**W3 -** PRACTICE

*Functions & Modular Design (Part 2)*

## *At the end of this practice, you should be able to…*

* Learn to **debug in VS Code** *(or another IDE)* properly
* Consolidate **Top-Down Design** and function-based refactoring
* Apply **scope**, **memory**, and **function call** concepts
* Practice **array manipulations** with functions

## *How do we structure exercises?*

We organize this practice into 4 parts:

| TOOLS | Practice the **usage of some tools** to code effectively |
| --- | --- |
| ANALYSE | **Understand** existing codes, find the **bugs** or **complete** missing gaps |
| MANIPULATE | Ensure you can **apply the theory** with some basic challenges |
| CREATE | **Express your creativity** with more complex challenges |

## *Are you lost?*

You can read the following documentation to be ready for this practice

<https://pseudocode.deepjain.com/guides/functions/>

<https://www.w3schools.com/c/c_functions.php>

**TOOLS**

**(BEFORE)** *Configure your VS Code for C programming*

* [Follow this tutorial](https://cadtedu-my.sharepoint.com/:u:/g/personal/cs_cadt_edu_kh/ETeBTE2Z6sRAjuXlrLFhnuYB7D0i6X8fRhKFJVfBdELsaw?e=NidFNb) to configure VS Code to **compile**, **run** and **debug** C code.
* *Note : You can also use another editor to debug C code, such as code Bloc, Eclipse…*

**EX 1 (The Mystery of the Wrong Sum)**

The below code has a bug, and we want you to find it, by using **the Debugger tool** !

* Create a file with the following code.
* Set a **breakpoint** in compute\_sum.
* Run the **debugger** and **step through** the loop.
* **Watch** i, sum, and arr[i].
* Fix the bug so the correct output is printed.

CODE

#include <stdio.h>

int compute\_sum(int arr[], int size) {

int sum = 0;

for (int i = 1; i <= size; i++) {

sum += arr[i];

}

return sum;

}

int main() {

int numbers[] = {2, 4, 6, 8, 10};

int total = compute\_sum(numbers, 5);

printf("The total is: %d\n", total);

return 0;

}

EXPECTED OUTPUT:

The total is: 30

ACTUAL OUTPUT:

The total is:  *an incorrect value*

#include <stdio.h>

int compute\_sum(int arr[], int size) {

int sum = 0;

for (int i = 0; i < size; i++) {

sum += arr[i];

}

return sum;

}

int main() {

int numbers[] = {2, 4, 6, 8, 10};

int total = compute\_sum(numbers, 5);

printf("The total is: %d\n", total);

return 0;

}

**ANALYSIS**

**EX 1 (Top-Down Design)**

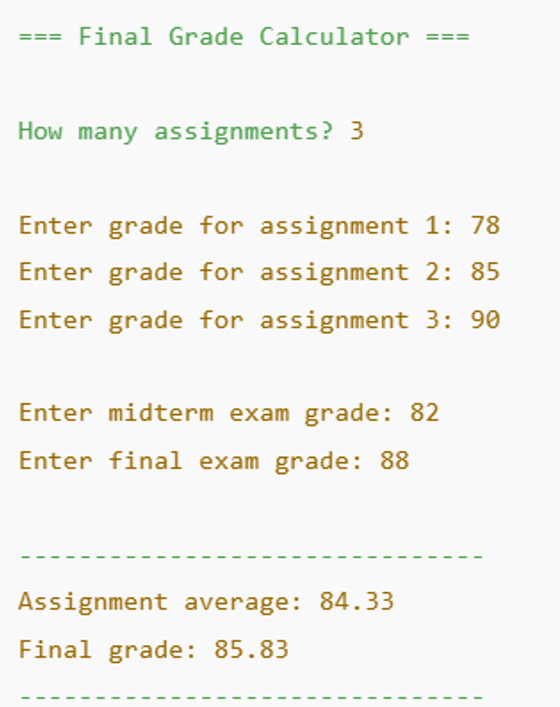
* Perform this activity **in team of 3**.
* **For this exercise, you don’t need to code !**
* You need to break **down the problem into small tasks** by defining functions.

