جامعة بيرزيت

BIRZEIT UNIVERSITY

Python Project #2
Student Course Manager

Student Names :

1- Abdallah Mohammad
2- Noor Aldeen Tirhi

Student ID:

1- 1190515
2- 1190081

Contents

Theory

The Program is meant to process files containing student information concerning college semester and the Electrical/Computer Engineering courses finished within them.

The basis that this program works with is as follows:
each student has a file with the name of the file being “STUDENT_ID”, all files are contained within a directory named “all”, the “all” directory is contained in the same directory as the source code, any change done within the program is written immediately on the concerned file, before any data processing, the files are read, this to make sure that the data produced is consistent.

We use a student class to keep track of all data concerning a particular student, and some global data, the student class also eases the data processing.

Data Structures

We used Tuple to hold all courses, since we don’t want this array to change at all.

We used dictionaries multiple times, we used it with students, this is since a student main Identification is the Student ID, we also used dictionaries for courses, same reason as student, we also used it for semesters, since we have no starting point, and we can easily identify each individual semester this way, also using a dictionary we won’t have two cells pointing to the same object.

We used lists for all remaining courses, we need a mutable list, that we can easily append new courses to.

Source Code Explaination

```
import re
import os
import matplotlib.pyplot as plt
```

These are the Libraries used in this program.
The re library is used mainly for checking user input, the os library was used for file management, and the matplotlib.pyplot library was used for plotting the histogram and graph.

```
def userInput(regex):
  answer = input("Enter the input: ")
  while (not re.match(regex, answer)):
    answer = input("Enter the input: ")
  return answer
```

As the name indicates, this function is used for user input, mainly for checking that checks the criteria for the input, for example when taking Student ID, the input needs to be a 7 digit number.

```
class Student:
```

Student class

```
Tuple = ("ENCS2340", "ENEE2307", "ENCS2110", "ENCS2380", "ENEE2304",
     "ENEE2312", "ENCS2380", "ENCS3130", "ENCS3310", "ENCS4110",
     "ENCS2360", "ENEE3309", "ENCS3320", "ENCS3330", "ENCS3340",
     "ENCS4370", "ENEE2103", "ENEE4113", "ENCS4130", "ENCS3210",
     "ENCS4310", "ENCS4320", "ENCS4380", "ENCS4330", "ENCS5140",
     "ENCS5200", "ENCS5150", "ENCS5300")
```

A Tuple containing all of the Computer/Electrical Engineering courses in BZU for Computer Engineering Students

`stdsAve = {}`	A dictionary to hold all student averages, the key used is student ID
`stdcount = 0`	A variable to keep count of all students
`SemAvgHours = {}`	A dictionary to hold all semester hours, the key used is the semester, eg: 2019-2020/3
`SemSumHours = {}`	A dictionary to hold average hours per semester, key used is the semester,
`SemTakers = {}`	A dictionary to hold number of students who enrolled in that semester, key is semester
`sems = []`	A list that holds all semesters

```
def __init__(self, id):
  self.id = str(id)
  self.stdSems = {}
  self.courses = {}
  self.semAve = {}
  self.semHours = {}
  self.takenHours = 0
  self.average = 0
  self.remainingCourses = []


  Student.stdcount = Student.stdcount + 1


  if (self.id in os.listdir("all")):
    self.readFromFile()
```

--stdSems, Dict of Dicts, first key is semseter, second key is course, holds all grades
--courses, Dict of grades for courses, key is course name
--semAve, Dict holding all semester averages, key is semester
--semHours, Dict holding all semester hours, key is semester
--takenHours, holds number of completed hours
--average, holds value of average
--remainingCourses, list of courses in the Tuple, but not yet completed

-- checks to see if student already has file, if so we can read information and give value to some of these variables

```
def readFromFile(self):
   file = open("all/" + str(self.id), 'r')
   file.readline()
   line = file.readline()
   while (line != '\n') and (line != ''):
      sem = (str.split(line, ';')[0])[:-1]
      self.addNewSem(sem)
      strings = str.split(str.split(line, ';')[1], ',')
      for s in strings:
         course = str.split(s, ' ')[1]
         grade = str.split(s, ' ')[2]
         self.addOrEditCourse(sem, course, grade)
      line = file.readline()
```

--This is the function used to read from the file of a particular student, it fills the stdSems dict of dicts, addNewSem and addOrEditCourse functions

```
def addNewSem(self, sem):
   self.stdSems[sem] = {}
```

--This function initialized dict (sem) of the stdSems dict of dicts

```
def addOrEditCourse(self, sem, course, grade):
   self.stdSems[sem][course] = int(grade)
```

--This function fills out a cell in the stdSems dict of dicts

```
def avePerSem(self, sem):
   sum = 0
   for course in self.stdSems[sem].keys():
      sum += self.stdSems[sem][course]
   print(sum / len(self.stdSems[sem]))
```

--This function produces the average of a particular semester for a particular student.

```
def overallAve(self):
   sum = 0
   courseNum = 0
   for sem in self.stdSems.keys():
      for course in self.stdSems[sem].keys():
         sum += self.stdSems[sem][course]
         courseNum += 1
   print(sum / courseNum)
```

--This function calculates the Average for a particular student

```
def writToFile(self):
   record = open("all/" + self.id, "w")
   record.write("Year\Semester ; Courses with grades\n")
   for sem in self.stdSems.keys():
      line = sem + " ;"
      for course in self.stdSems[sem].keys():
         line += " " + course + " " + str(self.stdSems[sem][course]) + ","
      line = line[:-1]
      record.write(line + '\n')
   record.close()
```

--This function overwrites a record with the current data for a student, this updates a record to the correct data.

```
def isRecordedSemester(self, sem):
   return sem in self.stdSems
```

--This function checks if sem is recorded, meaning it’s contained in the stdSem dictionary

```python
def isInSemester(self, sem, course):
    return course in self.stdSems[sem]
```

--This function checks a particular course is within a particular semister

```python
def setStdInfo(self):
    self.takenHours = 0
    gradevalue = 0
    totalGrades = 0
    for sem in self.stdSems:
        sumhours = 0
        gradevalue = 0
        for course in self.stdSems[sem]:
            self.courses[course] = self.stdSems[sem][course]
            sumhours += int(course[5])
            gradevalue += int(course[5]) * self.stdSems[sem][course]


        self.semHours[sem] = sumhours
        self.semAve[sem] = gradevalue / sumhours


        if not (sem in Student.SemTakers):
            Student.SemTakers[sem] = 0
        Student.SemTakers[sem] += 1
        if not (sem in Student.SemSumHours):
            Student.SemSumHours[sem] = 0
        Student.SemSumHours[sem] += sumhours
        totalGrades += gradevalue
        self.takenHours += sumhours
    self.average = totalGrades / self.takenHours
```

