Software Design and Development

Assessment Report

**Task 1 Evidence:**

**Add your algorithm below:**

1.0 **Input a Student’s Coursework Mark:**

1.1 Declare variable courseworkMark and assign to -1.

1.2 While courseworkMark is less than 0 or greater than 60:

1.3 Prompt the user to enter coursework mark.

1.4 Convert input to floating point number and assign to courseworkMark.

2.0 **Input Prelim Mark:**

2.1 Declare variable prelimMark and assign to -1.

2.2 While prelimMark is less than 0 or greater than 90:

2.3 Prompt the user to enter prelim mark.

2.4 Convert input to floating point number and assign to prelimMark.

3.0 **Calculate Mark Percentage:**

3.1 Calculate the mark percentage as the integer value of the sum of courseworkMark and prelimMark multiplied by 100 divided by 150.

4.0 **Determine Grade:**

4.1 If mark percentage is greater than or equal to 70:

4.2 Display student achieved grade A.

4.3 Else if mark percentage is between 60 and 69:

4.4 Display student achieved grade B.

4.5 Else if mark percentage is between 50 and 59:

4.6 Display student achieved grade C.

4.7 Else if mark percentage is between 45 and 49:

4.8 Display student achieved grade D.

4.9 Else no grade awarded.

START

1.1 INITIALISE courseworkMark to -1

1.2 WHILE 0 > courseworkMark > 60  
1.3 DISPLAY "Enter coursework mark: "  
1.4 INPUT courseworkMark

2.1 INITIALISE prelimMark to -1

2.2 WHILE 0 > prelimMark > 90

2.3 DISPLAY "Enter prelim mark: "

2.4 INPUT prelimMark

3.1 SET markPercentage to integer value of (courseworkMark + prelimMark) \* 100 / 150.

4.1 IF markPercentage >= 70:

4.2 DISPLAY "Student achieved grade A with total mark of”, markPercentage

4.3 ELSE IF markPercentage >= 60:

4.4 DISPLAY "Student achieved grade B with total mark of", markPercentage

4.5 ELSE IF markPercentage >= 50:

4.6 DISPLAY "Student achieved grade C with total mark of", markPercentage

4.7 ELSE IF markPercentage >= 45:

4.8 DISPLAY "Student achieved grade D with total mark of", markPercentage

4.9 ELSE:

4.10 DISPLAY "Student failed to achieve a grade with total mark of", markPercentage

**Task 2 Evidence:**

**Using the table below complete the test plan for the program:**

**Note: You may need to add additional rows in order to show a complete set of tests.**

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Reason for test (valid/invalid – Normal, exceptional, extreme etc.) | Test Data | Expected Output |
| coursework | Normal | 40 | Data accepted |
| coursework | Extreme | 0 | Data accepted |
| coursework | Extreme | 60 | Data accepted |
| coursework | Exceptional | 65 | Data rejected |
| coursework | Exceptional | Sixteen! | Data rejected:  “Please use numbers.” |
| prelim | Normal | 50 | Data accepted |
| prelim | Extreme | 0 | Data accepted |
| prelim | Extreme | 90 | Data accepted |
| prelim | Exceptional | 100 | Data rejected |
| prelim | Exceptional | Seventy | Data rejected:  “Please use numbers.” |
| calculate\_percentage | Normal | 40, 80 | Data accepted |
| calculate\_percentage | Extreme | 0, 0 | Data accepted |
| calculate\_percentage | Extreme | 60, 90 | Data accepted |
| display\_grade | Normal | 20, 30 | Data accepted |
| display\_grade | Extreme | 0, 0 | Data accepted |
| display\_grade | Extreme | 60, 90 | Data accepted |

**Task 3 and 4 Evidence:**

**Copy/ paste your source code and add below: Make sure to add internal commentary asked for.**

# Author: Gary Siu  
# Date: 18th Feb 2024  
# Description: Calculate student grade from coursework and prelim marks  
  
# Function to return coursework marks  
def coursework():  
 # Initialise variable  
 mark = -1  
 # Loop until coursework mark within limits  
 while mark < 0 or mark > 60:  
 # Catch ValueErrors such as strings  
 try:  
 # Input student coursework mark  
 mark = float(input("Enter coursework mark(0-60): "))  
 except ValueError:  
 print("Please use numbers.")  
 return mark  
  
