BACS2063 Data Structures and Algorithms

ASSIGNMENT 202401

Declaration

- I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University of Management and Technology's plagiarism policy.
- I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

Student Name	Student ID	Prog / Tut.Grp	Signature
Yam Jason	22WMR1366 2	RDS2S2G3	[illegible]
Wong Yee En	22WMR1365 9	RDS2S2G3	[illegible]

Note: The submission date and time will be according to the timestamp recorded in Google Classroom for Assignment Submission.

Yam Jason

Wong Yee En

Table of Contents

A. TEAM REPORT

1. Abstract Data Type (ADT)

1.1 ADT Specification

ADT Map
A Map functions as both an abstract data type and a collective entity designed to store key-value pairs, facilitating efficient storage and retrieval of values based on their corresponding keys. It offers functionality for adding, removing, and accessing these pairs.

put(K key, V value)
Description : Inserts a key-value pair into the Map.
Precondition : The key must not be null.
Postcondition : If the key already exists in the Map, its corresponding value will be updated. Otherwise, a new key-value pair will be added to the Map.

V get(K key)
Description : Retrieves the value associated with the specified key.
Precondition : The key must not be null.
Postcondition : The Map remains unchanged.
Returns : The value associated with the specified key, or null if the key is not found in the Map.

V remove(K key)
Description : Removes the key-value pair associated with the specified key from the Map.
Precondition : The key must not be null.
Postcondition : If the specified key exists in the Map, it will be removed along with its associated value.
Returns : The value associated with the specified key, or null if the key is not found in the Map.

ListInterface<K> keys()
Description : Returns a list of all keys present in the Map.
Postcondition : The Map remains unchanged.
Returns : a list of all keys present in the Map

ListInterface<V> values()
Description : Returns a list of all values present in the Map.
Postcondition : The Map remains unchanged.
Returns : a list of all values present in the Map.

boolean containsKey(K key)	
Description	: Checks whether the Map contains the specified key.
Postcondition	: The Map remains unchanged.
Returns	: True if the Map contains the specified key, false otherwise.

boolean containsValue(V value)	
Description	: Checks whether the Map contains the specified value.
Postcondition	: The Map remains unchanged.
Returns	: True if the Map contains the specified value, false otherwise.

int size()	
Description	: Returns the number of key-value pairs present in the Map.
Postcondition	: The Map remains unchanged.
Returns	: The number of key-value pairs present in the Map.

boolean isEmpty()	
Description	: Checks whether the Map is empty
Postcondition	: The Map remains unchanged.
Returns	: True if the Map contains no key-value pairs, false otherwise.

boolean isFull()	
Description	: Checks whether the Map is full.
Postcondition	: The Map remains unchanged.
Returns	: True if the Map has reached its maximum capacity, false otherwise.

clear()	
Description	: Removes all key-value pairs from the Map.
Postcondition	: The Map becomes empty.

1.2 ADT Implementation

a. MapInterface

```
package adt;

/**
 * An interface for the ADT Map, which stores key-value pairs.
 *
 * @param <K> The type of keys in the map.
 * @param <V> The type of values in the map.
 *
 * @author
 * Name: Wong Yee En, Yam Jason
 * RDS2Y2S2G3
 * 22WMR13659, 22WMR13662
 */
public interface MapInterface<K, V> {

   /**
    * Task: Adds a new key-value pair to the map. If the key already exists, the
    * corresponding value is overwritten.
    *
    * @param key The key to be added.
    * @param value The value to be associated with the key.
    */
   public void put(K key, V value);

   /**
    * Task: Retrieves the value associated with the specified key.
    *
    * @param key The key whose associated value is to be retrieved.
    * @return The value associated with the specified key, or null if the key
    *         is not found.
    */
```

```
public V get(K key);

/**
 * Task: Removes the key-value pair associated with the specified key from the
 * map.
 *
 * @param key The key of the key-value pair to be removed.
 * @return The value associated with the removed key, or null if the key is
 *         not found.
 */
public V remove(K key);

/**
 * Task: Retrieves a list of all keys in the map.
 *
 * @return A list containing all keys in the map.
 */
public ListInterface<K> keys();

/**
 * Task: Retrieves a list of all values in the map.
 *
 * @return A list containing all values in the map.
 */
public ListInterface<V> values();

/**
 * Task: Checks whether the map contains the specified key.
 *
 * @param key The key to be checked for existence.
 * @return true if the map contains the specified key, false otherwise.
 */
public boolean containsKey(K key);
```

```
/**
 * Task: Checks whether the map contains the specified value.
 *
 * @param value The value to be checked for existence.
 * @return true if the map contains the specified value, false otherwise.
 */
public boolean containsValue(V value);

/**
 * Task: Retrieves the number of key-value pairs in the map.
 *
 * @return The number of key-value pairs in the map.
 */
public int size();

/**
 * Task: Checks whether the map is empty.
 *
 * @return true if the map is empty, false otherwise.
 */
public boolean isEmpty();

/**
 * Task: Checks whether the map is full. This method may not be applicable for
 * all implementations of the Map ADT.
 *
 * @return true if the map is full, false otherwise.
 */
public boolean isFull();

/**
 * Task: Removes all key-value pairs from the map, resulting in an empty map.
 */
public void clear();
```

```
}
```

b. HashMap Class

```
package adt;

import java.io.Serializable;

/**
 * Array implementation of HashMap using Open Addressing with Double Hashing.
 *
 * @param <K> The type of keys in the map.
 * @param <V> The type of values in the map.
 * @author
 * Name: Wong Yee En, Yam Jason
 * RDS2Y2S2G3
 * 22WMR13659, 22WMR13662
 *
 */


public class HashMap<K, V> implements MapInterface<K, V>, Serializable {

   private class Entry<K, V> implements Serializable {

      private K key;
      private V value;

      public Entry(K key, V value) {
         this.key = key;
         this.value = value;
      }
   }

   private Entry<K, V>[] entries;
   private int numberOfEntries;
   private double loadFactor = 0.75;
   private int primeNumber;
   private static final int DEFAULT_CAPACITY = 20;

   public HashMap() {
      this(DEFAULT_CAPACITY);
   }

   public HashMap(int capacity) {
      entries = new Entry[capacity];
      numberOfEntries = 0;
```

```
        primeNumber = getPrimeNumber();
    }

    public HashMap(int capacity, double loadFactor) {
        this(capacity);
        this.loadFactor = loadFactor;
    }

    @Override
    public void put(K key, V value) {
        if (key == null || value == null) {
            throw new IllegalArgumentException("Key or value cannot be null");
        }

        // Update Value if key exists
        int index = getIndexForExistingEntries(key);
        if (index != -1) {
            entries[index].value = value;
            return;
        }

        // Add new entry
        if (isHashMapTooFull()) {
            rehash();
        }

        index = getIndexForNullEntries(key);

        // Find next available index
        while (index == -1) {
            rehash();
            index = getIndexForNullEntries(key);
        }

        entries[index] = new Entry<>(key, value);
        numberOfEntries++;
    }

    @Override
    public V get(K key) {
        if (key == null) {
            throw new IllegalArgumentException("Key cannot be null");
        }

        int index = getIndexForExistingEntries(key);
        if (index != -1) {
            return entries[index].value;
        }
        return null;
```

```
}

@Override
public V remove(K key) {
   if (key == null) {
      throw new IllegalArgumentException("Key cannot be null");
   }

   V removedValue = null;
   int index = getIndexForExistingEntries(key);
   if (index != -1) {
      removedValue = entries[index].value;
      entries[index] = null;
      numberOfEntries--;
   }
   return removedValue;
}

@Override
public ListInterface<K> keys() {
   ListInterface<K> keys = new ArrayList<>();
   for (int i = 0; i < entries.length; i++) {
      if (entries[i] != null) {
         keys.add(entries[i].key);
      }
   }
   return keys;
}

@Override
public ListInterface<V> values() {
   ListInterface<V> values = new ArrayList<>();
   for (int i = 0; i < entries.length; i++) {
      if (entries[i] != null) {
         values.add(entries[i].value);
      }
   }
   return values;
}

@Override
public boolean containsKey(K key) {
   if (key == null) {
      return false;
   }

   return getIndexForExistingEntries(key) != -1;
}
```

```
@Override
public boolean containsValue(V value) {
   if (value == null) {
      return false;
   }

   for (int i = 0; i < entries.length; i++) {
      if (entries[i] != null && entries[i].value.equals(value)) {
         return true;
      }
   }
   return false;
}

@Override
public int size() {
   return numberOfEntries;
}

@Override
public boolean isEmpty() {
   return numberOfEntries == 0;
}

@Override
public boolean isFull() {
   return entries.length == numberOfEntries;
}

@Override
public void clear() {
   for (int i = 0; i < entries.length; i++) {
      entries[i] = null;
   }
   numberOfEntries = 0;
}

@Override
public String toString() {
   StringBuilder sb = new StringBuilder();
   sb.append("HashMap: {");
   for (int i = 0; i < entries.length; i++) {
      if (entries[i] != null) {
         sb.append(entries[i].key).append("=").append(entries[i].value).append(", ");
      }
   }
   sb.append("}");
   return sb.toString();
}
```

```
// Check if HashMap needs rehashing
private boolean isHashMapTooFull() {
    return numberOfEntries >= loadFactor * entries.length;
}

// Get the prime number that is closest and lesser to the size of the array
private int getPrimeNumber() {
    for (int i = entries.length - 1; i >= 1; i--) {
        int count = 0;
        for (int j = 2; j * j <= i; j++) {
            if (i % j == 0) {
                count++;
                break;
            }
        }
        if (count == 0) {
            return i;
        }
    }
    return 3;
}

// Rehash the HashMap
private void rehash() {
    Entry<K, V>[] oldEntries = entries;
    entries = new Entry[oldEntries.length * 2];

    for (int i = 0; i < oldEntries.length; i++) {
        if (oldEntries[i] != null) {
            int index = getIndexForNullEntries(oldEntries[i].key);
            entries[index] = oldEntries[i];
        }
    }
}

// Get index for null entries
private int getIndexForNullEntries(K key) {
    int steps = 0;

    while (steps < entries.length) {
        int index = index(key, steps++);
        Entry<K, V> entry = entries[index];
        if (entry == null) {
            return index;
        }
    }
    return -1;
}
```

```
    // Get index for existing entries
    private int getIndexForExistingEntries(K key) {
        int steps = 0;

        while (steps < entries.length) {
            int index = index(key, steps++);
            Entry<K, V> entry = entries[index];
            if (entry != null && entry.key.equals(key)) {
                return index;
            }
        }
        return -1;
    }

    // Calculate index based on hash values and number of steps
    private int index(K key, int i) {
        int hash1 = hash1(key);
        int hash2 = hash2(key);
        return (hash1 + i * hash2) % entries.length;
    }

     // Calculate the first hash value
    private int hash1(K key) {
        int hashIndex1 = key.hashCode() % entries.length;
        if (hashIndex1 < 0) {
            hashIndex1 += entries.length;
        }
        return hashIndex1;
    }

    // Calculate the second hash value
    private int hash2(K key) {
        int hashIndex2 = primeNumber - (key.hashCode() % primeNumber);
        if (hashIndex2 < 0) {
            hashIndex2 += primeNumber;
        }
        return hashIndex2;
    }

    protected int getIndex(K key, int step) {
        return index(key, step);
    }

}
```

B. INDIVIDUAL REPORTS

2. Use of ADTs

Name	Student ID	Prog / Tut.Grp	Signature
Yam Jason	22WMR136 62	RDS/3	[illegible]

Subsystem : Student Management Subsystem

1. **Source codes for Control classes.**

```
package control;

import entity.*;
import adt.ArrayList;
import adt.ListInterface;
import adt.MapInterface;
import adt.SetInterface;
import boundary.StudentRegistrationManagementUI;
import dao.CourseDAO;
import dao.StudentDAO;
import java.util.Iterator;
import utility.MessageUI;
import java.io.Serializable;
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 *
 * @author Name: Yam Jason RDS2Y2S2G3 22WMR13662
 */
public class StudentRegistrationManagement implements Serializable {

    private ListInterface<Student> studentList = new ArrayList<>();
    private final CourseManagement courseManagement;
    private final StudentDAO studentDAO = new StudentDAO("students.dat");
    private final StudentRegistrationManagementUI studentUI = new
StudentRegistrationManagementUI();
    private final CourseDAO courseDAO = new CourseDAO("courses.dat");

    public static int studentEntries;
    public static int registrationEntries;
    public StudentRegistrationManagement() {

        courseManagement = new CourseManagement();
        studentList = studentDAO.retrieveFromFile();

    }

    public void mainMenu() {
        int choice = -1;

        do {

            choice = studentUI.getMenuChoice();
            switch (choice) {
                case 0:
                    //to exit
                    MessageUI.displayBackMessage();
                    break;
                case 1:
                    studentEntries = studentList.getNumberOfEntries();

                    addStudent();
                    break;
                case 2:
                    removeStudent();
                    break;
                case 3:
                    amendStudent();
                    break;
                case 4:
                    displayStudents();
                    break;
                case 5:
                    searchStudents();
                    break;
                case 6:
                    registrationEntries = getTotalRegistered();
                    register();
                    break;
                case 7:
//                    displayCourseRegistered();
                    removeFromCourse();
                    break;
                case 8:
                    calFeesRegCourse();
                    break;
                case 9:
                    filterStudents();
                    break;
                case 10:
                    generateReport1();
                    break;
                case 11:
                    generateReport2();
                    break;

                default:
                    MessageUI.displayInvalidChoiceMessage();
            }
        } while (choice != 0);
    }

    //Task 1
    public void addStudent() {

        String name = studentUI.inputStudentName();
        if (name.equals("999")) {
            return;
        }
        String DOB;
        boolean dobValid = false;
        do {
            DOB = studentUI.inputDOB();
            if (vldDOB(DOB)) {
                dobValid = true;
            } else {
                MessageUI.displayInvalidInput();
            }

        } while (!dobValid);

        String ic;
        boolean icValid = false;
        do {
            ic = studentUI.inputIC();
            if (vldIC(ic)) {
                icValid = true;
            } else {
                MessageUI.displayInvalidInput();
            }

        } while (!icValid);

        String phoneNo;
        boolean phoneValid = false;
        do {
            phoneNo = studentUI.inputPhoneNo();
            if (vldPhoneNumber(phoneNo)) {
                phoneValid = true;
             } else {
                MessageUI.displayInvalidInput();
             }
          } while (!phoneValid);

          String email;
          boolean emailValid = false;
          do {
             email = studentUI.inputEmail();
             if (vldEmail(email)) {
                emailValid = true;
             } else {
                MessageUI.displayInvalidInput();
             }
          } while (!emailValid);

          String programmeID;
          courseManagement.displayAllProgrammes();
          //remember to use return at the last point
          do {

             programmeID = studentUI.inputProgrammeID();
             if (courseManagement.getProgrammeMap().containsKey(programmeID)) {

                Student newStudent = new Student(name, DOB, ic, phoneNo, email,
    programmeID);
                studentList.add(newStudent);
                System.out.println("Student ID: " + newStudent.getStudentID());
                studentDAO.saveToFile(studentList);
                System.out.println("Student Sucessfully Added!");

                return;

             } else if (!programmeID.equals("999")) {
                System.out.println("Invalid Course ID!");
             }

          } while (!programmeID.equals("999"));

       }

       //Task 2
       public void removeStudent() {
          String studentId = studentUI.inputStudentID();
          for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
             Student student = studentList.getEntry(i);
             if (student.getStudentID().equals(studentId)) {
                studentList.getEntry(i).setWithdraw(true); //updated
                System.out.println("Student with ID " + studentId + " removed successfully.");
                studentDAO.saveToFile(studentList);
                return;
            }
        }
        System.out.println("Student with ID " + studentId + " not found.");
    }

    //For listing all students
    public String getAllStudents() {
        String outputStr = "";
        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
           if (studentList.getEntry(i).isWithdraw() == false) {  //updated
                outputStr += studentList.getEntry(i) + "\n";
            }

        }
        return outputStr;
    }

    //List all students (extra)
    public void displayStudents() {
        if (!studentList.isEmpty()) {
            studentUI.listAllStudents(getAllStudents());
        } else {
            System.out.println("Student list is empty, please add new student to view.");
        }

    }

    //Task 3
    public void amendStudent() {
        String studentId = studentUI.inputStudentID();
        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
            if (student.getStudentID().equals(studentId)) {
                int choice = 0;
                do {
                    choice = studentUI.getAmendChoice(studentId);
                    switch (choice) {
                        case 0:
                            MessageUI.displayBackMessage();
                            break;
                        case 1:
                            String studentName = studentUI.inputStudentName();
                            student.setStudentName(studentName);
                            MessageUI.displayUpdateMessage();
                            break;
                        case 2:
```

```
            String DOB;
            boolean dobValid = false;
            do {
                DOB = studentUI.inputDOB();
                if (vldDOB(DOB)) {
                    dobValid = true;
                } else {
                    MessageUI.displayInvalidInput();
                }

            } while (!dobValid);

            student.setStudentDOB(DOB);
            MessageUI.displayUpdateMessage();
            break;
        case 3:
            String phoneNo;
            boolean phoneValid = false;
            do {
                phoneNo = studentUI.inputPhoneNo();
                if (vldPhoneNumber(phoneNo)) {
                    phoneValid = true;
                } else {
                    MessageUI.displayInvalidInput();
                }
            } while (!phoneValid);

            student.setPhoneNo(phoneNo);
            MessageUI.displayUpdateMessage();
            break;
        case 4:
            String email;
            boolean emailValid = false;
            do {
                email = studentUI.inputEmail();
                if (vldEmail(email)) {
                    emailValid = true;
                } else {
                    MessageUI.displayInvalidInput();
                }
            } while (!emailValid);
            student.setStudentEmail(email);
            MessageUI.displayUpdateMessage();
            break;
        default:
            MessageUI.displayInvalidChoiceMessage();

    }
```

```
                } while (choice != 0);
                studentDAO.saveToFile(studentList);

                return;
            }
        }
        System.out.println("Student with ID " + studentId + " not found.");
    }

    //Task 4
    public void searchStudents() {
        String courseID;
        boolean printLabel;
        boolean studentExists;

        do {
            printLabel = true;
            studentExists = false;
            courseID = studentUI.inputCourseID();

            for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
                Student student = studentList.getEntry(i);
                MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

                // Iterate through the keys (registration numbers) of the registered courses map for
the current student
                for (String registrationNumber : registeredCourses.keys()) {
                    Registration registration = registeredCourses.get(registrationNumber);

                    // Check if the registration contains the specified course ID
                    if (registration.getCourse().getCourseId().equals(courseID) &&
!registration.isRegistrationIsCancelled() && !student.isWithdraw()) { //updated
                        studentExists = true;

                        if (printLabel) {
                            studentUI.printRegCourseLabel(courseID);
                            printLabel = false;
                        }
                        // If the student is registered for the course, you can perform further actions
here
                        System.out.printf("%-13s %-20s %-13s %-15s %-20s\n",
student.getStudentID(), student.getStudentName(), student.getStudentDOB(),
student.getPhoneNo(), student.getStudentEmail());
                        // No need to continue searching other registrations for this student
                    }
                }

            }
            if (!studentExists && !courseID.equals("999")) {
                studentUI.printNotExist();
            }
            System.out.println("");
        } while (!courseID.equals("999"));

    }

    //Task 5
    public void register() {
        String studentId = studentUI.inputStudentID();

        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
            if (student.getStudentID().equals(studentId) && student.isWithdraw() == false) {
//updated
                System.out.println("Valid student ID!");

                int choice = 0;
                do {
                    choice = studentUI.getRegChoice(studentId);
                    switch (choice) {
                        case 0:
                            MessageUI.displayBackMessage();
                            break;
                        case 1:
                            //display courses that only matches with the programme

//                          courseManagement.displayAllCourses();
                            //type courseID and make payment
                            registerProcess(i);
                            break;
                        default:
                            MessageUI.displayInvalidChoiceMessage();

                    }

                } while (choice != 0);

                return;
            }
        }
        System.out.println("Student with ID " + studentId + " not found.");
    }

    //For task 5
    public void registerProcess(int studentIndex) {

        String courseID;
        String type;
        Course course;
        SetInterface<String> courseStatuses;
        boolean valid;

        boolean isValidType;
        Payment payment;
        String approve;
        MapInterface<String, Course> courseMap = courseManagement.getCourseMap();
        ListInterface<ProgrammeCourse> programmeCourseList =
courseManagement.getProgrammeCourseList();
        ProgrammeCourse programmeCourse;
        int programmeCount = 0;

        //checks student's total credit hour
        int totalCreditHour = getTotalCreditHours(studentList.getEntry(studentIndex));
        System.out.println("Student's total credit hour in this semester: " + totalCreditHour);
        System.out.println("Max Credit Hour: 16");

        //stopping here
        // Iterate through the registered programme
        //shows courses that have connections with the student's program   *arraylist
        System.out.printf("\n%-15s%-35s%-30s%15s\n", "Course ID", "Course Name",
"Status(s)", "Credit Hours");
        for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
            programmeCourse = programmeCourseList.getEntry(i);
            // Check if the registration contains the given course ID
            if
(programmeCourse.getProgrammeID().equals(studentList.getEntry(studentIndex).getProgra
mmeID())) {
                programmeCount++;

                System.out.printf("%-15s%-35s%-30s%15s\n", programmeCourse.getCourseID(),

courseManagement.getCourseMap().get(programmeCourse.getCourseID()).getCourseName()
,

courseManagement.getCourseMap().get(programmeCourse.getCourseID()).getStatus(),

courseManagement.getCourseMap().get(programmeCourse.getCourseID()).getCreditHours()
);

            }

        }
        if (programmeCount == 0) {
            System.out.println("There is no avaliable courses to register for this student");
        } else {
            //remember to use return at the last point
        do {
            valid = false;
            courseID = studentUI.inputCourseID();

            for (ProgrammeCourse programmeCourse1 : programmeCourseList) {
                // Check if both programmeID and courseID match the input
                if
(programmeCourse1.getProgrammeID().equals(studentList.getEntry(studentIndex).getProgra
mmeID())
                        && programmeCourse1.getCourseID().equals(courseID)) {
                    valid = true;

                    //checks if the course has been registered by the student
                    if (isCourseAlreadyRegistered(studentList.getEntry(studentIndex), courseID))
{
                        System.out.println("This course is registered by the student!");
                    } else if (totalCreditHour +
courseManagement.getCourseMap().get(programmeCourse1.getCourseID()).getCreditHours(
) > 16) {
                        System.out.println("Unable to register for this course!");
                        System.out.println("Max Credit Hour is 16!");
                        return;

                    } else {
                        System.out.println("Course Not registered by the student!");
                        course = courseManagement.getCourseMap().get(courseID);
                        courseStatuses =
courseManagement.getCourseMap().get(courseID).getStatus();
                        // Get an iterator for the course statuses
                        Iterator<String> iterator;

                        do {
                            iterator = courseStatuses.getIterator();
                            isValidType = false;
                            type = studentUI.inputCourseType();

                            // Validate the type against the course statuses
                            while (iterator.hasNext()) {

                                String status = iterator.next();
                                if (type.equals(status)) {
                                    isValidType = true;
                                    break;
                                }
                            }

                            //if the course type entered is valid
                            if (isValidType) {
                                System.out.println("Course Type Valid!");
                        // The type matches one of the course statuses

                        generateBill(studentList.getEntry(studentIndex).getStudentID(),
                                studentList.getEntry(studentIndex).getStudentName(),
studentList.getEntry(studentIndex).getProgrammeID(),
                                courseID,
courseManagement.getCourseMap().get(courseID).getCourseName(),
courseManagement.getCourseMap().get(courseID).getCreditHours(),

courseManagement.getCourseMap().get(courseID).getCreditHours() *
Registration.courseRate);

                        // proceed to payment
                        payment =
payment(courseManagement.getCourseMap().get(courseID).getCreditHours() *
Registration.courseRate);
                        //test

                        do {

                            approve = studentUI.inputApprove();
                            if (approve.equals("Y")) {
                                //print the registration bill
                                System.out.println(payment);

                                //generate the registration object then add into that student
                                Registration registration = new Registration(course, type,
payment);

                                //add into student registered courses map

studentList.getEntry(studentIndex).getRegisteredCourses().put(registration.getRegNum(),
registration);

                                studentDAO.saveToFile(studentList);
                                courseDAO.saveToFile(courseMap);


                            } else if (approve.equals("N")) {
                                studentUI.printRejectedPayment();
                            } else {

                                System.out.println("Invalid input!");
                            }

                        } while (!approve.equals("Y") && !approve.equals("N"));

                        return;
                        } else if (!type.equals("999")) {
                            System.out.println("Invalid course type for the selected course!");
                        }
                    } while (!type.equals("999"));

                }

            }

        }
        if (!courseID.equals("999") && !valid) {
            System.out.println("Invalid Course ID!");
        }
    } while (!courseID.equals("999"));

    }

}

//Task 6
public void removeFromCourse() {
    String studentId = studentUI.inputStudentID();

    for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
        Student student = studentList.getEntry(i);
        if (student.getStudentID().equals(studentId) && !student.isWithdraw()) {  //updated
            System.out.println("Valid student ID!");
            // Get the registered courses of the student
            MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

            if (registeredCourses.isEmpty()) {
                System.out.println("This student has not registered for any courses.");
                return;
            } else {

System.out.println("=====================================================
=================================================");
                System.out.println("                    Courses registered by student with ID " +
studentId + ":");

System.out.println("=====================================================
=================================================");
                System.out.println("Note: you can only remove a student from a course (main,
elective) registration");
                System.out.printf("%-17s %-10s %-40s %-15s %-13s\n", "Registration ID",
"Course ID", "Course Name", "Credit Hours", "Course Type");
                for (String regNum : registeredCourses.keys()) {
                    Registration registration = registeredCourses.get(regNum);
                //print only if the registration is not a cancelled registration
                if (!registration.isRegistrationIsCancelled()) {

                    System.out.printf("%-17s %-10s %-40s %-15s %-13s\n", regNum,
registration.getCourse().getCourseId(), registration.getCourse().getCourseName(),
registration.getCourse().getCreditHours(), registration.getType());
                }

            }

            // Prompt user to enter course ID
            String regID = studentUI.inputRegID();

            // Remove the specified course ID if it exists in registeredCourses
            if (registeredCourses.containsKey(regID) &&
!registeredCourses.get(regID).isRegistrationIsCancelled() &&
(registeredCourses.get(regID).getType().equals("Main") ||
registeredCourses.get(regID).getType().equals("Elective"))) {
                registeredCourses.get(regID).setRegistrationIsCancelled(true);
                System.out.println("Course Registration with register ID " + regID + "
removed successfully.");

                studentDAO.saveToFile(studentList);
            } else if (registeredCourses.containsKey(regID) &&
registeredCourses.get(regID).isRegistrationIsCancelled()) {
                System.out.println("This registration was cancelled before!");
            } else if (registeredCourses.containsKey(regID) &&
!registeredCourses.get(regID).isRegistrationIsCancelled()) {
                System.out.println("You cant remove this registration because it is not a main
or elective registration");

            } else {
                System.out.println("Course Registration with register ID " + regID + " not
found in the registered courses.");
            }
            return;
        }
    }
  }
  System.out.println("Student with ID " + studentId + " not found.");

}

//Task 7
public void calFeesRegCourse() {
    MapInterface<String, Course> courseMap = courseManagement.getCourseMap();
    studentUI.displayFeesCourse();
    for (Course course : courseMap.values()) {
            System.out.printf("%-15s %-35s %.0f\n", course.getCourseId(),
course.getCourseName(), course.getFeePaid());
        }

    }

    //Task 8
    public void filterStudents() {

        String courseID;
        boolean printLabel;
        boolean studentExists;
        int criteria;
        String programmeID;

        do {
            printLabel = true;
            studentExists = false;
            courseID = studentUI.inputCourseID();

            for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
                Student student = studentList.getEntry(i);
                MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

                // Iterate through the keys (registration numbers) of the registered courses map for
the current student
                for (String registrationNumber : registeredCourses.keys()) {
                    Registration registration = registeredCourses.get(registrationNumber);

                    // Check if the registration contains the specified course ID
                    if (registration.getCourse().getCourseId().equals(courseID)) {

                        // IF TRUE THEN PROCEED WITH LOGIC
                        criteria = -1;
                        do {
                            try {
                                criteria = studentUI.getCriteria();

                                switch (criteria) {
                                    case 0:
                                        MessageUI.displayBackMessage();
                                        return;

                                    case 1: {

                                        programFilter(courseID);
                                        break;
                                    }
                                    case 2:
                                        //female filter
                                        femaleFilter(courseID);
                                        break;
                                    case 3:
                                        //male filter
                                        maleFilter(courseID);
                                        break;
                                    default:
                                        MessageUI.displayInvalidChoiceMessage();

                                }
                            } catch (InputMismatchException e) {
                                System.out.println("Invalid input. Please enter an integer.");

                            }
                        } while (criteria != 0);

                    }
                }

            }
            if (!studentExists && !courseID.equals("999")) {
                studentUI.printNotExist();
            }
            System.out.println("");
        } while (!courseID.equals("999"));

    }

    //Task 9 (report 1)
    public void generateReport1() {
        int maleCount = 0;
        int femaleCount = 0;
        double malePercent;
        double femalePercent;
        String gender;

System.out.println("=====================================================
===================================");
        System.out.println("                              Student Report");

System.out.println("=====================================================
===================================");
        System.out.printf("%-15s %-25s %-10s %-15s %-20s\n", "Student ID", "Student Name",
"Gender", "Date Of Birth", "Programme ID");
        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
```

