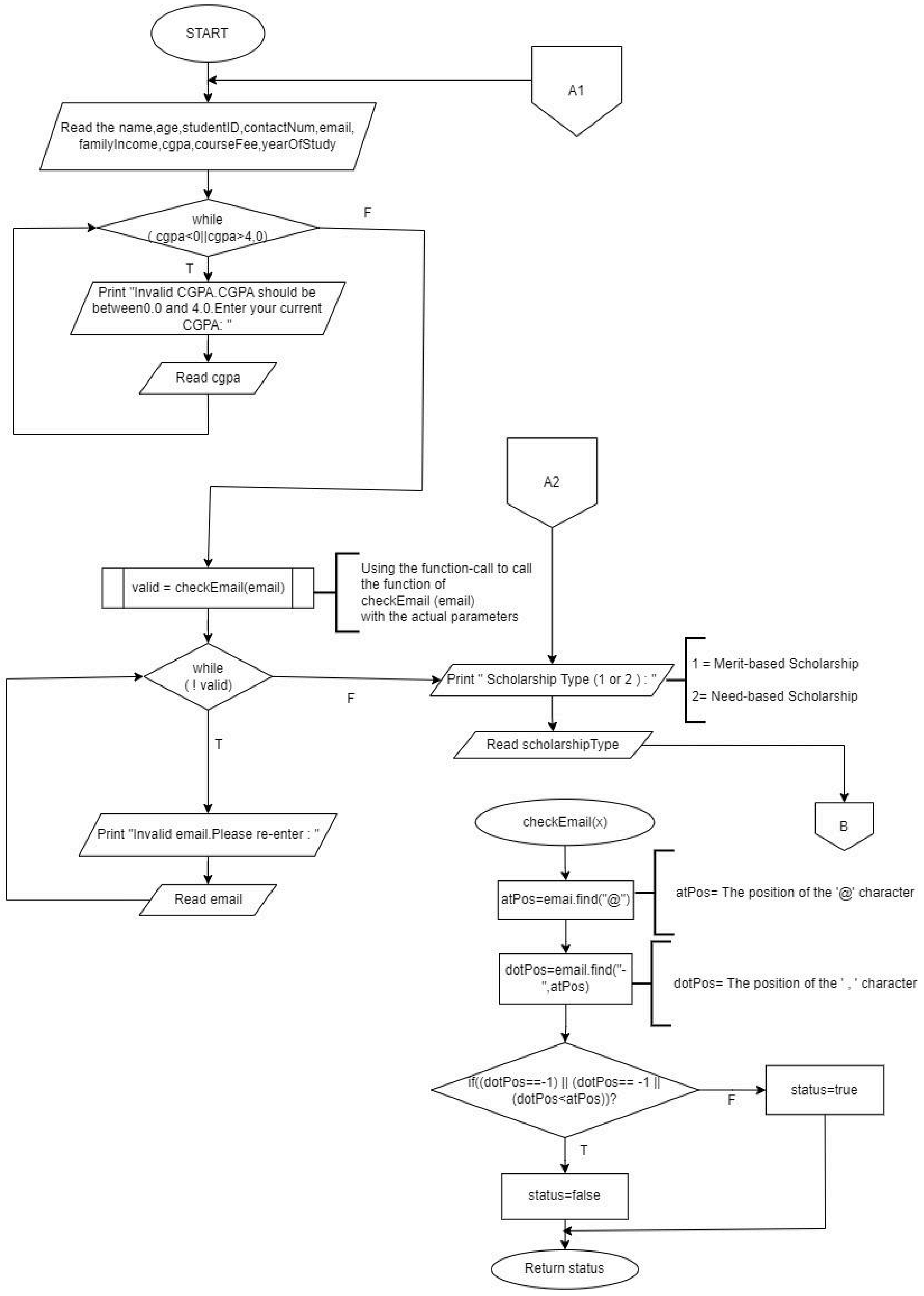
  Family income < 4849 && CGPA >= 3.0
  Total amount of scholarship received by the applicant is equal to the applicant's course fee.