--This function is used to fill out many variable initialized in the constructor, it fills out the takenHours,average,semHours dict, courses dict and the semAve dict.
This is done by going through the stdSems dict of dicts and the processing the data from there.

```python
def calcAllSemAvgHours():  # this calculates the average hours per each semester
    for sem in Student.SemTakers:
        Student.SemAvgHours[sem] = Student.SemSumHours[sem] / Student.SemTakers[sem]
```

--This function is used to calculate the avg hours of all semester , by using the two class dictionaries, SemTakers and SemSumHours.

```python
def setRemainingCourses(self):
    for course in Student.Tuple:
        if not (course in self.courses):
            self.remainingCourses.append(course)
```

--This function fills out the remainingCourses list, by seeing which course is in the Tuple tuple and isn’t in the courses list and adding them to the remainingCourses list

```python
def getID():
    print('Student ID')
    studentID = userInput("^\d{7}$")
    return studentID
```

--This is used to get user input for student Id and making sure it is a valid student ID.
"^\d{7}$" , regex for string that onlt contains 7 numbers.

```
def addNewRecord():
    studentID = getID()
    if (studentID in os.listdir("all")):
        print(studentID + " isn't unique")
        return 1
    student = Student(studentID)
    addSemester(student)
    student.writToFile()
```

--This Function Statifies the 1st Admin Option
-- It takes ID for new student record from user and checks for uniqueness against all records in the “all” directory

```
def getSemester():
    print('The year of the semester for example (enter 2019 for 2019-2020)')
    year = userInput('^\d{4}$')
    year += "-" + str(int(year) + 1)
    print('First semester (1) second semester (2), summer semester (3)Summer')
    semster = userInput('^[1-3]$')
    sem = year + "/" + semster
    return sem
```

--Takes user input for Semester with year and Semester number.
^\d{4}$', string made of only 4 digits.
'^[1-3]$', made of only one digit, 1 to 3

```
# Admin 2)
def addSemester(student):
    answer = 'y'
    while (answer == 'y') or (answer == 'Y'):
        sem = getSemester()
        if (student.isRecordedSemester(sem)):
            print("This semester was recorded")
        else:
            student.addNewSem(sem)
            updateStudentInformation(student, sem)


        print("Did you want to add any semester for "
              + student.id + " [y/n] (yes/no)")
        answer = userInput("[YyNn]")
```

--This function fulfils 2nd Admind option

--This is made to perform the second admin operation, adding a new semester to an already existing student record
--This function checks if the semister already exists in the student record, if it does, it raises an error, otherwise it continue normally using the addNewSem() function, and the UpdateStudentInformation(), the first is used to initialize a new semster and the latter is used to add courses to it.

```
# Admin 3)
def updateStudentInformation(student, sem):
    answer = 'y'
    while (answer == 'y') or (answer == 'Y'):
        print("[a/c] (Add course / change course grade) for "
              + student.id + " in " + sem)
        answer = userInput("[AaCc]")
        print("Course name")
        course = userInput('^[a-zA-Z]+\d+$')


        if ((answer == 'C') or (answer == 'c')) and (not student.isInSemester(sem, course)):
            print(student.id + " not have " + course + " course in " + sem + " to change")
        else:
            print("Enter course grade (like 90 without %)")
            grade = userInput('^\d{1,3}$')
            student.addOrEditCourse(sem, course, grade)


        print("Did you want to add more course or change for "
              + student.id + " in " + sem + " [y/n] (yes/no)")
        answer = userInput("[YyNn]")
```

--3rd admin function

--This function adds a course to a particular semester for a particular student, it first checks if the user wants to modify an existing or add a new one, if the user chooses to modify then it checks if the course exits within that semester, if it doesn't exits, then it raises an error, O.W it complies with the users commands.

^[a-zA-Z]+\d+$, String comprised of any number of alphabet characters followed by any number of digits.
^\d{1,3}$, String comprised from 1 to 3 numbers.

```
def AllStdAvg(students):  # takes in a list of all students
    sum = 0
    for stu in students:
        # first we calculate the overall student average
        sum = stu.average + sum
    return sum / Student.stdcount
```

--takes list containing all student objects, and uses that to calculate all overall student average.

```
def calcAllHours(Tuple):  # calcBAHours
    sum = 0
    for s in Tuple:
        sum += int(s[5])
    return sum
```

--Uses the Tuple to calculate the total number of hours in all the courses contained in the tuple since the second number in the course name is the number of hours in it, for example ENCS2380 is a three hours course.

```
def printGlobalAdmin():
    students = []
    Averages = []
    for id in os.listdir("all"):
        stu = Student(id)
        stu.setStdInfo()
        Averages.append(stu.average)
        students.append(stu)
    print("All students average grade: " + str(AllStdAvg(students)))
    Student.calcAllSemAvgHours()
    print("Average hours per semester : " + str(Student.SemAvgHours))
    print("Close figure to continue!!")
    plt.figure("Admin Mode Graph")
    plt.hist(Averages, bins=20)
    plt.xlabel("Average Grades")
    plt.ylabel("Number of students")
    plt.grid()
    plt.title("Histogram for Admin mode")
    plt.show()
```

--This function fulfils 5th admin option
--Scans all files in “all” directory and reads from them one by one, using the setStdInfo() function, the data gathered from this is then used to fill the students list and the Averages list.
--it then prints out all student averages, and all semester average hours

--The function also creates a histogram for the student averages using the matplotlib.pyplot library, the bins=20 is the number of bins allowed in this plot.

```
def printGlobalStudent():
    students = []
    Averages = []
    IDs = []
    for id in os.listdir("all"):
        stu = Student(id)
        stu.setStdInfo()
        Averages.append(stu.average)
        IDs.append(stu.id)
        students.append(stu)
    print("All students average grade: " + str(AllStdAvg(students)))
    Student.calcAllSemAvgHours()
    print("Average hours per semester : " + str(Student.SemAvgHours))
    print("Close figure to continue!!")
    plt.figure("Student Mode Graph")
    plt.plot(IDs, Averages)
    plt.xlabel("Student IDs")
    plt.ylabel("Student Averages")
    plt.grid()
    plt.title("graph for student mode")
    plt.show()
```

--This function fulfils the 2nd student option
--This is the exact same function as the admin, but instead of plotting a histogram, a normal plot is done instead