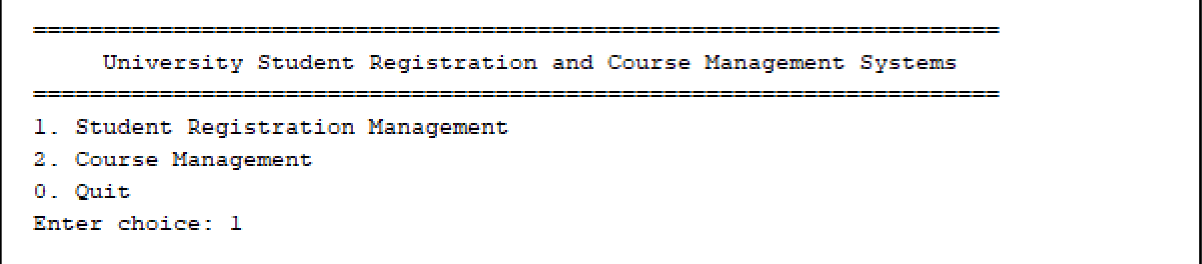
```
            // if not withdraw, display //updated
            if (student.isWithdraw() == false) {
                // Check the last digit of the student's IC number
                String ic = student.getIc();
                int lastDigit = Character.getNumericValue(ic.charAt(ic.length() - 1));

                // Check if the last digit has a remainder when divided by 2
                boolean isMale = lastDigit % 2 != 0;

                if (isMale) {
                    maleCount++;
                    gender = "Male";
                } else {
                    femaleCount++;
                    gender = "Female";
                }
                System.out.printf("%-15s %-25s %-10s %-15s %-20s\n", student.getStudentID(),
student.getStudentName(), gender, student.getStudentDOB(), student.getProgrammeID());

            }

        }

        System.out.println("\nNumber of Male Students: " + maleCount);
        System.out.println("Number of Female Students: " + femaleCount);
        System.out.println("Total Students: " + (maleCount + femaleCount));

        malePercent = (double) maleCount / (femaleCount + maleCount);
        femalePercent = 1 - malePercent;
        System.out.println("Percentage of Male Students: " + String.format("%.2f", malePercent
* 100) + "%");
        System.out.println("Percentage of Female Students: " + String.format("%.2f",
femalePercent * 100) + "%");

    }


    //Task 9 (report 2)
    public void generateReport2() {
        //main,elective,resit,repeat
        int mainCount = 0;
        int electiveCount = 0;
        int resitCount = 0;
        int repeatCount = 0;
        int total;

System.out.println("=======================================================
=============================================");
        System.out.println("                                   Registration Report");
System.out.println("==========================================================
==============================================");
        System.out.printf("%-20s %-20s %-40s %-10s\n", "Registration ID", "Course ID",
"Course Name", "Type");
        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
            if (!student.isWithdraw()) {  //updated
                // Get the registered courses of the student
                MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

                // If registeredCourses is null, the course is not registered
                if (registeredCourses == null) {
                    //do nothing
                } else {
                    // Iterate through the registered courses
                    for (Registration registration : registeredCourses.values()) {
                        if (!registration.isRegistrationIsCancelled()) {  //updated
                            // Check if the registration contains the given course ID
                            if (registration.getType().equals("Main")) {
                                mainCount++;
                            } else if (registration.getType().equals("Elective")) {
                                electiveCount++;
                            } else if (registration.getType().equals("Resit")) {
                                resitCount++;
                            } else {
                                repeatCount++;
                            }
                            System.out.println(registration);

                        }

                    }

                }

            }

        }
        total = mainCount + electiveCount + resitCount + repeatCount;
        if (total != 0) {
            System.out.println("\nNumber of Main Registrations: " + mainCount);
            System.out.println("Number of Elective Registrations: " + electiveCount);
            System.out.println("Number of Resit Registrations: " + resitCount);
            System.out.println("Number of Repeat Registrations: " + repeatCount);

            System.out.println("\nPercentage of Main Registrations: " + String.format("%.2f",
((double) mainCount / total) * 100) + "%");
```

```
            System.out.println("Percentage of Elective Registrations: " + String.format("%.2f",
((double) electiveCount / total) * 100) + "%");
            System.out.println("Percentage of Resit Registrations: " + String.format("%.2f",
((double) resitCount / total) * 100) + "%");
            System.out.println("Percentage of Repeat Registrations: " + String.format("%.2f",
((double) repeatCount / total) * 100) + "%");
        } else {
            System.out.println("There is no registration!");

        }

    }


    //For task 5 register process to make payment
    public Payment payment(double amountToPay) {
        Scanner s1 = new Scanner(System.in);

        //Make Payment
        int paymentNum = -1; // Initialize to an invalid value

        do {

            paymentNum = studentUI.inputPaymentOption(amountToPay);

            if (paymentNum < 1 || paymentNum > 2) {
                MessageUI.displayInvalidChoiceMessage();
            }

        } while (paymentNum < 1 || paymentNum > 2);

        //paymentAmount = event object's price
        //Create Card object if paymentNum = 1, 2 for cash
        if (paymentNum == 1) {

            //cardNum
            String cardNum = studentUI.inputCardNumber();
            while (Card.vldCardNum(cardNum) == false) {
                System.out.print("Invalid Card Number!\n");
                cardNum = studentUI.inputCardNumber();
            }

            //cardHolder
            String cardHolder = studentUI.inputCardHolder();

            //cardExp
            String cardExp = studentUI.inputCardExp();
            while (Card.vldCardExp(cardExp) == false) {
                System.out.print("Invalid Card Expiry Date!\n");
                cardExp = studentUI.inputCardExp();
            }

            //cardCVV
            String cardCVV = studentUI.inputCardCVV();
            while (Card.vldCardCvv(cardCVV) == false) {
                System.out.print("Invalid Card CVV!\n");
                cardCVV = studentUI.inputCardCVV();
            }
            //Create Payment Object
            Card payment = new Card(cardNum, cardHolder, cardExp, cardCVV, amountToPay);

            return payment;

        } else {

            //amount tendered
            double amountTendered = -1; // Initialize to an invalid value

            do {

                amountTendered = studentUI.inputAmountTendered();

                if ((amountTendered < amountToPay && amountTendered > 0) || amountTendered
< 0) {

                    MessageUI.displayInvalidInput();
                }

            } while ((amountTendered < amountToPay && amountTendered > 0) ||
amountTendered < 0);

            //create cash object
            Cash payment = new Cash(amountTendered, amountToPay);

            return payment;

        }
    }

    // Method to check if the course is already registered by the student
    private static boolean isCourseAlreadyRegistered(Student student, String courseID) {
        // Get the registered courses of the student
        MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

        // If registeredCourses is null, the course is not registered
        if (registeredCourses == null) {
            return false;
        }

        // Iterate through the registered courses
        for (Registration registration : registeredCourses.values()) {
            // Check if the registration contains the given course ID
            if (registration.getCourse().getCourseId().equals(courseID)) {
                // Course already registered
                return true;
            }
        }
        // Course not registered
        return false;
    }

    //to get total credit hours to check if it's eligible for registration
    private int getTotalCreditHours(Student student) {
        int totalCreditHours = 0;
        // Get the registered courses of the student
        MapInterface<String, Registration> registeredCourses = 
student.getRegisteredCourses();
        // Iterate through the registered courses
        for (Registration registration : registeredCourses.values()) {

            totalCreditHours += registration.getCourse().getCreditHours();
        }
        return totalCreditHours;

    }

    // to get registrationEntries for registration ID purpose
    public int getTotalRegistered() {

        // Iterate over all students in the studentList
        int totalRegisteredCourses = 0;
        for (int i = 0; i < studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i + 1);

            // Get the registered courses for the current student
            MapInterface<String, Registration> registeredCourses = 
student.getRegisteredCourses();

            // Add the number of registered courses for the current student to the total
            totalRegisteredCourses += registeredCourses.size();
        }

        System.out.println("Total registered courses across all students: " + 
totalRegisteredCourses);
        return totalRegisteredCourses;
    }
    //For task 8
    public void programFilter(String courseID) {

        boolean printLabel = true;
        int studCount = 0;
        String programmeID;

        programmeID = studentUI.inputProgrammeID();

        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
            MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

            // Iterate through the keys (registration numbers) of the registered courses map for the
current student
            for (String registrationNumber : registeredCourses.keys()) {
                Registration registration = registeredCourses.get(registrationNumber);

                // Check if the registration contains the specified course ID
                if (registration.getCourse().getCourseId().equals(courseID) &&
student.getProgrammeID().equals(programmeID) &&
!registration.isRegistrationIsCancelled() && !student.isWithdraw()) { //updated

                    if (printLabel) {
                        printLabel = false;
                        studCount++;
                        studentUI.printRegCourseLabel(courseID);

                    }
                    System.out.printf("%-13s %-20s %-13s %-15s %-20s\n", student.getStudentID(),
student.getStudentName(), student.getStudentDOB(), student.getPhoneNo(),
student.getStudentEmail());
//

                }
            }

        }
        if (studCount == 0) {
            System.out.println("There is no student in this course that meets the criteria!");
        }
    }



    //For task 8
    public void maleFilter(String courseID) {
        boolean printLabel = true;
        int studCount = 0;

        for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
            Student student = studentList.getEntry(i);
            MapInterface<String, Registration> registeredCourses = 
student.getRegisteredCourses();

            // Iterate through the keys (registration numbers) of the registered courses map for the 
current student
            for (String registrationNumber : registeredCourses.keys()) {
                Registration registration = registeredCourses.get(registrationNumber);
                if (!registration.isRegistrationIsCancelled() && !student.isWithdraw()) {  //updated
                    // Check the last digit of the student's IC number
                    String ic = student.getIc();
                    int lastDigit = Character.getNumericValue(ic.charAt(ic.length() - 1));

                    // Check if the last digit has a remainder when divided by 2
                    boolean isMale = lastDigit % 2 != 0;

                    // Check if the registration contains the specified course ID
                    if (registration.getCourse().getCourseId().equals(courseID) && isMale) {

                        if (printLabel) {
                            studCount++;
                            printLabel = false;
                            studentUI.printRegCourseLabel(courseID);

                        }
                        System.out.printf("%-13s %-20s %-13s %-15s %-20s\n", 
student.getStudentID(), student.getStudentName(), student.getStudentDOB(), 
student.getPhoneNo(), student.getStudentEmail());
//

                    }

                }

            }

        }
        if (studCount == 0) {
            System.out.println("There is no student in this course that meets the criteria!");
        }
    }


    //For task 8
```

```
  public void femaleFilter(String courseID) {

     boolean printLabel = true;
     int studCount = 0;

     for (int i = 1; i <= studentList.getNumberOfEntries(); i++) {
        Student student = studentList.getEntry(i);
        MapInterface<String, Registration> registeredCourses =
student.getRegisteredCourses();

        // Iterate through the keys (registration numbers) of the registered courses map for the
current student
        for (String registrationNumber : registeredCourses.keys()) {
           Registration registration = registeredCourses.get(registrationNumber);
           if (!registration.isRegistrationIsCancelled() && !student.isWithdraw()) {//updated

              // Check the last digit of the student's IC number
              String ic = student.getIc();
              int lastDigit = Character.getNumericValue(ic.charAt(ic.length() - 1));

              // Check if the last digit has a remainder when divided by 2
              boolean isMale = lastDigit % 2 != 0;

              // Check if the registration contains the specified course ID
              if (registration.getCourse().getCourseId().equals(courseID) && !isMale) {

                 if (printLabel) {
                    printLabel = false;
                    studCount++;
                    studentUI.printRegCourseLabel(courseID);

                 }
                 System.out.printf("%-13s %-20s %-13s %-15s %-20s\n",
student.getStudentID(), student.getStudentName(), student.getStudentDOB(),
student.getPhoneNo(), student.getStudentEmail());
//

              }
           }

        }

     }
     if (studCount == 0) {
        System.out.println("There is no student in this course that meets the criteria!");
     }

  }
```

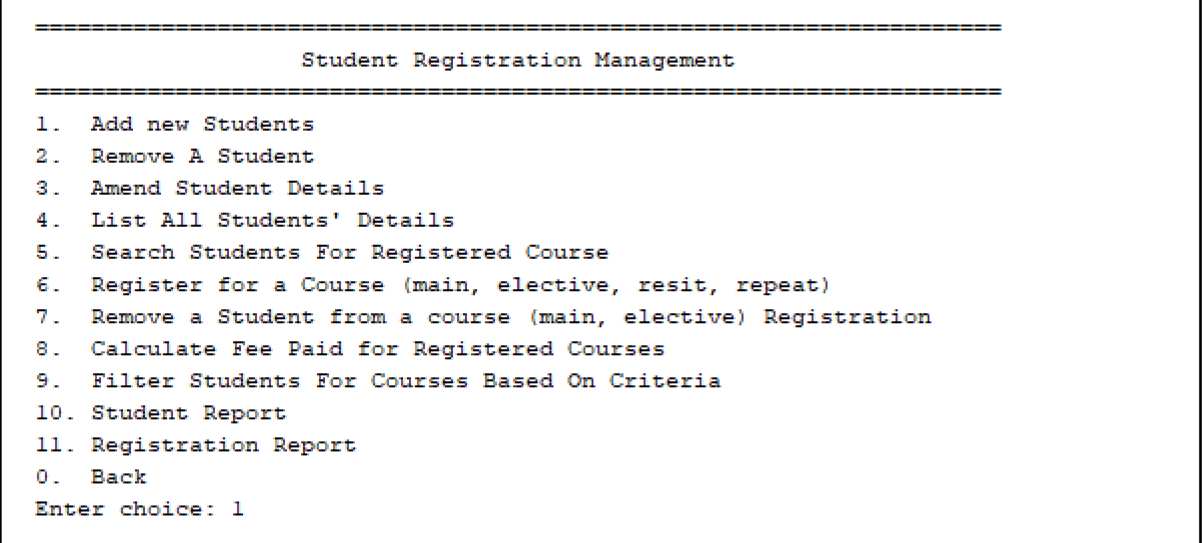
```
   //for registration
   public void generateBill(String ID, String name, String programmeID, String courseID,
String courseName, int creditH, double fees) {

System.out.println("\n=====================================================
==================================================");
       System.out.println("                                        STUDENT BILL");

System.out.println("======================================================
================================================");
       System.out.println("Student ID: " + ID);
       System.out.println("Student Name: " + name);
       System.out.println("Programme: " + programmeID);

System.out.println("-----------------------------------------------------------------------------------------------
--------");
       System.out.printf("%-10s %-40s %-15s %-20s\n", "CourseID", "Course Name", "Credit
Hours", "Fees");
       System.out.printf("%-10s %-40s %-15s %-20s\n", courseID, courseName, creditH,
fees);

   }

   //to validate IC
   public static boolean vldIC(String IC) {
       String ICRegex = "^[0-9]{12}$";
       Pattern pattern = Pattern.compile(ICRegex);
       Matcher matcher = pattern.matcher(IC);
       return matcher.matches();
   }

   //to validate email
   public static boolean vldEmail(String email) {
       // Regular expression for a valid email address
       String emailRegex = "^[A-Za-z0-9+_.-]+@(.+)$";

       // Compile the regex pattern
       Pattern pattern = Pattern.compile(emailRegex);

       // Match the email against the pattern
       Matcher matcher = pattern.matcher(email);

       // Check if the email matches the pattern
       return matcher.matches(); // True for valid, false for invalid

   }

   //to validate date of birth
   public static boolean vldDOB(String dob) {
       // Regular expression for a valid date of birth (dd/MM/yyyy)
       String dobRegex = "^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/\\d{4}$";

       // Compile the regex pattern
       Pattern pattern = Pattern.compile(dobRegex);

       // Match the DOB against the pattern
       Matcher matcher = pattern.matcher(dob);

       // Check if the DOB matches the pattern
       return matcher.matches(); // True for valid, false for invalid
   }

   //to validate validate Phone Number
   public static boolean vldPhoneNumber(String phoneNumber) {
       // Regular expression for a Malaysian phone number starting with "01" followed by 8
digits
       String phoneRegex = "^(01[0-9])-[0-9]{7,8}$";

       // Compile the regex pattern
       Pattern pattern = Pattern.compile(phoneRegex);

       // Match the phone number against the pattern
       Matcher matcher = pattern.matcher(phoneNumber);

       // Check if the phone number matches the pattern
       return matcher.matches(); // True for valid, false for invalid
   }
}
```

2. Screenshots



```
=====================================================================
     University Student Registration and Course Management Systems
=====================================================================
1. Student Registration Management
2. Course Management
0. Quit
Enter choice: 1
```

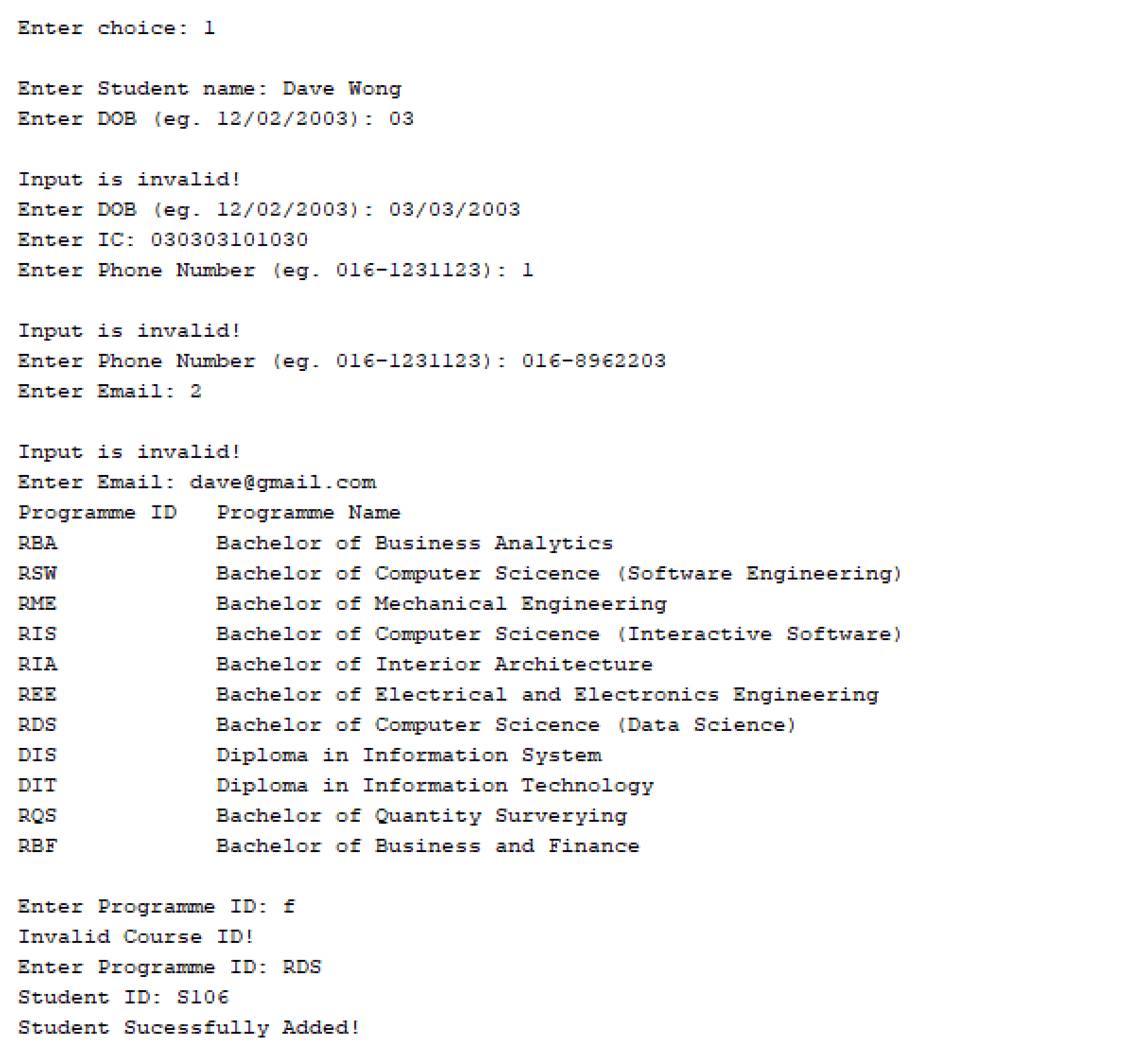
The main menu of the system shows the option to enter the student registration management screen and course management screen. 0 to quit the program.



```
=====================================================================
                   Student Registration Management
=====================================================================
1.  Add new Students
2.  Remove A Student
3.  Amend Student Details
4.  List All Students' Details
5.  Search Students For Registered Course
6.  Register for a Course (main, elective, resit, repeat)
7.  Remove a Student from a course (main, elective) Registration
8.  Calculate Fee Paid for Registered Courses
9.  Filter Students For Courses Based On Criteria
10. Student Report
11. Registration Report
0.  Back
Enter choice: 1
```

Student Registration Management screen, the options shown are the things that one can do in the student registration management subsystem. In this diagram, 1 is entered to add new students.

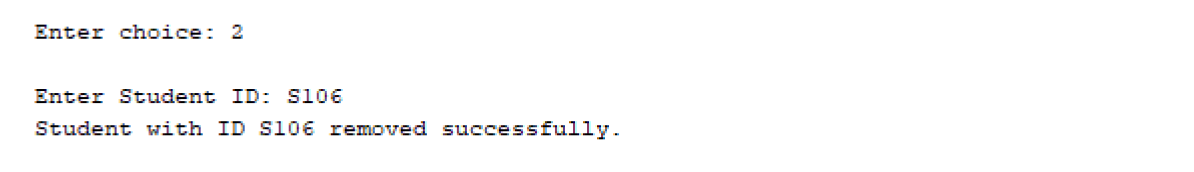
A. Add new students



```
Enter choice: 1

Enter Student name: Dave Wong
Enter DOB (eg. 12/02/2003): 03

Input is invalid!
Enter DOB (eg. 12/02/2003): 03/03/2003
Enter IC: 030303101030
Enter Phone Number (eg. 016-1231123): 1

Input is invalid!
Enter Phone Number (eg. 016-1231123): 016-8962203
Enter Email: 2

Input is invalid!
Enter Email: dave@gmail.com
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Scicence (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance

Enter Programme ID: f
Invalid Course ID!
Enter Programme ID: RDS
Student ID: S106
Student Sucessfully Added!
```

In this “add new students” screen, the user will be prompted to enter the name, date of birth, IC, phone number, and email of the student. Validation is implemented to make sure the correct format of date of birth, IC, phone number, and email of the student.

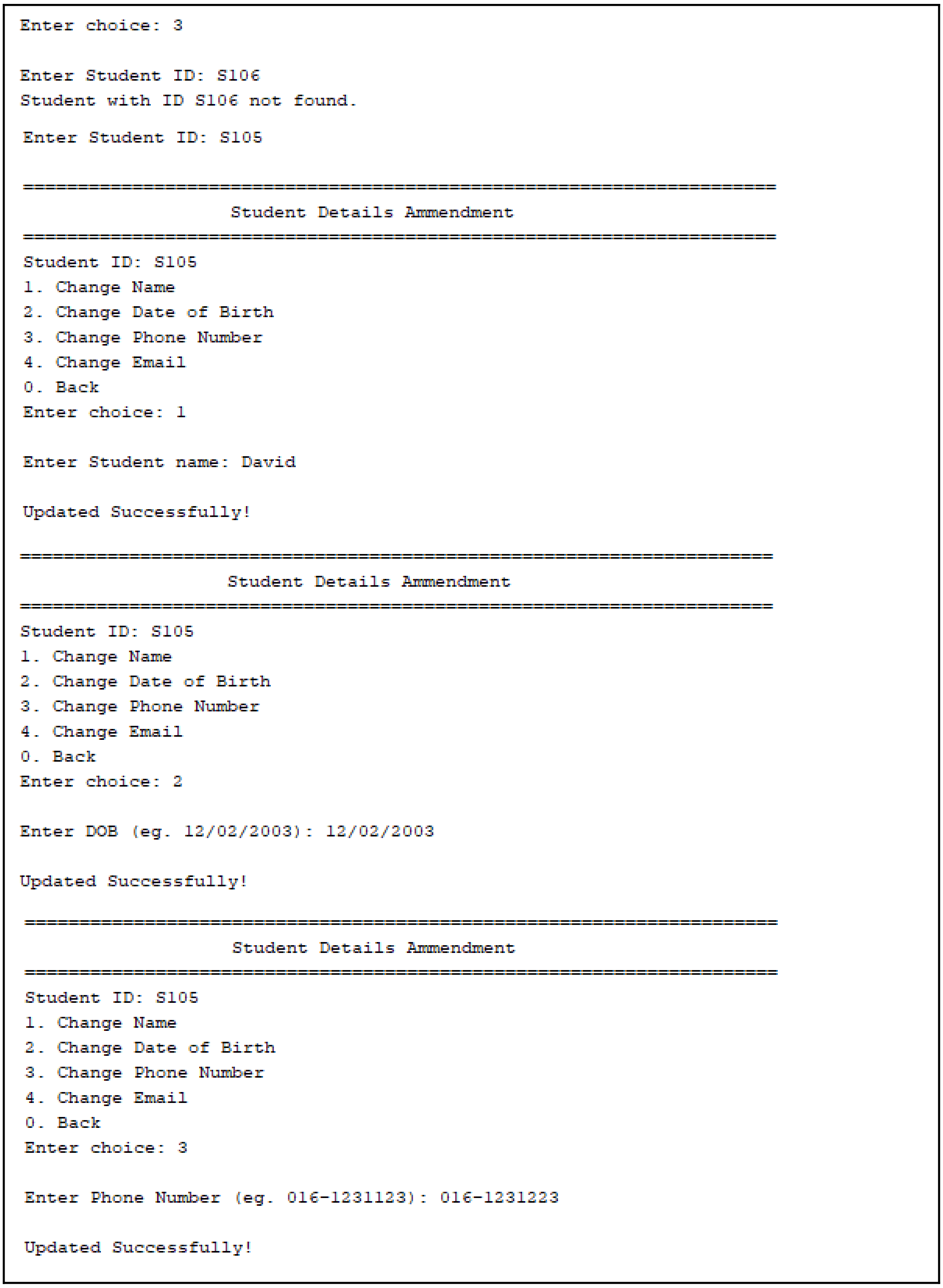
B. Remove a student



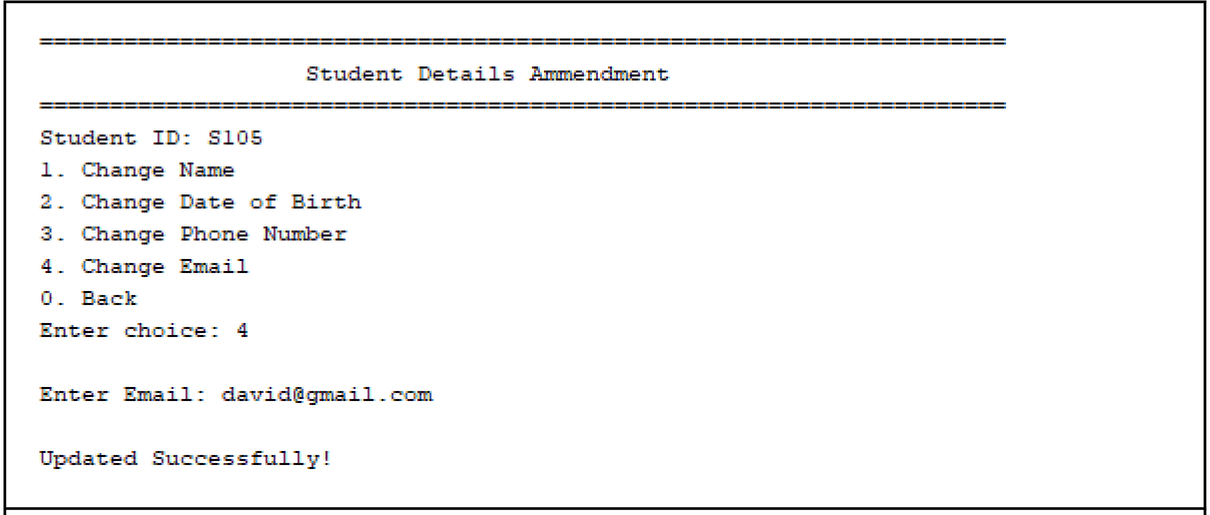
```
Enter choice: 2

Enter Student ID: S106
Student with ID S106 removed successfully.
```

When ‘2’ is entered in the student subsystem menu, the system will take the user to the screen to remove a student. The student can be removed by entering the student id. For example, “S101” and “S106”.

C. Amend student details



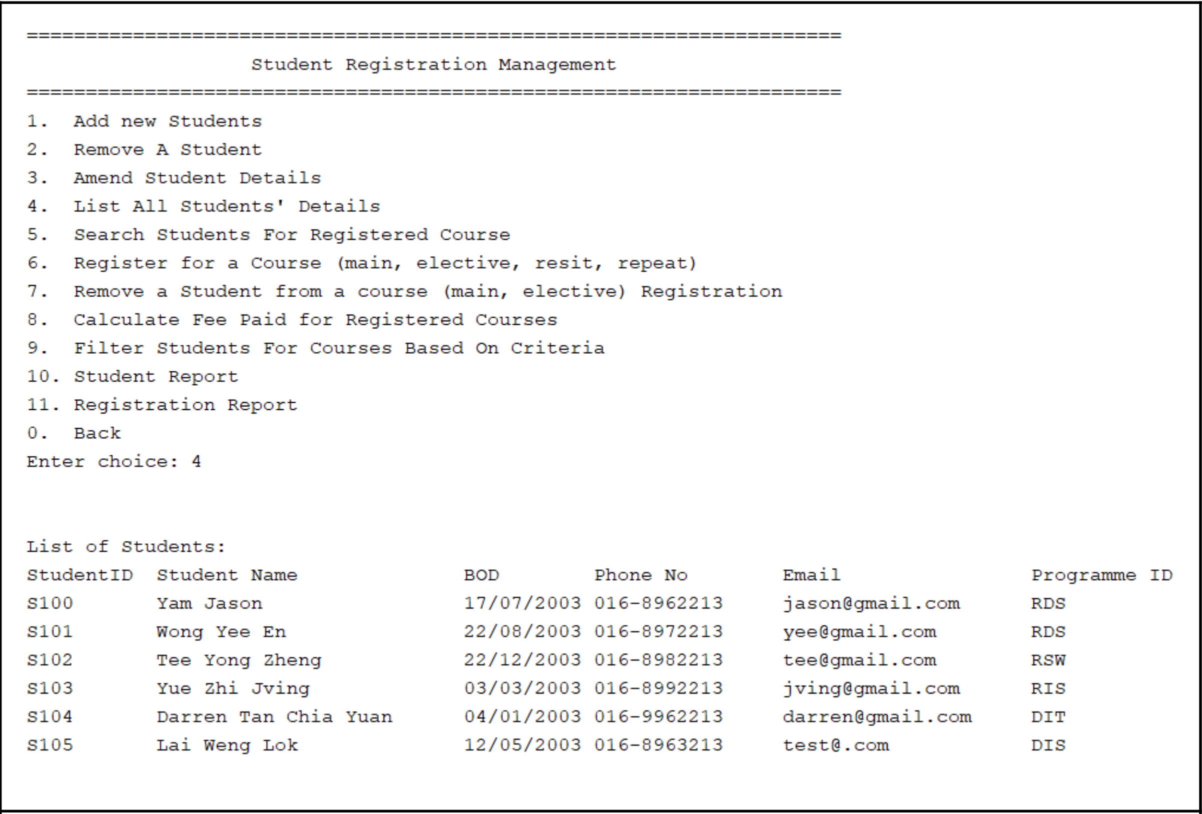
```
Enter choice: 3

Enter Student ID: S106
Student with ID S106 not found.

Enter Student ID: S105

======================================================================
                   Student Details Ammendment
======================================================================
Student ID: S105
1. Change Name
2. Change Date of Birth
3. Change Phone Number
4. Change Email
0. Back
Enter choice: 1

Enter Student name: David

Updated Successfully!

======================================================================
                   Student Details Ammendment
======================================================================
Student ID: S105
1. Change Name
2. Change Date of Birth
3. Change Phone Number
4. Change Email
0. Back
Enter choice: 2

Enter DOB (eg. 12/02/2003): 12/02/2003

Updated Successfully!

======================================================================
                   Student Details Ammendment
======================================================================
Student ID: S105
1. Change Name
2. Change Date of Birth
3. Change Phone Number
4. Change Email
0. Back
Enter choice: 3

Enter Phone Number (eg. 016-1231123): 016-1231223

Updated Successfully!
```



```
====================================================================
                   Student Details Ammendment
====================================================================
Student ID: S105
1. Change Name
2. Change Date of Birth
3. Change Phone Number
4. Change Email
0. Back
Enter choice: 4

Enter Email: david@gmail.com

Updated Successfully!
```

The "Amend student details" screen can be entered by entering "3" in the student management subsystem screen. User will get prompted to enter a student's id, if the id is invalid, the user will get prompted for input again. Once the user has entered a valid student id, the user will be able to change the student's name, date of birth, phone number, and email. Validations are implemented here as well for date of birth, phone number and email input, in order to make sure the input is correct.

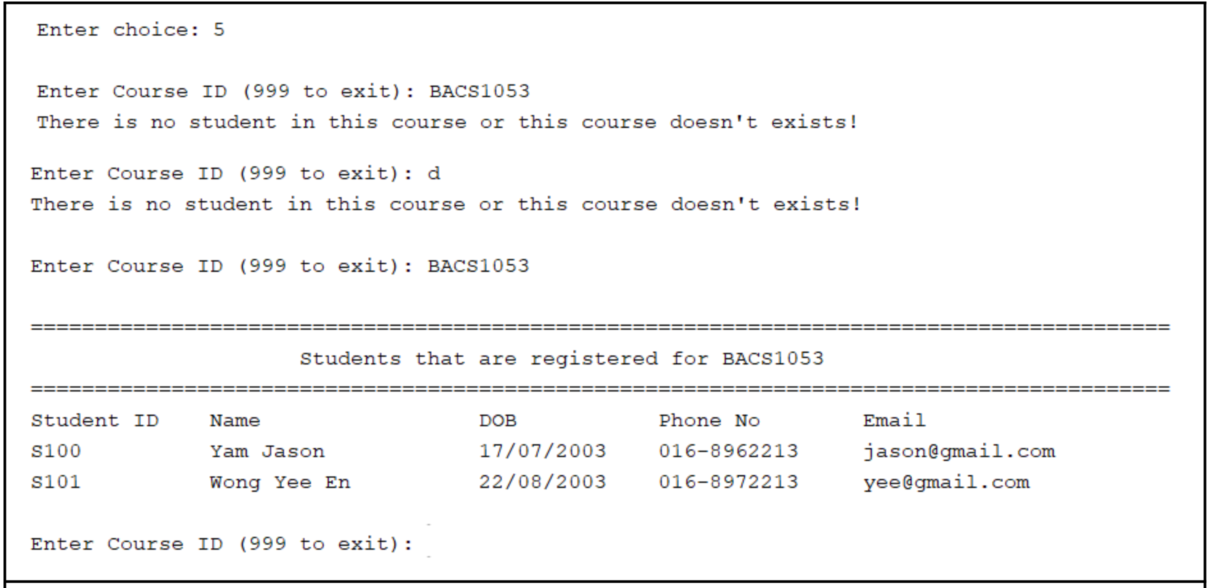
D. List all students' details



```
=====================================================================
                   Student Registration Management
=====================================================================
1.  Add new Students
2.  Remove A Student
3.  Amend Student Details
4.  List All Students' Details
5.  Search Students For Registered Course
6.  Register for a Course (main, elective, resit, repeat)
7.  Remove a Student from a course (main, elective) Registration
8.  Calculate Fee Paid for Registered Courses
9.  Filter Students For Courses Based On Criteria
10. Student Report
11. Registration Report
0.  Back
Enter choice: 4


List of Students:
StudentID  Student Name          BOD         Phone No       Email               Programme ID
S100       Yam Jason             17/07/2003  016-8962213    jason@gmail.com     RDS
S101       Wong Yee En           22/08/2003  016-8972213    yee@gmail.com       RDS
S102       Tee Yong Zheng        22/12/2003  016-8982213    tee@gmail.com       RSW
S103       Yue Zhi Jving         03/03/2003  016-8992213    jving@gmail.com     RIS
S104       Darren Tan Chia Yuan  04/01/2003  016-9962213    darren@gmail.com    DIT
S105       Lai Weng Lok          12/05/2003  016-8963213    test@.com           DIS
```

If the user enters 4 in the student registration management screen, the system will list out all the students available. This extra feature is added to make it convenient to manage and view students' details.