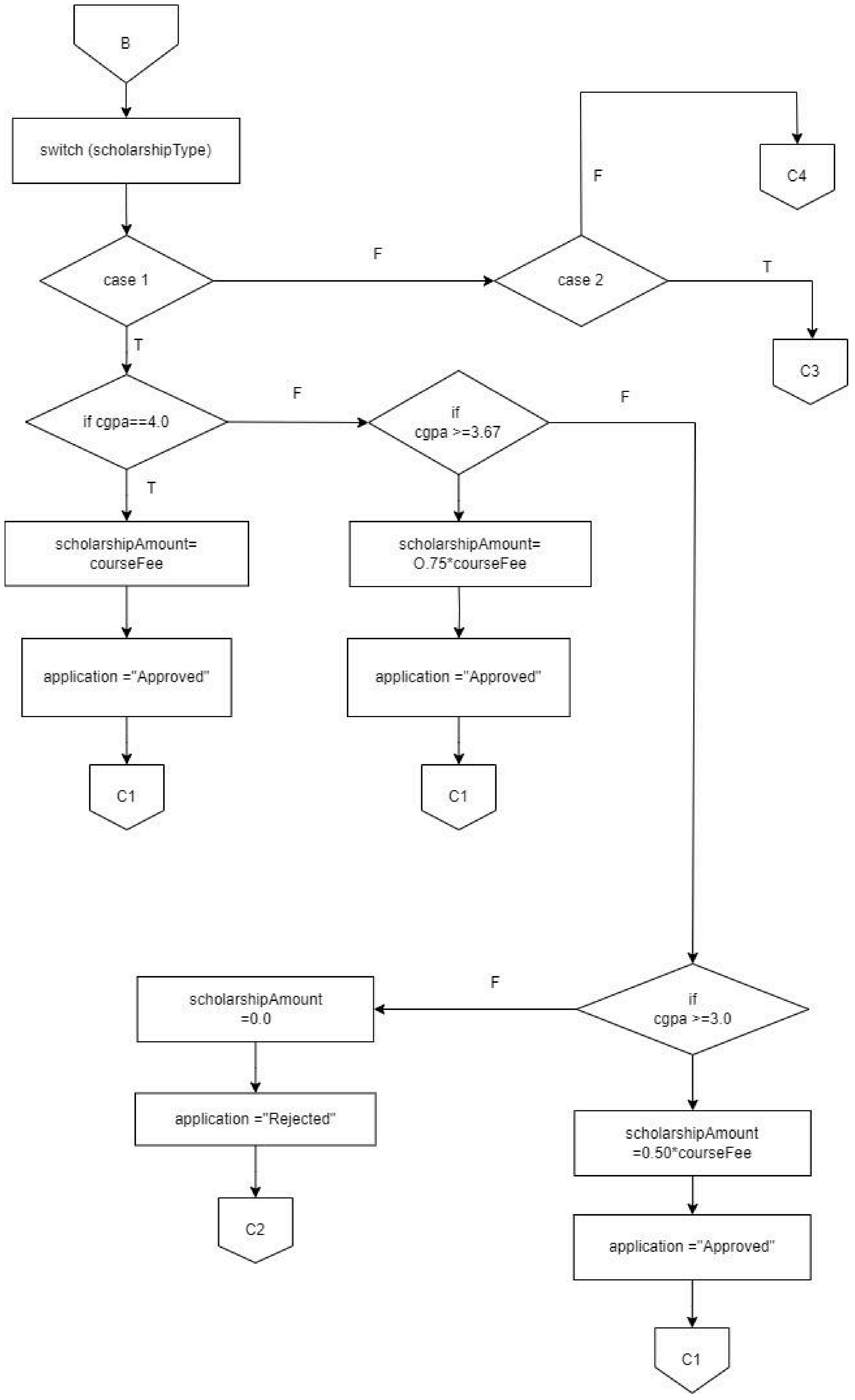
c. Application review
After key in all the information of the applicant and the scholarship's type that the applicant wishes to apply, the system will check whether the information key in by the students are accurate. Any inaccurate data will cause the application of the student to be withdrawn.

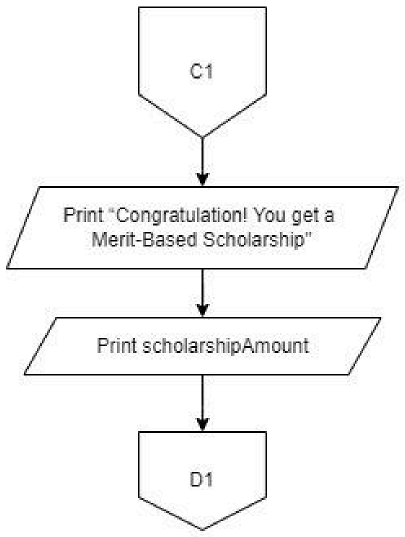
4. Reporting

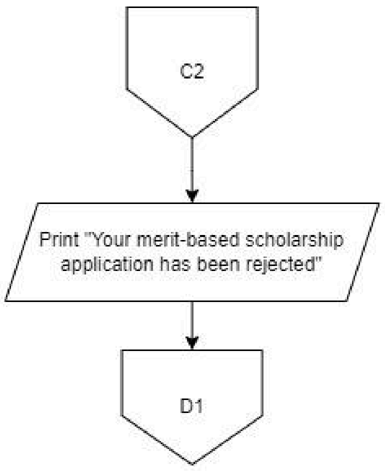
The users would get different outputs based on the different information they key in. The merit based scholarship will be given to the students based on their CGPA. However, the need based scholarship will be given to students based on family income group especially B40 and their academic performance.