--This function fulfils the 1st student Option

--This function will print out all data concerning a single student

```python
# Student 1)
def studentStatistics(studentID):
  stu = Student(studentID)
  stu.setStdInfo()
  stu.setRemainingCourses()
  print("Taken hours: " + str(stu.takenHours))
  print("Remaining hours " + str(calcAllHours(Student.Tuple) - stu.takenHours))
  print("Remaining courses: " + str(stu.remainingCourses))
  print("Overall average: " + str(stu.average))


  print("Did you want to see any semester average for "
      + stu.id + " [y/n] (yes/no)")
  answer = userInput("[YyNn]")
  while (answer == 'y') or (answer == 'Y'):
    sem = getSemester()
    if not (stu.isRecordedSemester(sem)):
      print("This semester wasn't recorded")
    else:
      print(sem + " semester average: " + str(stu.semAve[sem]))
    print("Did you want to see any semester average for "
        + stu.id + " [y/n] (yes/no)")
    answer = userInput("[YyNn]")
```

--after that it asks the user if the user wants to see student average for a particualr semester, if the student was enrolled in that semester it shows the output, otherwise it raises an error

```python
# Admin 4)
# the same as studentStatis() but ask about student ID
def studentStatisticsForAdmin():
  studentID = getID()
  if not (studentID in os.listdir("all")):
    print(studentID + " wasn't recorded")
    return 1
  studentStatistics(studentID)
```

--This function fulfils the 4th Admin Option
--Exactly the same function as before, excpet this time it asks for studentID

```
# Admin 6)
def searching():
  print("please enter parameter you want to search by, eg (<80)")
  parameter = userInput("[><=][6-9][0-9]")
  op = parameter[0]
  threshold = int(parameter[1:])
  students = {}
  for id in os.listdir("all"):
    stu = Student(id)
    stu.setStdInfo()
    if (op == '>'):
      if (stu.average > threshold):
        students[id] = stu.average
    elif (op == '='):
      if (stu.average == threshold):
        students[id] = stu.average
    else:
      if (stu.average < threshold):
        students[id] = stu.average
  if (students):
    print("students with Average " + op + " " + str(threshold))
    print(students)
  else:
    print("no students in searched parameter")
```

--This function fulfils the 6th Admin functionality

--asks user for search threshold and operator, after the input format is correct the function begins reading all the records one by one, it then compares them to user given parameter and adds the ones that fulfill the parameter to the students dict
--it then prints out the dictionary onto the terminal

Driver:

```
print("User ID")
id = userInput("^\d{7}$")
Student.stdcount = 0
```

Takes in user Id and makes sure it's a String made of 7 digits
--We also reset student count each run

```
if(id == '0002244'):
  print("\n Admin Mode")
  print("------------")
  print("1) Add a new record")
  print("2) Add new semester")
  print("3) Update student information")
  print("4) Student statistics")
  print("5) Global statistics")
  print("6) Searching")
  choice = int(userInput("^[1-6]$"))
```

If the user enter the Admin ID, which we set to 0002244 for the sake of testing, the user will get a menu with the Admin options, and promt to enter the required option number

```
if (choice == 1):
   addNewRecord()
elif (choice == 2):
   studentID = getID()
   if (not studentID in os.listdir("all")):
      print(studentID + " isn't recorded")
   else:
      stu = Student(studentID)
      addSemester(stu)
      stu.writToFile()
elif (choice == 3):
   studentID = getID()
   if (not studentID in os.listdir("all")):
      print(studentID + " isn't recorded")
   else:
      stu = Student(studentID)
      sem = getSemester()
      if (stu.isRecordedSemester(sem)):
         updateStudentInformation(stu, sem)
         stu.writToFile()
      else:
         print("This semester wasn't recorded")


elif (choice == 4):
   studentStatisticsForAdmin()
elif (choice == 5):
   globalStatisticsWithHistogram()
elif (choice == 6):
   searching()
```

--Control structure that gives user acces to the required function, in option 1 through 3 there is always a writeToFile function call, this is because these operation all update the records, and therefore the files need to be updates

--Option 4 through 6, all just read from the files so there's not need to update files since there's no change.

```
else:
   if not (id in os.listdir("all")):
      print(id + " wasn't recorded")
      exit()
   else:
      print("\n Student Mode")
      print("--------------")
      print("1) Student statistics")
      print("2) Global statistics")
      choice = int(userInput("[1-2]"))
      if (choice == 1):
         studentStatistics(id)
      elif (choice == 2):
         globalStatisticsWithGraph()
      print("exit ? [y,n]")
      answer = userInput("^[YyNn]$")
```

--if the input is not that of the admins, we assume user to be student, so we search for his records, if they are found we continue otherwise, we raise an error and exits

--print a menu for the student for the options and prompt the user to choose the desired option

```
print("exit ? [y,n]")
answer = userInput("^[YyNn]$")
if (answer == 'Y' or answer == 'y'):
   print("Thank you for using our program!")
   exit()
```

--ask the user if he is done with the program, if yes then exit, otherwise print menu again.

Program Results

we will be using these five files as our testing material:

1190081

```
Year\Semester ; Courses with grades
2021-2022/1 ; ENCS2380 90, ENCS2110 87, ENCS311 90, ENEE2304 75, ENEE2307 93
2021-2022/2 ; ENCS3340 67, ENEE3320 72, ENCS4370 66, ENCS5150 83, ENCS3210 78
2020-2021/1 ; ENCS4113 75, ENCS3330 65, ENCS3340 78, ENEE4320 65, ENEE5150 63
```

1190082

```
Year/Semester ; Courses with grades
2021-2022/1 ; ENCS2380 87, ENCS2110 84, ENCS311 73, ENEE2304 72, ENEE2307 77
2021-2022/2 ; ENCS3340 63, ENEE3320 76, ENCS4370 76, ENCS5150 86, ENCS3210 88
2020-2021/1 ; ENCS4113 94, ENCS3330 82, ENCS3340 69, ENEE4320 79, ENEE5150 73
```

1190083

```
Year/Semester ; Courses with grades
2021-2022/1 ; ENCS2380 85 ENCS2110 76, ENCS311 68, ENEE2304 76, ENEE2307 91
2021-2022/2 ; ENCS3340 77, ENEE3320 77, ENCS4370 76, ENCS5150 95, ENCS3210 81
2020-2021/1 ; ENCS4113 94, ENCS3330 77, ENCS3340 60, ENEE4320 77, ENEE5150 85
```