We want to design a program to **calculate a student's final grade** based on:

* 3 assignment grades
* Midterm exam grade
* Final exam grade

The final grade is computed as follows:

* Assignments: 40%
* Midterm: 25%
* Final Exam: 35%

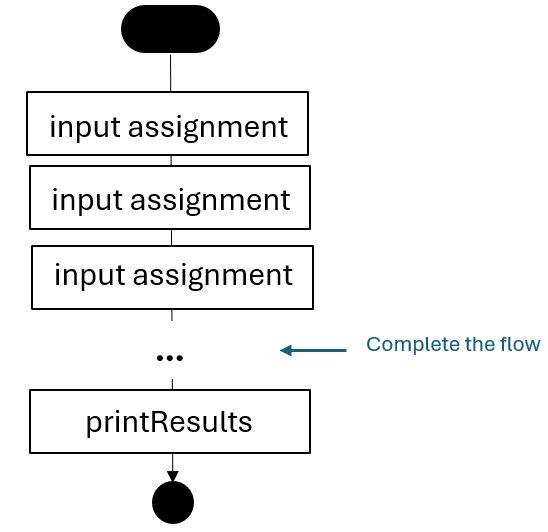


*Example of console interaction*

–**Q1 –** Complete **the below table** to describe your functions (small tasks)

| **Function** | **Parameters** | **Return** | **Description** |
| --- | --- | --- | --- |
| inputScore | void | int assignment | Input a number from 0 to 100 on the console |
| calculateAvg | int score  int size | double | to calculate the average of the assignments |
| finalGrade | double assignmentAvg  double midtermScore  double finalGradeScore | double | to calculate the final grade |
|  |  |  |  |
|  |  |  |  |
| printResults | double assigmentAverage  double finalGrade | void | Print the results on the console |

**Q2 –** Complete **the flowchart below** to describe how you will sequence the small tasks (ON PAPER)

****

**Q3 –** Now the number of assignments grade is **10**.

*What would you change in your previous functions and flowchart to comply with this need?*

**Q4 –** Present and discuss your solution to other groups (or to the whole class).

**#include <stdio.h>**

**// to input elements**

**void inPutScore(int \*score){**

**scanf("%d", score);**

**}**

**// to calculate avg of assignment**

**double calculateAvg(int score, int size){**

**return (double)score/size;**

**}**

**// to calculate the final grade**

**double finalGrade(double assignmentAvg, double midtermScore, double finalGradeScore){**

**return (double)assignmentAvg\*40/100 + midtermScore\*25/100 + finalGradeScore\*35/100;**

**}**

**//to print the result**

**void printResult(double avgAssignment, double avgFinalGrade){**

**printf("\n--- Results ---\n");**

**printf("Assignment average : %.2lf\n", avgAssignment);**

**printf("Final grade : %.2lf\n", avgFinalGrade);**

**}**

**// the main code**

**int main(){**

**int scoreAssignment[3], size = 3;**

**int scoreFinal;**

**int scoreMidterm;**

**double assignmentAvg, finalGradeAvg;**

**// to input the assignment score**

**for(int i = 0; i < size; i++){**

**printf("Enter grade for assignment %d : ", i+1);**

**inPutScore(&scoreAssignment[i]);**

**}**

**// to input the score of the midterm**

**printf("Enter grade for midterm exam : ");**

**inPutScore(&scoreMidterm);**

**// to input the score of the final**

**printf("Enter grade for final exam : ");**

**inPutScore(&scoreFinal);**

**//to calculate the assignment avg**

**for(int i = 0; i < size; i++){**

**assignmentAvg += calculateAvg(scoreAssignment[i], size);**

**}**

**// to get the value of the final grade**

**finalGradeAvg = finalGrade(assignmentAvg, scoreMidterm, scoreFinal );**

**// print result**

**printResult(assignmentAvg,finalGradeAvg);**

**return 0;**

**}**

**EX 2 (Variables scope)**

**Q1 –** Read the code and for **each marked point**, list the **visible variables** and their **scope** (*global, local, parameter)*