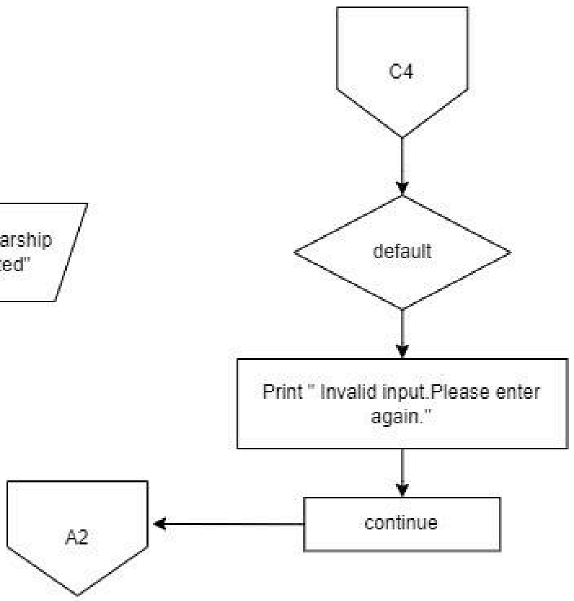
Flow Chart

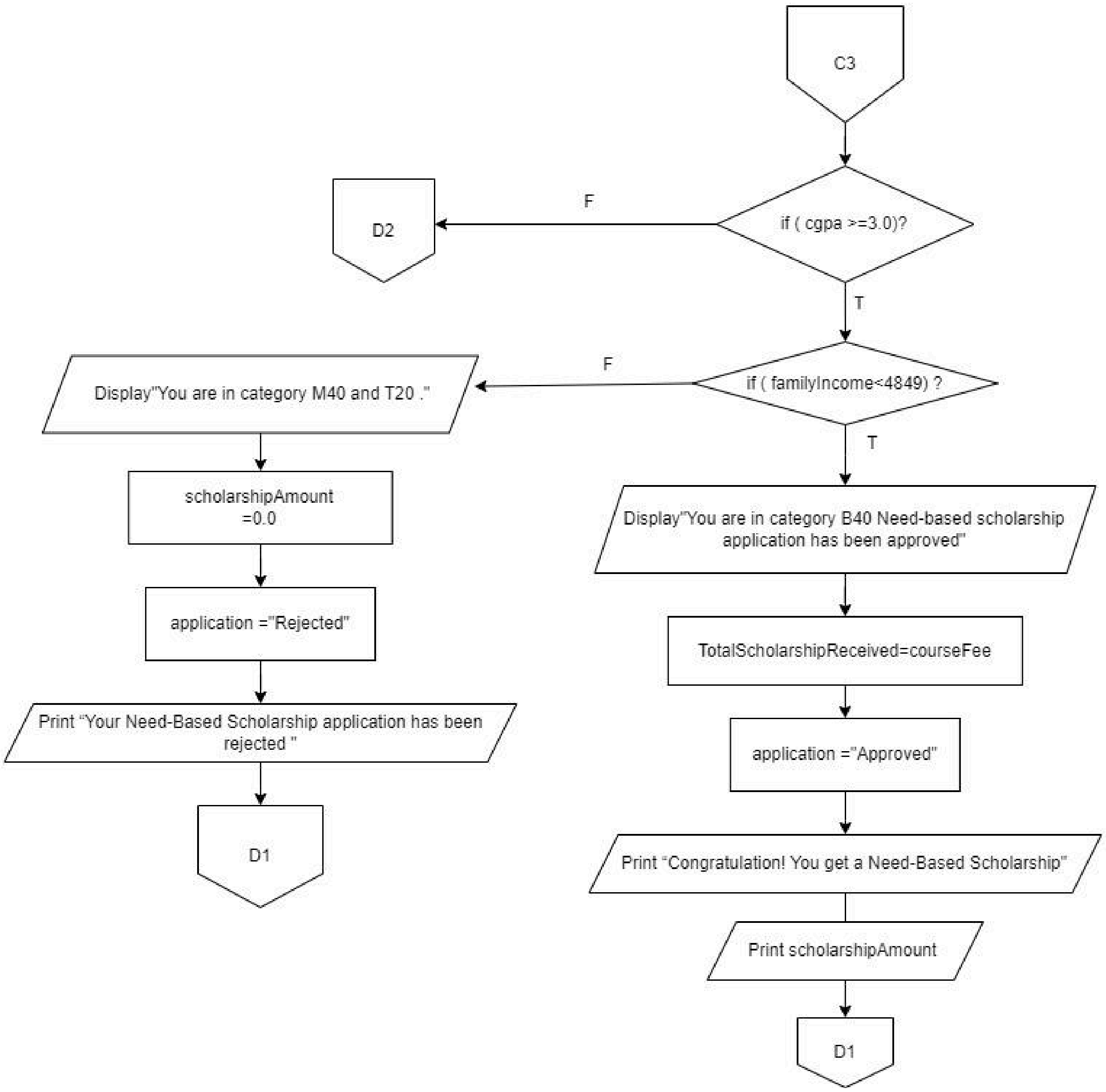

START
A1
Read the name,age,studentID,contactNum,email, familyIncome,cgpa,courseFee,yearOfStudy
while (cgpa<0||cgpa>4.0)
F
T
Print "Invalid CGPA.CGPA should be between0.0 and 4.0.Enter your current CGPA: "
Read cgpa
A2
valid = checkEmail(email)
Using the function-call to call the function of checkEmail (email) with the actual parameters
while (! valid)
F
T
Print " Scholarship Type (1 or 2) : "
1 = Merit-based Scholarship
2= Need-based Scholarship
Read scholarshipType
B
Print "Invalid email.Please re-enter : "
Read email
checkEmail(x)
atPos=emai.find("@")
atPos= The position of the '@' character
dotPos=email.find("-",atPos)
dotPos= The position of the ' , ' character
if((dotPos==-1) || (dotPos== -1 || (dotPos<atPos))?
F
status=true
T
status=false
Return status