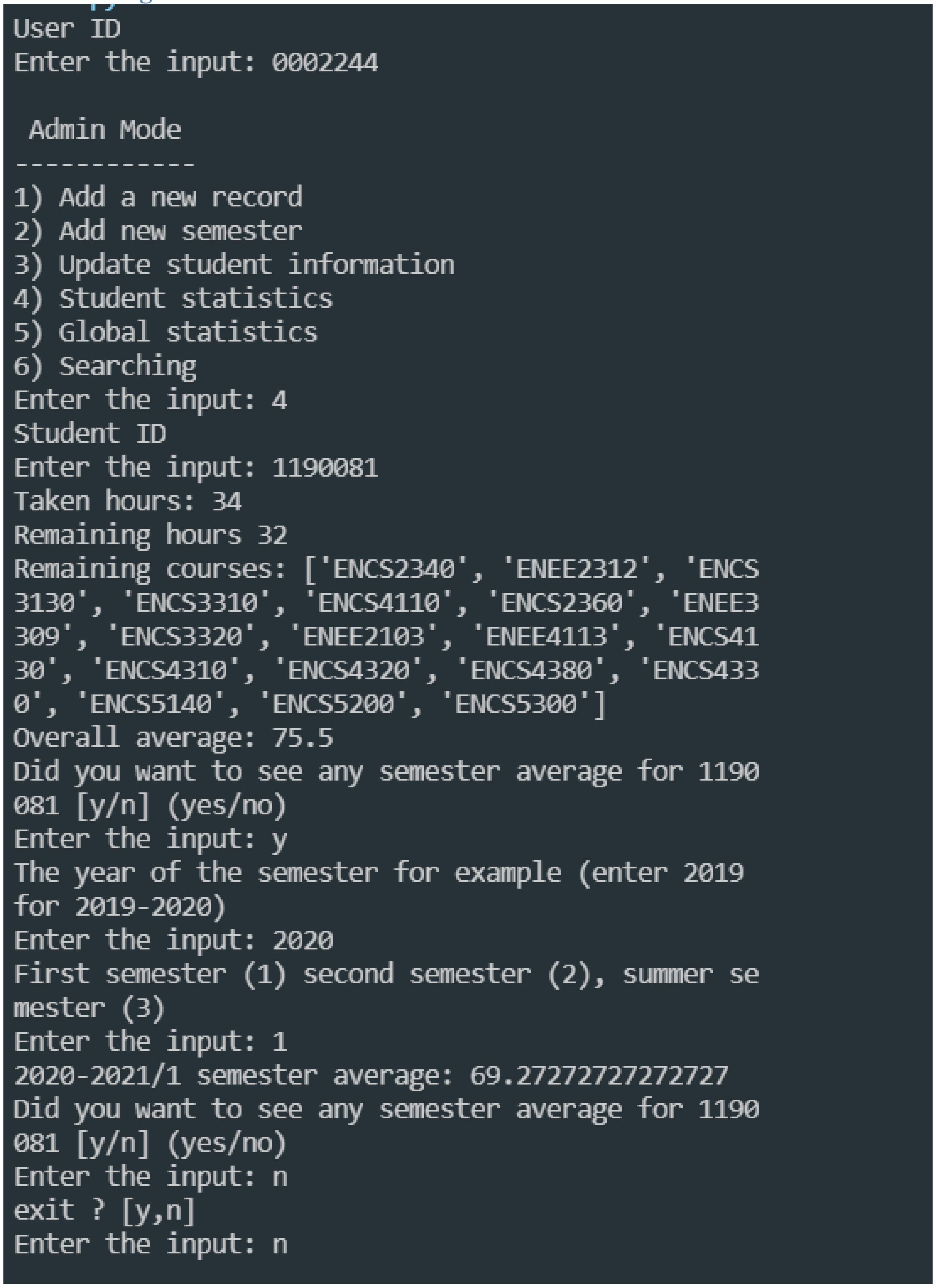
1190084

```
Year/Semester ; Courses with grades
2021-2022/1 ; ENCS2380 70, ENCS2110 93, ENCS311 64, ENEE2304 74, ENEE2307 87
2021-2022/2 ; ENCS3340 66, ENEE3320 65, ENCS4370 76, ENCS5150 69, ENCS3210 78
2020-2021/1 ; ENCS4113 74, ENCS3330 79, ENCS3340 87, ENEE4320 69, ENEE5150 65
```

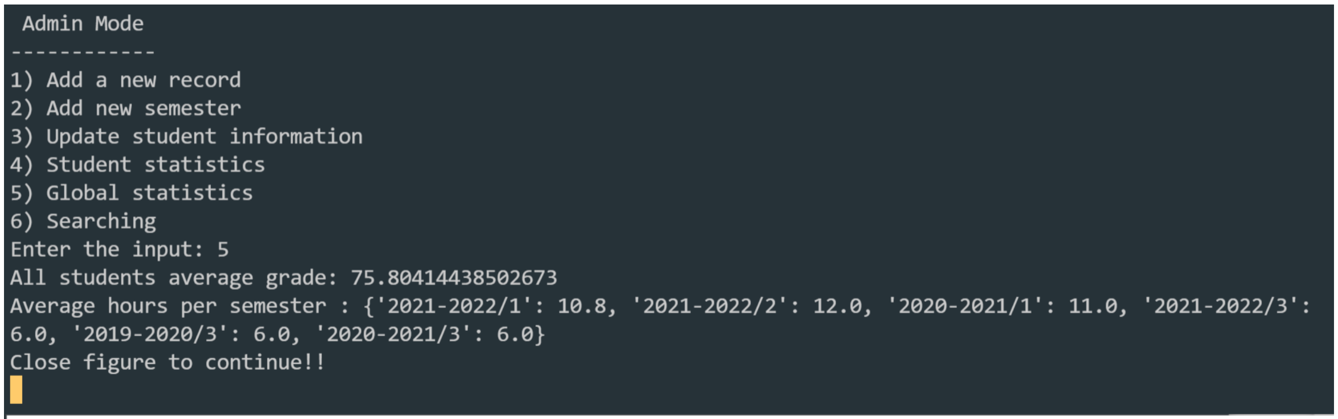
1190086

```
Year\Semester ; Courses with grades
2019-2020/3 ; ENCS2340 86, ENEE2304 92
2020-2021/3 ; ENEE2307 77, ENCS4370 80
```