  
# Function to return prelim marks  
def prelim():  
 # Initialise variable  
 mark = -1  
 # Loop until prelim mark within limits  
 while mark < 0 or mark > 90:  
 # Catch ValueErrors such as strings  
 try:  
 # Input student prelim mark  
 mark = float(input("Enter prelim mark (0-90): "))  
 except ValueError:  
 print("Please use numbers.")  
 return mark  
  
  
# Function to calculate mark percentage  
def calculate\_percentage(coursework\_mark, prelim\_mark):  
 # Use coursework and prelim marks  
 return int(((coursework\_mark + prelim\_mark) \* 100) / 150)  
  
  
# Function to return student grade  
def display\_grade(coursework, prelim):  
 # Pass in marks to calculate\_percentage function  
 mark\_percentage = calculate\_percentage(coursework, prelim)  
 # IF conditionals to determine grade achieved  
 if mark\_percentage >= 70:  
 return f"Student achieved grade A with total mark of **{**mark\_percentage**}.**"  
 elif mark\_percentage >= 60:  
 return f"Student achieved grade B with total mark of **{**mark\_percentage**}.**"  
 elif mark\_percentage >= 50:  
 return f"Student achieved grade C with total mark of **{**mark\_percentage**}.**"  
 elif mark\_percentage >= 45:  
 return f"Student achieved grade D with total mark of **{**mark\_percentage**}.**"  
 else:  
 return (f"Student failed to achieve a grade with total mark of "  
 f"**{**mark\_percentage**}.**")  
  
  
def main():  
 # Call functions within main  
 coursework\_mark = coursework()  
 prelim\_mark = prelim()  
 # Pass values returned from coursework and prelim functions into  
 # display\_grade() and display the result returned  
 print(display\_grade(coursework\_mark, prelim\_mark))  
  
  
# Call main to start the program  
main()

**Task 5 Evidence:**

**Use the table below to complete the testing outlined in above test plan.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function Name | Reason for test (valid/invalid – Normal, exceptional, extreme etc.) | Test Data | Expected Output | Actual Output | Comments |
| coursework | Normal | 40 | Data accepted | As expected |  |
| coursework | Extreme | 0 | Data accepted | As expected |  |
| coursework | Extreme | 60 | Data accepted | As expected |  |
| coursework | Exceptional | 65 | Data rejected:  “Enter coursework mark (0-60): ” | As expected |  |
| coursework | Exceptional | Sixteen! | Data rejected:  “Please use numbers.” | As expected |  |
| prelim | Normal | 50 | Data accepted | As expected |  |
| prelim | Extreme | 0 | Data accepted | As expected |  |
| prelim | Extreme | 90 | Data accepted | As expected |  |
| prelim | Exceptional | 100 | Data rejected:  “Enter prelim mark (0-90): ” | As expected |  |
| prelim | Exceptional | Seventy | Data rejected:  “Please use numbers.” | As expected |  |
| calculate\_percentage | Normal | 40, 80 | Data accepted | As expected |  |
| calculate\_percentage | Extreme | 0, 0 | Data accepted | As expected |  |
| calculate\_percentage | Extreme | 60, 90 | Data accepted | As expected |  |
| display\_grade | Normal | 20, 30 | Data accepted:  “Student failed to achieve a grade with total mark of 33.” | As expected |  |
| display\_grade | Extreme | 0, 0 | Data accepted:  “Student failed to achieve a grade with total mark of 0.” | As expected |  |
| display\_grade | Extreme | 60, 90 | Data accepted:  “Student achieved grade A with total mark of 100.” | As expected |  |

**If any of the above tests failed, then outline how you fixed it and rerun the test to show that it now works correctly (include screenshot below to show this).**

**Task 6 Evidence:**

**Copy/ paste your source code (read from file included) and add below: Make sure to add internal commentary asked for.**

# Author: Gary Siu  
# Date: 19th Feb 2024  
# Description: Calculate students' grades from external files  
  
# Function to read data from external files  
def read\_files():  
 # Open names.txt and read contents  
 with open("names.txt", "r") as f:  
 # Read each line in file and append to list  
 names = f.read().splitlines()  
 with open("mark1.txt", "r") as f:  
 mark1 = f.read().splitlines()  
 with open("mark2.txt", "r") as f:  
 mark2 = f.read().splitlines()  
 # Initialise empty list  
 students = []  
 # For loop to iterate over the length of the "names" list  
 for i in range(len(names)):  
 # Create list of student name, mark1, mark2  
 student = [names[i], mark1[i], mark2[i]]  
 # Create nested list of students  
 students.append(student)  
 return students  
  