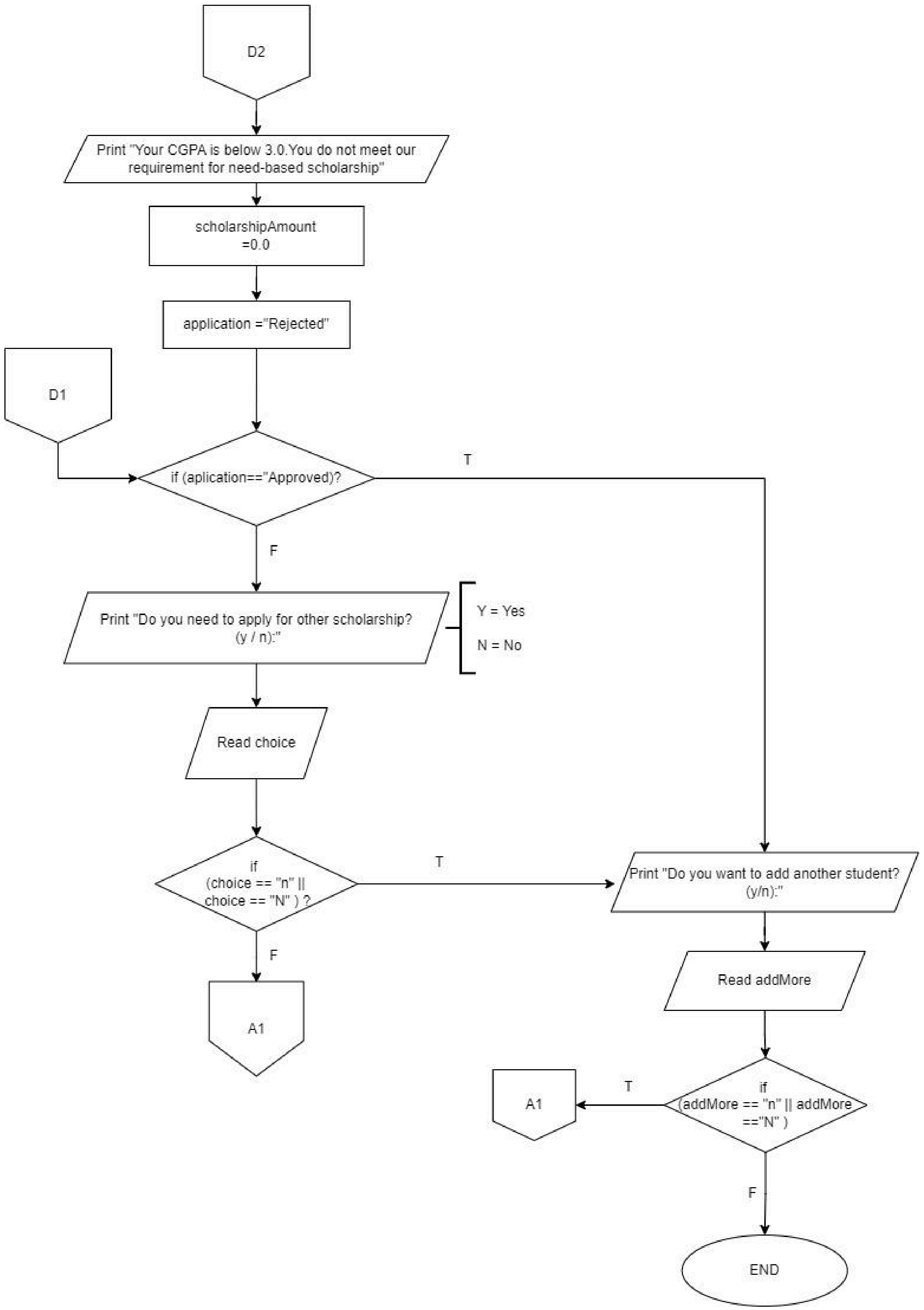

B
switch (scholarshipType)
case 1
F
case 2
F
C4
T
C3
T
if cgpa==4.0
F
if cgpa >=3.67
F
T
scholarshipAmount= courseFee
scholarshipAmount= O.75*courseFee
application ="Approved"
application ="Approved"
C1
C1
F
scholarshipAmount =0.0
if cgpa >=3.0
application ="Rejected"
scholarshipAmount =0.50*courseFee
C2
application ="Approved"
C1


C1
Print "Congratulation! You get a Merit-Based Scholarship"
Print scholarshipAmount
D1


C2
Print "Your merit-based scholarship application has been rejected"
D1


C4
default
Print " Invalid input.Please enter again."
continue
A2


C3
if (cgpa >=3.0)?
F
D2
T
if (familyIncome<4849) ?
F
Display"You are in category M40 and T20 ."
scholarshipAmount =0.0
application ="Rejected"
Print "Your Need-Based Scholarship application has been rejected "
D1
T
Display"You are in category B40 Need-based scholarship application has been approved"
TotalScholarshipReceived=courseFee
application ="Approved"
Print "Congratulation! You get a Need-Based Scholarship"
Print scholarshipAmount
D1


D2
Print "Your CGPA is below 3.0.You do not meet our requirement for need-based scholarship"
scholarshipAmount =0.0
application ="Rejected"
D1
if (aplication=="Approved)?
T
F
Print "Do you need to apply for other scholarship? (y / n):"
Y = Yes
N = No
Read choice
if (choice == "n" || choice == "N") ?
T
F
A1
Print "Do you want to add another student? (y/n):"
Read addMore
if (addMore == "n" || addMore =="N")
T
A1
F
END

Validate Email

- checkEmail ()
  This is a function call to check whether the entered email address is in a valid format. The email of the student (string data type) as a parameter of the function so that the email can pass to function and undergo validation.
- atPos = email.find("@")
  This is to find the position of ‘@’ character in an email.
- dotPos = email.find(".", atPos)
  This is to find the position of the ‘.’ character after the ‘@” character in the email address.
- if ((dotPos == -1) || (atPos == -1) || (dotPos < atPos))?
  This is to ensure that the email contains both ‘.’ and ‘@’ characters and also make sure that ‘.’appears after ‘@’. If the condition does not meet, the status will return false, meaning that the email is in a invalid format. Else, if the condition is met, the status will return to true, indicating that the email is in a valid format. If the email is invalid, the system will request the applicant to re-enter the email. This is to ease the process for further communication with the applicant.

Read Scholarship Type

- Print "Scholarship Type (1 or 2): 1 = Merit-based Scholarship, 2 = Need-based Scholarship"
  This is to prompt the applicant to enter the scholarship type that they want to apply. We use an integer to represent the type of scholarship as we want to ease the users to key in the data. On the other hand, we also want to ensure the data can be read by the compiler accurately to reduce error.