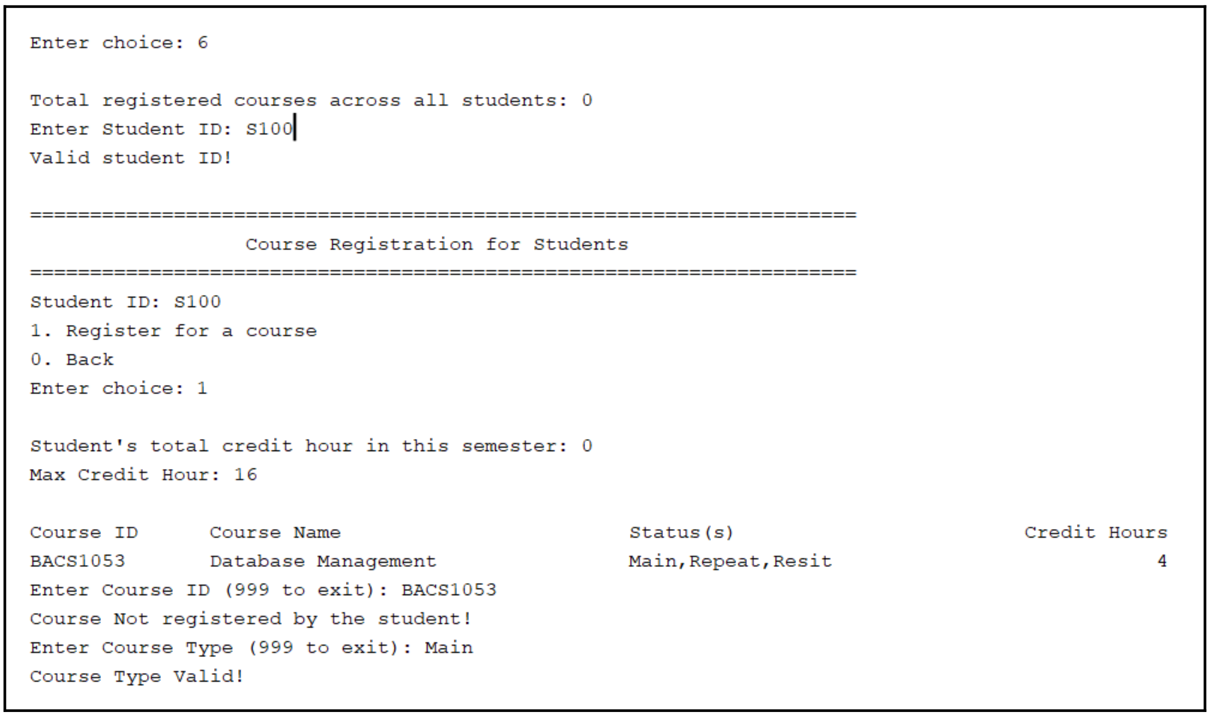
E. Search Students For Registered Course



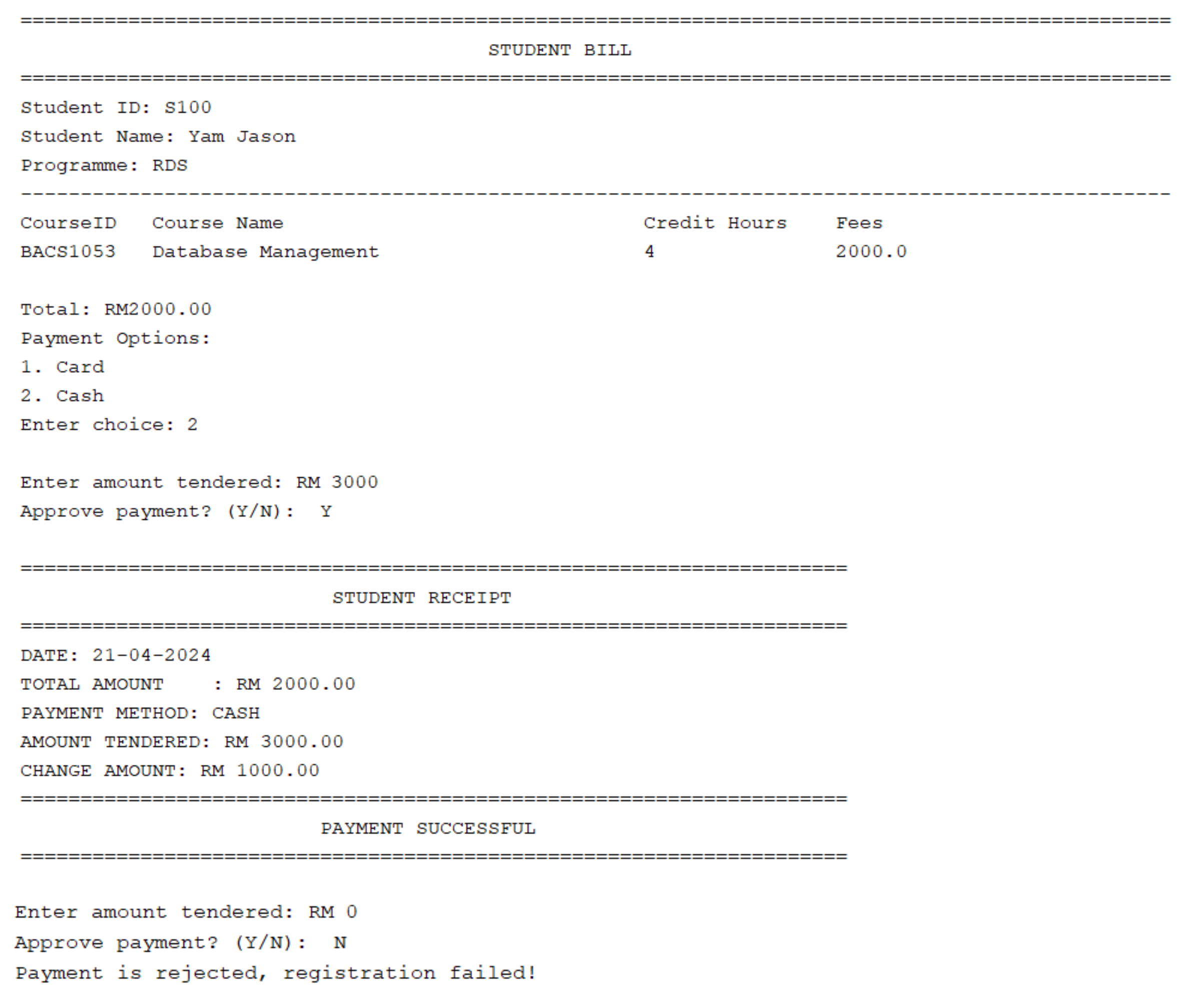
```
Enter choice: 5

Enter Course ID (999 to exit): BACS1053
There is no student in this course or this course doesn't exists!

Enter Course ID (999 to exit): d
There is no student in this course or this course doesn't exists!

Enter Course ID (999 to exit): BACS1053

==========================================================================================
                     Students that are registered for BACS1053
==========================================================================================
Student ID    Name                 DOB           Phone No       Email
S100          Yam Jason            17/07/2003    016-8962213    jason@gmail.com
S101          Wong Yee En          22/08/2003    016-8972213    yee@gmail.com

Enter Course ID (999 to exit):
```

In the 'search students for registered course' screen, the user will be prompted to enter the course id, a message will pop up informing there is no student in the course or course doesn't exist if the course id is valid but there is no student in it or if the course id is invalid. If the course id is valid and there is at least one student, a list will appear.

F. Register for a Course (main, elective, resit, repeat)

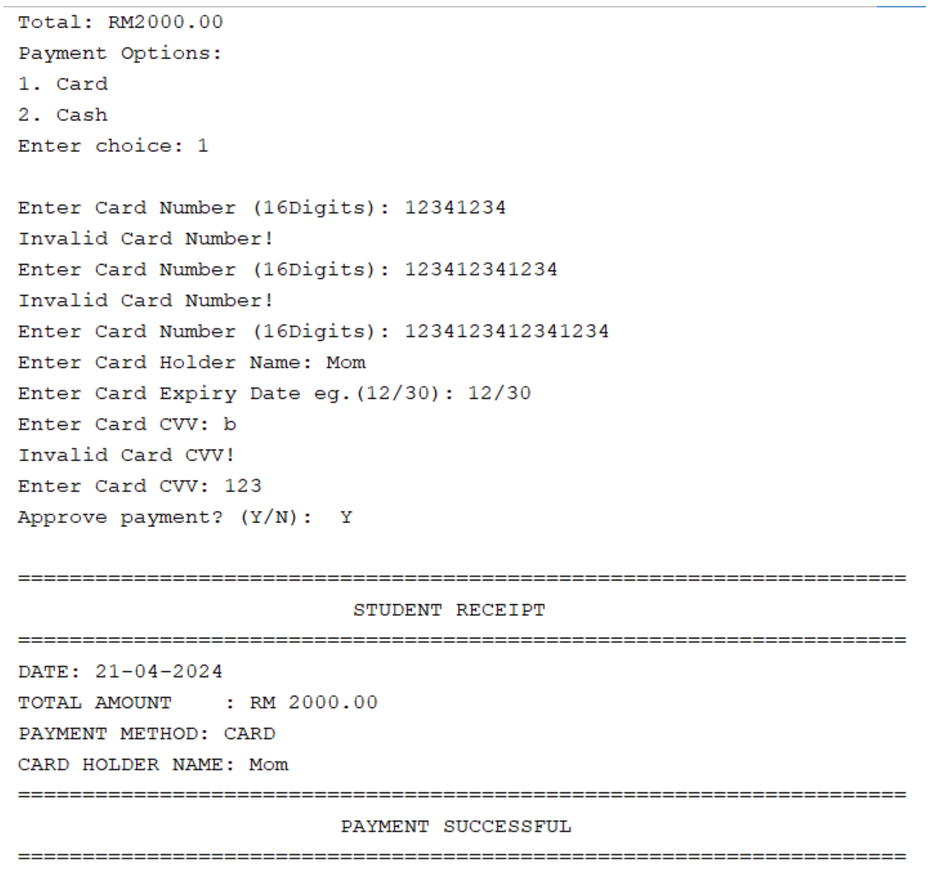


```
Enter choice: 6

Total registered courses across all students: 0
Enter Student ID: S100
Valid student ID!

=====================================================================
                 Course Registration for Students
=====================================================================
Student ID: S100
1. Register for a course
0. Back
Enter choice: 1

Student's total credit hour in this semester: 0
Max Credit Hour: 16

Course ID      Course Name                        Status(s)                        Credit Hours
BACS1053       Database Management                Main,Repeat,Resit                           4
Enter Course ID (999 to exit): BACS1053
Course Not registered by the student!
Enter Course Type (999 to exit): Main
Course Type Valid!
```



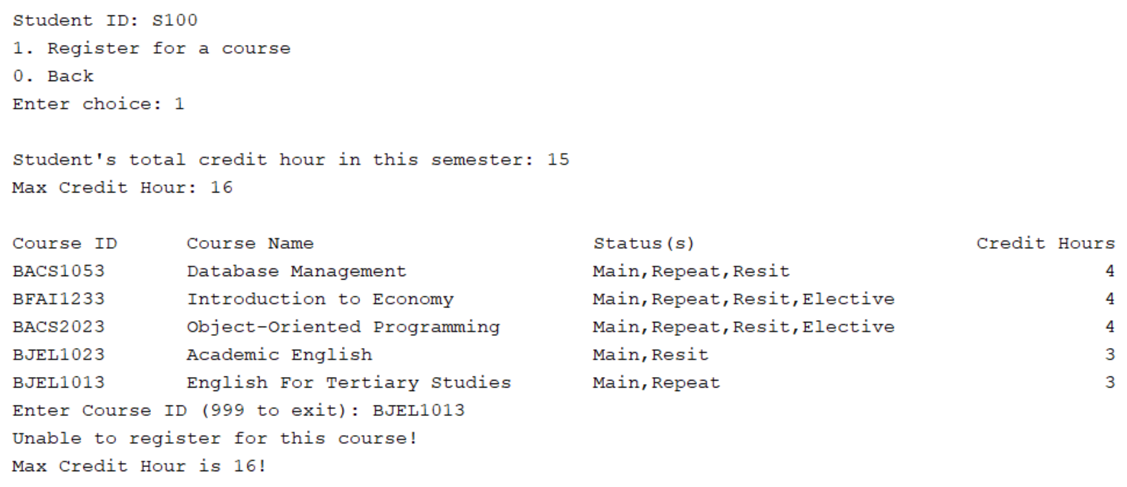
```
================================================================================================
                                        STUDENT BILL
================================================================================================
Student ID: S100
Student Name: Yam Jason
Programme: RDS
------------------------------------------------------------------------------------------------
CourseID   Course Name                                 Credit Hours    Fees
BACS1053   Database Management                         4               2000.0

Total: RM2000.00
Payment Options:
1. Card
2. Cash
Enter choice: 2

Enter amount tendered: RM 3000
Approve payment? (Y/N):  Y

=====================================================================
                          STUDENT RECEIPT
=====================================================================
DATE: 21-04-2024
TOTAL AMOUNT     : RM 2000.00
PAYMENT METHOD: CASH
AMOUNT TENDERED: RM 3000.00
CHANGE AMOUNT: RM 1000.00
=====================================================================
                         PAYMENT SUCCESSFUL
=====================================================================

Enter amount tendered: RM 0
Approve payment? (Y/N):  N
Payment is rejected, registration failed!
```

Cash Payment



```
Total: RM2000.00
Payment Options:
1. Card
2. Cash
Enter choice: 1

Enter Card Number (16Digits): 12341234
Invalid Card Number!
Enter Card Number (16Digits): 123412341234
Invalid Card Number!
Enter Card Number (16Digits): 1234123412341234
Enter Card Holder Name: Mom
Enter Card Expiry Date eg.(12/30): 12/30
Enter Card CVV: b
Invalid Card CVV!
Enter Card CVV: 123
Approve payment? (Y/N):  Y

=====================================================================
                          STUDENT RECEIPT
=====================================================================
DATE: 21-04-2024
TOTAL AMOUNT    : RM 2000.00
PAYMENT METHOD: CARD
CARD HOLDER NAME: Mom
=====================================================================
                         PAYMENT SUCCESSFUL
=====================================================================
```

Card payment

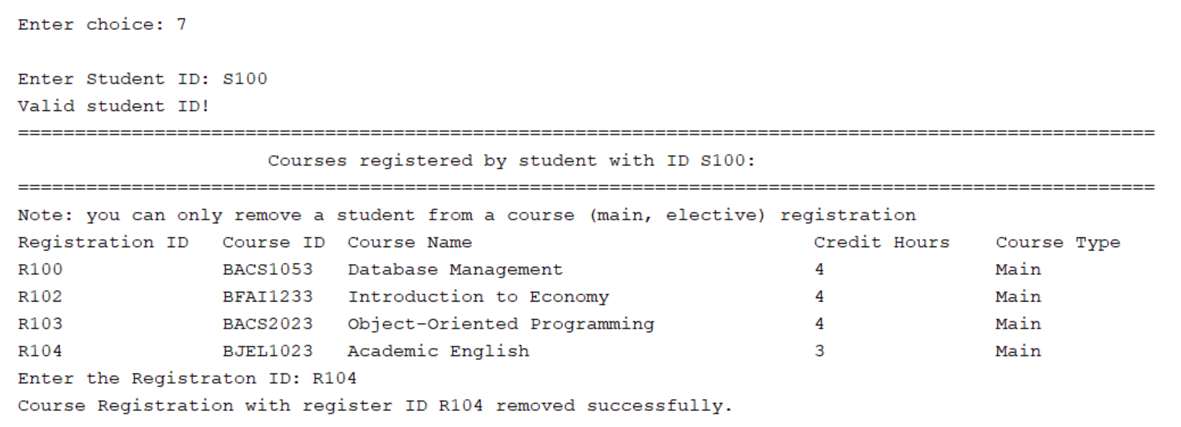


```
Student ID: S100
1. Register for a course
0. Back
Enter choice: 1

Student's total credit hour in this semester: 15
Max Credit Hour: 16

Course ID      Course Name                     Status(s)                       Credit Hours
BACS1053       Database Management             Main,Repeat,Resit                          4
BFAI1233       Introduction to Economy         Main,Repeat,Resit,Elective                 4
BACS2023       Object-Oriented Programming     Main,Repeat,Resit,Elective                 4
BJEL1023       Academic English                Main,Resit                                 3
BJEL1013       English For Tertiary Studies    Main,Repeat                                3
Enter Course ID (999 to exit): BJEL1013
Unable to register for this course!
Max Credit Hour is 16!
```

If the registration exceeds a student’s credit hour limit

To help a student register for a course, the user can enter “6” in the student management subsystem to enter the “Register for a Course (main, elective, resit, repeat)” screen. The user will need to enter the student id first before the registration process, a student can have a maximum of 16 hours of credit hours. If the registration exceeds the limit, registration will not be allowed, else continue the registration process. Once the user enters a valid course ID, the user will get prompted to enter the course type such as “Main” or “Repeat”. Next, a student will be generated then the user will have to select an option for the payment. The user can select either cash or card. If a card is selected, the card number, cvv, expiry date will be validated to avoid fraud. If cash is entered, the amount tender has to be more than the fees. The user can enter 0 if they wish to cancel registration. Lastly, the user will be prompted to approve the payment. Once the payment is approved, the registration will be successful and a receipt will be shown, else the registration will be cancelled.

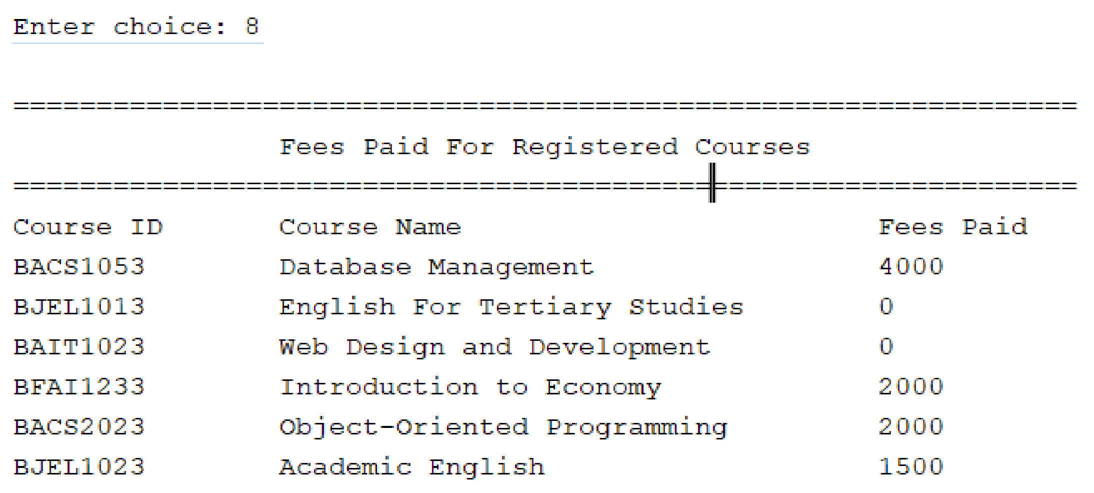
G. Remove a Student from a course (main, elective) Registration



```
Enter choice: 7

Enter Student ID: S100
Valid student ID!
====================================================================================================
                      Courses registered by student with ID S100:
====================================================================================================
Note: you can only remove a student from a course (main, elective) registration
Registration ID  Course ID  Course Name                             Credit Hours   Course Type
R100             BACS1053   Database Management                     4              Main
R102             BFAI1233   Introduction to Economy                 4              Main
R103             BACS2023   Object-Oriented Programming             4              Main
R104             BJEL1023   Academic English                        3              Main
Enter the Registraton ID: R104
Course Registration with register ID R104 removed successfully.
```

Enter 7 in the student management subsystem menu to enter the “Remove a Student from a course (main, elective) Registration” screen. In this screen, the user will have to enter the student id then the registration id to remove a student from a course registration. If the user doesn’t want to remove anything, just type anything else during the input for registration id to exit.

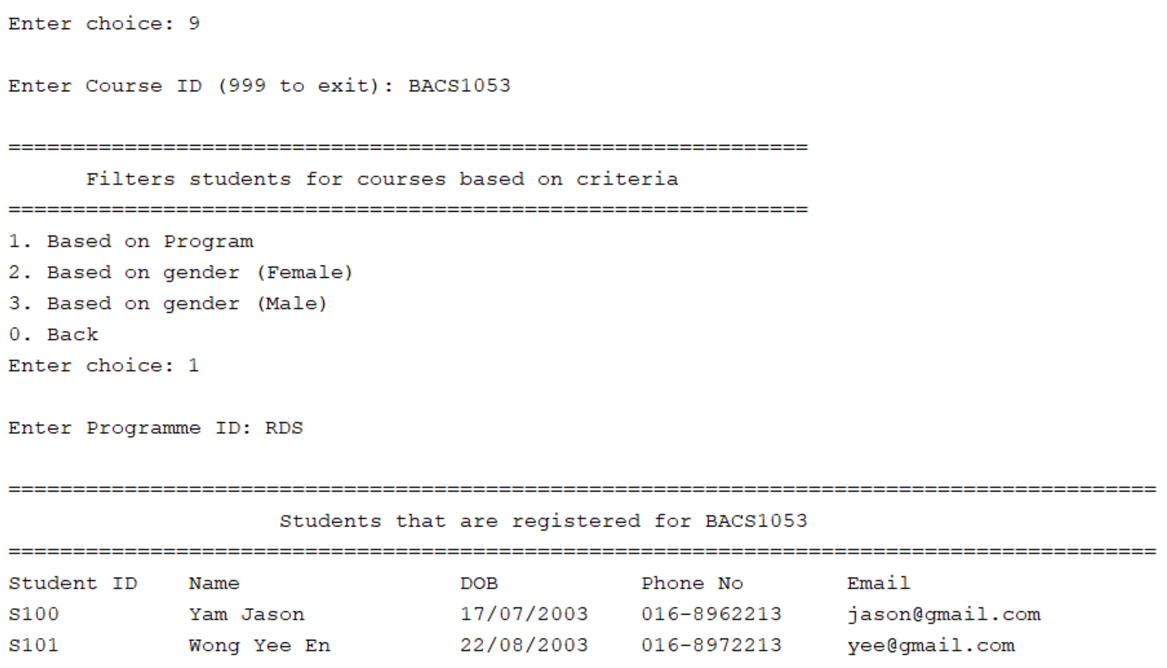
H. Calculate Fee Paid for Registered Courses



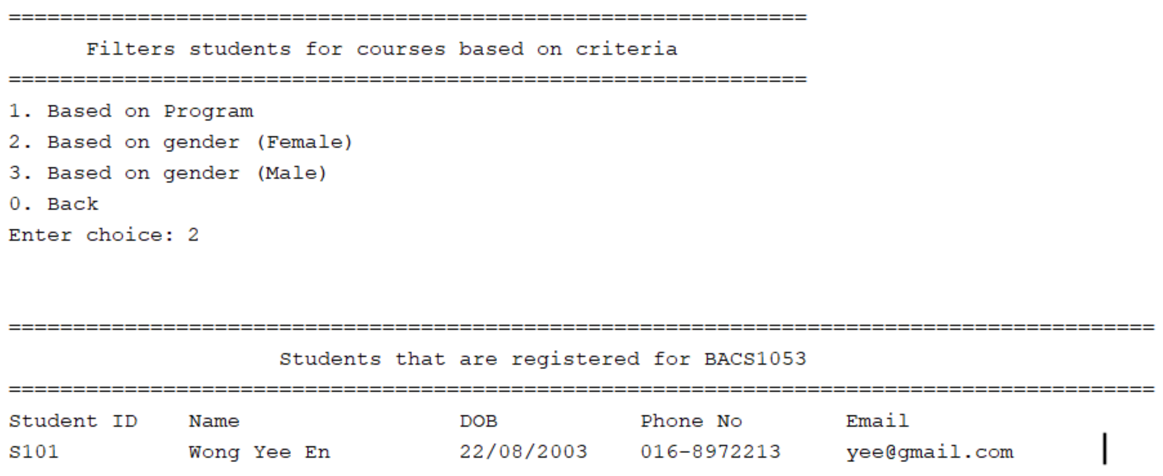
```
Enter choice: 8

================================================================
                Fees Paid For Registered Courses
================================================================
Course ID       Course Name                         Fees Paid
BACS1053        Database Management                 4000
BJEL1013        English For Tertiary Studies        0
BAIT1023        Web Design and Development          0
BFAI1233        Introduction to Economy             2000
BACS2023        Object-Oriented Programming         2000
BJEL1023        Academic English                    1500
```

To calculate the fee paid for registered courses, the user can enter 8 in the student management subsystem menu. In this screen, the total fees paid for registration for each existing course will be shown in a list.

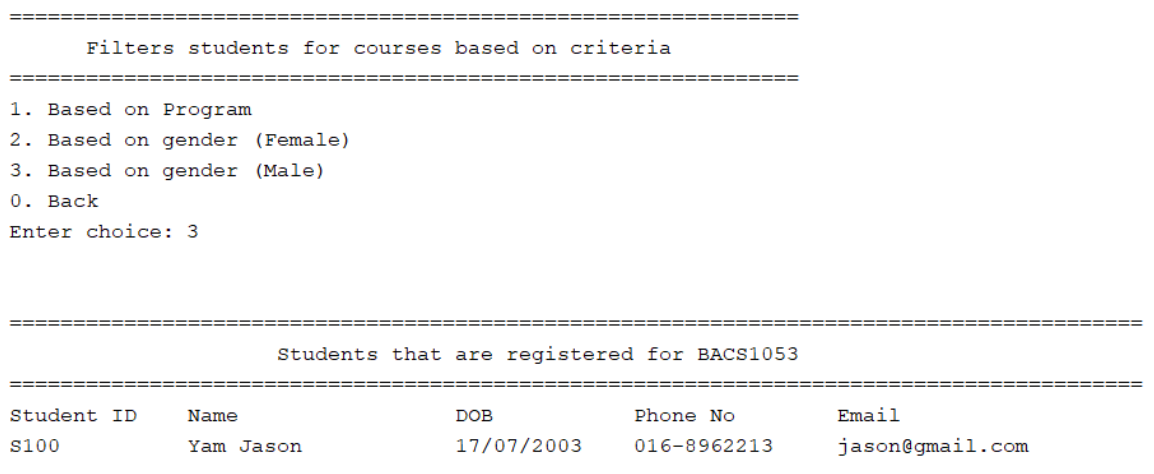
I. Filter Students For Courses Based On Criteria



```
Enter choice: 9

Enter Course ID (999 to exit): BACS1053

==============================================================
      Filters students for courses based on criteria
==============================================================
1. Based on Program
2. Based on gender (Female)
3. Based on gender (Male)
0. Back
Enter choice: 1

Enter Programme ID: RDS

=========================================================================================
                     Students that are registered for BACS1053
=========================================================================================
Student ID    Name                DOB           Phone No         Email
S100          Yam Jason           17/07/2003    016-8962213      jason@gmail.com
S101          Wong Yee En         22/08/2003    016-8972213      yee@gmail.com
```



```
==============================================================
      Filters students for courses based on criteria
==============================================================
1. Based on Program
2. Based on gender (Female)
3. Based on gender (Male)
0. Back
Enter choice: 2


=========================================================================================
                     Students that are registered for BACS1053
=========================================================================================
Student ID    Name                DOB           Phone No         Email
S101          Wong Yee En         22/08/2003    016-8972213      yee@gmail.com
```

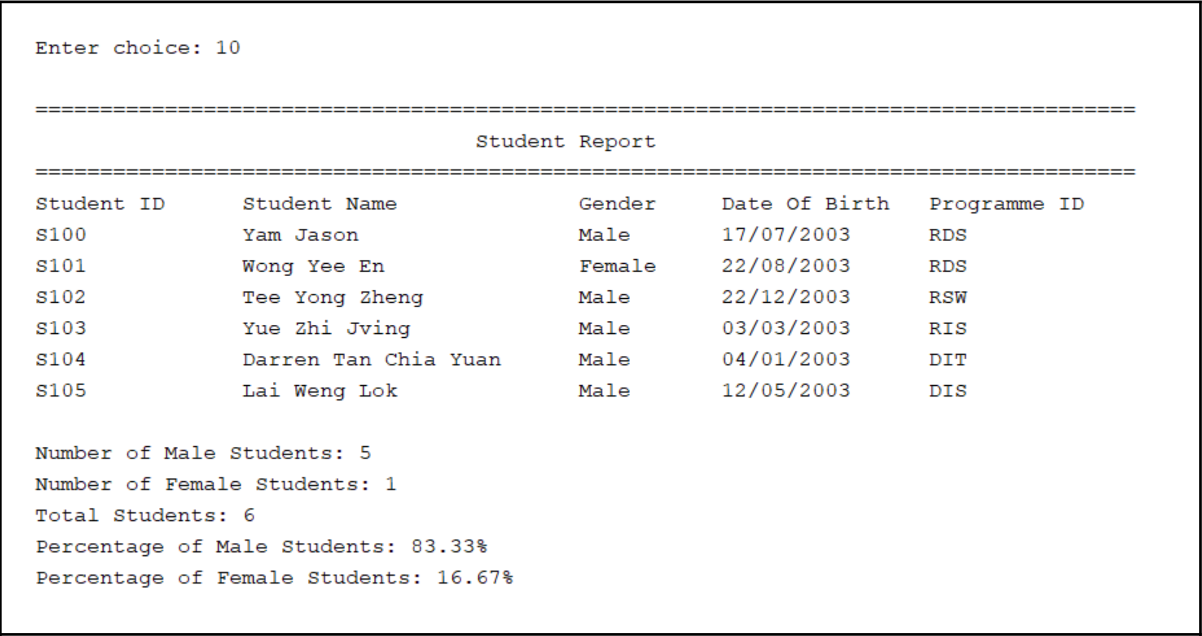


```
==============================================================
      Filters students for courses based on criteria
==============================================================
1. Based on Program
2. Based on gender (Female)
3. Based on gender (Male)
0. Back
Enter choice: 3


=========================================================================================
                     Students that are registered for BACS1053
=========================================================================================
Student ID    Name                DOB           Phone No         Email
S100          Yam Jason           17/07/2003    016-8962213      jason@gmail.com
```

9 is to enter the “Filter Students For Courses Based On Criteria” screen. The user will be prompted to enter the course id they want. Then 3 options will be shown for the user to

select. 1 is to filter based on program, 2 is filter based on female, and 3 is based on male. If the user enters a valid course id that has zero student or invalid course id, a message will pop up saying “There is no student in this course or this course doesn't exists!”.

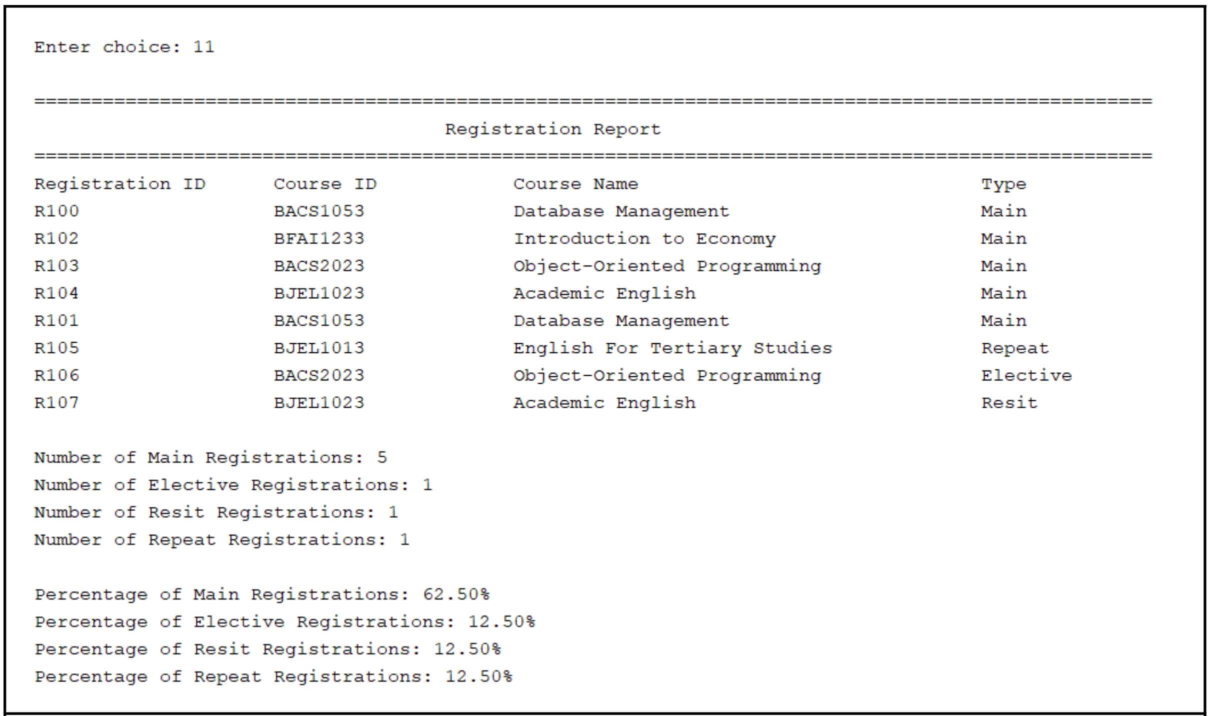
J. Summary Report 1 (Student Report)



```
Enter choice: 10

======================================================================================
                                  Student Report
======================================================================================
Student ID      Student Name              Gender     Date Of Birth   Programme ID
S100            Yam Jason                 Male       17/07/2003      RDS
S101            Wong Yee En               Female     22/08/2003      RDS
S102            Tee Yong Zheng            Male       22/12/2003      RSW
S103            Yue Zhi Jving             Male       03/03/2003      RIS
S104            Darren Tan Chia Yuan      Male       04/01/2003      DIT
S105            Lai Weng Lok              Male       12/05/2003      DIS

Number of Male Students: 5
Number of Female Students: 1
Total Students: 6
Percentage of Male Students: 83.33%
Percentage of Female Students: 16.67%
```

The summary report 1 which is a student report can be entered by entering 10 in the student management subsystem screen. In this report, a student list will be shown and the number of students based on gender and percentage of male and female students will also be shown.