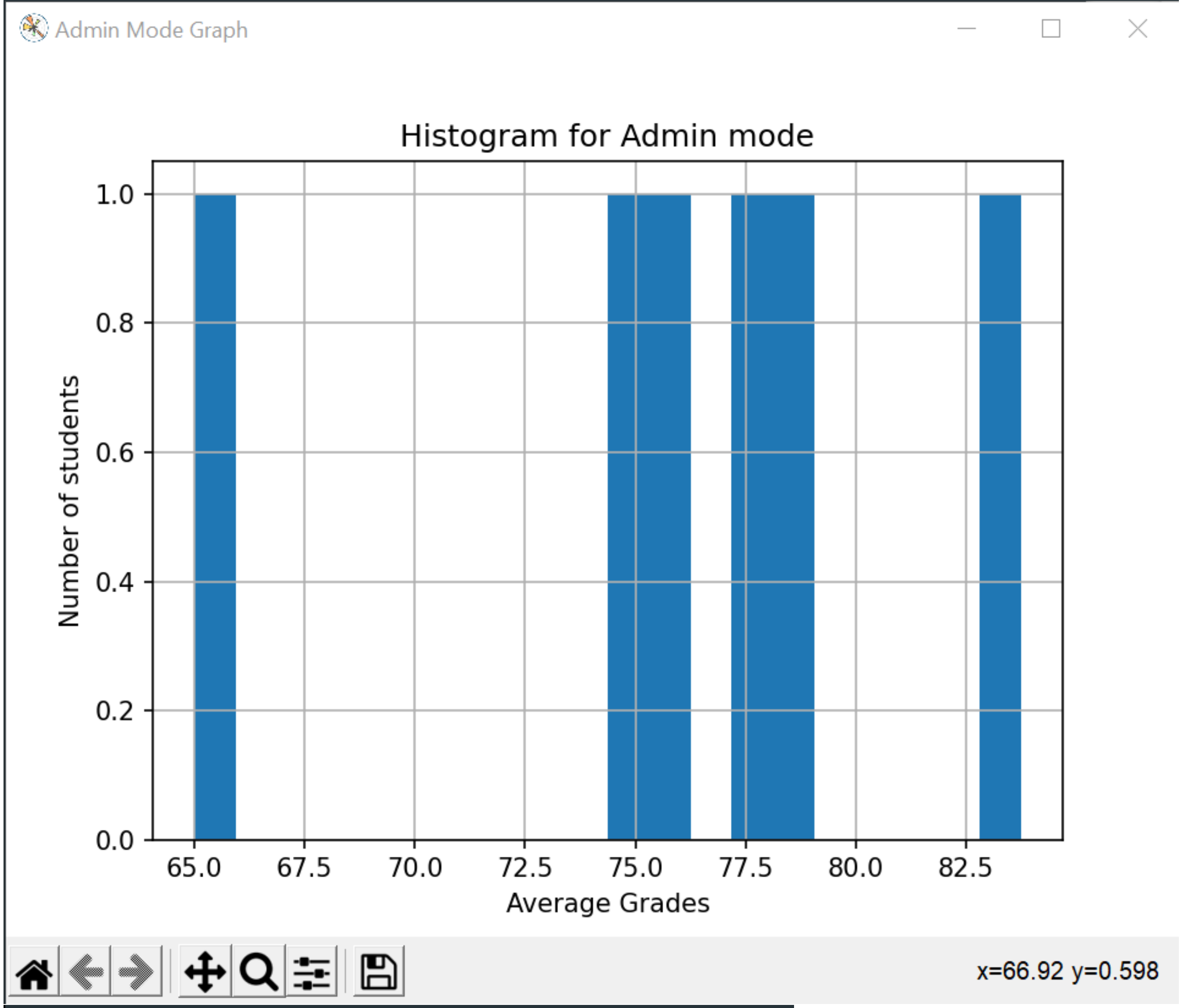
1- Producing Student and Global results as Admin:

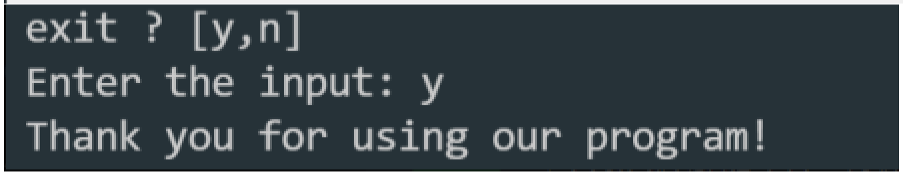


```
User ID
Enter the input: 0002244

 Admin Mode
------------
1) Add a new record
2) Add new semester
3) Update student information
4) Student statistics
5) Global statistics
6) Searching
Enter the input: 4
Student ID
Enter the input: 1190081
Taken hours: 34
Remaining hours 32
Remaining courses: ['ENCS2340', 'ENEE2312', 'ENCS
3130', 'ENCS3310', 'ENCS4110', 'ENCS2360', 'ENEE3
309', 'ENCS3320', 'ENEE2103', 'ENEE4113', 'ENCS41
30', 'ENCS4310', 'ENCS4320', 'ENCS4380', 'ENCS433
0', 'ENCS5140', 'ENCS5200', 'ENCS5300']
Overall average: 75.5
Did you want to see any semester average for 1190
081 [y/n] (yes/no)
Enter the input: y
The year of the semester for example (enter 2019
for 2019-2020)
Enter the input: 2020
First semester (1) second semester (2), summer se
mester (3)
Enter the input: 1
2020-2021/1 semester average: 69.27272727272727
Did you want to see any semester average for 1190
081 [y/n] (yes/no)
Enter the input: n
exit ? [y,n]
Enter the input: n
```



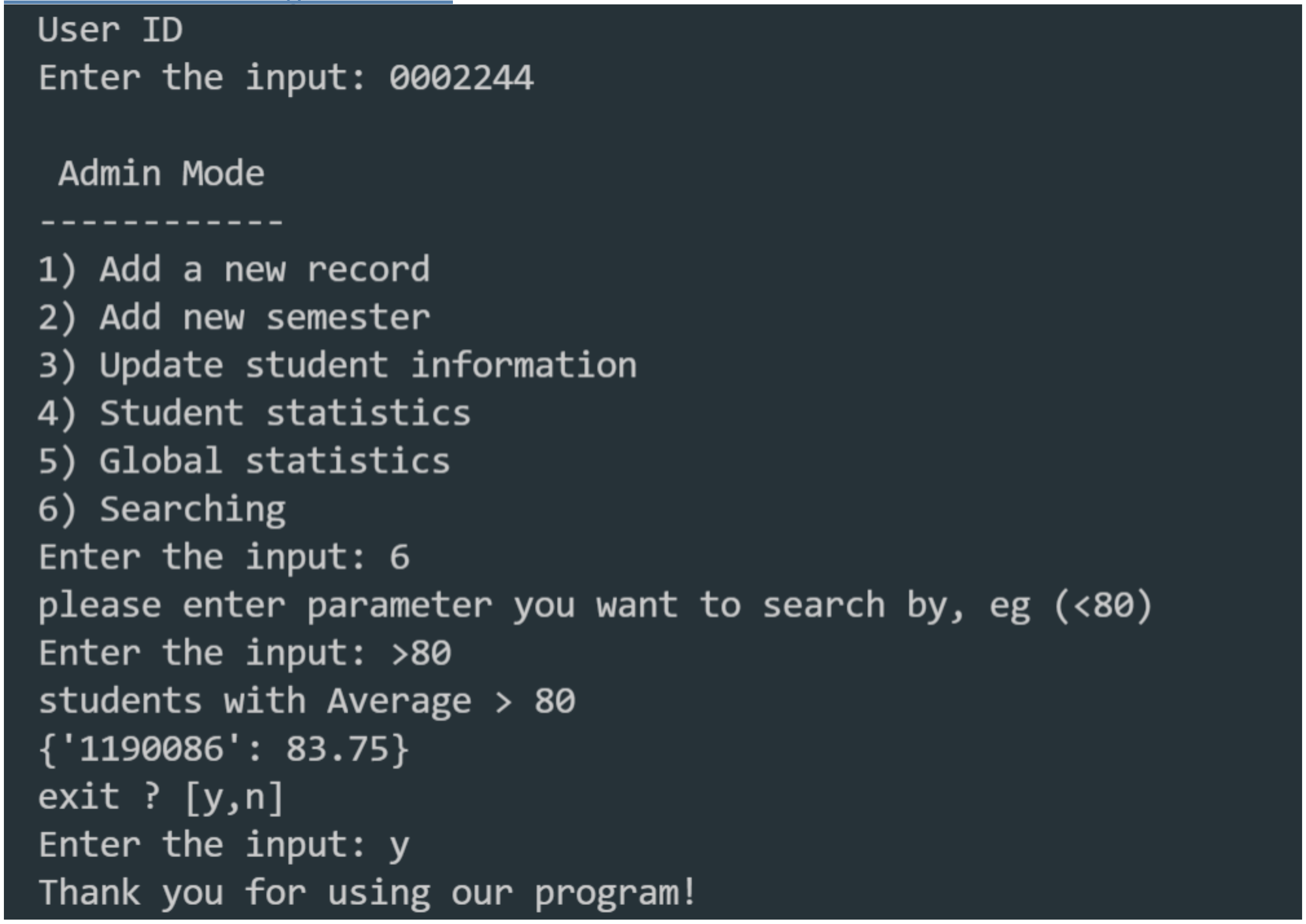
```
 Admin Mode
------------
1) Add a new record
2) Add new semester
3) Update student information
4) Student statistics
5) Global statistics
6) Searching
Enter the input: 5
All students average grade: 75.80414438502673
Average hours per semester : {'2021-2022/1': 10.8, '2021-2022/2': 12.0, '2020-2021/1': 11.0, '2021-2022/3':
6.0, '2019-2020/3': 6.0, '2020-2021/3': 6.0}
Close figure to continue!!
```





