Software Design and Development

Assessment Report

**Task 1 Evidence:**

1. **Start**
2. **Input**: Prompt the user to enter the coursework mark (out of 60).
3. **Input**: Prompt the user to enter the prelim mark (out of 90).
4. **Validate Inputs**:
   * Ensure that the coursework mark is between 0 and 60.
   * Ensure that the prelim mark is between 0 and 90.
   * If the input is invalid, display an error message and ask the user to re-enter the mark.
5. **Calculate Total Marks**:
   * Add the coursework mark and the prelim mark.
6. **Calculate Percentage**:
   * Use the formula:  
     Percentage = ((coursework mark + prelim mark) \* 100) / 150
7. **Determine Grade**:
   * If the percentage >= 70%, assign an "A" grade.
   * If the percentage >= 60% and < 70%, assign a "B" grade.
   * If the percentage >= 50% and < 60%, assign a "C" grade.
   * If the percentage >= 45% and < 50%, assign a "D" grade.
   * If the percentage < 45%, assign "No Grade".
8. **Output**: Display the percentage and grade.
9. **End**

**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 |
| Percentage Calculation | Normal Case | Coursework: 50, Prelim: 70 | Percentage: 66.67%, Grade: B |
| Percentage Calculation | Extreme Case | Coursework: 60, Prelim: 90 | Percentage: 100%, Grade: A |
| Percentage Calculation | Exceptional Case | Coursework: -5, Prelim: 70 | Error: Invalid coursework mark |
| Percentage Calculation | Exceptional Case | Coursework: 50, Prelim: 95 | Error: Prelim mark too high (should be <= 90) |
| Percentage Calculation | No Grade | |  | | --- | |  |  |  | | --- | | Coursework: 20, Prelim: 20 | | Percentage: 26.67%, Grade: No Grade |
| Percentage Calculation | Edge Case (just passing) | Coursework: 45, Prelim: 50 | Percentage: 63.33%, Grade: B |

**Task 3 and 4 Evidence:**

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

elow is a Python program that implements the design, including meaningful variable names, internal commentary, and subprograms (functions).

python

Copy code

# Function to calculate the percentage and grade

def calculate\_grade(coursework, prelim):

# Calculate the total marks and percentage

total\_marks = coursework + prelim

percentage = (total\_marks \* 100) / 150

# Determine grade based on percentage

if percentage >= 70:

grade = "A"

elif percentage >= 60:

grade = "B"

elif percentage >= 50:

grade = "C"

elif percentage >= 45:

grade = "D"

else:

grade = "No Grade"

# Return percentage and grade

return percentage, grade

# Function to validate input

def get\_valid\_input(prompt, max\_value):

while True:

try:

value = float(input(prompt))

if 0 <= value <= max\_value:

return value

else:

print(f"Please enter a value between 0 and {max\_value}.")

except ValueError:

print("Invalid input. Please enter a valid number.")

# Main program to calculate percentage and grade

def main():

# Get valid coursework and prelim marks

coursework = get\_valid\_input("Enter the coursework mark (out of 60): ", 60)

prelim = get\_valid\_input("Enter the prelim mark (out of 90): ", 90)

# Calculate percentage and grade

percentage, grade = calculate\_grade(coursework, prelim)

# Display the results

print(f"Percentage: {percentage:.2f}%")

print(f"Grade: {grade}")

# Call the main function

if \_\_name\_\_ == "\_\_main\_\_":

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 |
| calculate\_grade | Normal Test | Coursework: 50, Prelim: 70 | Percentage: 66.67%, Grade: B | Correct | Normal test passed |
| calculate\_grade | Extreme Case | Coursework: 60, Prelim: 90 | Percentage: 100%, Grade: A | Correct | Extreme test passed |
| get\_valid\_input | Invalid Input | Coursework: -5, Prelim: 70 | Error: Invalid coursework mark | Correct | Handled invalid input |
| get\_valid\_input | Invalid Input | Coursework: 50, Prelim: 95 | Error: Prelim mark too high | Correct | Handled invalid input |
|  |  |  |  |  |  |

**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.

To read the student data from files, we can modify the program to read the student names and marks from external files.

python

Copy code

# Function to read student data from files and process each student

def process\_student\_data(filename1, filename2, filename3):

with open(filename1, 'r') as f1, open(filename2, 'r') as f2, open(filename3, 'r') as f3:

names = f1.readlines()

coursework\_marks = f2.readlines()

prelim\_marks = f3.readlines()

for i in range(len(names)):

name = names[i].strip()

coursework = float(coursework\_marks[i].strip())

prelim = float(prelim\_marks[i].strip())

# Calculate grade for each student

percentage, grade = calculate\_grade(coursework, prelim)

print(f"Name: {name}, Percentage: {percentage:.2f}%, Grade: {grade}")

# Call this function for 15 students (3 files as mentioned)

process\_student\_data("Name.txt", "Mark1.txt", "Mark2.txt")

**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.

Below is the modified program that counts the number of "A" grades and finds the highest percentage in the class.

python

Copy code

# Function to count occurrences of a specific grade

def count\_occurrences(grades, target\_grade):

count = 0

for grade in grades:

if grade == target\_grade:

count += 1

return count

# Function to find the maximum percentage

def find\_max\_percentages(percentages):

max\_percentage = max(percentages)

return max\_percentage

# Main function modified to include these new features

def main():

names = ["Duncan", "Grant", "Alistair", "Fergus", "Ivor", "Isobel", "Moyra", "Kirstin", "Shona", "Beitris", "Ainsley", "Islay", "Allan", "Niven", "Katriona"]

coursework\_marks = [59, 33, 55, 10, 47, 60, 57, 45, 20, 39, 55, 25, 23, 30, 54]

prelim\_marks = [75, 80, 88, 45, 78, 90, 82, 65, 40, 59, 75, 50, 48, 60, 86]

grades = []

percentages = []

for i in range(len(names)):

coursework = coursework\_marks[i]

prelim = prelim\_marks[i]

percentage, grade = calculate\_grade(coursework, prelim)

grades.append(grade)

percentages.append(percentage)

print(f"Name: {names[i]}, Percentage: {percentage:.2f}%, Grade: {grade}")

# Count A grades

a\_count = count\_occurrences(grades, "A")

print(f"Number of A grades: {a\_count}")

# Find the maximum percentage

max\_percentage = find\_max\_percentages(percentages)

print(f"Highest percentage: {max\_percentage:.2f}%")

# Call the main function

main()

**Task 8 Evidence**

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

The fetch-execute cycle refers to the steps the CPU goes through to execute an instruction. Here’s a breakdown of the cycle:

1. Fetch: The CPU fetches the instruction from memory (RAM) that is pointed to by the program counter (PC).
2. Decode: The CPU decodes the instruction to understand what operation is to be performed (addition, subtraction, etc.).
3. Execute: The CPU performs the operation, using the appropriate registers and memory addresses.
4. Store: The result of the operation is stored back in memory or a register.
5. Repeat: The cycle repeats for the next instruction.

The Processor executes the instructions, Memory stores the data (including the program itself and variables), and Buses transfer the data between memory and the processor.

Diagram: (Simplified)

css

Copy code

[Memory] <--> [Processor] <--> [Registers & ALU]