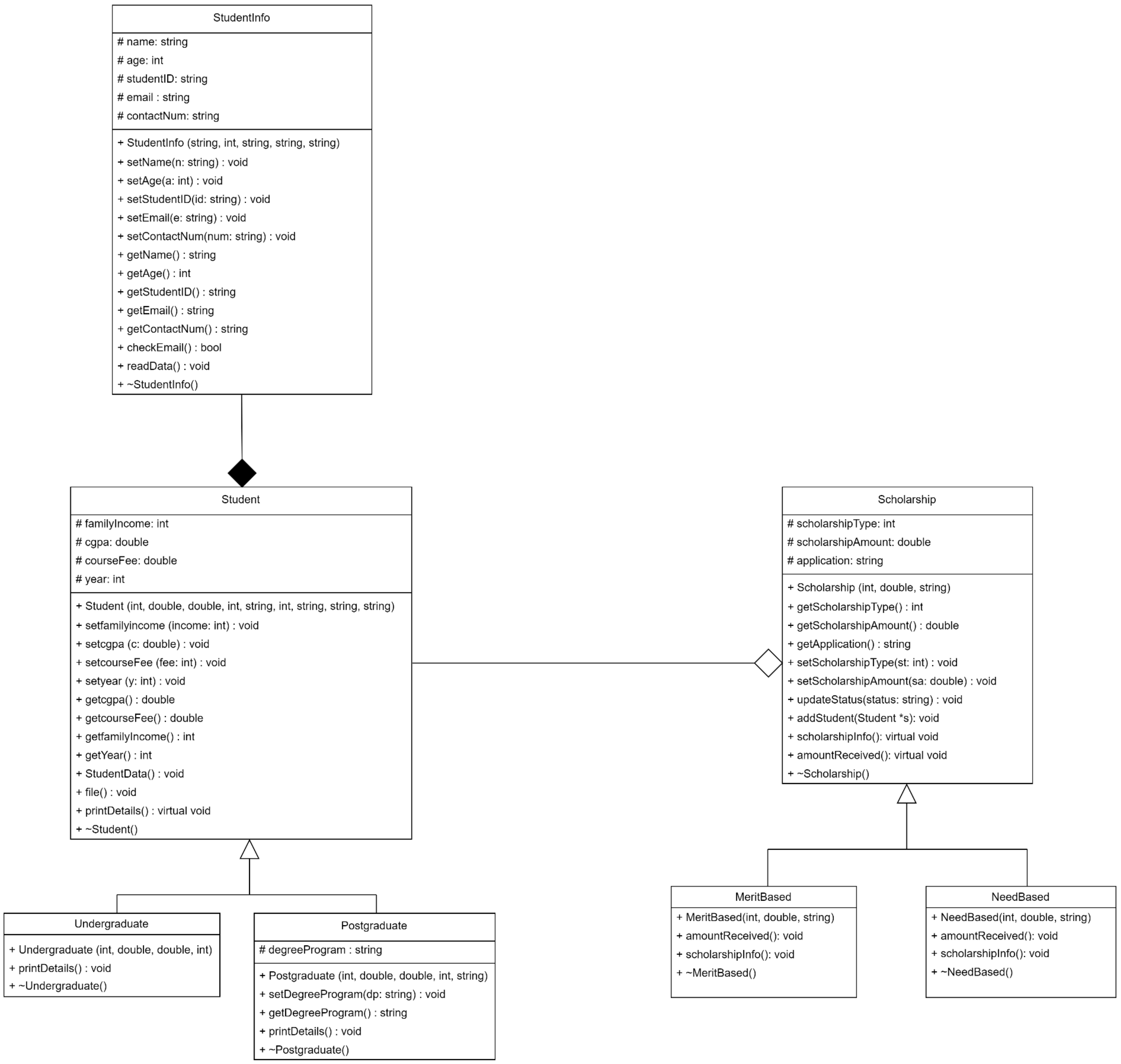
Amount of scholarship that assigned to Merit-Based Scholarship applicants

- If the applicant chooses a Merit-based scholarship, they are required to key in their CGPA for the system to evaluate.
- If the user gets a CGPA of 4.0 the applicant will be awarded the full course fee as the scholarship amount.
- If the CGPA is in range 3.67 to 3.99. The applicant will be awarded a scholarship with 75% of the course fee.
- On the other hand, if the applicant gets a CGPA in the range of 3.66 to 3.01. They will be awarded with 50% of course fee as scholarship amount.
- If the applicant gets a CGPA less or equal 3.00, the system will reject the application as the applicant does not meet the minimum requirement.

Apply for another scholarship

- Only the applicants who fail to receive the scholarship can apply for another scholarship type.
- Print "Do you need to apply for another scholarship's type?" (Y or N)
  The system will read the applicants' responses.
- If the applicant presses 'Y' means that they wish to apply for another scholarship. Then, the process will loop back to allow for another application.
- If the applicant presses 'N' means that they reject to apply for another scholarship. Then, the process will be over and the information of the student would not be stored in the array. This step is important because we wish the applicants to have a higher chance to receive the financial aid.

Class Diagram


StudentInfo
# name: string
# age: int
# studentID: string
# email : string
# contactNum: string
+ StudentInfo (string, int, string, string, string)
+ setName(n: string) : void
+ setAge(a: int) : void
+ setStudentID(id: string) : void
+ setEmail(e: string) : void
+ setContactNum(num: string) : void
+ getName() : string
+ getAge() : int
+ getStudentID() : string
+ getEmail() : string
+ getContactNum() : string
+ checkEmail() : bool
+ readData() : void
+ ~StudentInfo()
Student
# familyIncome: int
# cgpa: double
# courseFee: double
# year: int
+ Student (int, double, double, int, string, int, string, string, string)
+ setfamilyincome (income: int) : void
+ setcgpa (c: double) : void
+ setcourseFee (fee: int) : void
+ setyear (y: int) : void
+ getcgpa() : double
+ getcourseFee() : double
+ getfamilyIncome() : int
+ getYear() : int
+ StudentData() : void
+ file() : void
+ printDetails() : virtual void
+ ~Student()
Scholarship
# scholarshipType: int
# scholarshipAmount: double
# application: string
+ Scholarship (int, double, string)
+ getScholarshipType() : int
+ getScholarshipAmount() : double
+ getApplication() : string
+ setScholarshipType(st: int) : void
+ setScholarshipAmount(sa: double) : void
+ updateStatus(status: string) : void
+ addStudent(Student *s): void
+ scholarshipInfo(): virtual void
+ amountReceived(): virtual void
+ ~Scholarship()
Undergraduate
+ Undergraduate (int, double, double, int)
+ printDetails() : void
+ ~Undergraduate()
Postgraduate
# degreeProgram : string
+ Postgraduate (int, double, double, int, string)
+ setDegreeProgram(dp: string) : void
+ getDegreeProgram() : string
+ printDetails() : void
+ ~Postgraduate()
MeritBased
+ MeritBased(int, double, string)
+ amountReceived(): void
+ scholarshipInfo(): void
+ ~MeritBased()
NeedBased
+ NeedBased(int, double, string)
+ amountReceived(): void
+ scholarshipInfo(): void
+ ~NeedBased()

Section B: Implementation of the Concepts

1. **Objects and Classes**

   Objects:
   - Applicant
     It represents the students who apply for the scholarship.
     It contains an array that holds students' personal details, academic information and finance status.
     The array will only hold the information of the applicant that got the scholarship.