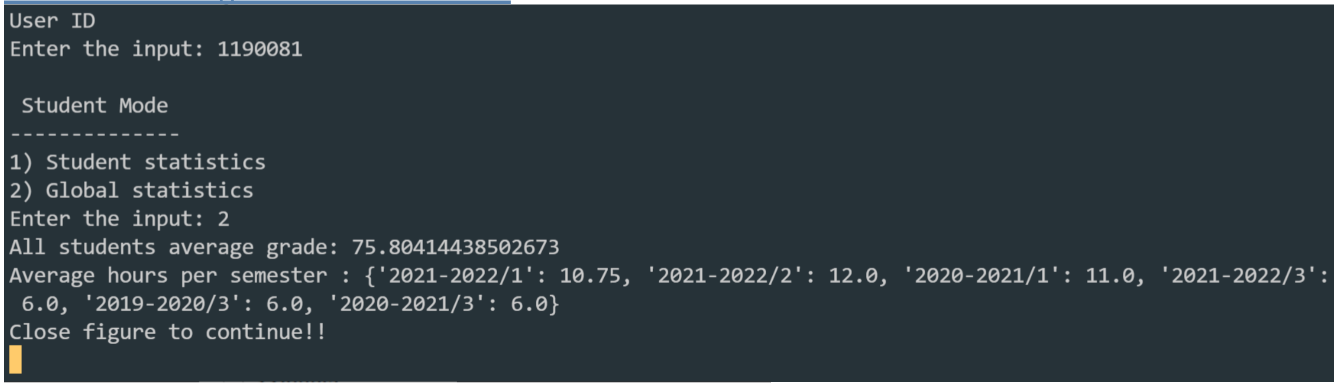
```
exit ? [y,n]
Enter the input: y
Thank you for using our program!
```

2-Admin to see averages above 80

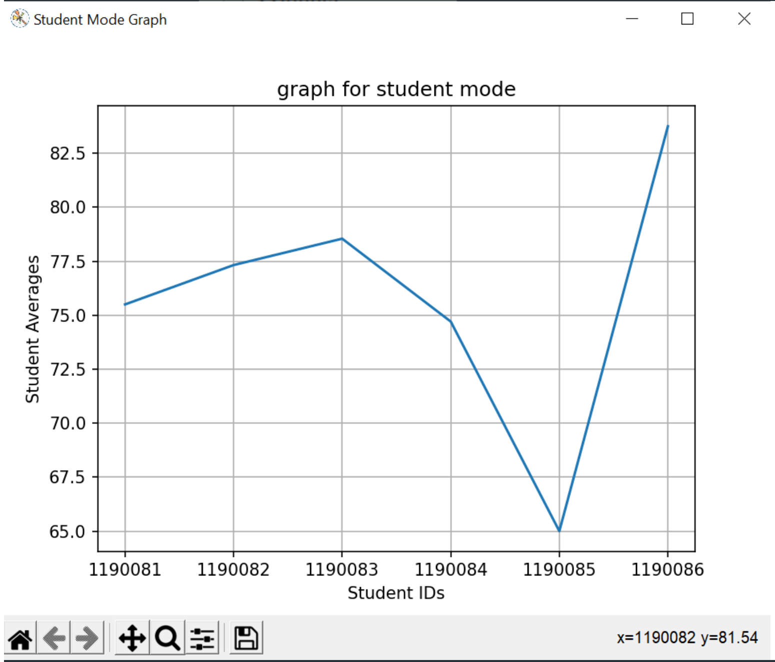


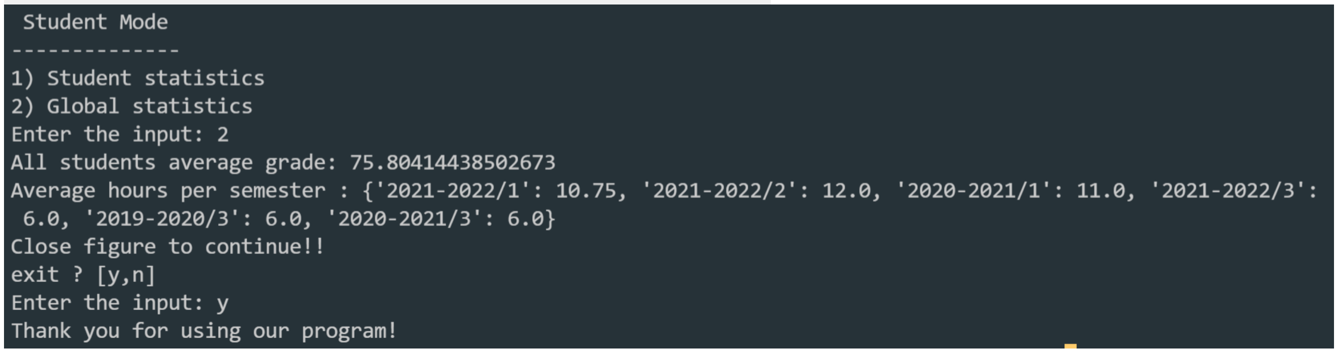
```
User ID
Enter the input: 0002244

 Admin Mode
------------
1) Add a new record
2) Add new semester
3) Update student information
4) Student statistics
5) Global statistics
6) Searching
Enter the input: 6
please enter parameter you want to search by, eg (<80)
Enter the input: >80
students with Average > 80
{'1190086': 83.75}
exit ? [y,n]
Enter the input: y
Thank you for using our program!
```