K. Summary Report 2 (Registration Report)



```
Enter choice: 11

===================================================================================================
                                    Registration Report
===================================================================================================
Registration ID     Course ID           Course Name                             Type
R100                BACS1053            Database Management                     Main
R102                BFAI1233            Introduction to Economy                 Main
R103                BACS2023            Object-Oriented Programming             Main
R104                BJEL1023            Academic English                        Main
R101                BACS1053            Database Management                     Main
R105                BJEL1013            English For Tertiary Studies            Repeat
R106                BACS2023            Object-Oriented Programming             Elective
R107                BJEL1023            Academic English                        Resit

Number of Main Registrations: 5
Number of Elective Registrations: 1
Number of Resit Registrations: 1
Number of Repeat Registrations: 1

Percentage of Main Registrations: 62.50%
Percentage of Elective Registrations: 12.50%
Percentage of Resit Registrations: 12.50%
Percentage of Repeat Registrations: 12.50%
```

To go to the second summary report, which is the registration report, the user can enter 11 in the student management subsystem screen. In this report, the number of main, elective, resit, and repeat registration will be shown in numbers and also in percentage. A list will also be shown.

Name	Student ID	Prog / Tut.Grp	Signature
Wong Yee En	22WMR365 9	RDS/3	[illegible]

Subsystem: Course Management Subsystem

1. Source codes for Control classes

```
package control;

import adt.*;
import entity.*;
import dao.*;
import boundary.*;
import java.io.Serializable;
import java.util.InputMismatchException;
import java.util.Iterator;
import utility.MessageUI;

/**
 *
 * @author Name: Wong Yee En RDS2Y2S2G3 22WMR13659
 */
public class CourseManagement implements Serializable {

    private MapInterface<String, Faculty> facultyMap = new HashMap<>();
    private MapInterface<String, Programme> programmeMap = new HashMap<>();
//focs,fafb,...
    private MapInterface<String, Course> courseMap = new HashMap<>();
    private ListInterface<ProgrammeCourse> programmeCourseList = new ArrayList<>(); //
[rds,aaa],[rda,bbb],....

    SetInterface<String> programmesThatHasCourses = new ArraySet<>();
    SetInterface<String> coursesThatHaveProgramme = new ArraySet<>();

    private ListInterface<String> selectedProgrammeList = new ArrayList<>(); //rds,....

    private final FacultyDAO facultyDAO = new FacultyDAO("faculties.dat");
    private final ProgrammeDAO programmeDAO = new
ProgrammeDAO("programmes.dat");
    private final CourseDAO courseDAO = new CourseDAO("courses.dat");
    private final ProgrammeCourseDAO programmeCourseDAO = new
ProgrammeCourseDAO("programmeCourses.dat");
    private final CourseManagementUI courseManagementUI = new CourseManagementUI();
```

```
public CourseManagement() {
    facultyMap = facultyDAO.retrieveFromFile();
    programmeMap = programmeDAO.retrieveFromFile();
    courseMap = courseDAO.retrieveFromFile();
    programmeCourseList = programmeCourseDAO.retrieveFromFile();

}

public void start() {
    int choice = 0;

    do {
        choice = courseManagementUI.getMenuChoice();
        switch (choice) {
            case 1: {
                addProgrammetoCourses();
                break;
            }

            case 2: {
                removeProgrammeFromCourse();
                break;
            }

            case 3: {
                addNewCourseToProgrammes();
                break;
            }
            case 4: {
                removeCourseFromProgramme();
                break;
            }

            case 5: {
                searchCoursesOfferedInSemester();
                break;
            }

            case 6: {
                amendCourseDetailsForProgramme();
                break;
            }
            case 7: {
                listCoursesTakenByDifferentFaculties();
                break;
            }
            case 8: {
                listAllCoursesForAProgramme();
                break;
                }

                case 9: {
                    courseSummaryReport();
                    break;
                }
                case 10: {
                    programmeSummaryReport();
                    break;
                }

                case 0: {
                    MessageUI.displayBackMessage();
                    break;
                }
                default:
                    courseManagementUI.displayInvalidChoice();
            }
        } while (choice != 0);
    }

    // TASK 1
    // Add a Prorgramme to Courses
    public void addProgrammetoCourses() {
        // Display Title first
        courseManagementUI.displayAddProgrammeTitle();

        // Display all programmes available
        displayAllProgrammes();

        String programmeID = validateInputProgrammeID();

        if (programmeID == null) {
            return;
        }

        Programme programme = programmeMap.get(programmeID);
        displayAllCourses();

        int totalCreditHours = calculateTotalCreditHours(programme); // Calculate total credit
  hours

        boolean continueToAdd = true;

        do {
            if (!continueToAdd) {
                return;
            }
            String courseID = validateInputCourseID();

            if (courseID == null) {
                continueToAdd = false;
            } else {
                Course course = courseMap.get(courseID);
                int newTotalCreditHours = totalCreditHours + course.getCreditHours(); // Calculate
new total credit hours

                if (newTotalCreditHours <= 18) { // Check if adding the course exceeds the limit
                    ProgrammeCourse programmeCourse = new
ProgrammeCourse(programme.getProgrammeId(), course.getCourseId());

                    if (programmeCourseList.contains(programmeCourse)) {
                        courseManagementUI.displayProgrammeHasBeenAddedBefore(programme);
                    } else {
                        programmeCourseList.add(programmeCourse);
                        programmeCourseDAO.saveToFile(programmeCourseList);

courseManagementUI.displayProgrammeIsSuccessfullyAddedToCourse(course,
programme);
                        totalCreditHours = newTotalCreditHours; // Update total credit hours
                    }
                } else {
                    courseManagementUI.displayExceed18();
                    //System.out.println("Exceed 18 total credit hours! Cant add anymore. Fail to add
course.");
                    // Display message for exceeding limit
                }
            }
        } while (continueToAdd);
    }

    // FOR TASK 1 & 3 (CODE REUSE)
    private int calculateTotalCreditHours(Programme programme) {
        int totalCreditHours = 0;
        for (ProgrammeCourse pc : programmeCourseList) {
            if (pc.getProgrammeID().equals(programme.getProgrammeId())) {
                Course course = courseMap.get(pc.getCourseID());
                totalCreditHours += course.getCreditHours();
            }
        }
        return totalCreditHours;
    }

    // FOR TASK 1
    public void displayAllProgrammes() {
        StringBuilder sb = new StringBuilder();
        for (Programme programme : programmeMap.values()) {
```

```
        sb.append(programme.toString());
        sb.append("\n");
    }
    courseManagementUI.listProgrammes(sb.toString());
}

// FOR TASK 1
public void displayAllCourses() {
    StringBuilder sb = new StringBuilder();
    for (Course course : courseMap.values()) {
        sb.append(course.toString());
        sb.append("\n");
    }
    courseManagementUI.listCourses(sb.toString());
}

// FOR TASK 1
private String validateInputProgrammeID() {
    String programmeID = null;
    boolean isValidFormat = false;
    boolean programmeIDExist = false;
    String regexProgrammeID = "[A-Z]{3}";
    do {
        System.out.println("");
        try {
            programmeID = courseManagementUI.inputProgrammeID().toUpperCase();

            if (!programmeID.equals("999")) {
                if (programmeID.matches(regexProgrammeID)) {
                    isValidFormat = true;
                    if (programmeMap.containsKey(programmeID)) {
                        programmeIDExist = true;
                    } else {
                        courseManagementUI.displayNoMatchProgrammeID();
                    }
                } else {
                    courseManagementUI.displayProgrammeIDFormatIncorrect();
                }
            } else {
                programmeID = null;
                break;

            }

        } catch (InputMismatchException e) {
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidFormat || !programmeIDExist);
    return programmeID;
```

```
}

// FOR TASK 1
private String validateInputCourseID() {
    String courseID = null;
    boolean isValidFormat = false;
    boolean courseIDExist = false;
    String regexCourseID = "[A-Z]{4}\\d{4}";
    do {
        System.out.println("");
        try {
            courseID = courseManagementUI.inputCourseID().toUpperCase();

            if (!courseID.equals("999")) {
                if (courseID.matches(regexCourseID)) {
                    isValidFormat = true;
                    if (courseMap.containsKey(courseID)) {
                        courseIDExist = true;
                    } else {
                        courseManagementUI.displayNoMatchCourseID();
                    }
                } else {
                    courseManagementUI.displayCourseIDFormatIncorrectAndExample();
                }
            } else {
                courseID = null;
                break;

            }

        } catch (InputMismatchException e) {
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidFormat || !courseIDExist);
    return courseID;
}

//Task 4
//Remove a course from a programme
public void removeCourseFromProgramme() {
    courseManagementUI.displayRemoveCourseTitle();

    // if no any record in bridge table, display error message, exit this method
    if (programmeCourseList.isEmpty()) {
        courseManagementUI.noRecordFoundInBridgeTableMsg();
        return;
    }
    // if has record, display the programme(s) that inside the bridge table only,
    //because only when the bridge table has the entry about the programme, user can
```

remove the programme from a course

```
        courseManagementUI.displayOnlyProgrammesBelowHaveCourses();

        displayProgrammesThatHasCourses();
        String programmeID = validateInputProgrammeIDThatHasCourse();

        if (programmeID == null) {
            return;
        }

        Programme selectedProgramme = programmeMap.get(programmeID);

        SetInterface<String> coursesOfSelectedProgramme = new ArraySet<>();
        courseManagementUI.displayCoursesOfSelectProgrammeNote();
        displayCoursesOfSelectedProgramme(selectedProgramme,
coursesOfSelectedProgramme);

        String courseID =
validateInputCourseIdOfSelectedProgramme(coursesOfSelectedProgramme);

        //if valid courseID is keyed in
        if (courseID != null) {
            Course selectedCourse = courseMap.get(courseID);
            ProgrammeCourse programmeCourseToBeRemoved = new
ProgrammeCourse(programmeID, courseID);

            for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
                ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
                if (programmeCourseToBeRemoved.equals(programmeCourse)) {
                    programmeCourseList.remove(i);
                    programmesThatHasCourses.remove(selectedProgramme.getProgrammeId());
                    coursesThatHaveProgramme.remove(selectedCourse.getCourseId());
                    coursesOfSelectedProgramme.remove(selectedCourse.getCourseId());

courseManagementUI.removedCourseFromProgrammeSuccessMsg(selectedCourse,
selectedProgramme);
//                  System.out.println("Course " + selectedCourse.getCourseName() + "is removed
successfully from programme " + selectedProgramme.getProgrammeName());
                    programmeCourseDAO.saveToFile(programmeCourseList);
                    return;
                    //if the entry that want to be removed is found, exit method
                }
            }
            //if the entry that want to be removed is not found
            //display error message
            System.out.println("The course is not in the programme.");

        }
```

```
    }

    // FOR TASK 4
    private String validateInputCourseIdOfSelectedProgramme(SetInterface<String> 
coursesOfSelectedProgramme) {
        String courseID = null;
        boolean isValidFormat = false;
        boolean courseIDExist = false;
        String regexCourseID = "[A-Z]{4}\\d{4}";
        do {
            System.out.println("");
            try {
                courseID = courseManagementUI.inputCourseID().toUpperCase();

                if (!courseID.equals("999")) {
                    if (courseID.matches(regexCourseID)) {
                        isValidFormat = true;
                        if (courseMap.containsKey(courseID)) {
                            if (coursesOfSelectedProgramme.contains(courseID)) {
                                courseIDExist = true;
                            } else {
                                courseManagementUI.displayCourseDontHaveThisProgramme();
                            }
                        } else {
                            courseManagementUI.displayNoThisCourse();
                        }

                    } else {
                        courseManagementUI.displayCourseIDFormatIncorrect();
                    }
                } else {
                    courseID = null;
                    break;

                }

            } catch (InputMismatchException e) {
                courseManagementUI.displayInvalidInput();
            }
        } while (!isValidFormat || !courseIDExist);
        return courseID;
    }

    // FOR TASK 4
    private String validateInputProgrammeIDThatHasCourse() {
        String programmeID = null;
        boolean isValidFormat = false;
        boolean programmeIDExist = false;
        String regexProgrammeID = "[A-Z]{3}";
        do {
            System.out.println("");
            try {
                programmeID = courseManagementUI.inputProgrammeID().toUpperCase();

                if (!programmeID.equals("999")) {
                    if (programmeID.matches(regexProgrammeID)) {
                        isValidFormat = true;
                        if (programmeMap.containsKey(programmeID)) {
                            if (programmesThatHasCourses.contains(programmeID)) {
                                programmeIDExist = true;
                            } else {
                                courseManagementUI.displayThisProgrammeDontHaveAnyCourse();
                            }
                        } else {
                            courseManagementUI.displayProgrammeDoesNotExist();
                        }

                    } else {
                        courseManagementUI.displayProgrammeIDFormatIncorrect();
                    }
                } else {
                    programmeID = null;
                    break;

                }

            } catch (InputMismatchException e) {
                courseManagementUI.displayInvalidInput();
            }
        } while (!isValidFormat || !programmeIDExist);
        return programmeID;
    }

    //FUNCTION FOR TASK 4
    public void displayProgrammesThatHasCourses() {

        for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
            ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
            String programmeID = programmeCourse.getProgrammeID();
            programmesThatHasCourses.add(programmeID);
            coursesThatHaveProgramme.add(programmeCourse.getCourseID());

        }

        String sb = "";
        for (int i = 0; i < programmesThatHasCourses.getNumberOfEntries(); i++) {

            sb += (programmeMap.get(programmesThatHasCourses.getEntry(i)) + "\n");
        }

        courseManagementUI.listProgrammes(sb);
    }

    //FUNCTION FOR TASK 4
    public void displayCoursesOfSelectedProgramme(Programme selectedProgramme, 
SetInterface<String> coursesOfSelectedProgramme) {

        for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
            ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
            String programmeID = programmeCourse.getProgrammeID();
            if (selectedProgramme.getProgrammeId().equals(programmeID)) {
                coursesOfSelectedProgramme.add(programmeCourse.getCourseID());
            }
        }

        String sb = "";
        for (int i = 0; i < coursesOfSelectedProgramme.getNumberOfEntries(); i++) {

            sb += (courseMap.get(coursesOfSelectedProgramme.getEntry(i)) + "\n");
        }

        courseManagementUI.listCourses(sb);
    }
    //FOR TASK 3 & 6 (CODE REUSABILITY)
    public SetInterface<String> selectStatusChoice() {

        int statusChoice = validateInputStatusChoice();

        SetInterface<String> status;
        SetInterface<String> status1 = new ArraySet<>();
        status1.add("Main");
        status1.add("Repeat");
        status1.add("Resit");
        status1.add("Elective");

        SetInterface<String> status2 = new ArraySet<>();
        status2.add("Main");
        status2.add("Repeat");
        status2.add("Resit");

        SetInterface<String> status3 = new ArraySet<>();
        status3.add("Main");
        status3.add("Repeat");

        SetInterface<String> status4 = new ArraySet<>();
        status4.add("Main");
        status4.add("Resit");
    switch (statusChoice) {
        case 1:
            status = status1;
            break;
        case 2:
            status = status2;
            break;
        case 3:
            status = status3;
            break;
        default:
            status = status4;
            break;
    }

    return status;
}

//TASK 3
public void addNewCourseToProgrammes() {
    // Display Title first
    courseManagementUI.displayAddNewCourseTitle();

    String courseID = validateInputCourseIDForNew();

    if (courseID == null) {
        return;
    }

    String courseName = courseManagementUI.inputCourseName();

    courseManagementUI.displayStatusChoice();

    SetInterface<String> status = selectStatusChoice();

    int creditHours = validateInputCreditHours();

    Course course = new Course(courseID, courseName, status, creditHours);

    courseManagementUI.displayEnterProgrammeIDTitle();
    displayAllProgrammes();

    boolean continueAddCourse = true;
    do {
        String programmeID = validateInputProgrammeID();

        if (programmeID == null) {
            continueAddCourse = false;
        } else {

            int totalCreditHours =
calculateTotalCreditHours(programmeMap.get(programmeID)); // Calculate total credit hours
for the programme

            if (totalCreditHours + creditHours <= 18) { // Check if adding the course exceeds
the limit

                if (!courseMap.containsKey(courseID)) {
                    courseMap.put(courseID, course);
                    courseDAO.saveToFile(courseMap);
                }
                ProgrammeCourse programmeCourse = new ProgrammeCourse(programmeID,
courseID);
                if (!programmeCourseList.contains(programmeCourse)) {
                    programmeCourseList.add(programmeCourse);
                    courseManagementUI.newCourseAddedMsg(programmeID);
                    programmeCourseDAO.saveToFile(programmeCourseList);
                } else {
                    courseManagementUI.alreadyAddedBeforeMsg(programmeID);
                }
            } else {
                courseManagementUI.displayExceed18();
                //System.out.println("Exceeds 18 total credit hours!");
                // Display message for exceeding limit
            }
        }
    } while (continueAddCourse);
}

// FOR TASK 3
private String validateInputCourseIDForNew() {

    String courseID = null;
    boolean isValidFormat;
    boolean courseIDExist;
    String regexCourseID = "[A-Z]{4}\\d{4}"; //check format of input
    do {
        isValidFormat = false;
        courseIDExist = false;
        System.out.println("");
        try {
            courseID = courseManagementUI.inputCourseID();

            if (!courseID.equals("999")) { //999 to exit
                if (courseID.matches(regexCourseID)) { //validate the format of the input here
                    isValidFormat = true;
                if (courseMap.containsKey(courseID)) {
                    courseIDExist = true;    // forbade the course registration since the ID exists
                    courseManagementUI.courseIDExistErrorMsg();
                }
            } else {
                courseManagementUI.displayCourseIDFormatIncorrectAndExample();
            }
        } else {
            courseID = null;
            break;

        }

    } catch (InputMismatchException e) {
        courseManagementUI.displayInvalidInput();
    }
} while (!isValidFormat || courseIDExist);
return courseID;
}

// FOR TASK 3
private int validateInputStatusChoice() {
    int statusChoice = 0;
    boolean isValidInput = false;
    do {
        try {
            statusChoice = Integer.parseInt(courseManagementUI.inputCourseStatusChoice());
            if (statusChoice >= 1 && statusChoice <= 4) {
                isValidInput = true;
            } else {
                System.out.println("Only choose 1 to 4!\n");
            }

        } catch (NumberFormatException e) {
            isValidInput = false;
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidInput);
    return statusChoice;
}

// FOR TASK 3
private int validateInputCreditHours() {
    int creditHours = 0;
    boolean isValidInput = false;
    do {
        try {
            creditHours = Integer.parseInt(courseManagementUI.inputCreditHours());
            if (creditHours >= 3 && creditHours <= 4) {
                isValidInput = true;
            } else {
                System.out.println("Only choose 3 or 4!\n");
            }

        } catch (NumberFormatException e) {
            isValidInput = false;
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidInput);
    return creditHours;
}
// FOR STUDENT REGISTRATION MANAGEMENT CONTROL TO RETRIEVE
LATEST DATA
public MapInterface<String, Course> getCourseMap() {
    return courseMap;
}
// FOR STUDENT REGISTRATION MANAGEMENT CONTROL TO RETRIEVE
LATEST DATA
public MapInterface<String, Programme> getProgrammeMap() {
    return programmeMap;
}

// FOR STUDENT REGISTRATION MANAGEMENT CONTROL TO RETRIEVE
LATEST DATA
public ListInterface<ProgrammeCourse> getProgrammeCourseList() {
    return programmeCourseList;
}

// TASK 2
// Remove a course from a programme
public void removeProgrammeFromCourse() {
    courseManagementUI.displayRemoveProgrammeTitle();

    // if no any record in bridge table, display error message, exit this method
    if (programmeCourseList.isEmpty()) {
        courseManagementUI.noRecordFoundInBridgeTableMsg();
        return;
    }

    courseManagementUI.displayOnlyCoursesBelowHaveProgrammes();
    displayCoursesThatHasProgrammes();
    String courseID = validateInputCourseIDThatHaveProgramme();

    if (courseID == null) {
        return;
    }

    Course selectedCourse = courseMap.get(courseID);
    SetInterface<String> programmesOfSelectedCourse = new ArraySet<>();

    courseManagementUI.displayProgrammesOfSelectCourseNote();
    displayProgrammesOfSelectedCourse(selectedCourse, programmesOfSelectedCourse);

    String programmeID =
validateInputProgrammeIdOfSelectedCourse(programmesOfSelectedCourse);

    //if valid courseID is keyed in
    if (programmeID != null) {
        Programme selectedProgramme = programmeMap.get(programmeID);
        ProgrammeCourse programmeCourseToBeRemoved = new
ProgrammeCourse(programmeID, courseID);

        for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
            ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
            if (programmeCourseToBeRemoved.equals(programmeCourse)) {
                programmeCourseList.remove(i);
                programmesThatHasCourses.remove(selectedProgramme.getProgrammeId());
                coursesThatHaveProgramme.remove(selectedCourse.getCourseId());
                programmesOfSelectedCourse.remove(selectedProgramme.getProgrammeId());
                courseManagementUI.removeSuccessfullyFromCourseMsg(selectedProgramme,
selectedCourse);
//                  System.out.println("Programme " + selectedProgramme.getProgramemeName()
+ "is removed successfully from course " + selectedCourse.getCourseName() + "!\n");
                programmeCourseDAO.saveToFile(programmeCourseList);

                return;
                //if the entry that want to be removed is found, exit method
            }
        }
        //if the entry that want to be removed is not found
        //display error message
        System.out.println("The programme is not in the course.");

    }

}

// FOR TASK 2
private String validateInputProgrammeIdOfSelectedCourse(SetInterface<String>
programmesOfSelectedCourse) {
    String programmeID = null;
    boolean isValidFormat = false;
    boolean programmeIDExist = false;
    String regexProgrammeID = "[A-Z]{3}";
    do {
        System.out.println("");
        try {
            programmeID = courseManagementUI.inputProgrammeID().toUpperCase();

            if (!programmeID.equals("999")) {
                if (programmeID.matches(regexProgrammeID)) {
                    isValidFormat = true;
                    if (programmeMap.containsKey(programmeID)) {
                        if (programmesOfSelectedCourse.contains(programmeID)) {
                            programmeIDExist = true;
                        } else {
                            courseManagementUI.displayProgrammeDontHaveThisCourse();
                        }
                    } else {
                        courseManagementUI.displayProgrammeDoesNotExist();
                    }

                } else {
                    courseManagementUI.displayProgrammeIDFormatIncorrect();
                }
            } else {
                programmeID = null;
                break;

            }

        } catch (InputMismatchException e) {
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidFormat || !programmeIDExist);
    return programmeID;
}

// FOR TASK 2
public void displayProgrammesOfSelectedCourse(Course selectedCourse,
SetInterface<String> programmesOfSelectedCourse) {

    for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
        ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
        String courseID = programmeCourse.getCourseID();
        if (selectedCourse.getCourseId().equals(courseID)) {
            programmesOfSelectedCourse.add(programmeCourse.getProgrammeID());
        }
    }

    String sb = "";
    for (int i = 0; i < programmesOfSelectedCourse.getNumberOfEntries(); i++) {

        sb += (programmeMap.get(programmesOfSelectedCourse.getEntry(i)) + "\n");
    }
    courseManagementUI.listProgrammes(sb);
}
// FOR TASK 2
private String validateInputCourseIDThatHaveProgramme() {
    String courseID = null;
    boolean isValidFormat = false;
    boolean courseIDExist = false;
    String regexCourseID = "[A-Z]{4}\\d{4}";
    do {
        System.out.println("");
        try {
            courseID = courseManagementUI.inputCourseID().toUpperCase();

            if (!courseID.equals("999")) {
                if (courseID.matches(regexCourseID)) {
                    isValidFormat = true;
                    if (courseMap.containsKey(courseID)) {
                        if (coursesThatHaveProgramme.contains(courseID)) {
                            courseIDExist = true;
                        } else {
                            courseManagementUI.displayThisCourseDontHaveAnyProgramme();
                        }
                    } else {
                        courseManagementUI.displayNoThisCourse();
                    }

                } else {
                    courseManagementUI.displayCourseIDFormatIncorrect();
                }
            } else {
                courseID = null;
                break;

            }

        } catch (InputMismatchException e) {
            courseManagementUI.displayInvalidInput();
        }
    } while (!isValidFormat || !courseIDExist);
    return courseID;
}

// FOR TASK 2
public void displayCoursesThatHasProgrammes() {

    for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
        ProgrammeCourse programmeCourse = programmeCourseList.getEntry(i);
        String courseID = programmeCourse.getCourseID();
        coursesThatHaveProgramme.add(courseID);
        programmesThatHasCourses.add(programmeCourse.getProgrammeID());

    }

    String sb = "";
    for (int i = 0; i < coursesThatHaveProgramme.getNumberOfEntries(); i++) {

        sb += (courseMap.get(coursesThatHaveProgramme.getEntry(i)) + "\n");
    }

    courseManagementUI.listCourses(sb);

}

//TASK 5
public void searchCoursesOfferedInSemester() {
    String fuzzyInput;
    do {
        courseManagementUI.displaySearchCoursesTitle();
        fuzzyInput = courseManagementUI.inputFuzzy().toUpperCase(); // Convert input to
uppercase
        if (!fuzzyInput.equals("999")) {
            boolean matchFound = false;
            if (fuzzyInput.matches("[A-Z]{4}\\d{4}")) {
                Course matchingCourse = courseMap.get(fuzzyInput);
                if (matchingCourse != null) {
                    courseManagementUI.courseTitle();
                    courseManagementUI.listCourses(matchingCourse.toString());
                    matchFound = true;
                }
            } else {
                courseManagementUI.courseTitle();
                for (Course course : courseMap.values()) {
                    if (course.getCourseId().toUpperCase().matches(".*" + fuzzyInput + ".*")
                            || course.getCourseName().toUpperCase().matches(".*" + fuzzyInput +
".*")) {
                        courseManagementUI.listCoursesPrefix(course.toString());
                        matchFound = true;
                    }
                }
            }
            // If no matches found, display a message
            if (!matchFound) {
                courseManagementUI.displayNoMatchCourse();
            }
            System.out.println("\n\n");
        }
    } while (!fuzzyInput.equals("999"));
```

```
}

// TASK 6
public void amendCourseDetailsForProgramme() {

    courseManagementUI.displayAmendTitle();
    if (programmeCourseList.isEmpty()) {
        courseManagementUI.noRecordFoundInBridgeTableMsg();
        return;
    }
    // if has record, display the programme(s) that inside the bridge table only,
    //because only when the bridge table has the entry about the programme, user can
remove the programme from a course

    courseManagementUI.displayOnlyProgrammesBelowHaveCourses();
    displayProgrammesThatHasCourses();
    String programmeID = validateInputProgrammeIDThatHasCourse();

    if (programmeID == null) {
        return;
    }

    Programme selectedProgramme = programmeMap.get(programmeID);

    SetInterface<String> coursesOfSelectedProgramme = new ArraySet<>();

    courseManagementUI.displayCoursesOfSelectProgrammeNote();
    displayCoursesOfSelectedProgramme(selectedProgramme,
coursesOfSelectedProgramme);

    String courseID =
validateInputCourseIdOfSelectedProgramme(coursesOfSelectedProgramme);

    //if valid courseID is keyed in
    if (courseID != null) {
        //locate the course in HashMap
        Course course = courseMap.get(courseID);

        int choice = 0;
        do {
            choice = courseManagementUI.getAmendChoice(courseID);
            switch (choice) {
                case 999:
                    MessageUI.displayBackMessage();
                    return;
                case 1:
                    boolean isDifferentName;
                    do {
                        isDifferentName = true;
                        String courseName = courseManagementUI.inputCourseName();
                        String prevName = course.getCourseName();
                        if (prevName.equals(courseName)) {
                            System.out.println("Same name as initial.\n");
                            isDifferentName = false;
                        } else {
                            course.setCourseName(courseName);
                            MessageUI.displayUpdateMessage();
                            courseManagementUI.displayChangedName(prevName, courseName);
                        }

                    } while (!isDifferentName);

                    break;
                case 2:
                    boolean isDifferentStatus;
                    do {
                        isDifferentStatus = true;
                        courseManagementUI.displayStatusChoice();
                        SetInterface<String> status = selectStatusChoice();
                        String prevStatus = course.getStatus().toString();

                        if (prevStatus.equals(status.toString())) {
                            System.out.println("Same status as initial.\n");
                            isDifferentStatus = false;
                        } else {
                            course.setStatus(status);
                            MessageUI.displayUpdateMessage();
                            courseManagementUI.displayChangedName(prevStatus,
status.toString());
                        }

                    } while (!isDifferentStatus);

                    break;
                case 3:

                    boolean isDifferentCreditHours;
                    do {
                        isDifferentCreditHours = true;
                        int creditHours = validateInputCreditHours();
                        int prevCreditHours = course.getCreditHours();
                        if (prevCreditHours == creditHours) {
                            System.out.println("Same credit hours as initial.\n");
                            isDifferentCreditHours = false;
                        } else {
                            course.setCreditHours(creditHours);
                            MessageUI.displayUpdateMessage();
                            courseManagementUI.displayChangedCreditHours(prevCreditHours,
creditHours);
                        }
                    } while (!isDifferentCreditHours);

                    break;
                default:
                    MessageUI.displayInvalidChoiceMessage();

            }

            courseDAO.saveToFile(courseMap);

        } while (choice != 999);

    }

}

// TASK 7
// List courses taken by different faculties
public void listCoursesTakenByDifferentFaculties() {

    for (String facultyID : facultyMap.keys()) {
        SetInterface<String> coursesTakenByFaculty = new ArraySet<>();
        for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {

            if
(facultyMap.get(facultyID).getFacultyProgrammesMap().containsKey(programmeCourseList
.getEntry(i).getProgrammeID())) {
                coursesTakenByFaculty.add(programmeCourseList.getEntry(i).getCourseID());
            }
        }


courseManagementUI.displayDifferentFacultiesTitle(facultyMap.get(facultyID).getFacultyN
ame());

        if (!coursesTakenByFaculty.isEmpty()) {

            StringBuilder sb = new StringBuilder();

            Iterator ite = coursesTakenByFaculty.getIterator();
            while (ite.hasNext()) {
                sb.append(courseMap.get(ite.next().toString()));
                sb.append("\n");
            }

            courseManagementUI.listCourses(sb.toString());
        } else {
            System.out.println("\nNo courses taken by this faculty.\n");
        }
    }

}

//TASK 8
// List all courses for a programme
public void listAllCoursesForAProgramme() {
    courseManagementUI.displayListAllCoursesForAProgrammeTitle();
    courseManagementUI.displayOnlyProgrammesBelowHaveCourses();

    if(programmeCourseList.isEmpty()){
        courseManagementUI.displayNoProgrammeHaveCourse();
        return;
    }

    displayProgrammesThatHasCourses();
    String programmeID = validateInputProgrammeIDThatHasCourse();

    if (programmeID == null) {
        return;
    }
    SetInterface<String> coursesIDInAProgramme = new ArraySet<>();
    for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
        if (programmeCourseList.getEntry(i).getProgrammeID().equals(programmeID)) {
            coursesIDInAProgramme.add(programmeCourseList.getEntry(i).getCourseID());
        }
    }

    Iterator ite = coursesIDInAProgramme.getIterator();
    StringBuilder sb = new StringBuilder();

    while (ite.hasNext()) {
        sb.append(courseMap.get(ite.next().toString()));
        sb.append("\n");

    }

    courseManagementUI.displayCoursesInSpecificProgrammeTitle(programmeID);
    courseManagementUI.listCoursesInProgramme(sb.toString());
}

//SUMMARY REPORT 1
public void courseSummaryReport() {
    courseManagementUI.displaySummaryReportTitle();

    if (programmeCourseList.isEmpty()) {
        System.out.println("\nThere is no any record yet!\n");
```