   - Scholarship
     It represents the scholarship awarded to the students.
     It contains an array that holds details on the type of scholarship that students get and coverage percentage.

   Classes:
   a. Student
      Attributes:
      - familyIncome: int
      - cgpa: double
      - courseFee: double
      - year: int
      - s1 : StudentInfo

      Method:
      - Student(int, double, double, int, string, int, string, string, string)
      - setfamilyincome(int): void
      - setcgpa(double): void
      - setcourseFee(int): void
      - setyear(int): void
      - getcgpa(): double
      - getcourseFee(): double
      - getfamilyIncome(): int
      - getYear(): int
      - StudentData():void
      - file(): void
      - printDetails()=0: virtual void
      - ~Student()

   b. StudentInfo
      Attributes:
      - name: string

  - age: int
  - studentID: string
  - email: string
  - contactNum: string

  Method:
  - StudentInfo (string, int, string, string, string)
  - setName (string): void
  - setAge (int): void
  - setStudentID (string): void
  - setEmail (string): void
  - setContactNum(string): void
  - getName(): string
  - getAge(): int
  - getStudentID(): string
  - getEmail(): string
  - getContactNum(): string
  - checkEmail(): bool
  - readData(): void
  - ~StudentInfo ()

c. Scholarship

   Attributes:
   - scholarshipType: int
   - scholarshipAmount: double
   - application: string
   - *student : Student

   Method:
   - Scholarship (int, double, string)
   - getScholarshipType() : int
   - getScholarshipAmount() : double
   - getApplication() : string
   - setScholarshipType(int) : void
   - setScholarshipAmount(double) : void
   - updateStatus(string): string
   - addStudent(Student): void
   - scholarshipInfo(): virtual void
   - amountReceived(): virtual void
   - ~Scholarship()

d. Undergraduate
   Method :
   - Undergraduate (int, double, double, int)
   - printDetails(): void
   - ~Undergraduate()

e. Postgraduate
   Attributes:
   - degreeProgram: string

   Method :
   - Postgraduate(int, double, double, int, string)
   - setDegreeProgram(string) : void
   - getDegreeProgram() : string
   - printDetails(): void
   - ~Postgraduate()

f. MeritBased
   Method:
   - MeritBased(int, double, string)
   - amountReceive(): void
   - scholarshipInfo(): void
   - ~MeritBased()

g. NeedBased
   Method:
   - NeedBased(int, double, string)
   - amountReceive(): void
   - scholarshipInfo(): void
   - ~NeedBased()

2. Object-Oriented Programming Concepts

Encapsulation:

- Combining the attributes and methods of class StudentInfo in one package.

Association:

- Relationships between Student and Scholarship could be done by aggregation. This is because a Student can have a Scholarship but Scholarship does not necessarily need to have a Student.
- Relationships between Student and StudentInfo could be done by composition. This is because a Student consists of StudentInfo, when a Student object is created, the StudentInfo is also created.

Inheritance:

- Base Class : Student
  Derived Class: Undergraduate, Postgraduate

  Undergraduate and Postgraduate class inherits from Student class because an undergraduate student and postgraduate student are a special type of student. They are having a "is a" relationship as an undergraduate student is a student and the postgraduate student is also a student. By inheriting from students, both undergraduate and postgraduate classes can reuse common attributes and methods from student class while they can also create additional methods to specify their own class.

- Base Class : Scholarship
  Derived Class: MeritBased, NeedBased

  NeedBased and MeritBased classes inherit from Scholarship because need-based and merit-based are both specific types of scholarship. Three of the classes have the same criteria which are related to financial need. Thus, by inheriting, both MeritBased and NeedBased classes can use the same attribute and method of Scholarship class and they can add some method to specify their class. This ensures different types of scholarships are organized under a common hierarchy and ensure that they are easy to manage.

Polymorphism:

- The Student and Scholarship classes have virtual functions.Specifically, printDetails() in Student class while scholarshipInfo() and amountReceived() in Scholarship class are declared as virtual functions.
- The Student and Scholarship classes are abstract classes because they have pure virtual functions.This is indicated by '=0' syntax in the virtual function declaration.
- Polymorphism is utilized when dealing with the Student pointer to *stu and Scholarship pointer to *_scholarship.According to the type of student (Undergraduate or Postgraduate) or scholarship (MeritBased or NeedBased), different implementations of the virtual functions are called.

Array of Object:

- Student Array
  Student *stud[MAXSTUDENT] declares an array of pointers to Student objects. Each element of the array can hold the address of a Student object or any of its derived classes. In the main function, objects of Undergraduate and Postgraduate are dynamically allocated and stored in this array based on user input.

- Scholarship Array
  Scholarship *_scholarship[MAXSTUDENT] declares an array of pointer to Scholarship objects. Each element of the array can hold the address of a Scholarship object or any of its derived classes. In the main function, objects of MeritBased and NeedBased are dynamically allocated and stored in this array based on user input.