3-Student Seeing Global Information



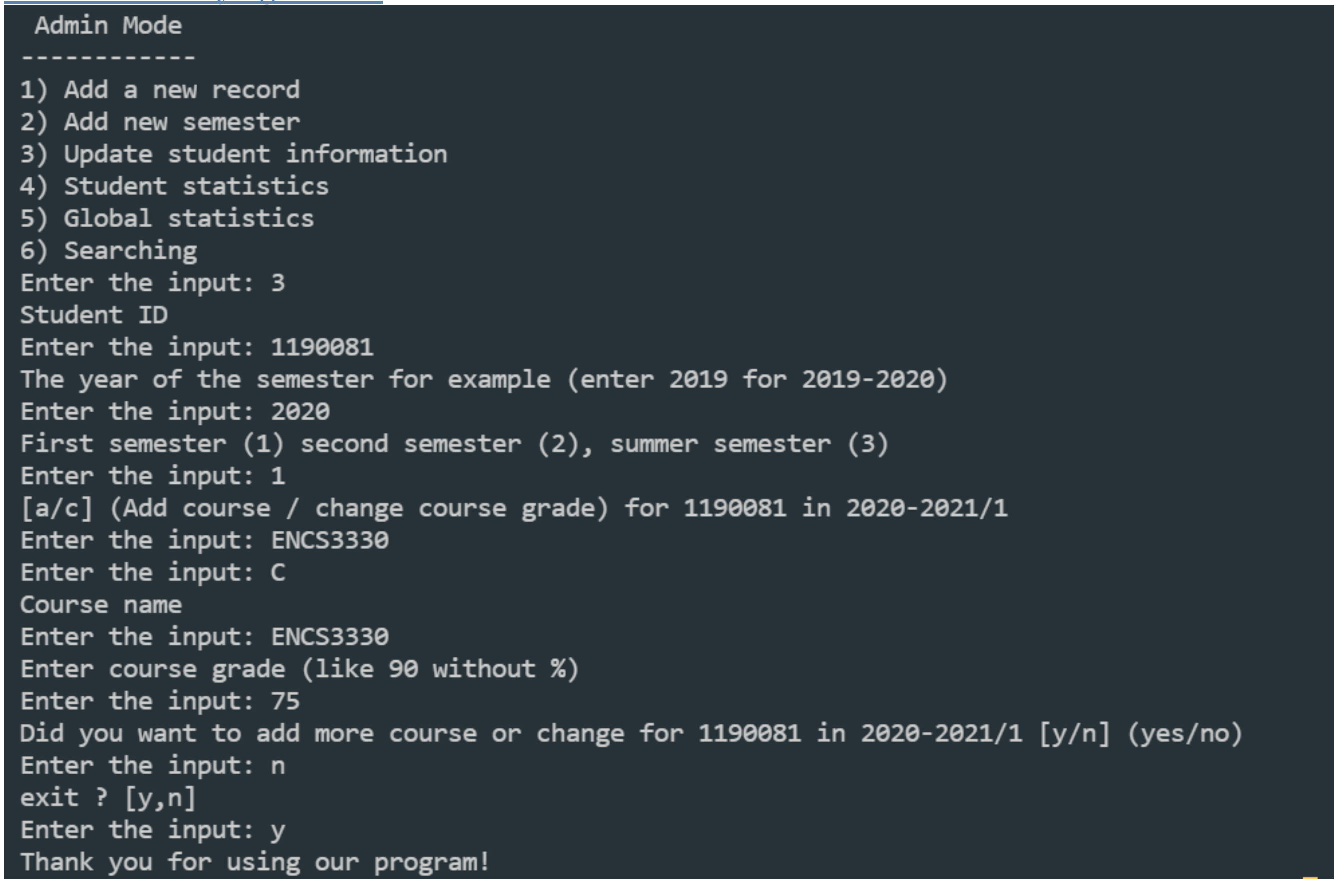
```
User ID
Enter the input: 1190081

 Student Mode
--------------
1) Student statistics
2) Global statistics
Enter the input: 2
All students average grade: 75.80414438502673
Average hours per semester : {'2021-2022/1': 10.75, '2021-2022/2': 12.0, '2020-2021/1': 11.0, '2021-2022/3':
 6.0, '2019-2020/3': 6.0, '2020-2021/3': 6.0}
Close figure to continue!!
```





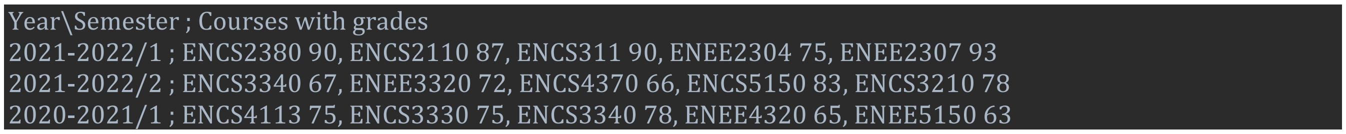
```
 Student Mode
--------------
1) Student statistics
2) Global statistics
Enter the input: 2
All students average grade: 75.80414438502673
Average hours per semester : {'2021-2022/1': 10.75, '2021-2022/2': 12.0, '2020-2021/1': 11.0, '2021-2022/3':
 6.0, '2019-2020/3': 6.0, '2020-2021/3': 6.0}
Close figure to continue!!
exit ? [y,n]
Enter the input: y
Thank you for using our program!
```