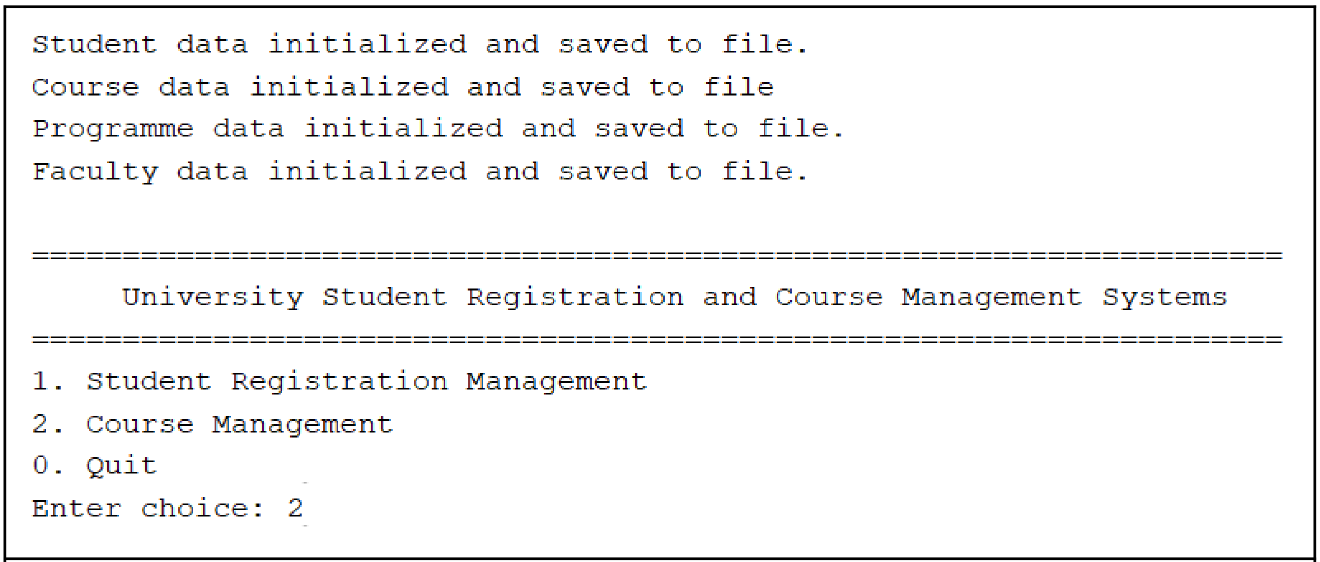
```
            courseManagementUI.endSummaryReport();
            return;
        }

        int numberOfCourses = 0;
        int numberOfMain = 0, numberOfRepeat = 0, numberOfResit = 0, numberOfElective =
0;

        int maxNumberOfTakenByProgrammes = 0;
        int minNumberOfTakenByProgrammes = Integer.MAX_VALUE;
        ListInterface<String> coursesWithMaxProgramme = new ArrayList<>();
        ListInterface<String> coursesWithMinProgramme = new ArrayList<>();

        int maxNumberOfFaculty = 0;
        int minNumberOfFaculty = Integer.MAX_VALUE;
        ListInterface<String> courseWithMaxFaculty = new ArrayList<>();
        ListInterface<String> courseWithMinFaculty = new ArrayList<>();

        StringBuilder sb = new StringBuilder();
        for (Course course : courseMap.values()) {
            int numberOfTakenByProgrammes = 0;
            SetInterface<String> takenByFaculty = new ArraySet<>();
            for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
                if (programmeCourseList.getEntry(i).getCourseID().equals(course.getCourseId()))
{
                    numberOfTakenByProgrammes++;

                    for (Faculty faculty : facultyMap.values()) {
                        if
(faculty.getFacultyProgrammesMap().containsKey(programmeCourseList.getEntry(i).getPro
grammeID())) {
                            takenByFaculty.add(faculty.getFacultyId());

                        }
                    }

                }
            }

            sb.append(++numberOfCourses);
            sb.append("\t");
            sb.append(course.toString());
            sb.append("\t\t");
            sb.append(numberOfTakenByProgrammes);
            sb.append(" / ");
            sb.append(takenByFaculty.getNumberOfEntries());
            sb.append("\n");

            SetInterface<String> status = course.getStatus();
```

```
Iterator ite = status.getIterator();
while (ite.hasNext()) {
   String currentStatus = (String) ite.next(); // Get the current status
   switch (currentStatus) {
      case "Main" ->
         numberOfMain++;
      case "Resit" ->
         numberOfResit++;
      case "Repeat" ->
         numberOfRepeat++;
      case "Elective" ->
         numberOfElective++;
      default -> {
      }
   }
}

if (numberOfTakenByProgrammes > maxNumberOfTakenByProgrammes) {
   maxNumberOfTakenByProgrammes = numberOfTakenByProgrammes;
   coursesWithMaxProgramme.clear();
   coursesWithMaxProgramme.add(course.getCourseId());

} else if (numberOfTakenByProgrammes == maxNumberOfTakenByProgrammes) {
   coursesWithMaxProgramme.add(course.getCourseId()); // Add the course to the set
if it has the same minimum number of programmes
}

if (numberOfTakenByProgrammes < minNumberOfTakenByProgrammes) {
   minNumberOfTakenByProgrammes = numberOfTakenByProgrammes;
   coursesWithMinProgramme.clear(); // Clear the previous set
   coursesWithMinProgramme.add(course.getCourseId()); // Add the new course to
the set
} else if (numberOfTakenByProgrammes == minNumberOfTakenByProgrammes) {
   coursesWithMinProgramme.add(course.getCourseId()); // Add the course to the set
if it has the same minimum number of programmes
}

if (takenByFaculty.getNumberOfEntries() > maxNumberOfFaculty) {
   maxNumberOfFaculty = takenByFaculty.getNumberOfEntries();
   courseWithMaxFaculty.clear();
   courseWithMaxFaculty.add(course.getCourseId());
} else if (takenByFaculty.getNumberOfEntries() == maxNumberOfFaculty) {
   courseWithMaxFaculty.add(course.getCourseId());
}

if (takenByFaculty.getNumberOfEntries() < minNumberOfFaculty) {
   minNumberOfFaculty = takenByFaculty.getNumberOfEntries();
   courseWithMinFaculty.clear();
```

```
                courseWithMinFaculty.add(course.getCourseId());
            } else if (takenByFaculty.getNumberOfEntries() == minNumberOfFaculty) {
                courseWithMinFaculty.add(course.getCourseId());
            }

        }
        courseManagementUI.listCoursesSummaryReport(sb.toString());
        courseManagementUI.displaySummaryReport1Middle(numberOfCourses,
numberOfMain, numberOfResit, numberOfRepeat, numberOfElective);

        // if there are more than 1 courses with the same highest number of programmes offered,
        Iterator coursesWithMaxProgrammesIte = coursesWithMaxProgramme.iterator();
        StringBuilder maxProgrammeString = new StringBuilder();
        int num = 0;
        while (coursesWithMaxProgrammesIte.hasNext()) {
            String courseId = coursesWithMaxProgrammesIte.next().toString();
            maxProgrammeString.append("\n");
            maxProgrammeString.append(++num);
            maxProgrammeString.append(".  <");
            maxProgrammeString.append(courseMap.get(courseId).getCourseId());
            maxProgrammeString.append("> ");
            maxProgrammeString.append(courseMap.get(courseId).getCourseName());
            maxProgrammeString.append("\n");
        }

//      System.out.println("Highest Programmes Offered: [" +
maxNumberOfTakenByProgrammes + " Programmes] \n" + ss);

courseManagementUI.displayHighestNoOfProgrammes(maxNumberOfTakenByProgrammes
, maxProgrammeString);
        courseManagementUI.displayLineSummaryReport();

        Iterator coursesWithMinProgrammesIte = coursesWithMinProgramme.iterator();
        StringBuilder minProgrammeString = new StringBuilder();
        int no = 0;
        while (coursesWithMinProgrammesIte.hasNext()) {
            String courseId = coursesWithMinProgrammesIte.next().toString();
            minProgrammeString.append("\n");
            minProgrammeString.append(++no);
            minProgrammeString.append(".  <");
            minProgrammeString.append(courseMap.get(courseId).getCourseId());
            minProgrammeString.append("> ");
            minProgrammeString.append(courseMap.get(courseId).getCourseName());
            minProgrammeString.append("\n");
        }


courseManagementUI.displayLowestNoOfProgrammes(minNumberOfTakenByProgrammes,
minProgrammeString);
```

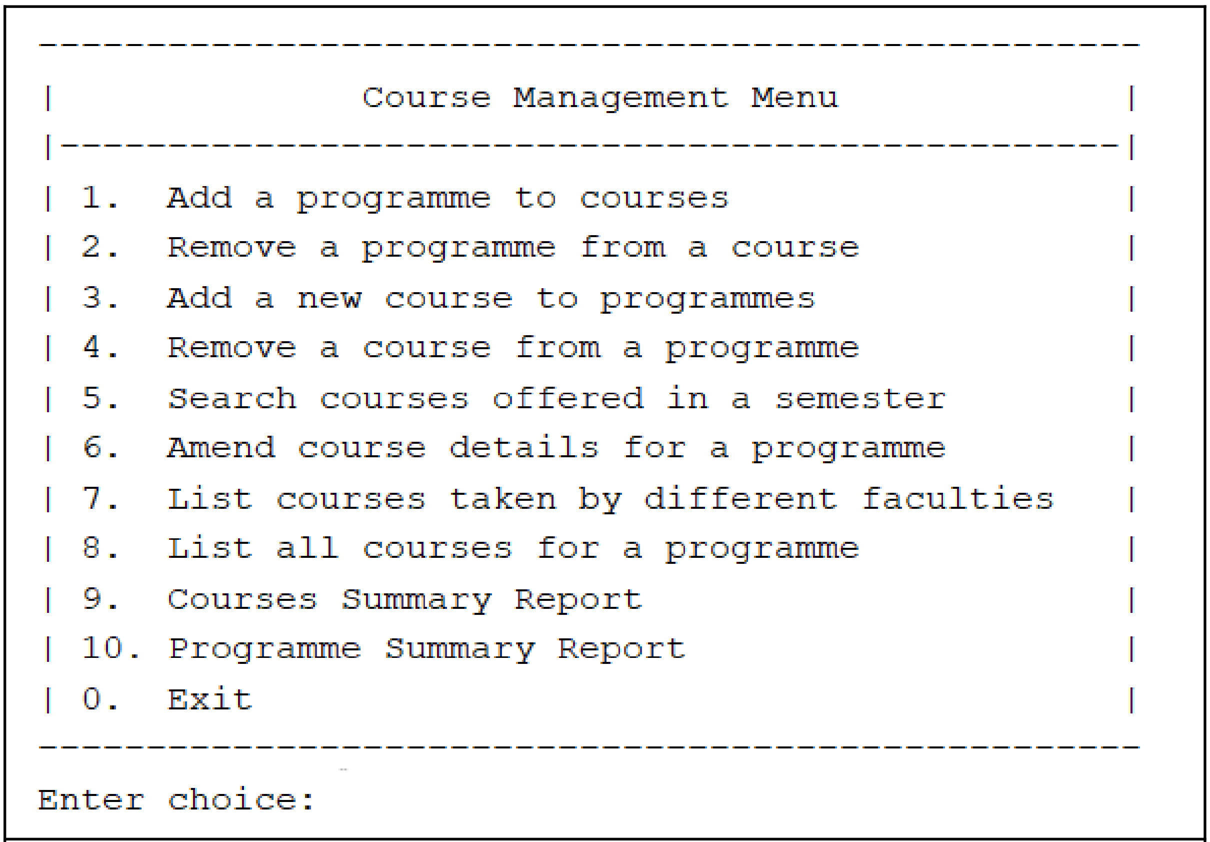
```
        Iterator coursesWithMaxFacultyIte = courseWithMaxFaculty.iterator();
        StringBuilder maxFacultyString = new StringBuilder();
        int indexMaxFaculty = 0;
        while (coursesWithMaxFacultyIte.hasNext()) {
            String courseId = coursesWithMaxFacultyIte.next().toString();
            maxFacultyString.append("\n");
            maxFacultyString.append(++indexMaxFaculty);
            maxFacultyString.append(".  <");
            maxFacultyString.append(courseMap.get(courseId).getCourseId());
            maxFacultyString.append("> ");
            maxFacultyString.append(courseMap.get(courseId).getCourseName());
            maxFacultyString.append("\n");
        }

        courseManagementUI.displayHighestNoOfFaculties(maxNumberOfFaculty,
maxFacultyString);
        courseManagementUI.displayLineSummaryReport();

        Iterator coursesWithMinFacultyIte = courseWithMinFaculty.iterator();
        StringBuilder minFacultyString = new StringBuilder();
        int indexMinFaculty = 0;
        while (coursesWithMinFacultyIte.hasNext()) {
            String courseId = coursesWithMinFacultyIte.next().toString();
            minFacultyString.append("\n");
            minFacultyString.append(++indexMinFaculty);
            minFacultyString.append(".  <");
            minFacultyString.append(courseMap.get(courseId).getCourseId());
            minFacultyString.append("> ");
            minFacultyString.append(courseMap.get(courseId).getCourseName());
            minFacultyString.append("\n");
        }

        courseManagementUI.displayLowestNoOfFaculties(minNumberOfFaculty,
minFacultyString);

        courseManagementUI.endSummaryReport();
    }

// SUMMARY REPORT 2
    public void programmeSummaryReport() {
        courseManagementUI.displaySummaryReportTitle();
        if (programmeCourseList.isEmpty()) {
            System.out.println("\nThere is no any record yet!\n");
            courseManagementUI.endSummaryReport();
            return;
        }

        int maxTotalCreditHours = 0;
        int minTotalCreditHours = Integer.MAX_VALUE;
        int maxTotalCourses = 0; // Variable to track the maximum total courses
        int minTotalCourses = Integer.MAX_VALUE; // Variable to track the minimum total
courses
        ListInterface<String> programmesWithMaxTotalCredit = new ArrayList<>();
        ListInterface<String> programmesWithMinTotalCredit = new ArrayList<>();
        ListInterface<String> programmesWithMaxTotalCourses = new ArrayList<>(); // List
for programmes with the maximum total courses
        ListInterface<String> programmesWithMinTotalCourses = new ArrayList<>(); // List
for programmes with the minimum total courses

        int noOfProgrammesUnderFOCS = 0;
        int noOfProgrammesUnderFOET = 0;
        int noOfProgrammesUnderFAFB = 0;
        int noOfProgrammesUnderFOAS = 0;

        StringBuilder sb = new StringBuilder();
        for (Programme programme : programmeMap.values()) {
            int totalCreditHours = 0;
            int totalCourses = 0; // New variable to count total courses for each program
            for (int i = 1; i <= programmeCourseList.getNumberOfEntries(); i++) {
                if
(programmeCourseList.getEntry(i).getProgrammeID().equals(programme.getProgrammeId())
) {
                    totalCreditHours +=
courseMap.get(programmeCourseList.getEntry(i).getCourseID()).getCreditHours();
                    totalCourses++; // Increment the total courses count
                }
            }

            sb.append(programme.toString());
            sb.append("\t\t");
            sb.append(totalCreditHours);
            sb.append("\t\t");
            sb.append(totalCourses);
            sb.append("\n");

            // Check for max and min total credit hours
            if (totalCreditHours > maxTotalCreditHours) {
                maxTotalCreditHours = totalCreditHours;
                programmesWithMaxTotalCredit.clear(); // Clear the previous set
                programmesWithMaxTotalCredit.add(programme.getProgrammeId()); // Add the
new programme to the set
            } else if (totalCreditHours == maxTotalCreditHours) {
                programmesWithMaxTotalCredit.add(programme.getProgrammeId()); // Add the
programme to the set if it has the same maximum total credit hours
            }

            if (totalCreditHours < minTotalCreditHours) {
                minTotalCreditHours = totalCreditHours;
                programmesWithMinTotalCredit.clear(); // Clear the previous set
                programmesWithMinTotalCredit.add(programme.getProgrammeId()); // Add the
new programme to the set
            } else if (totalCreditHours == minTotalCreditHours) {
                programmesWithMinTotalCredit.add(programme.getProgrammeId()); // Add the
programme to the set if it has the same minimum total credit hours
            }

            // Check for max and min total courses
            if (totalCourses > maxTotalCourses) {
                maxTotalCourses = totalCourses;
                programmesWithMaxTotalCourses.clear(); // Clear the previous set
                programmesWithMaxTotalCourses.add(programme.getProgrammeId()); // Add the
new programme to the set
            } else if (totalCourses == maxTotalCourses) {
                programmesWithMaxTotalCourses.add(programme.getProgrammeId()); // Add the
programme to the set if it has the same maximum total courses
            }

            if (totalCourses < minTotalCourses) {
                minTotalCourses = totalCourses;
                programmesWithMinTotalCourses.clear(); // Clear the previous set
                programmesWithMinTotalCourses.add(programme.getProgrammeId()); // Add the
new programme to the set
            } else if (totalCourses == minTotalCourses) {
                programmesWithMinTotalCourses.add(programme.getProgrammeId()); // Add the
programme to the set if it has the same minimum total courses
            }

            for (Faculty faculty : facultyMap.values()) {
                if
(faculty.getFacultyProgrammesMap().containsKey(programme.getProgrammeId())) {
                    switch (faculty.getFacultyId()) {
                        case "FOCS" ->
                            noOfProgrammesUnderFOCS++;
                        case "FOET" ->
                            noOfProgrammesUnderFOET++;
                        case "FAFB" ->
                            noOfProgrammesUnderFAFB++;
                        case "FOAS" ->
                            noOfProgrammesUnderFOAS++;
                        default -> {
                        }
                    }

                }
            }
        }
        courseManagementUI.listProgrammesSummaryReport2(sb.toString());
        courseManagementUI.displayTotalNoOfProgrammes(programmeMap);
        courseManagementUI.summaryReport2Middle(noOfProgrammesUnderFOCS,
noOfProgrammesUnderFOET, noOfProgrammesUnderFAFB,
noOfProgrammesUnderFOAS);

        courseManagementUI.displayProgrammeWithHighestCreditHr(maxTotalCreditHours);
        for (String programmeId : programmesWithMaxTotalCredit) {
            Programme programme = programmeMap.get(programmeId);
            System.out.println("-  <" + programme.getProgrammeId() + "> " +
programme.getProgrammeName() + "\n");

        }
        courseManagementUI.displayLineSummaryReport();

        courseManagementUI.displayProgrammeWithLowestCreditHr(minTotalCreditHours);
        for (String programmeId : programmesWithMinTotalCredit) {
            Programme programme = programmeMap.get(programmeId);
            System.out.println("-  <" + programme.getProgrammeId() + "> " +
programme.getProgrammeName() + "\n");
        }
        courseManagementUI.displayLineSummaryReport();

        // Display programmes with the highest and lowest total courses
        courseManagementUI.displayProgrammeWithMostCourses(maxTotalCourses);
        for (String programmeId : programmesWithMaxTotalCourses) {
            Programme programme = programmeMap.get(programmeId);
            System.out.println("-  <" + programme.getProgrammeId() + "> " +
programme.getProgrammeName() + "\n");
        }

        courseManagementUI.displayLineSummaryReport();
        courseManagementUI.displayProgrammeWithLeastCourses(minTotalCourses);
        for (String programmeId : programmesWithMinTotalCourses) {
            Programme programme = programmeMap.get(programmeId);
            System.out.println("-  <" + programme.getProgrammeId() + "> " +
programme.getProgrammeName() + "\n");
        }

        courseManagementUI.displayLineSummaryReport();
        courseManagementUI.endSummaryReport();
    }

}
```

2. Screenshots



```
Student data initialized and saved to file.
Course data initialized and saved to file
Programme data initialized and saved to file.
Faculty data initialized and saved to file.

=====================================================================
     University Student Registration and Course Management Systems
=====================================================================
1. Student Registration Management
2. Course Management
0. Quit
Enter choice: 2
```

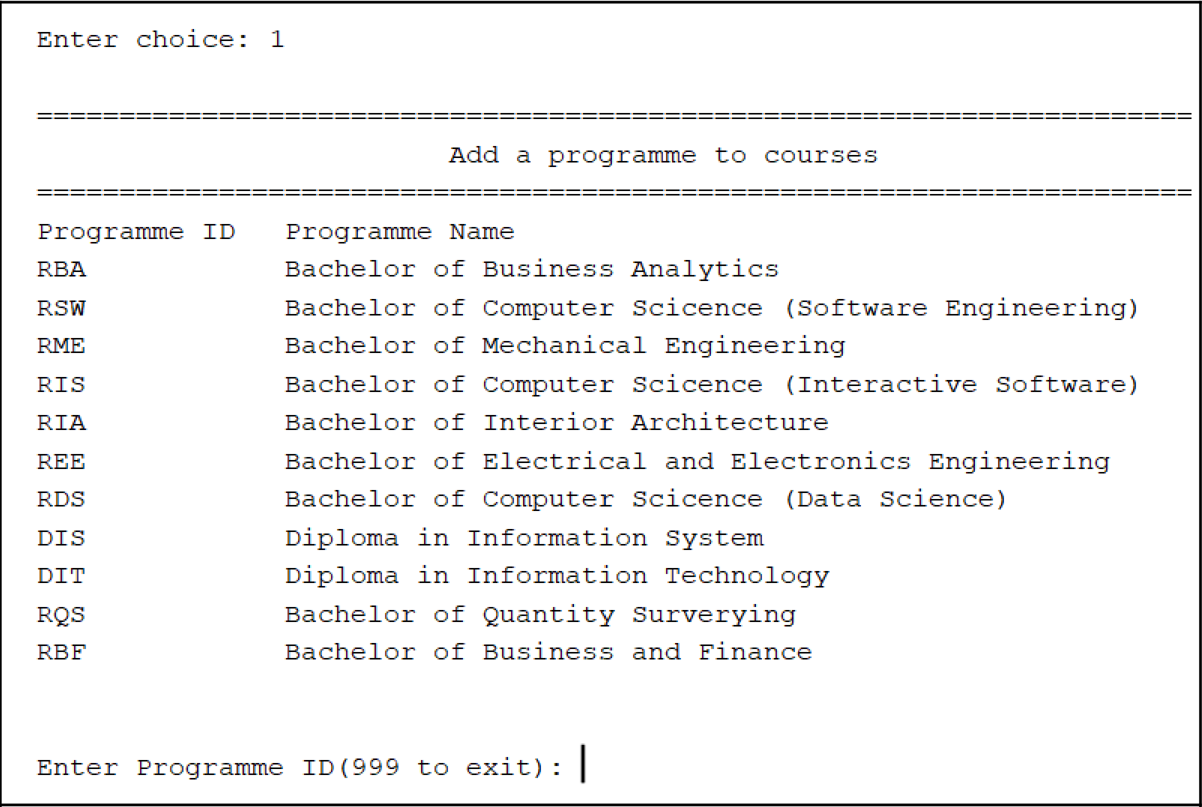
The main menu of our system. You can enter choices 1,2 or 0. Any invalid input will be prompted to reenter again.



```
--------------------------------------------------
|               Course Management Menu           |
|------------------------------------------------|
| 1.   Add a programme to courses                |
| 2.   Remove a programme from a course          |
| 3.   Add a new course to programmes            |
| 4.   Remove a course from a programme          |
| 5.   Search courses offered in a semester      |
| 6.   Amend course details for a programme      |
| 7.   List courses taken by different faculties |
| 8.   List all courses for a programme          |
| 9.   Courses Summary Report                    |
| 10.  Programme Summary Report                  |
| 0.   Exit                                      |
--------------------------------------------------
Enter choice:
```

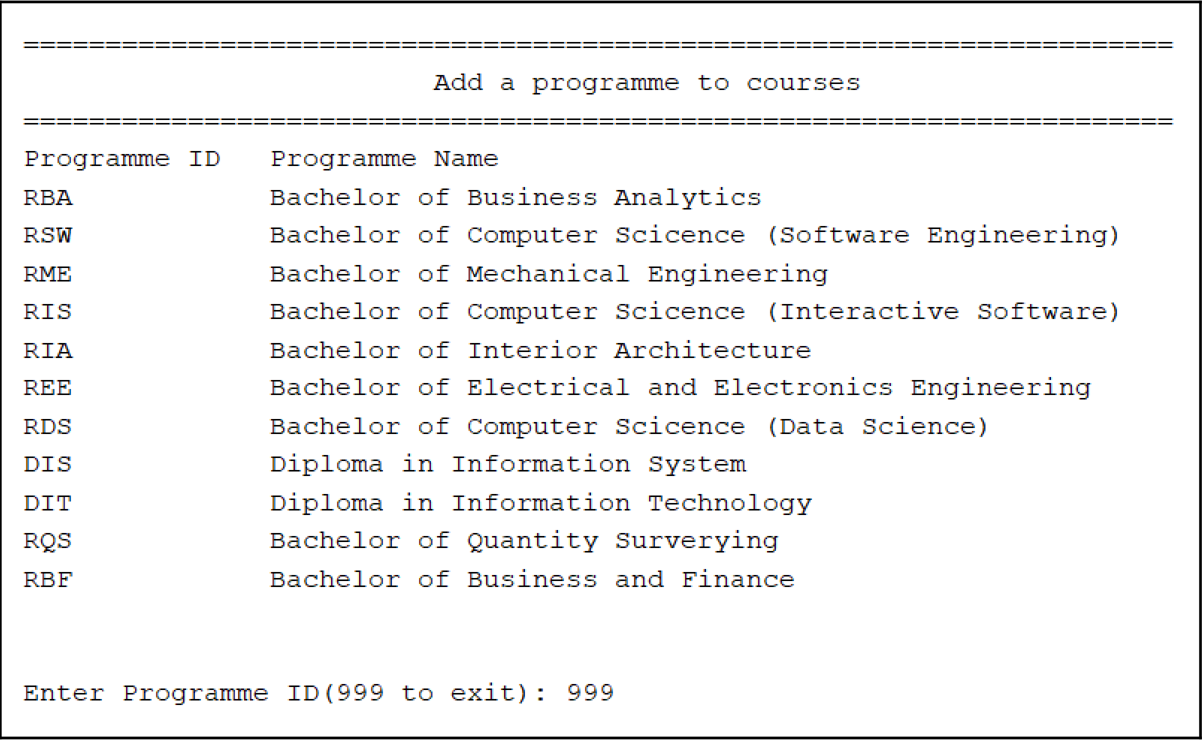
The menu for the course management subsystem.

A. Add a programme to courses (Limit total credit hours for each programme)



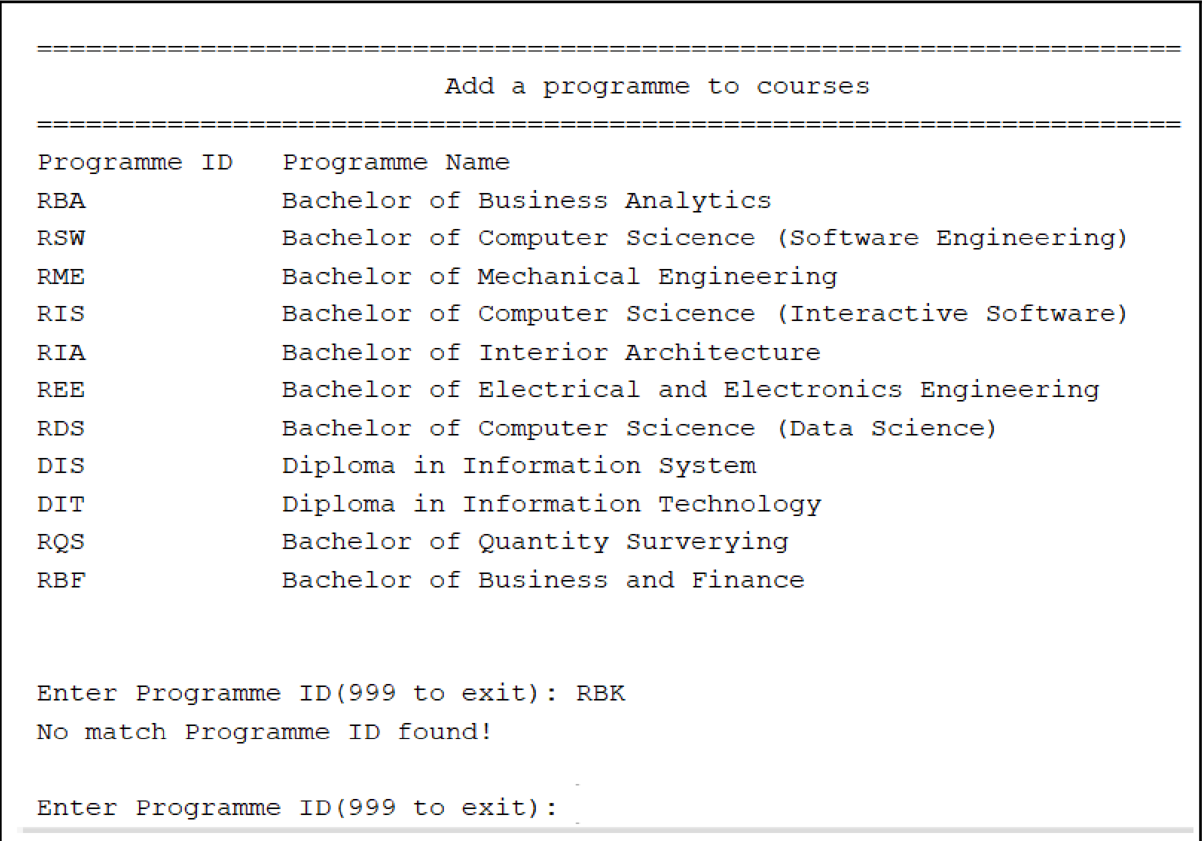
```
Enter choice: 1

======================================================================
                         Add a programme to courses
======================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Scicence (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance


Enter Programme ID(999 to exit):
```

If the user chooses 1, it will show this menu and ask the user to input the programme ID which he wants to add courses.



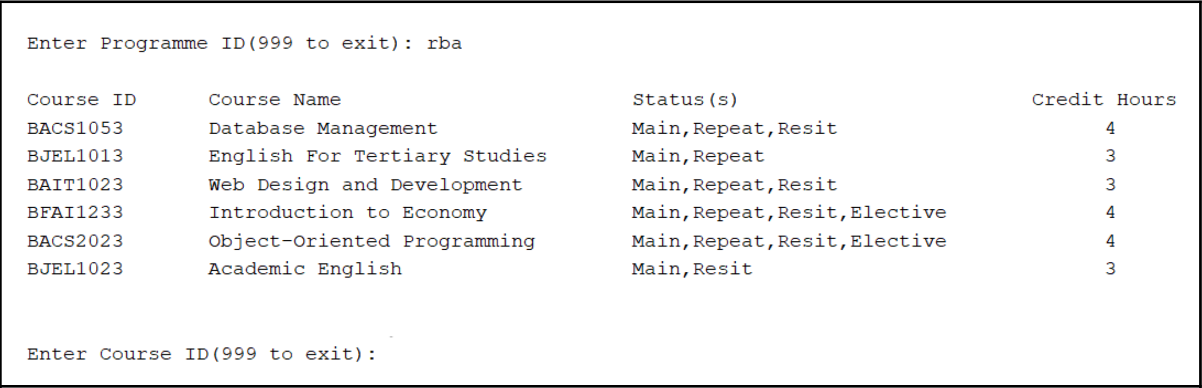
```
======================================================================
                         Add a programme to courses
======================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Scicence (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance


Enter Programme ID(999 to exit): 999
```

If the user accidentally press 1 and come to this function, he can exit this function by entering 999 and he will be back to course management subsystem menu.