Section C : Code

🟨	Encapsulation
🟩	Aggregation
🟥	Composition
🟦	Inheritance
🟪	Polymorphism
🟧	Array of Object

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

#define MAXSTUDENT 100
class StudentInfo{
    protected:
      string name,studentID,email,contactNum;
      int age;

    public:
      StudentInfo(string n="",int a=0,string id="",string e="",string contact=""){
        name=n;
        age=a;
        studentID=id;
        email=e;
        contactNum=contact;
      }

      void setName(string n)
      {name=n;}

      void setAge(int a)
      {age=a;}

      void setStudentID(string id)
      {studentID=id;}



      void setEmail(string e)
      {email=e;}

      void setContactNum(string num)
      {contactNum=num;}

      string getName()
      {return name;}

      int getAge()
      {return age;}

      string getStudentID()
      {return studentID;}

     string getEmail()
     {return email;}

      string getContactNum()
      {return contactNum;}

      //This function is to check whether the email is valid or not
      bool checkEmail() {
        int atPos = email.find("@");
        int dotPos = email.find(".", atPos);
        if (atPos == -1  || atPos == -1 || atPos > dotPos) {
            return false;
        }
        return true;
      }

      void readData() {
        cout << "Enter your name: ";
        getline(cin, name);
        cout << "Enter your age: ";
        cin >> age;
        cin.ignore();
        cout << "Enter your student ID: ";
        cin >> studentID;
        cout << "Enter your contact number: ";
        cin >> contactNum;
        cout << "Enter your email: ";
        cin >> email;
        while (!checkEmail()) {
            cout << "Invalid email. Please re-enter: ";
            cin >> email;
        }
      }

    ~StudentInfo()
    {}
};

class Student{
    protected:
      int familyincome,year;
      double cgpa,courseFee;



      StudentInfo s1;

    public:
       Student(int income=0,double c=0.0,double fee=0.0,int y=0,string name="",int age=0,string
id="",string email="",string contact=""):
       s1(name,age,id,email,contact)
       {
          familyincome=income;
          cgpa=c;
          courseFee=fee;
          year=y;
       }

       class NegativeSign{};

       void setfamilyincome(int income)
       {familyincome=income;}

       void setcgpa(double c)
       {cgpa=c;}

       void setcourseFee(int fee)
       {courseFee=fee;}

       void setyear(int y)
       {year=y;}

       double getcgpa()
       {return cgpa;}
```

```
   double getcourseFee()
   {return courseFee;}

   int getfamilyIncome()
   {return familyincome;}

   int getYear()
   {return year;}

void StudentData() {
    s1.readData();
    cout << "Enter your family income (RM): ";
    cin >> familyincome;
    if (familyincome<0)
    throw NegativeSign();
    cout << "Enter your current CGPA: ";
    cin >> cgpa;

    //Check the validate CGPA
    while (cgpa < 0 || cgpa > 4.0) {
       cout << "Invalid CGPA. CGPA should be between 0.0 and 4.0." << endl;
       cout << "Enter your current CGPA: ";
       cin >> cgpa;
    }
    if(cgpa<0)
    throw NegativeSign();
    cout << "Enter your course fee (RM): ";
    cin >> courseFee;


    if (courseFee<0)
    throw NegativeSign();
    cout << "Enter your year of study: ";
    cin >> year;
    if (year<0)
    throw NegativeSign();
}

//Store the student's data to a file named "data.txt".
void file(){

    ofstream out;
    out.open("data.txt", ios::app); // Open in append mode
    if (!out) {
        cerr << "Error opening file!" << endl;
        return;
    }

    string upperName;
    upperName = s1.getName();

    //Convert the name to uppercase.
    for(int i=0; i<upperName.length();++i){
      upperName[i]=toupper(upperName[i]);
    }

    s1.setName(upperName);

    out << "Name: " << s1.getName()<< endl;
```

```
        out << "Age: " << s1.getAge() << endl;
        out << "Student ID: " << s1.getStudentID() << endl;
        out << "Contact Number: " << s1.getContactNum() << endl;
        out << "Email: " << s1.getEmail() << endl;
        out << "Enter your family income (RM): "<<familyincome<<endl;
        out << "Enter your current CGPA: "<<cgpa<<endl;
        out << "Enter your course fee (RM): "<<courseFee<<endl;
        out << "Enter your year of study: "<<year<<endl;
        out << endl;

        out.close();
    }

    virtual void printDetails()=0;

    ~Student(){}
};



class Undergraduate : public Student {
  public:
    Undergraduate(int income = 0, double c = 0.0, double fee = 0.0, int y = 0) : Student(income, c,
fee, y) {}  //inheritance

    void printDetails() {
        cout << endl << "Student Details:" << endl;
        cout << "Name: " << s1.getName() << endl;
        cout << "Age: " << s1.getAge() << endl;



        cout << "Student ID: " << s1.getStudentID() << endl;
        cout << "Contact Number: " << s1.getContactNum() << endl;
        cout << "Email: " << s1.getEmail() << endl;
        cout << "Family income (RM): " << familyincome << endl;
        cout << "Current CGPA: " << cgpa << endl;
        cout << "Course fee (RM): " << courseFee << endl;
        cout << "Year of study: " << year << endl;
    }

    ~Undergraduate(){}
};

class Postgraduate : public Student{
  protected:
    string degreeProgram;

  public:
    Postgraduate(int income = 0, double c = 0.0, double fee = 0.0, int y = 0, string dP="") :
Student(income, c, fee, y), degreeProgram(dP) {}

    string getDegreeProgram()
    {return degreeProgram;}

    void setDegreeProgram(string dp)
    {degreeProgram=dp;}

    void printDetails(){
        cout << endl << "Student Details:" << endl;
        cout << "Name: " << s1.getName() << endl;
        cout << "Age: " << s1.getAge() << endl;
        cout << "Student ID: " << s1.getStudentID() << endl;
        cout << "Contact Number: " << s1.getContactNum() << endl;
        cout << "Email: " << s1.getEmail() << endl;
        cout << "Family income (RM): " << familyincome << endl;
        cout << "Current CGPA: " << cgpa << endl;
        cout << "Course fee (RM): " << courseFee << endl;
        cout << "Year of study: " << year << endl;
        cout << "Degree Program: " << degreeProgram << endl;
    }