4-Admin Modifying Record



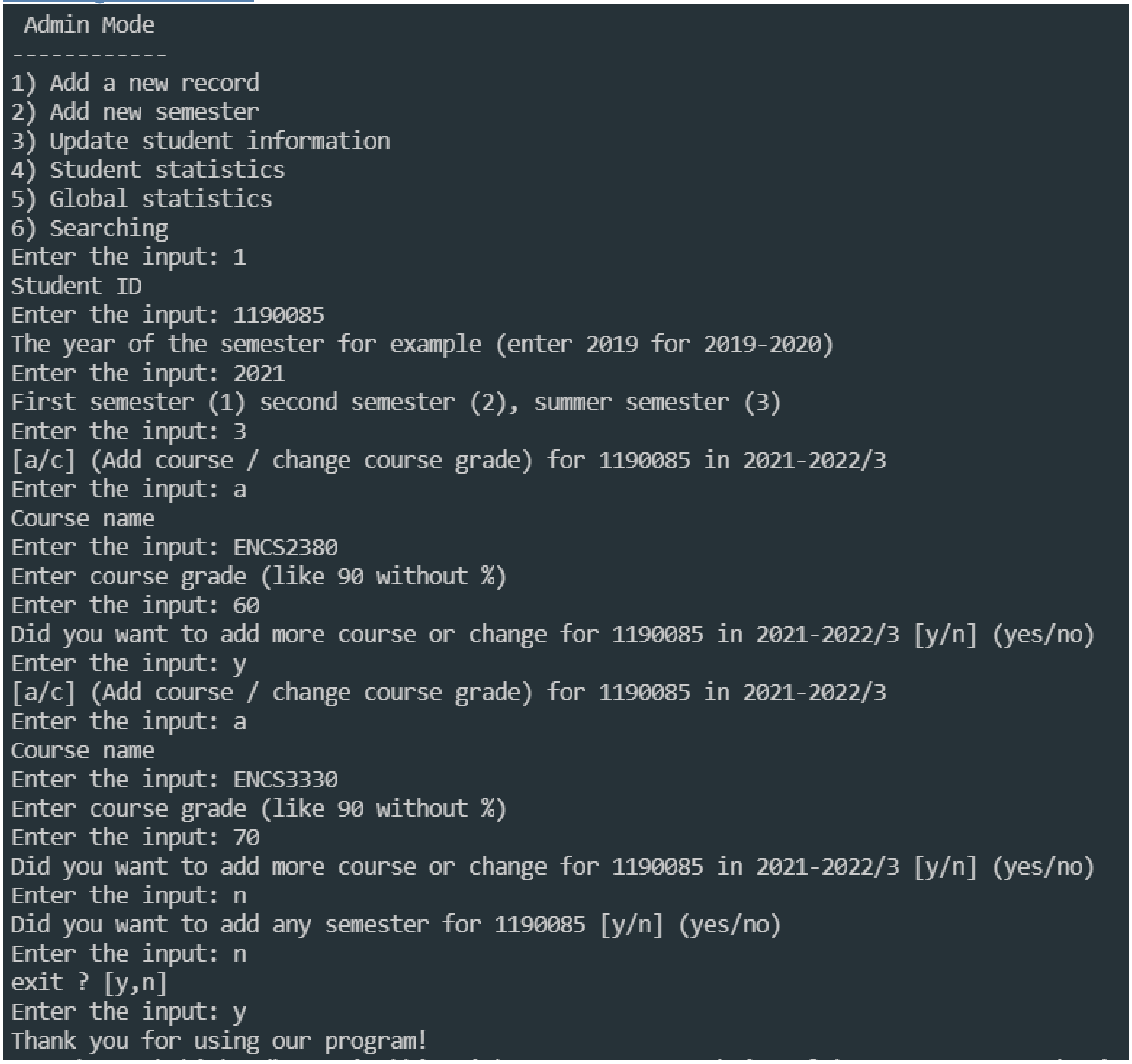
```
 Admin Mode
------------
1) Add a new record
2) Add new semester
3) Update student information
4) Student statistics
5) Global statistics
6) Searching
Enter the input: 3
Student ID
Enter the input: 1190081
The year of the semester for example (enter 2019 for 2019-2020)
Enter the input: 2020
First semester (1) second semester (2), summer semester (3)
Enter the input: 1
[a/c] (Add course / change course grade) for 1190081 in 2020-2021/1
Enter the input: ENCS3330
Enter the input: C
Course name
Enter the input: ENCS3330
Enter course grade (like 90 without %)
Enter the input: 75
Did you want to add more course or change for 1190081 in 2020-2021/1 [y/n] (yes/no)
Enter the input: n
exit ? [y,n]
Enter the input: y
Thank you for using our program!
```

1190081 After this run



```
Year\Semester ; Courses with grades
2021-2022/1 ; ENCS2380 90, ENCS2110 87, ENCS311 90, ENEE2304 75, ENEE2307 93
2021-2022/2 ; ENCS3340 67, ENEE3320 72, ENCS4370 66, ENCS5150 83, ENCS3210 78
2020-2021/1 ; ENCS4113 75, ENCS3330 75, ENCS3340 78, ENEE4320 65, ENEE5150 63
```

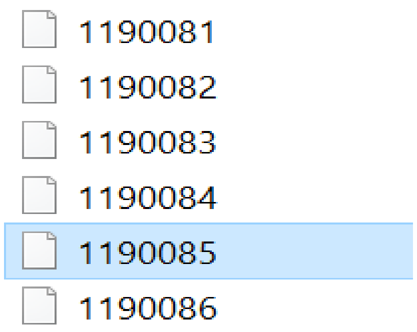
5-Adding a new record



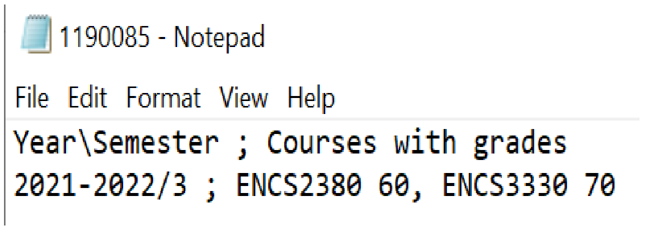
```
 Admin Mode
------------
1) Add a new record
2) Add new semester
3) Update student information
4) Student statistics
5) Global statistics
6) Searching
Enter the input: 1
Student ID
Enter the input: 1190085
The year of the semester for example (enter 2019 for 2019-2020)
Enter the input: 2021
First semester (1) second semester (2), summer semester (3)
Enter the input: 3
[a/c] (Add course / change course grade) for 1190085 in 2021-2022/3
Enter the input: a
Course name
Enter the input: ENCS2380
Enter course grade (like 90 without %)
Enter the input: 60
Did you want to add more course or change for 1190085 in 2021-2022/3 [y/n] (yes/no)
Enter the input: y
[a/c] (Add course / change course grade) for 1190085 in 2021-2022/3
Enter the input: a
Course name
Enter the input: ENCS3330
Enter course grade (like 90 without %)
Enter the input: 70
Did you want to add more course or change for 1190085 in 2021-2022/3 [y/n] (yes/no)
Enter the input: n
Did you want to add any semester for 1190085 [y/n] (yes/no)
Enter the input: n
exit ? [y,n]
Enter the input: y
Thank you for using our program!
```

Files After this run

1190085



```
1190081
1190082
1190083
1190084
1190085
1190086
```



```
Year\Semester ; Courses with grades
2021-2022/3 ; ENCS2380 60, ENCS3330 70
```