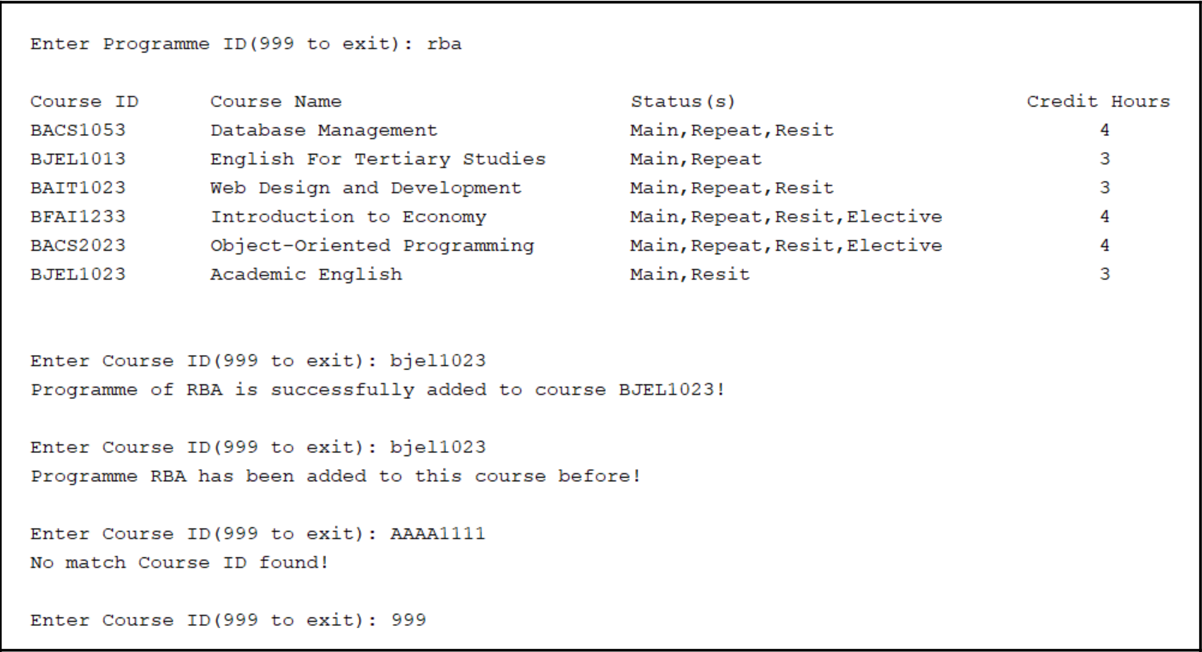
```
======================================================================
                         Add a programme to courses
======================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Science (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Science (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Science (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance


Enter Programme ID(999 to exit): RBK
No match Programme ID found!

Enter Programme ID(999 to exit):
```

If the input does not match the programme ID shown in the menu, it will show an error message and prompt for programme ID again.



```
Enter Programme ID(999 to exit): rba

Course ID      Course Name                        Status(s)                        Credit Hours
BACS1053       Database Management                Main,Repeat,Resit                     4
BJEL1013       English For Tertiary Studies       Main,Repeat                           3
BAIT1023       Web Design and Development         Main,Repeat,Resit                     3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective            4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective            4
BJEL1023       Academic English                   Main,Resit                            3


Enter Course ID(999 to exit):
```

The user can input the programme ID in either lower case or upper case. For example, the programme ID of "RBA" can be matched to input in lowercase of "rba". Then, all the available courses will be shown. It will prompt the user to input the course ID of the course that he wants to add to the selected programme.



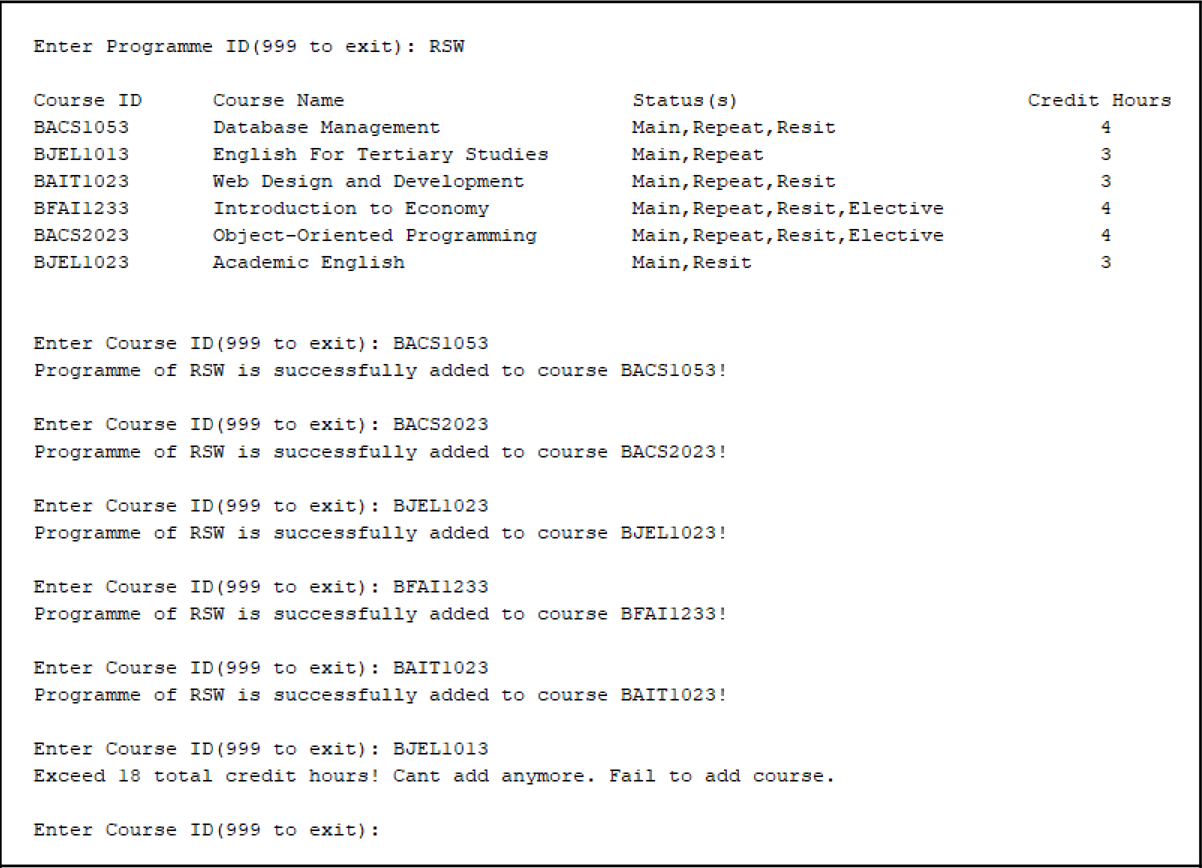
```
Enter Programme ID(999 to exit): rba

Course ID       Course Name                       Status(s)                          Credit Hours
BACS1053        Database Management               Main,Repeat,Resit                       4
BJEL1013        English For Tertiary Studies      Main,Repeat                             3
BAIT1023        Web Design and Development        Main,Repeat,Resit                       3
BFAI1233        Introduction to Economy           Main,Repeat,Resit,Elective              4
BACS2023        Object-Oriented Programming       Main,Repeat,Resit,Elective              4
BJEL1023        Academic English                  Main,Resit                              3


Enter Course ID(999 to exit): bjel1023
Programme of RBA is successfully added to course BJEL1023!

Enter Course ID(999 to exit): bjel1023
Programme RBA has been added to this course before!

Enter Course ID(999 to exit): AAAA1111
No match Course ID found!

Enter Course ID(999 to exit): 999
```

The user can input the course ID in either lowercase or uppercase as long as it matches the course ID shown. Then, a successful message will be shown. The system will prompt for course ID again since the assignment question is “ add a programme to courses” which means multiple courses can be added to the selected programme.

If the same course ID is input, it will show an error message indicating that the programme has been added to this course before.

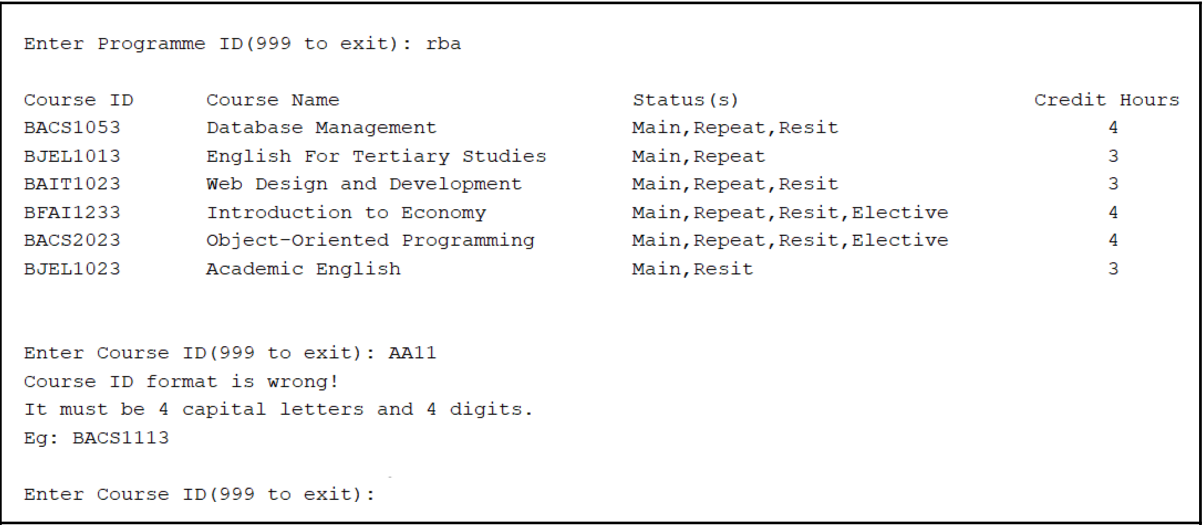
If the input does not match with the available courses shown. An error message will be printed and prompt for course ID again.

If the user wants to stop adding a programme to courses. He can enter 999 to exit the function. Then, he will be led to the course management subsystem menu.



```
Enter Programme ID(999 to exit): RSW

Course ID      Course Name                        Status(s)                          Credit Hours
BACS1053       Database Management                Main,Repeat,Resit                       4
BJEL1013       English For Tertiary Studies       Main,Repeat                             3
BAIT1023       Web Design and Development         Main,Repeat,Resit                       3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective              4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective              4
BJEL1023       Academic English                   Main,Resit                              3


Enter Course ID(999 to exit): BACS1053
Programme of RSW is successfully added to course BACS1053!

Enter Course ID(999 to exit): BACS2023
Programme of RSW is successfully added to course BACS2023!

Enter Course ID(999 to exit): BJEL1023
Programme of RSW is successfully added to course BJEL1023!

Enter Course ID(999 to exit): BFAI1233
Programme of RSW is successfully added to course BFAI1233!

Enter Course ID(999 to exit): BAIT1023
Programme of RSW is successfully added to course BAIT1023!

Enter Course ID(999 to exit): BJEL1013
Exceed 18 total credit hours! Cant add anymore. Fail to add course.

Enter Course ID(999 to exit):
```

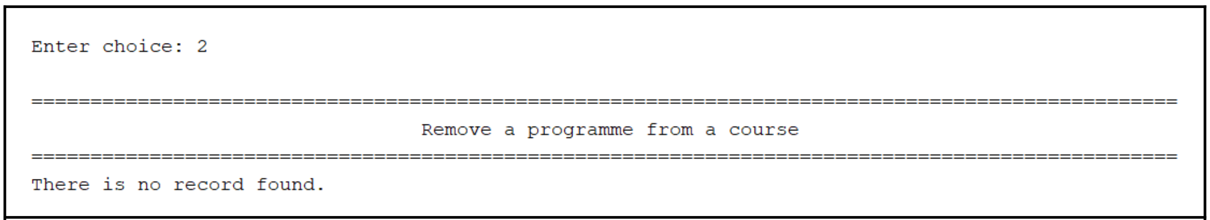
If the programme RSW ‘s total credit hours exceed 18 after adding to a course, an error message will be shown and the course will fail to be added.



```
Enter Programme ID(999 to exit): rba

Course ID      Course Name                        Status(s)                          Credit Hours
BACS1053       Database Management                Main,Repeat,Resit                       4
BJEL1013       English For Tertiary Studies       Main,Repeat                             3
BAIT1023       Web Design and Development         Main,Repeat,Resit                       3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective              4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective              4
BJEL1023       Academic English                   Main,Resit                              3


Enter Course ID(999 to exit): AA11
Course ID format is wrong!
It must be 4 capital letters and 4 digits.
Eg: BACS1113

Enter Course ID(999 to exit):
```

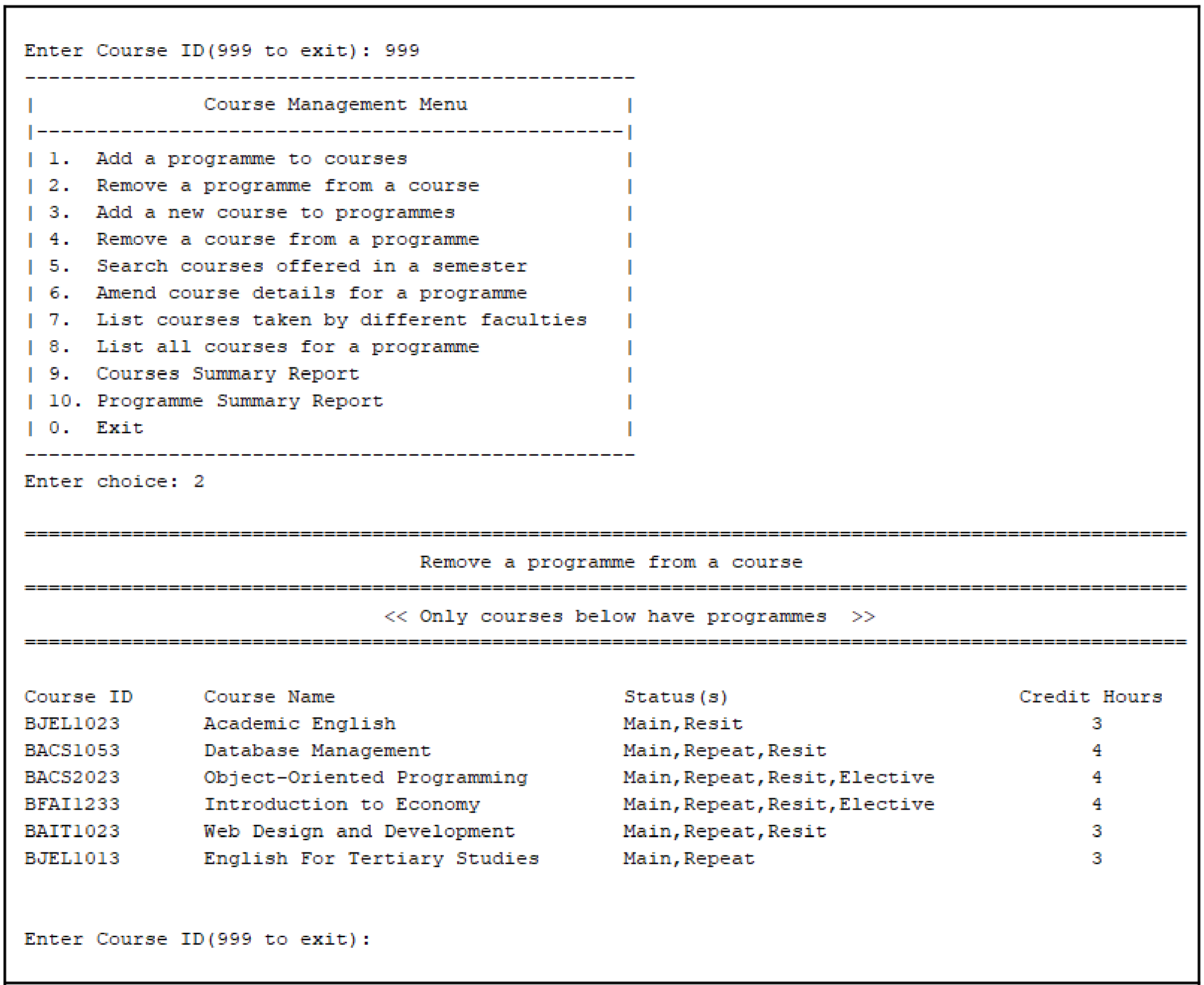
If the input course ID format is wrong, an error message will be shown and prompt again. The correct course ID is four capital letters followed by 4 digits.

B. Remove a programme from a course



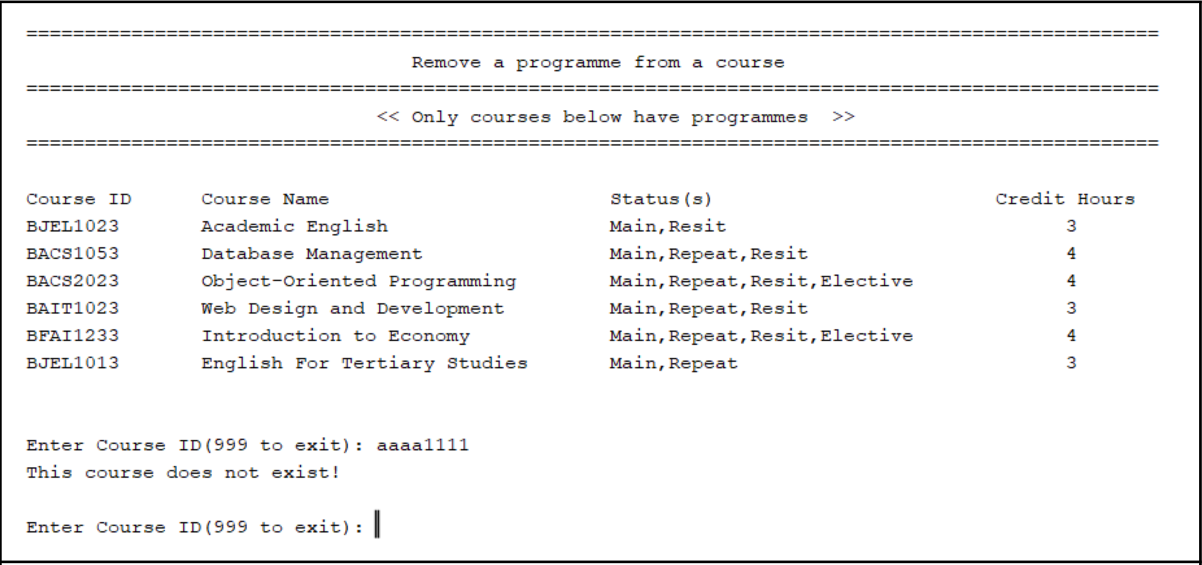
```
Enter choice: 2

================================================================================================
                                Remove a programme from a course
================================================================================================
There is no record found.
```

When there is no any course is taken by any programme, an error message is shown.



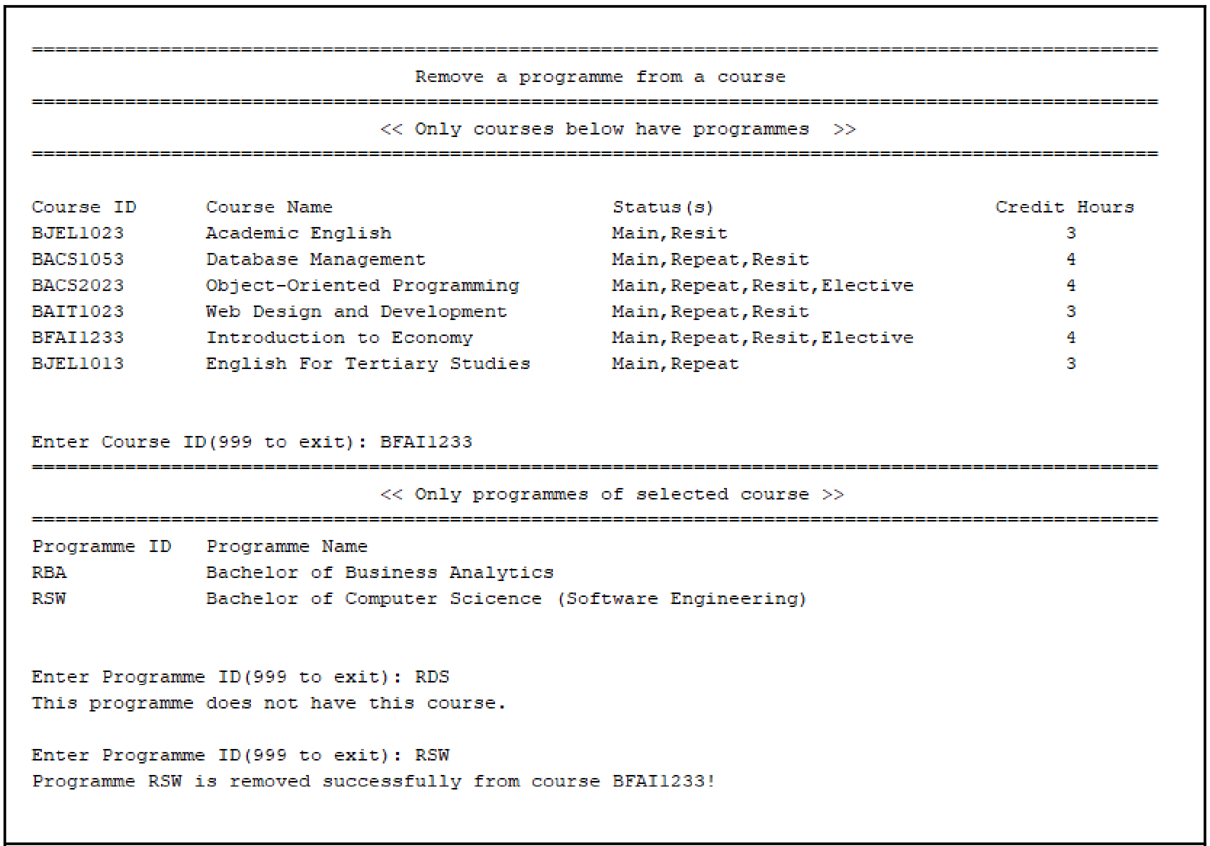
```
Enter Course ID(999 to exit): 999
---------------------------------------------------
|              Course Management Menu             |
|-------------------------------------------------|
| 1.  Add a programme to courses                  |
| 2.  Remove a programme from a course            |
| 3.  Add a new course to programmes              |
| 4.  Remove a course from a programme            |
| 5.  Search courses offered in a semester        |
| 6.  Amend course details for a programme        |
| 7.  List courses taken by different faculties   |
| 8.  List all courses for a programme            |
| 9.  Courses Summary Report                      |
| 10. Programme Summary Report                    |
| 0.  Exit                                        |
---------------------------------------------------
Enter choice: 2

=================================================================================================
                                 Remove a programme from a course
=================================================================================================
                              << Only courses below have programmes  >>
=================================================================================================

Course ID      Course Name                        Status(s)                        Credit Hours
BJEL1023       Academic English                   Main,Resit                             3
BACS1053       Database Management                Main,Repeat,Resit                      4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
BAIT1023       Web Design and Development         Main,Repeat,Resit                      3
BJEL1013       English For Tertiary Studies       Main,Repeat                            3


Enter Course ID(999 to exit):
```

When the user wants to remove a programme from a course, option 2 will be chosen. It will only show the courses that are added to any programme before. Any course that exists but have not been added to any programme will not be shown. Then, course ID is prompted. The user can only input course ID shown in the menu.



```
================================================================================================
                              Remove a programme from a course
================================================================================================
                           << Only courses below have programmes  >>
================================================================================================

Course ID      Course Name                        Status(s)                        Credit Hours
BJEL1023       Academic English                   Main,Resit                             3
BACS1053       Database Management                Main,Repeat,Resit                      4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BAIT1023       Web Design and Development         Main,Repeat,Resit                      3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
BJEL1013       English For Tertiary Studies       Main,Repeat                            3


Enter Course ID(999 to exit): aaaa1111
This course does not exist!

Enter Course ID(999 to exit):
```

If the input does not match with any course in the courseMap, an error message will be shown.

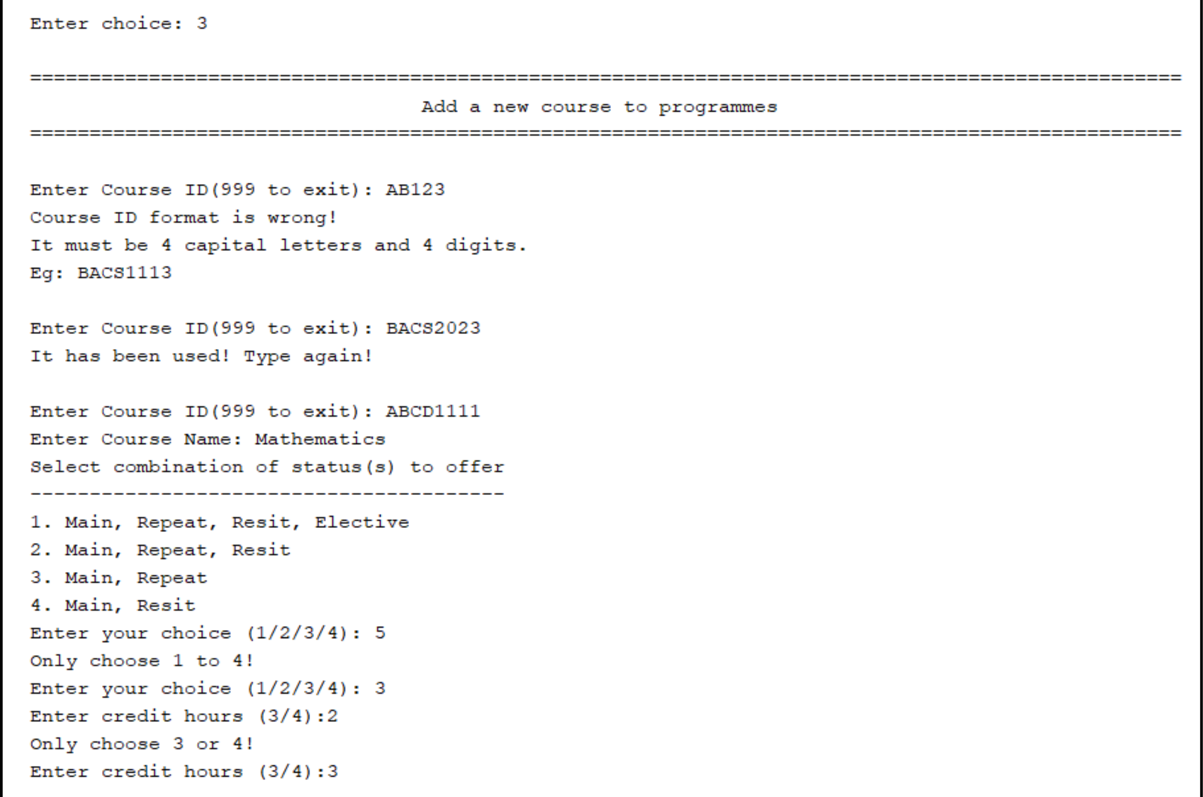


```
================================================================================================
                              Remove a programme from a course
================================================================================================
                           << Only courses below have programmes  >>
================================================================================================

Course ID      Course Name                        Status(s)                        Credit Hours
BJEL1023       Academic English                   Main,Resit                             3
BACS1053       Database Management                Main,Repeat,Resit                      4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BAIT1023       Web Design and Development         Main,Repeat,Resit                      3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
BJEL1013       English For Tertiary Studies       Main,Repeat                            3


Enter Course ID(999 to exit): BFAI1233
================================================================================================
                           << Only programmes of selected course >>
================================================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)


Enter Programme ID(999 to exit): RDS
This programme does not have this course.

Enter Programme ID(999 to exit): RSW
Programme RSW is removed successfully from course BFAI1233!
```

The user can input the course ID in either lowercase or uppercase. Then, it will only show the programmes that took the selected course. The programmes that do not take the course “BFAI1233” will not be shown. If the other programme ID that does not take the course “BFAI1233” is input, an error message will be shown.

If the correct programme ID is input, it will show a successful message. Then, it will exit the function and back to course management subsystem menu.

C. Add a new course to programmes (Limit total credit hours for each programme)



```
Enter choice: 3

===================================================================================================
                                 Add a new course to programmes
===================================================================================================

Enter Course ID(999 to exit): AB123
Course ID format is wrong!
It must be 4 capital letters and 4 digits.
Eg: BACS1113

Enter Course ID(999 to exit): BACS2023
It has been used! Type again!

Enter Course ID(999 to exit): ABCD1111
Enter Course Name: Mathematics
Select combination of status(s) to offer
----------------------------------------
1. Main, Repeat, Resit, Elective
2. Main, Repeat, Resit
3. Main, Repeat
4. Main, Resit
Enter your choice (1/2/3/4): 5
Only choose 1 to 4!
Enter your choice (1/2/3/4): 3
Enter credit hours (3/4):2
Only choose 3 or 4!
Enter credit hours (3/4):3
```

Option 3 is to add a new course to programmes. The user will be prompted to enter a course ID. If the input does not follow the correct format, an error message will be shown and ask the user to input again.

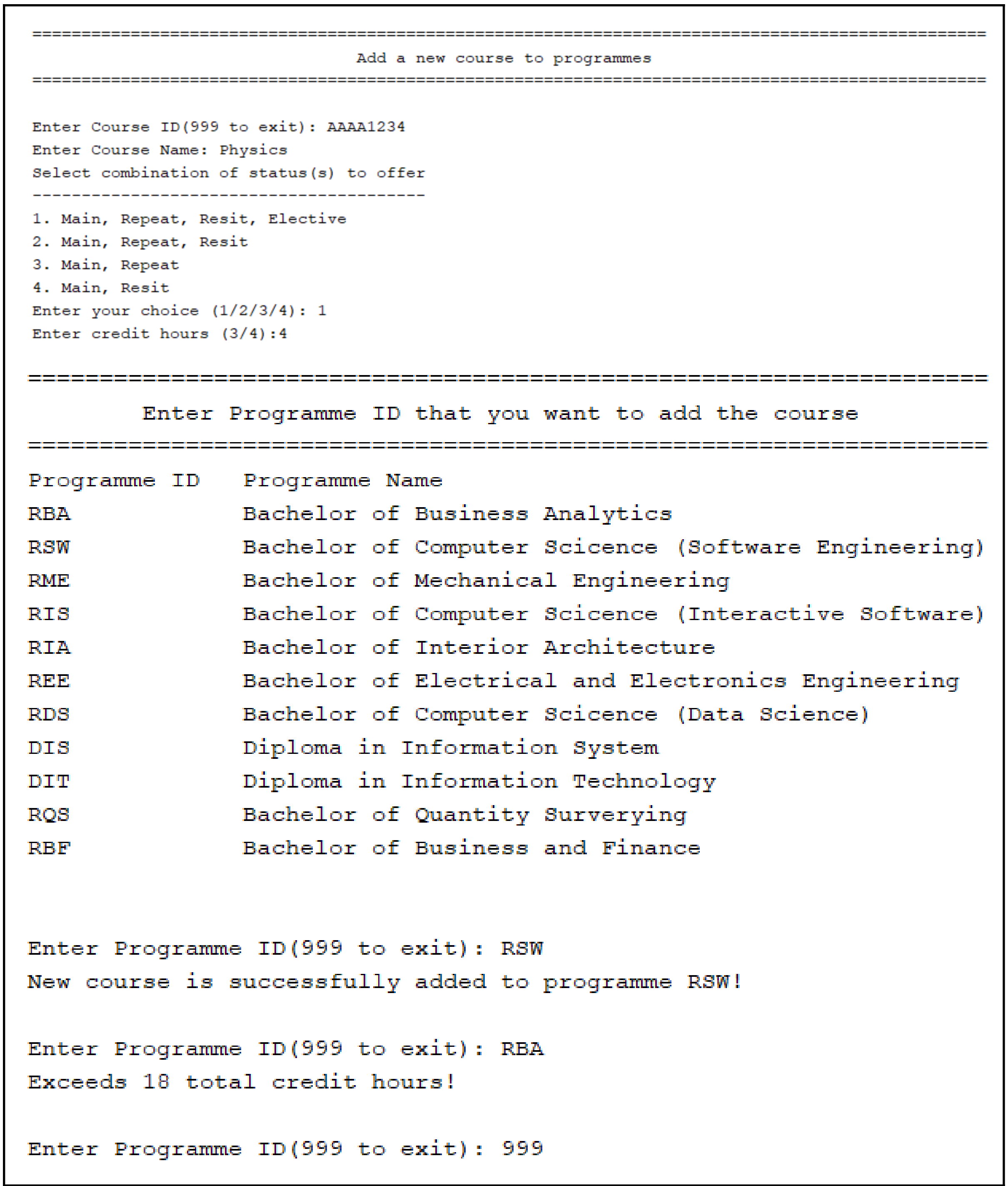
If the input is the same with the existing course's course ID, an error message will be shown.

After a correct course ID is input, it will prompt for course name, status to offer and credit hours of the new course,

If the choice for status to offer and credit hours is not in the valid range, an error message will be shown.

```
===================================================================
        Enter Programme ID that you want to add the course
===================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Scicence (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance


Enter Programme ID(999 to exit): RBA
New course is successfully added to programme RBA!

Enter Programme ID(999 to exit): RQS
New course is successfully added to programme RQS!

Enter Programme ID(999 to exit): OKK
No match Programme ID found!

Enter Programme ID(999 to exit): 999
```

Then, the newly created course can be added to multiple programmes. If the programme ID input is not matched with the existing programme's ID, an error message will be shown. The new course will only be created after adding to 1 programme.

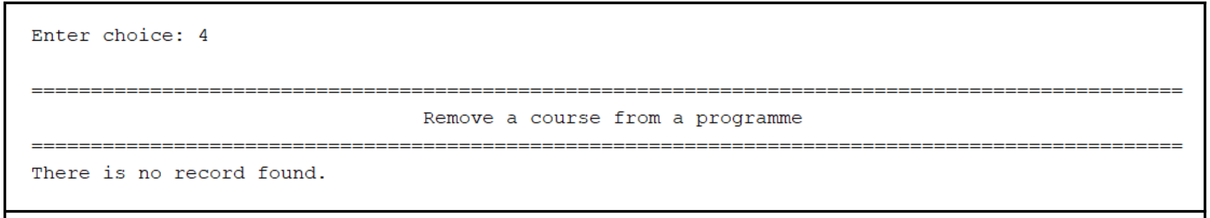


```
===========================================================================================
                                 Add a new course to programmes
===========================================================================================

Enter Course ID(999 to exit): AAAA1234
Enter Course Name: Physics
Select combination of status(s) to offer
----------------------------------------
1. Main, Repeat, Resit, Elective
2. Main, Repeat, Resit
3. Main, Repeat
4. Main, Resit
Enter your choice (1/2/3/4): 1
Enter credit hours (3/4):4

===================================================================
        Enter Programme ID that you want to add the course
===================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIS            Bachelor of Computer Scicence (Interactive Software)
RIA            Bachelor of Interior Architecture
REE            Bachelor of Electrical and Electronics Engineering
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying
RBF            Bachelor of Business and Finance


Enter Programme ID(999 to exit): RSW
New course is successfully added to programme RSW!

Enter Programme ID(999 to exit): RBA
Exceeds 18 total credit hours!

Enter Programme ID(999 to exit): 999
```

In this scenario, another new course is created. When the user adds the new course to the programme RBA, an error message will be shown because RBA's total credit hours will exceed 18 if the new course is added.