    ~Postgraduate(){}
};

class Scholarship
{
  protected:
  int scholarshipType;
  double scholarshipAmount;
  string application;
  Student *student;

  public:
  Scholarship(int st=1, double sAmount=0.00, string
a="Pending"):scholarshipType(st),scholarshipAmount(sAmount),application(a)
  {student = NULL;}

  int getScholarshipType()
  {return scholarshipType;}



  double getScholarshipAmount()
  {return scholarshipAmount;}

  string getApplication()
  {return application;}

  void setScholarshipType(int st)
  {scholarshipType=st;}

  void setScholarshipAmount(double sa)
  {scholarshipAmount=sa;}

  void updateStatus(string status)
  {application=status; }

  void addStudent(Student *s)
  {student=s;}

  virtual void scholarshipInfo()=0;

  virtual void amountReceived()=0;

  ~Scholarship(){}
};

class MeritBased:public Scholarship
{
  public:
```

```
  MeritBased(int sT = 1,double amount=0.0, string aS = "Pending"):Scholarship(sT,amount,aS){}

  //Calculates the scholarship amount a student is eligible for based on their CGPA.
  void amountReceived(){
    if (student->getcgpa() == 4.0) {
      scholarshipAmount=student->getcourseFee() ;
      application="Approved";
    }
    else if (student->getcgpa() >= 3.67) {
      scholarshipAmount= 0.75 * student->getcourseFee();
      application="Approved";
    }
    else if (student->getcgpa() >= 3.0) {
      scholarshipAmount= 0.5 * student->getcourseFee();
      application="Approved";
    }
    else {
      scholarshipAmount= 0.0;
      application="Rejected";
    }
  }

  void scholarshipInfo(){
    if (application == "Approved"){
      cout << endl << "Congratulation! You get a Merit-Based Scholarship"<< endl;
      cout << "The amount that you get is RM " << scholarshipAmount << endl;
    }
    else
      cout << endl << "Your merit-based scholarship application has been rejected" << endl;



    }

  ~MeritBased(){}
};

class NeedBased:public Scholarship
{
  public:
  NeedBased(int sT = 2, double amount=0.0,string aS = "Pending"):Scholarship(sT,amount,aS){}

  //Calculates the scholarship amount a student is eligible based on the student's CGPA and family
income.
  void amountReceived() {
    if(student->getcgpa()>=3.0){

      if(student->getfamilyIncome() < 4849){
        cout << endl << "You are in category B40. Need-based scholarship application has been
approved."<<endl;
        scholarshipAmount=student->getcourseFee();
        application="Approved";
      }

      else{
        cout << endl << "You are in category M40 and T20."<<endl;
        scholarshipAmount= 0.0;
        application="Rejected";
        }
      }
    else{
      cout << endl << "Your CGPA is below 3.0. You do not meet our requirement for need-based
scholarship."<< endl;
      scholarshipAmount=0.0;
      application= "Rejected";
      }
    }

  void scholarshipInfo(){
    if (application == "Approved"){
      cout << endl << "Congratulation! You get a Need-Based Scholarship"<< endl;
      cout << "The amount that you get is RM " << scholarshipAmount << endl;
    }
    else
    cout << "Your need-based scholarship application has been rejected." << endl;
  }

  ~NeedBased(){}
};


int main(){
  Student *stud[MAXSTUDENT];
  Scholarship *_scholarship[MAXSTUDENT];
  int studentCount = 0;
  int numApply=0;

  while (true){
    string degreeProgram;



    cout << "Enter your degree program (leave empty and press Enter if you are undergraduate): ";
    getline(cin, degreeProgram);

    Student *stu = NULL;

    if (degreeProgram.empty()){
      stu = new Undergraduate();
    }
    else{
      stu = new Postgraduate(0,0,0,0,degreeProgram);
    }
    try{
    stu->StudentData();
    stu->file();
    stu->printDetails();
    }

    catch(Student::NegativeSign)
    {cout << "Something wrong!!! Negtive sign exist";}


    while(true){

      int scholarship;
      cout << endl << "Choose the Scholarship Type you want to apply: " << endl;
      cout << "1 - Merit-based" << endl << "2 - Need-based" << endl;

      cout << "Enter your choice: ";
      cin >> scholarship;
      Scholarship *scholar;

      switch(scholarship){
        case 1: scholar=new MeritBased();
                break;
        case 2: scholar=new NeedBased();
                break;
        default:
                cout<<"Invalid input.Please enter again."<<endl;
                continue;
    }

    scholar->addStudent(stu);
    scholar->amountReceived();
    scholar->scholarshipInfo();

    if (scholar->getApplication() == "Approved") {
      _scholarship[numApply] = scholar;
      numApply++;
      stud[studentCount] = stu;
      studentCount++;
      break;
    }
    else {
      char choice;
      cout << "Do you want to apply for another scholarship? (y/n): ";
      cin >> choice;
      if (choice == 'n' || choice == 'N') {



        break;
      }
    }
  }



  char addMore;
  cout << "Do you want to add another student? (y/n): ";
  cin >> addMore;
  cin.ignore(); // Ignore newline character left in the buffer

    if (addMore == 'n' || addMore == 'N') {
      break;
    }
  }

  // Clean up
  for (int i = 0; i < studentCount; i++) {
    delete stud[i];
  }
  for (int j=0; j<numApply;j++){
    delete _scholarship[j];
  }

system ("pause");
return 0;
}
```