  
# Function to calculate mark percentage  
def calculate\_percentage():  
 # Call read\_files(), assign students with respective marks to "students"  
 students = read\_files()  
 # Initialise empty list  
 students\_grade = []  
 # Loop each student in students list  
 for student in students:  
 # Use student's coursework and prelim marks to calculate percentage  
 percentage = int(((int(student[1]) + int(student[2])) \* 100) / 150)  
 # Call determine\_grade function and pass percentage as parameter  
 grade = determine\_grade(percentage)  
 # Create list with student name, percentage, and grade  
 student\_grade = [student[0], percentage, grade]  
 # Create nested list of students with their percentages and grades  
 students\_grade.append(student\_grade)  
 return students\_grade  
  
  
# Function to return student grade  
def determine\_grade(percentage):  
 # IF conditionals to determine grade achieved  
 if percentage >= 70:  
 return "A"  
 elif percentage >= 60:  
 return "B"  
 elif percentage >= 50:  
 return "C"  
 elif percentage >= 45:  
 return "D"  
 else:  
 return "Fail"  
  
  
def main():  
 # Display list of students returned from calculate\_percentage()  
 print(calculate\_percentage())  
  
  
# Call main to start the program  
main()

**Task 7 Evidence:**

**Copy/ paste your source code (count occurences / max algorithm now included) and add below: Make sure to add internal commentary asked for.**

# Author: Gary Siu  
# Date: 19th Feb 2024  
# Description: Calculate students' grades from external files, and get number   
# of grade A's achieved and the highest percentage scored  
  
# Function to read data from external files  
def read\_files():  
 # Open names.txt and read contents  
 with open("names.txt", "r") as f:  
 # Read each line in file and append to list  
 names = f.read().splitlines()  
 with open("mark1.txt", "r") as f:  
 mark1 = f.read().splitlines()  
 with open("mark2.txt", "r") as f:  
 mark2 = f.read().splitlines()  
 # Initialise empty list  
 students = []  
 # For loop to iterate over the length of the "names" list  
 for i in range(len(names)):  
 # Create list of student name, mark1, mark2  
 student = [names[i], mark1[i], mark2[i]]  
 # Create nested list of students  
 students.append(student)  
 return students  
  
  
# Function to calculate mark percentage  
def calculate\_percentage():  
 # Call read\_files(), assign students with respective marks to "students"  
 students = read\_files()  
 # Initialise empty list  
 students\_grade = []  
 # Loop each student in students list  
 for student in students:  
 # Use student's coursework and prelim marks to calculate percentage  
 percentage = int(((int(student[1]) + int(student[2])) \* 100) / 150)  
 # Call determine\_grade function and pass percentage as parameter  
 grade = determine\_grade(percentage)  
 # Create list with student name, percentage, and grade  
 student\_grade = [student[0], percentage, grade]  
 # Create nested list of students with their percentages and grades  
 students\_grade.append(student\_grade)  
 return students\_grade  
  
  
# Function to return student grade  
def determine\_grade(percentage):  
 # IF conditionals to determine grade achieved  
 if percentage >= 70:  
 return "A"  
 elif percentage >= 60:  
 return "B"  
 elif percentage >= 50:  
 return "C"  
 elif percentage >= 45:  
 return "D"  
 else:  
 return "Fail"  
  
  
# Function to return number of A's and best percentage  
def results(students):  
 # Initialise counter  
 count = 0  
 # Initialise max percentage to first student in students list  
 max\_percentage = students[0][1]  
 # Loop through students list with student as iterative  
 for student in students:  
 # Conditional if student grade is "A", add 1 to count  
 if student[2] == "A":  
 count += 1  
 # Conditional if max\_percentage value is less than current iterative  
 if max\_percentage < student[1]:  
 # Assign current iterative percentage to max percentage  
 max\_percentage = student[1]  
 # Return the count and best percentage  
 return count, max\_percentage  
  
  
def main():  
 # Call calculate\_percentage() and assign result to students  
 students = calculate\_percentage()  
 # Call results(), pass "students" list as parameter, assign the returned  
 # count and best percentage to "counter" and "best\_percentage"  
 counter, best\_percentage = results(students)  
 print(f"**{**counter**}** students achieved grade A, with a best "  
 f"percentage of **{**best\_percentage**}**%.")  
  
  
# Call main to start the program  
main()

**Task 8 Evidence**

**Write your description of the fetch execute cycle below: You may include a diagram as well.**