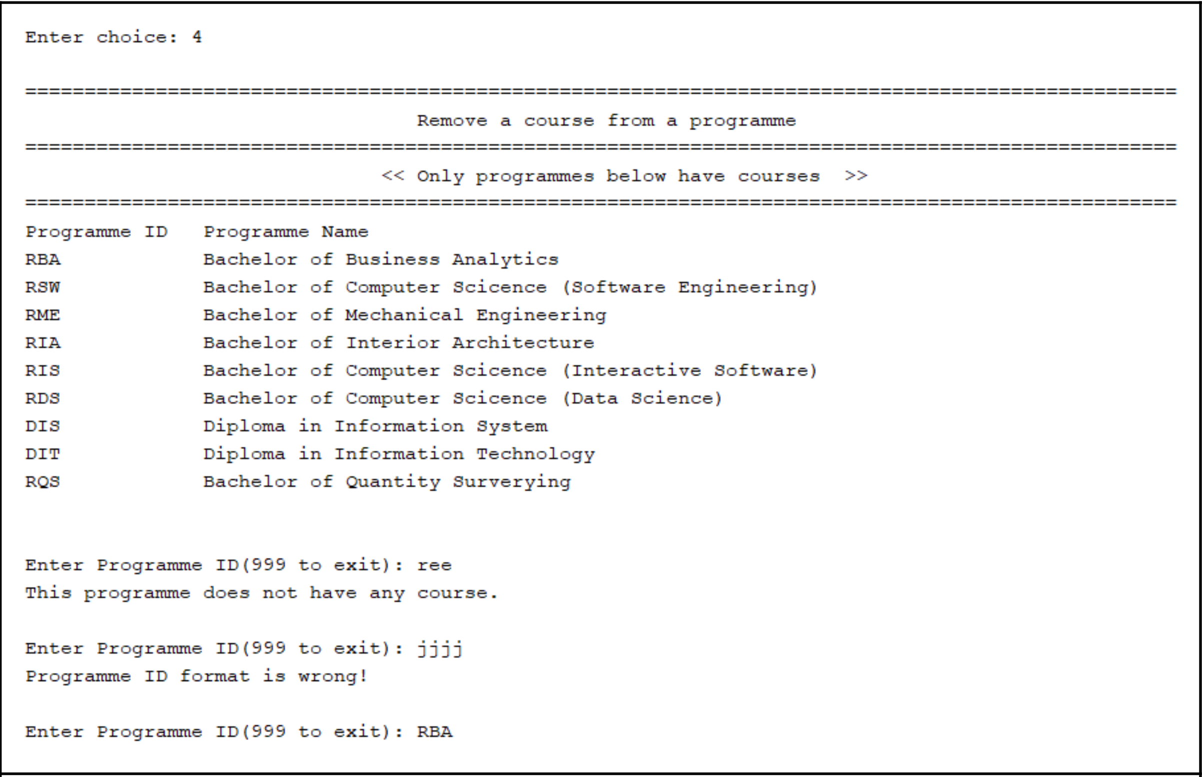
The user can enter 999 to exit this function.

D. Remove a course from a programme



```
Enter choice: 4

================================================================================================
                                 Remove a course from a programme
================================================================================================
There is no record found.
```

When there is no programme has taken any courses, an error message will be shown.

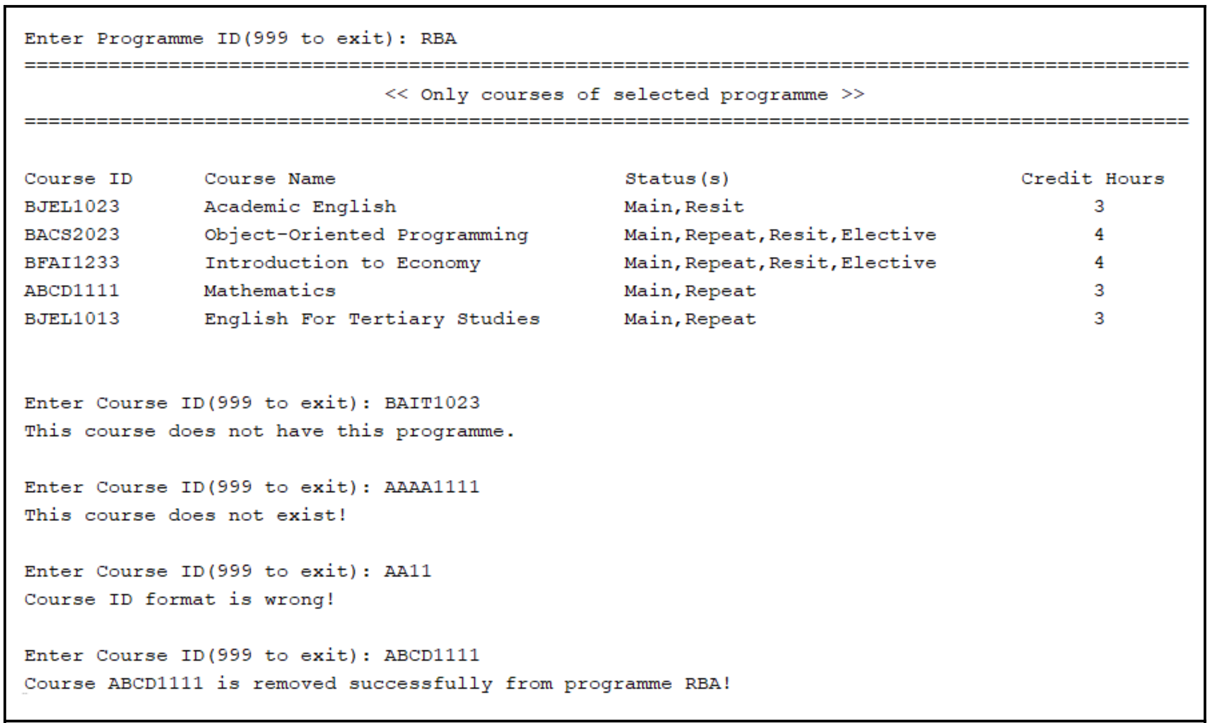


```
Enter choice: 4

================================================================================================
                                 Remove a course from a programme
================================================================================================
                              << Only programmes below have courses  >>
================================================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Science (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIA            Bachelor of Interior Architecture
RIS            Bachelor of Computer Science (Interactive Software)
RDS            Bachelor of Computer Science (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying


Enter Programme ID(999 to exit): ree
This programme does not have any course.

Enter Programme ID(999 to exit): jjjj
Programme ID format is wrong!

Enter Programme ID(999 to exit): RBA
```

Option 4 is to remove a course from a programme. It will only show the programmes that took any courses before. Any programme that exists but has not taken any course will not be shown. Then, the programme ID is prompted. The user can only input the programme ID shown in the menu, either in lowercase or uppercase.

If the input does not match with the programmes shown in the menu, an error message indicating “This programme does not have any courses.” will be displayed.

If the input does not follow the correct format, which is 3 letters, an error message indicating “ Programme ID format is wrong!” will be shown.



```
Enter Programme ID(999 to exit): RBA
=================================================================================================
                              << Only courses of selected programme >>
=================================================================================================

Course ID      Course Name                        Status(s)                         Credit Hours
BJEL1023       Academic English                   Main,Resit                             3
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
ABCD1111       Mathematics                        Main,Repeat                            3
BJEL1013       English For Tertiary Studies       Main,Repeat                            3


Enter Course ID(999 to exit): BAIT1023
This course does not have this programme.

Enter Course ID(999 to exit): AAAA1111
This course does not exist!

Enter Course ID(999 to exit): AA11
Course ID format is wrong!

Enter Course ID(999 to exit): ABCD1111
Course ABCD1111 is removed successfully from programme RBA!
```

After a programme ID is input correctly, it will only show the courses that have been taken by the programme.

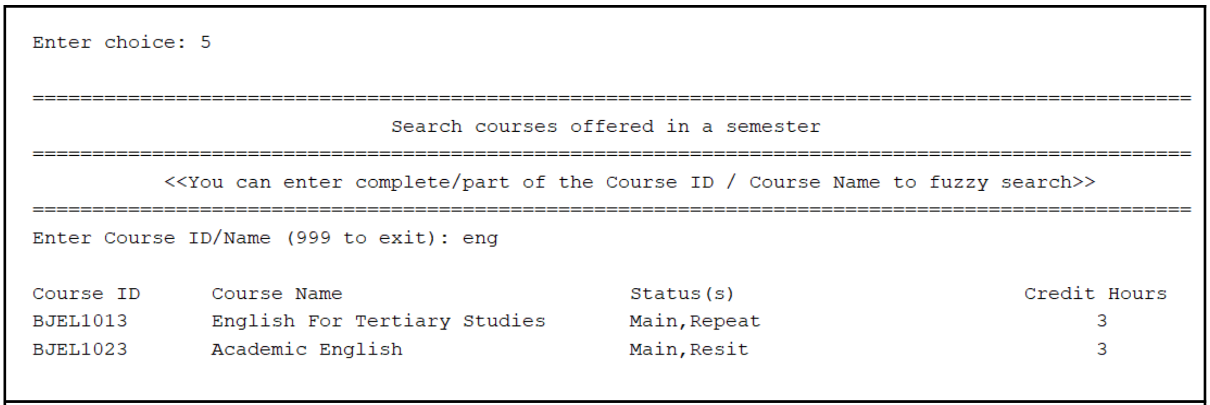
If the course of the course ID input exists but is not taken by the selected programme, an error message indicating that “ This course does not have this programme” will be shown .

If the course ID input does not match will any key in the courseMap, an error message indicating that “This course does not exist” will be shown.

If the course ID input does not follow the correct format, an error message indicating that “ Course ID format is wrong“ will be shown.

Once the correct course ID is input, a successful message will be shown..

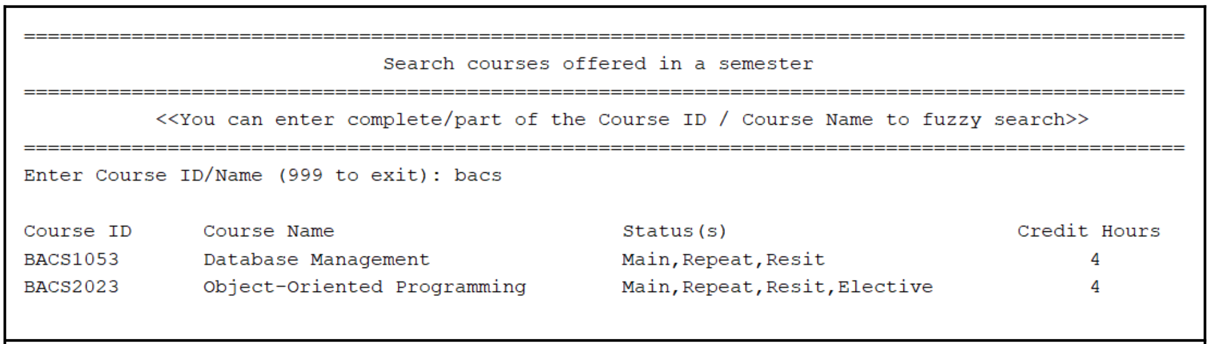
E. Search courses offered in a semester (Fuzzy Search)



```
Enter choice: 5

===============================================================================================
                              Search courses offered in a semester
===============================================================================================
           <<You can enter complete/part of the Course ID / Course Name to fuzzy search>>
===============================================================================================
Enter Course ID/Name (999 to exit): eng

Course ID      Course Name                        Status(s)                        Credit Hours
BJEL1013       English For Tertiary Studies       Main,Repeat                            3
BJEL1023       Academic English                   Main,Resit                             3
```

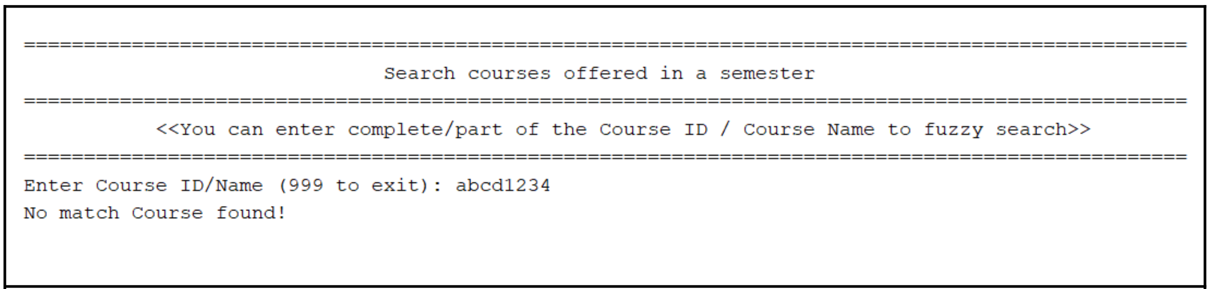
Option 5 is to search courses offered in a semester. I applied fuzzy search where the user can input either complete or part of the course ID or course name.

The example above shows input of “eng“ will display relevant courses with course name that have “eng”. I handled the user input and the course name in uppercase for easy comparison.



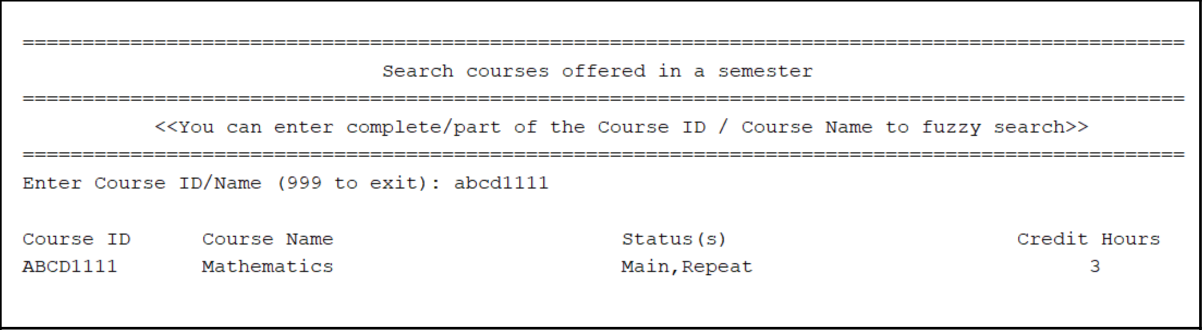
```
===============================================================================================
                              Search courses offered in a semester
===============================================================================================
           <<You can enter complete/part of the Course ID / Course Name to fuzzy search>>
===============================================================================================
Enter Course ID/Name (999 to exit): bacs

Course ID      Course Name                        Status(s)                        Credit Hours
BACS1053       Database Management                Main,Repeat,Resit                      4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
```

The example above shows input of “bacs“ will display relevant courses with course ID that have “bacs”.



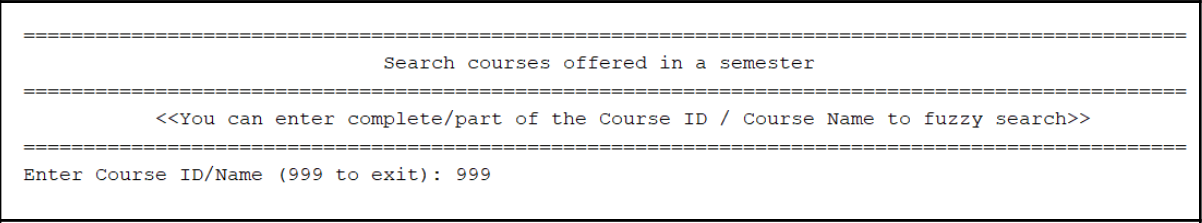
```
===============================================================================================
                              Search courses offered in a semester
===============================================================================================
           <<You can enter complete/part of the Course ID / Course Name to fuzzy search>>
===============================================================================================
Enter Course ID/Name (999 to exit): abcd1234
No match Course found!
```

If the input does not match with any course’s ID and course name, an error message will be displayed.



```
=================================================================================================
                              Search courses offered in a semester
=================================================================================================
           <<You can enter complete/part of the Course ID / Course Name to fuzzy search>>
=================================================================================================
Enter Course ID/Name (999 to exit): abcd1111

Course ID      Course Name                        Status(s)                        Credit Hours
ABCD1111       Mathematics                        Main,Repeat                            3
```

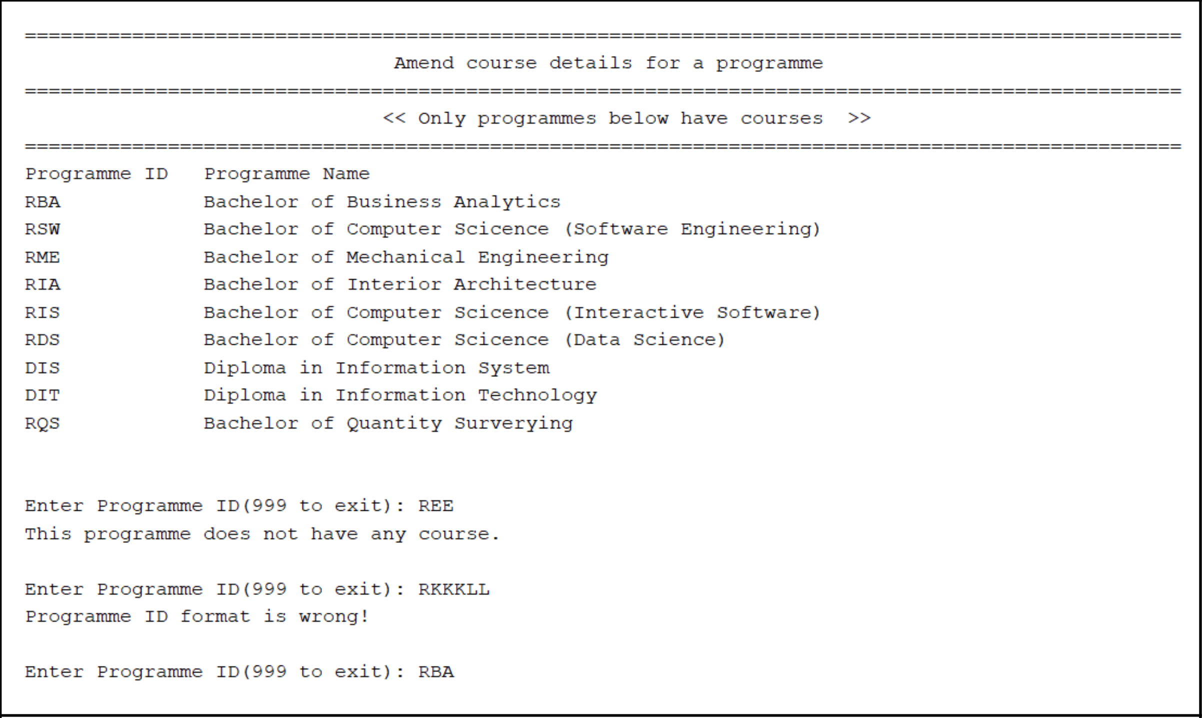
If the input matches will any course's course ID, the course's details will be displayed.



```
=================================================================================================
                              Search courses offered in a semester
=================================================================================================
           <<You can enter complete/part of the Course ID / Course Name to fuzzy search>>
=================================================================================================
Enter Course ID/Name (999 to exit): 999
```

The user can enter 999 to exit the function.

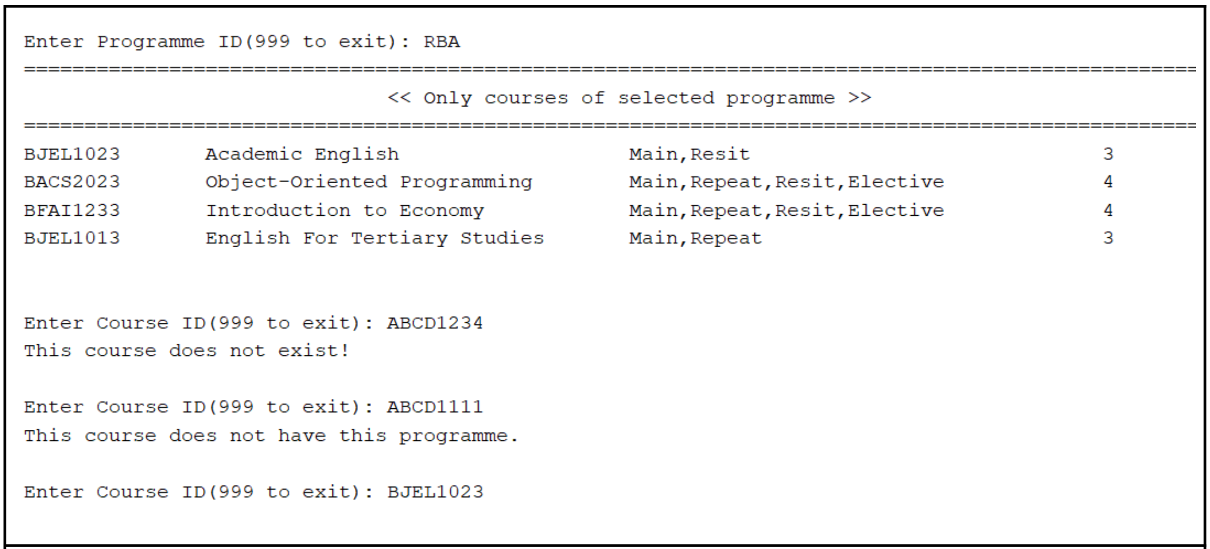
F. Amend course details for a programme



```
=================================================================================================
                              Amend course details for a programme
=================================================================================================
                             << Only programmes below have courses   >>
=================================================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIA            Bachelor of Interior Architecture
RIS            Bachelor of Computer Scicence (Interactive Software)
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
RQS            Bachelor of Quantity Surverying


Enter Programme ID(999 to exit): REE
This programme does not have any course.

Enter Programme ID(999 to exit): RKKKLL
Programme ID format is wrong!

Enter Programme ID(999 to exit): RBA
```

Option 6 is to amend course details for a programme. It will only display the programmes that have any courses. Those programmes without any courses will not be shown like "REE" is not shown. When "REE" is input, an error message indicating that "This programme does not have any course." will be shown.

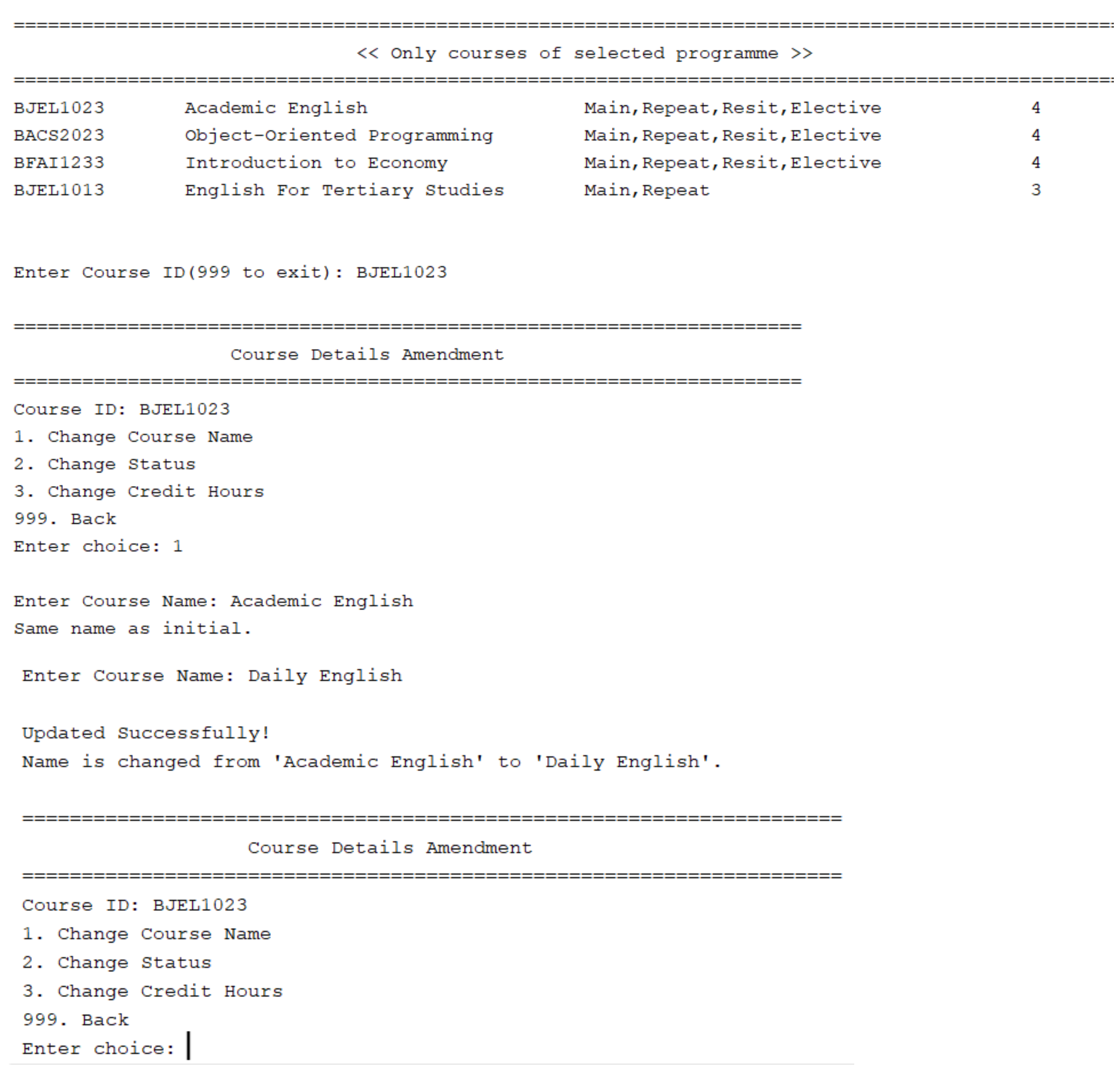
If the input does not follow correct programme ID format, an error message indicating "Programme ID format is wrong" will be shown.


```
Enter Programme ID(999 to exit): RBA
===============================================================================================
                              << Only courses of selected programme >>
===============================================================================================
BJEL1023       Academic English                   Main,Resit                            3
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective            4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective            4
BJEL1013       English For Tertiary Studies       Main,Repeat                           3


Enter Course ID(999 to exit): ABCD1234
This course does not exist!

Enter Course ID(999 to exit): ABCD1111
This course does not have this programme.

Enter Course ID(999 to exit): BJEL1023
```

Once the correct programme ID is input, it will only show the courses taken by the programme.

If a course ID with the correct format but does not match with any key in the courseMap, an error message indicating "This course does not exist!" will be shown.

If a valid course ID is input but it is not taken by the programme, an error message indicating "This course does not have this programme" will be shown.



```
=============================================================================================
                              << Only courses of selected programme >>
=============================================================================================
BJEL1023       Academic English                   Main,Repeat,Resit,Elective           4
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective           4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective           4
BJEL1013       English For Tertiary Studies       Main,Repeat                          3


Enter Course ID(999 to exit): BJEL1023

=====================================================================
                   Course Details Amendment
=====================================================================
Course ID: BJEL1023
1. Change Course Name
2. Change Status
3. Change Credit Hours
999. Back
Enter choice: 1

Enter Course Name: Academic English
Same name as initial.

 Enter Course Name: Daily English

 Updated Successfully!
 Name is changed from 'Academic English' to 'Daily English'.

 =====================================================================
                    Course Details Amendment
 =====================================================================
 Course ID: BJEL1023
 1. Change Course Name
 2. Change Status
 3. Change Credit Hours
 999. Back
 Enter choice: 
```

After a correct course ID is input, it will ask the user to choose the course details that he wants to amend. There are few amendments can be done such as changing course name, status and credit hours.

In the example above, if the new course name entered is the same as the previous name. An error message indicating “ Same name as initial.” will be shown and ask the user to enter again.

Then, a different name is input, and a successful message is shown.

Then, the amendment menu is prompted again.

```
Enter choice: 2

Select combination of status(s) to offer
----------------------------------------
1. Main, Repeat, Resit, Elective
2. Main, Repeat, Resit
3. Main, Repeat
4. Main, Resit
Enter your choice (1/2/3/4): 0
Only choose 1 to 4!

Enter your choice (1/2/3/4): 1
Same status as initial.

Select combination of status(s) to offer
----------------------------------------
1. Main, Repeat, Resit, Elective
2. Main, Repeat, Resit
3. Main, Repeat
4. Main, Resit
Enter your choice (1/2/3/4): 4

Updated Successfully!
Name is changed from 'Main,Repeat,Resit,Elective' to 'Main,Resit'.
```

When the user changes the status of the course, he should input valid choice within 1 to 4. Otherwise, an error message indicating "Only choose 1 to 4" will be displayed".

If the new status selected is the same as initial, an error message will be shown and prompt again.

Once it is successfully updated, a successful message will be shown.

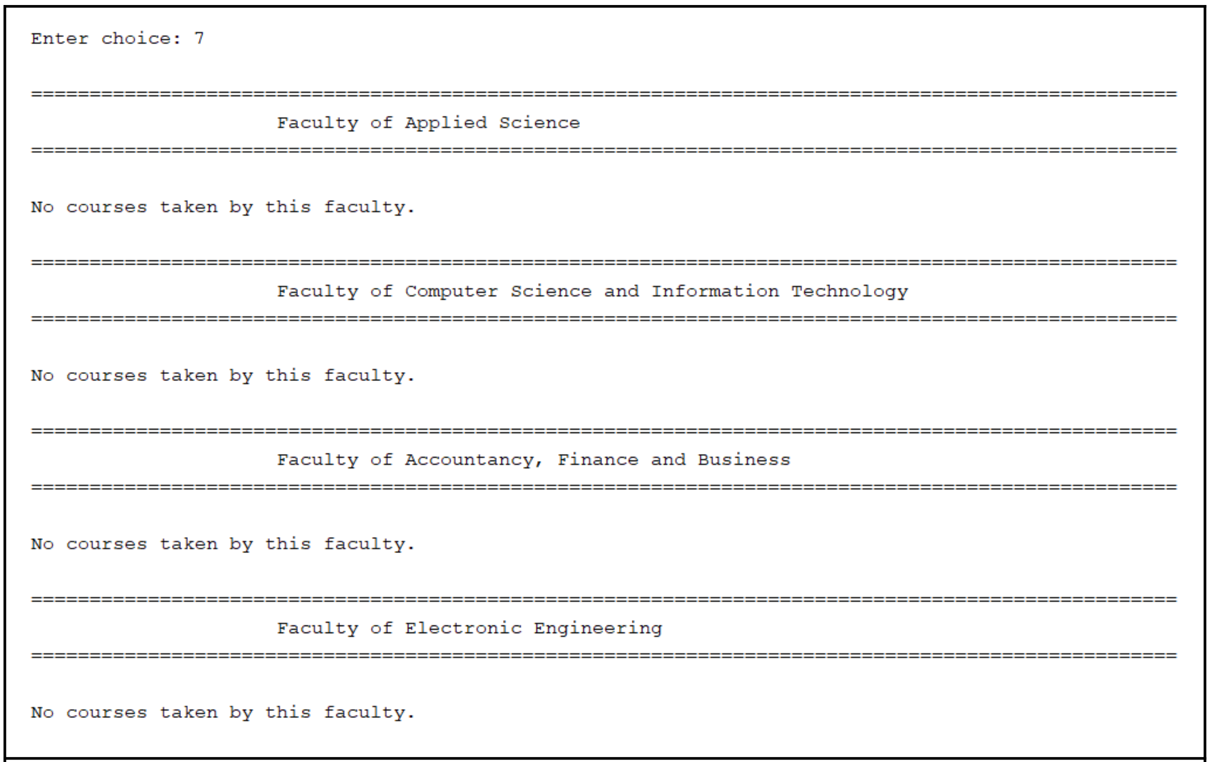
```
======================================================================
                   Course Details Amendment
======================================================================
Course ID: BJEL1023
1. Change Course Name
2. Change Status
3. Change Credit Hours
999. Back
Enter choice: 3

Enter credit hours (3/4):0
Only choose 3 or 4!

Enter credit hours (3/4):3
Same credit hours as initial.

Enter credit hours (3/4):4

Updated Successfully!
Credit hours is changed from '3' to '4'.
```

If the user choose to change credit hours of the course, it must be within valid range from 3 to 4. Otherwise, an error message will be shown.

Moreover, if the new credit hours is the same as the initial credit hours, an error message will be shown.

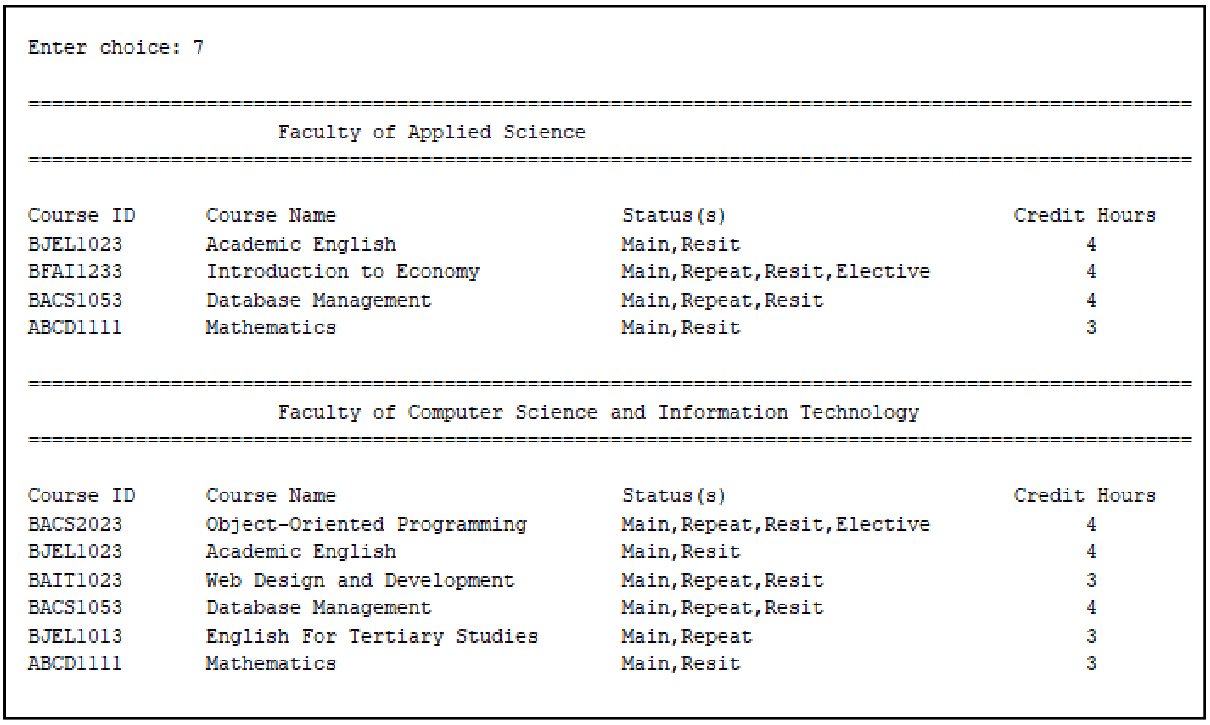
Once the credit hours are updated successfully, it will show a successful message.

G. List courses taken by different faculties

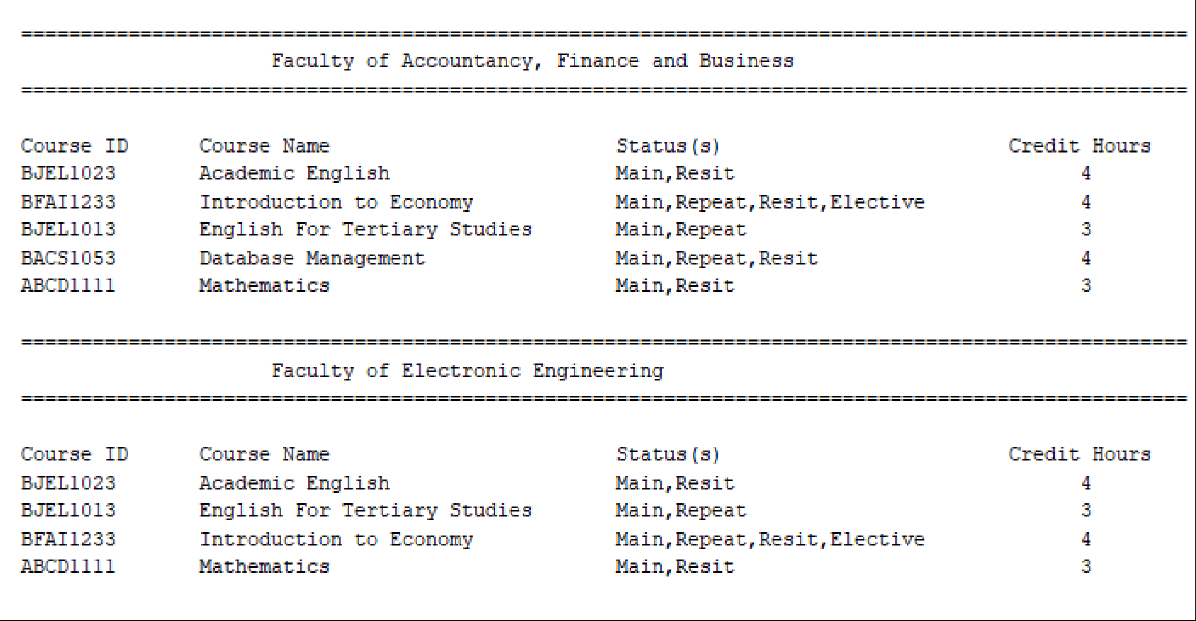


```
Enter choice: 7

==================================================================================================
                     Faculty of Applied Science
==================================================================================================

No courses taken by this faculty.

==================================================================================================
                     Faculty of Computer Science and Information Technology
==================================================================================================

No courses taken by this faculty.

==================================================================================================
                     Faculty of Accountancy, Finance and Business
==================================================================================================

No courses taken by this faculty.

==================================================================================================
                     Faculty of Electronic Engineering
==================================================================================================

No courses taken by this faculty.
```

When there is no record in the programmeCourseList, an error message will be shown.



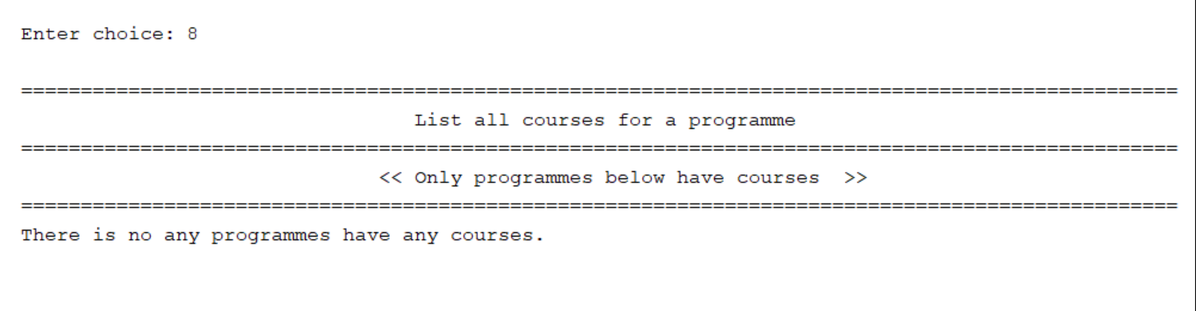
```
Enter choice: 7

==================================================================================================
                     Faculty of Applied Science
==================================================================================================

Course ID      Course Name                        Status(s)                        Credit Hours
BJEL1023       Academic English                   Main,Resit                             4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
BACS1053       Database Management                Main,Repeat,Resit                      4
ABCD1111       Mathematics                        Main,Resit                             3

==================================================================================================
                     Faculty of Computer Science and Information Technology
==================================================================================================

Course ID      Course Name                        Status(s)                        Credit Hours
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BJEL1023       Academic English                   Main,Resit                             4
BAIT1023       Web Design and Development         Main,Repeat,Resit                      3
BACS1053       Database Management                Main,Repeat,Resit                      4
BJEL1013       English For Tertiary Studies       Main,Repeat                            3
ABCD1111       Mathematics                        Main,Resit                             3
```



```
================================================================================================
                     Faculty of Accountancy, Finance and Business
================================================================================================

Course ID      Course Name                        Status(s)                         Credit Hours
BJEL1023       Academic English                   Main,Resit                             4
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
BJEL1013       English For Tertiary Studies       Main,Repeat                            3
BACS1053       Database Management                Main,Repeat,Resit                      4
ABCD1111       Mathematics                        Main,Resit                             3

================================================================================================
                     Faculty of Electronic Engineering
================================================================================================

Course ID      Course Name                        Status(s)                         Credit Hours
BJEL1023       Academic English                   Main,Resit                             4
BJEL1013       English For Tertiary Studies       Main,Repeat                            3
BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4
ABCD1111       Mathematics                        Main,Resit                             3
```

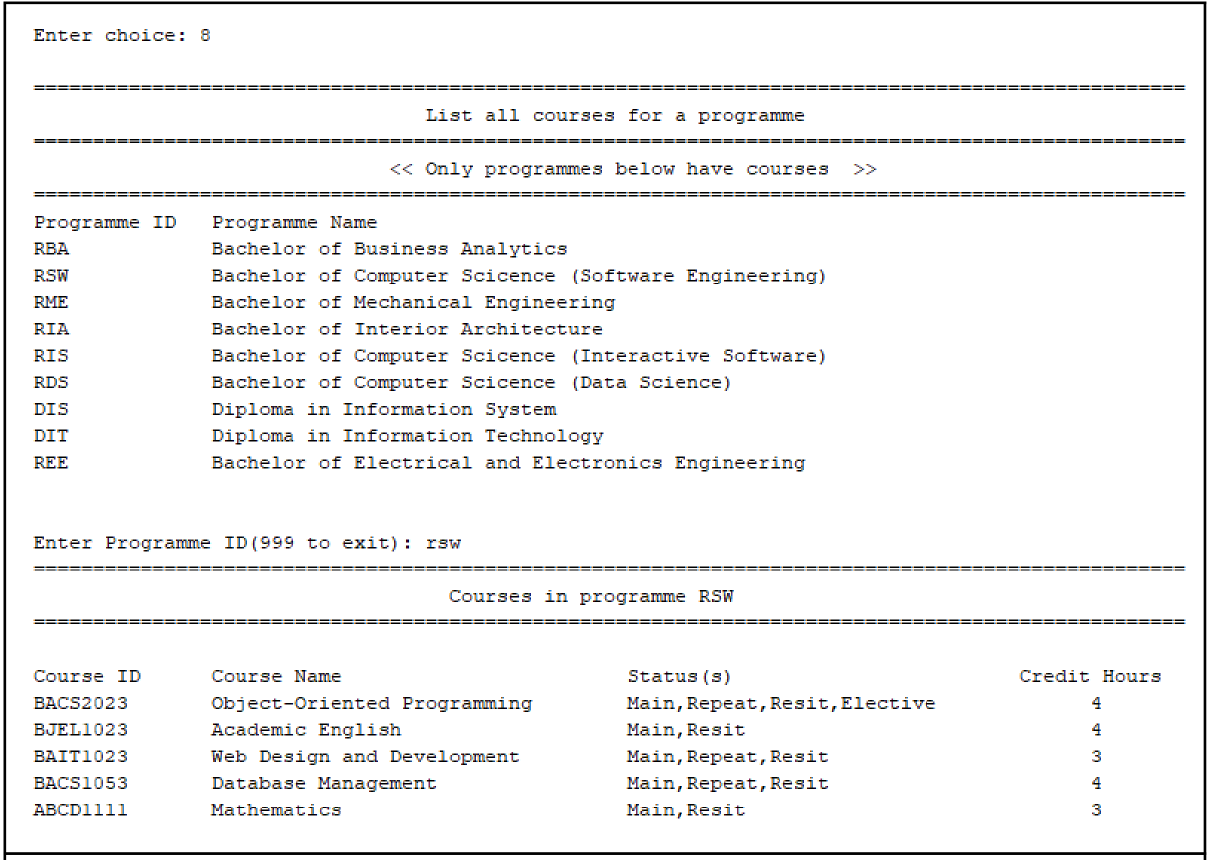
Option 7 is to list all courses taken by different faculties.

H. List all courses for a programme



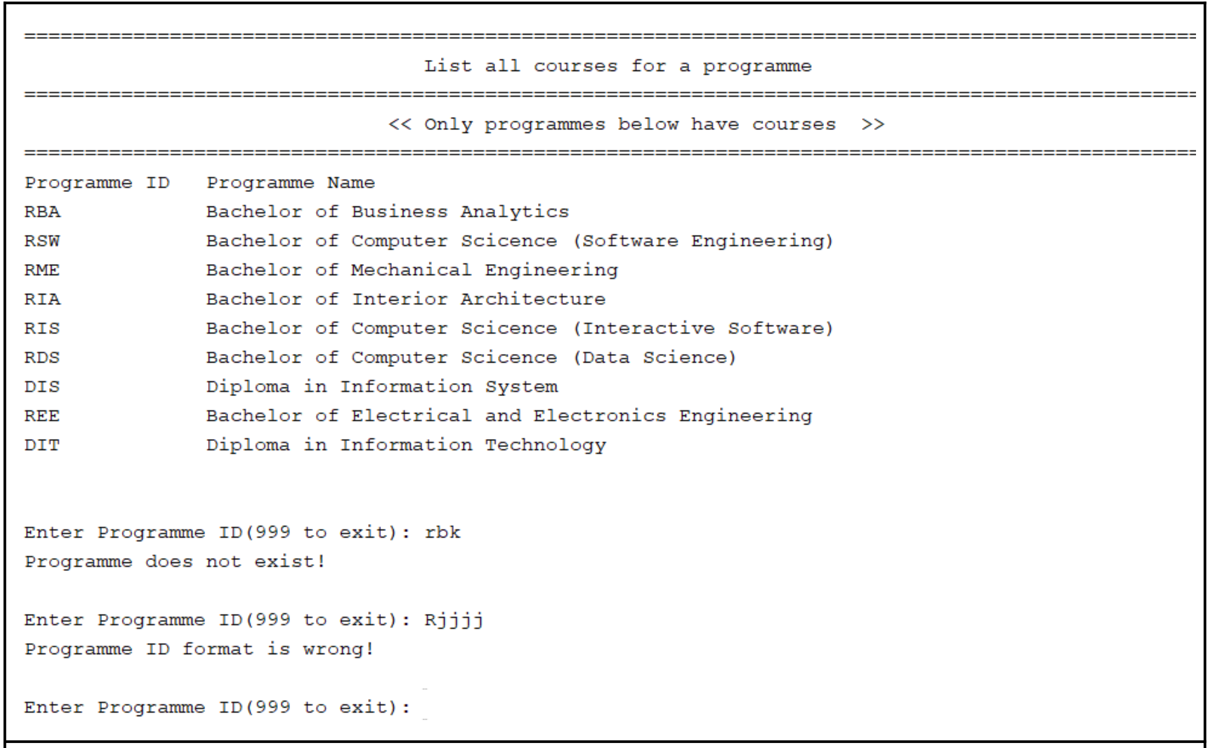
```
Enter choice: 8

================================================================================================
                                   List all courses for a programme
================================================================================================
                                << Only programmes below have courses  >>
================================================================================================
There is no any programmes have any courses.
```

When there is no any programmes take courses, an error message will be shown.



```
Enter choice: 8

================================================================================================
                                 List all courses for a programme
================================================================================================
                              << Only programmes below have courses  >>
================================================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIA            Bachelor of Interior Architecture
RIS            Bachelor of Computer Scicence (Interactive Software)
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
DIT            Diploma in Information Technology
REE            Bachelor of Electrical and Electronics Engineering


Enter Programme ID(999 to exit): rsw
================================================================================================
                                   Courses in programme RSW
================================================================================================

Course ID      Course Name                        Status(s)                         Credit Hours
BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4
BJEL1023       Academic English                   Main,Resit                             4
BAIT1023       Web Design and Development         Main,Repeat,Resit                      3
BACS1053       Database Management                Main,Repeat,Resit                      4
ABCD1111       Mathematics                        Main,Resit                             3
```

Option 8 is to list all courses for a programme. It will prompt user to input programme ID. Then all the courses taken by the programme will be displayed..

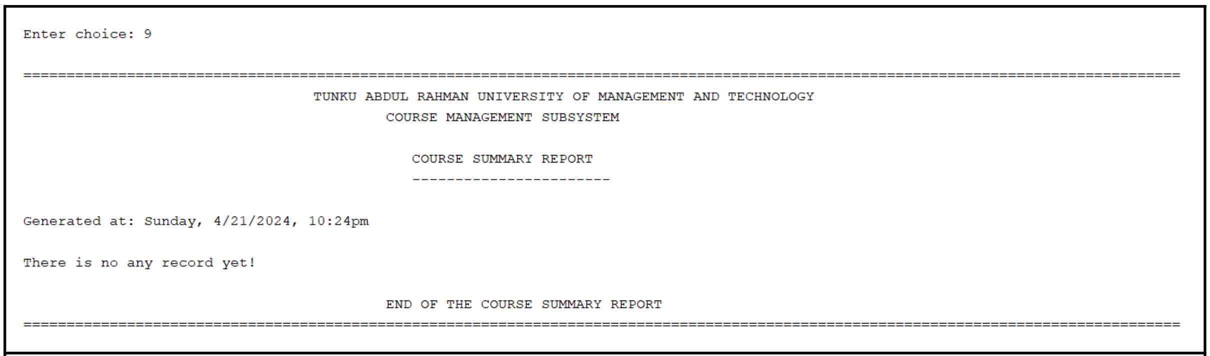


```
================================================================================================
                                 List all courses for a programme
================================================================================================
                              << Only programmes below have courses  >>
================================================================================================
Programme ID   Programme Name
RBA            Bachelor of Business Analytics
RSW            Bachelor of Computer Scicence (Software Engineering)
RME            Bachelor of Mechanical Engineering
RIA            Bachelor of Interior Architecture
RIS            Bachelor of Computer Scicence (Interactive Software)
RDS            Bachelor of Computer Scicence (Data Science)
DIS            Diploma in Information System
REE            Bachelor of Electrical and Electronics Engineering
DIT            Diploma in Information Technology


Enter Programme ID(999 to exit): rbk
Programme does not exist!

Enter Programme ID(999 to exit): Rjjjj
Programme ID format is wrong!

Enter Programme ID(999 to exit):
```

If the input does not match with any programme ID in the programmeMap, an error

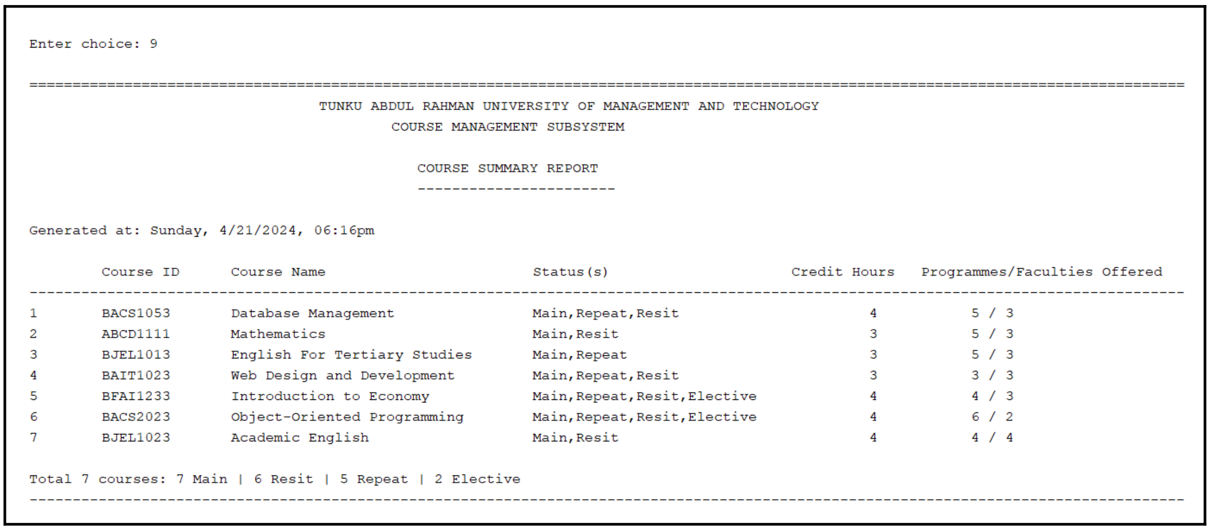
message will be shown and reprompt. If the input does not follow the correct format, an error message will be shown and reprompt.

I. Summary Report 1 (Course Summary Report)

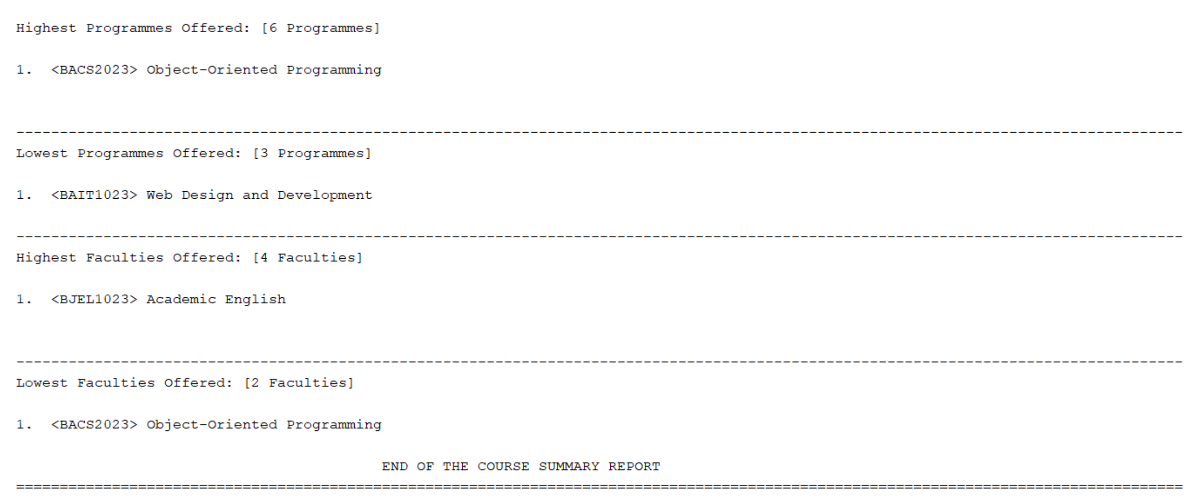


```
Enter choice: 9

======================================================================================================================================
                                  TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
                                          COURSE MANAGEMENT SUBSYSTEM

                                             COURSE SUMMARY REPORT
                                             -----------------------

Generated at: Sunday, 4/21/2024, 10:24pm

There is no any record yet!

                                          END OF THE COURSE SUMMARY REPORT
======================================================================================================================================
```

When there is no record in the programmeCourseList, an error message will be shown.



```
Enter choice: 9

======================================================================================================================================
                                  TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
                                          COURSE MANAGEMENT SUBSYSTEM

                                             COURSE SUMMARY REPORT
                                             -----------------------

Generated at: Sunday, 4/21/2024, 06:16pm

        Course ID      Course Name                        Status(s)                     Credit Hours   Programmes/Faculties Offered
--------------------------------------------------------------------------------------------------------------------------------------
1       BACS1053       Database Management                Main,Repeat,Resit                      4           5 / 3
2       ABCD1111       Mathematics                        Main,Resit                             3           5 / 3
3       BJEL1013       English For Tertiary Studies       Main,Repeat                            3           5 / 3
4       BAIT1023       Web Design and Development         Main,Repeat,Resit                      3           3 / 3
5       BFAI1233       Introduction to Economy            Main,Repeat,Resit,Elective             4           4 / 3
6       BACS2023       Object-Oriented Programming        Main,Repeat,Resit,Elective             4           6 / 2
7       BJEL1023       Academic English                   Main,Resit                             4           4 / 4

Total 7 courses: 7 Main | 6 Resit | 5 Repeat | 2 Elective
--------------------------------------------------------------------------------------------------------------------------------------
```

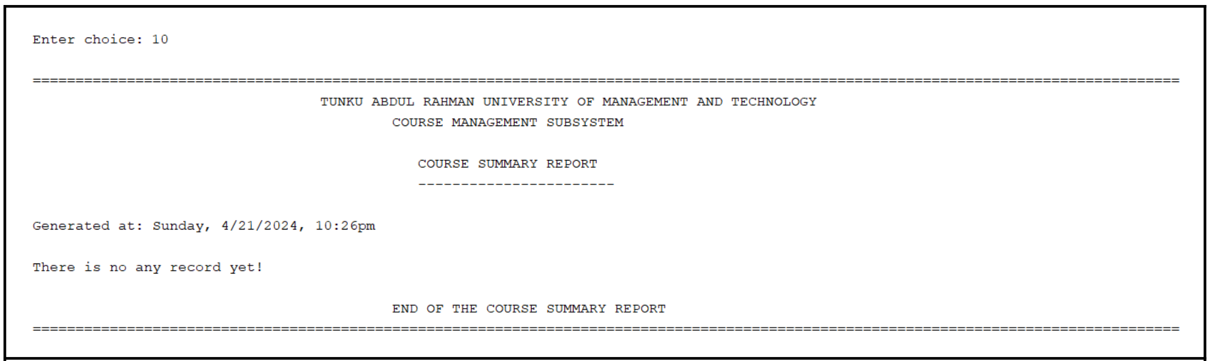


```
Highest Programmes Offered: [6 Programmes]

1.  <BACS2023> Object-Oriented Programming


-----------------------------------------------------------------------------------------------------------------------------
Lowest Programmes Offered: [3 Programmes]

1.  <BAIT1023> Web Design and Development

-----------------------------------------------------------------------------------------------------------------------------
Highest Faculties Offered: [4 Faculties]

1.  <BJEL1023> Academic English


-----------------------------------------------------------------------------------------------------------------------------
Lowest Faculties Offered: [2 Faculties]

1.  <BACS2023> Object-Oriented Programming

                                          END OF THE COURSE SUMMARY REPORT
=============================================================================================================================
```

Option 9 is a summary report about all the courses.

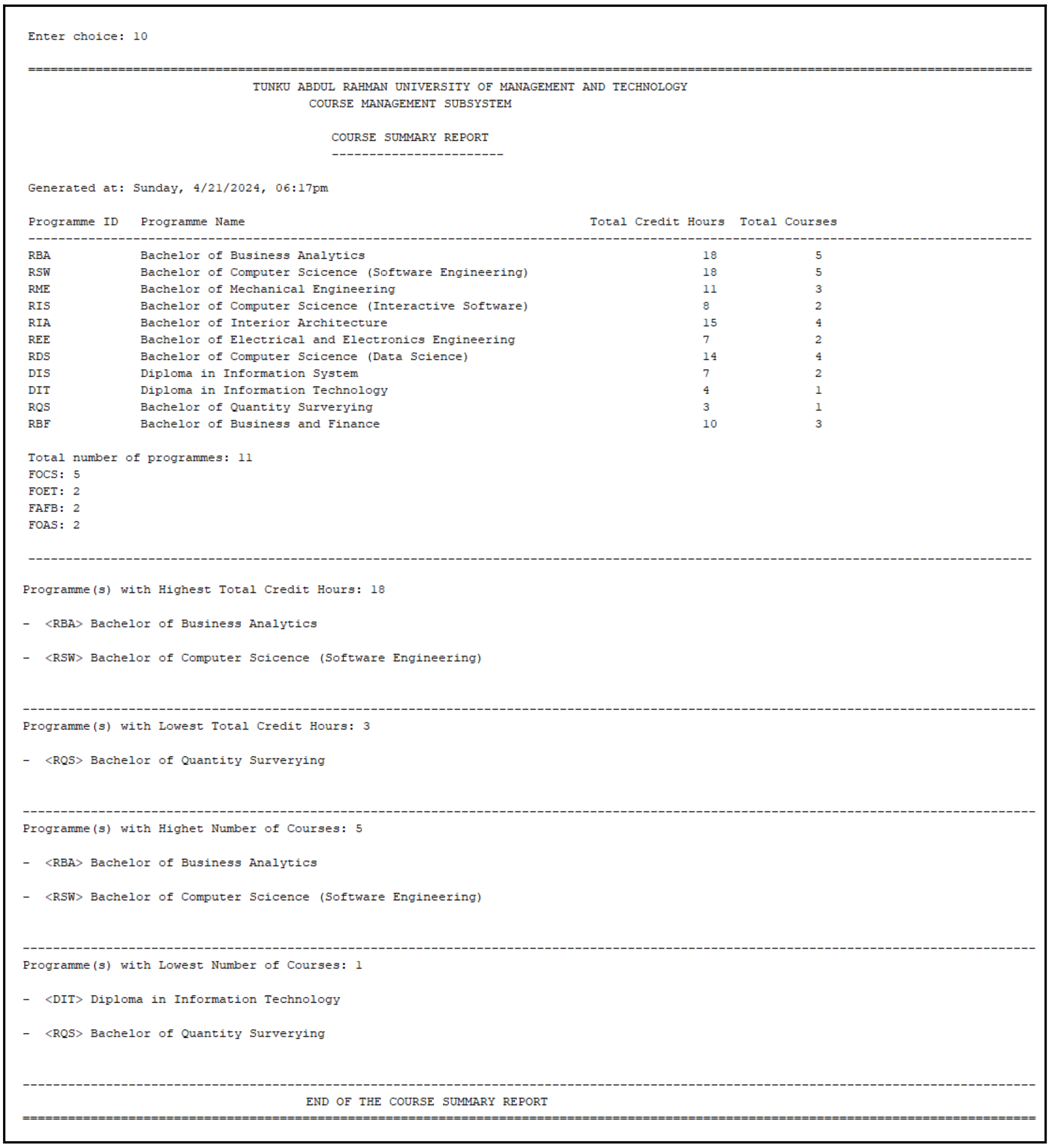
- The total number of programmes offered and the total number of faculties offered are calculated for each course.
- The total number of courses is calculated.
- The total number of courses that offer status “Main”, “Resit”, “Repeat” and “Elective” is calculated.
- The course with highest total number of programmes offered is shown.
- The course with lowest total number of programmes offered is shown.
- The course with highest total number of faculties offered is shown.
- The course with lowest total number of faculties offered is shown.

J. Summary Report 2 (Programme Summary Report)



```
Enter choice: 10

==========================================================================================================================
                              TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
                                      COURSE MANAGEMENT SUBSYSTEM

                                         COURSE SUMMARY REPORT
                                         ----------------------

Generated at: Sunday, 4/21/2024, 10:26pm

There is no any record yet!

                                      END OF THE COURSE SUMMARY REPORT
==========================================================================================================================
```

When there is no record in the programmeCourseList, an error message will be shown.



```
Enter choice: 10

==========================================================================================================================
                              TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
                                      COURSE MANAGEMENT SUBSYSTEM

                                         COURSE SUMMARY REPORT
                                         ----------------------

Generated at: Sunday, 4/21/2024, 06:17pm

Programme ID   Programme Name                                            Total Credit Hours  Total Courses
--------------------------------------------------------------------------------------------------------------------------
RBA            Bachelor of Business Analytics                                           18             5
RSW            Bachelor of Computer Scicence (Software Engineering)                     18             5
RME            Bachelor of Mechanical Engineering                                       11             3
RIS            Bachelor of Computer Scicence (Interactive Software)                     8              2
RIA            Bachelor of Interior Architecture                                        15             4
REE            Bachelor of Electrical and Electronics Engineering                       7              2
RDS            Bachelor of Computer Scicence (Data Science)                             14             4
DIS            Diploma in Information System                                            7              2
DIT            Diploma in Information Technology                                        4              1
RQS            Bachelor of Quantity Surverying                                          3              1
RBF            Bachelor of Business and Finance                                         10             3

Total number of programmes: 11
FOCS: 5
FOET: 2
FAFB: 2
FOAS: 2

--------------------------------------------------------------------------------------------------------------------------

Programme(s) with Highest Total Credit Hours: 18

-  <RBA> Bachelor of Business Analytics

-  <RSW> Bachelor of Computer Scicence (Software Engineering)


--------------------------------------------------------------------------------------------------------------------------
Programme(s) with Lowest Total Credit Hours: 3

-  <RQS> Bachelor of Quantity Surverying


--------------------------------------------------------------------------------------------------------------------------
Programme(s) with Highet Number of Courses: 5

-  <RBA> Bachelor of Business Analytics

-  <RSW> Bachelor of Computer Scicence (Software Engineering)


--------------------------------------------------------------------------------------------------------------------------
Programme(s) with Lowest Number of Courses: 1

-  <DIT> Diploma in Information Technology

-  <RQS> Bachelor of Quantity Surverying


--------------------------------------------------------------------------------------------------------------------------
                                      END OF THE COURSE SUMMARY REPORT
==========================================================================================================================
```

Option 10 is a summary report about all the programmes.

- The total credit hours of each programme is calculated.
- The total number of courses taken by each programme is calculated.
- The total number of programmes is calculated.
- The number of programmes for each faculty is calculated.
- The programmes with the highest total credit hours are shown.
- The programmes with the lowest total credit hours are shown.
- The programmes with the highest number of courses are shown.
- The programmes with the lowest number of courses are shown.