#include <stdio.h>

int total = 100;

void printScore(int score) {

int bonus = 5;

if (score > 50) {

int total = score + bonus;

printf("The total = %d\n", total); // Point A

}

printf("The bonus = %d\n", bonus); // Point B

}

int main() {

int score = 80;

process(score);

return 0; // Point C

}

| **POINT** | **VISIBLE VARIABLES** |
| --- | --- |
| A | Int total(local), score(local), bonus(local); |
| B | int total(global), bonus(local), score(local) |
| C | total (global)  score (local) |
|  |  |

**EX 3 (Variables scope)**

**Q1 –** Read the code and for **each marked point**, list the **visible variables** and their **scope** (*global, local, parameter)*

#include <stdio.h>

int counter = 0;

void increment() {

int counter = 10;

counter++;

printf("Inside function: counter = %d\n", counter); // Point A

}

int main() {

printf("Before function call: counter = %d\n", counter);

increment(); // Point B

printf("After function call: counter = %d\n", counter); // Point C

return 0;

}

| **POINT** | **VISIBLE VARIABLES** |
| --- | --- |
| A | Counter(global), counter(local) |
| B | Counter(global) |
| C | Counter (global) |

**Q2 –** Predict the code outputs (fill up the gaps)

Before function: counter = 0

Inside function: counter = \_\_\_\_11\_\_\_\_\_\_

After function: counter = \_\_\_0\_\_\_\_\_\_\_

**EX 4 (Refactoring code)**

* **Read** the given code.
* Identify **repeated or reusable code** blocks.
* Refactor the program by creating **functions to avoid duplication**.
* *Make sure to choose* ***meaningful function*** *names and* ***use parameters*** *where needed!*

int main() {

printf("Welcome A ! You have 2 messages.\n");

printf("Welcome J ! You have 5 messages.\n");

printf("Welcome B ! You have 0 messages.\n");

}

#include <stdio.h>

void print(char name, int numberOfMessage){

    printf("Welcome %c ! You have %d messages.\n", name,numberOfMessage);

}

int main() {

    print('A', 2);

    print('J', 5);

    print('B', 0);

return 0;

}

**EX 5 (Refactoring code)**

int main() {

// Calculate the average of 2 grades (assignment + exam)

float g1 = (15 + 40) / 2.0;

float g2 = (10 + 50) / 2.0;

float g3 = (18 + 32) / 2.0;

}

#include <stdio.h>

void average(double assigment, double exam, int g){

    double result = (assigment + exam)/2;

    printf("float g%d = %.2lf\n", g,result);

}

int main (){

    average(20, 50, 1);

    average(18, 53, 2);

    average(19, 48, 3);

    return 0;

}

**EX 6 (Refactoring code)**

#include <stdio.h>

int countA(char text[], int textCountA, int length){

for (int i = 0; i < length; i++) {

if (text[i] == 'a') {

textCountA++;

}

}

return textCountA;

}

int main() {

// Count the number of A in text 1

char text1[4] = {'a', 'b', 'b', 'a'};

int text1CountA = countA(text1, text1CountA, 4);

// Count the number of A in text 2

char text2[5] = {'b', 'a', 'a', 't', 't'};

int text2CountA = countA(text2, text2CountA, 5);

// Count the number of A in text 3

char text3[6] = {'r', 'w', 'q', 'q', 'i', 'a'};

int text3CountA = countA(text3, text3CountA, 6);

}

int main() {

    // Count the number of A in text 1

    char text1[4] = {'a', 'b', 'b', 'a'};

    int text1CountA  = 0;

    for (int i = 0; i< 4; i++) {

        if (text1[i] == 'a')  {

            text1CountA++;

        }

    }

    // Count the number of A in text 2

    char text2[5] = {'b', 'a', 'a', 't', 't'};

    int text2CountA  = 0;

    for (int i = 0; i< 5; i++) {

        if (text2[i] == 'a')  {

            text2CountA++;

        }

    }

    // Count the number of A in text 3

    char text3[6] = {'r', 'w', 'q', 'q', 'i', 'a'};

    int text3CountA  = 0;

    for (int i = 0; i< 6; i++) {

        if (text3[i] == 'a')  {

            text3CountA++;

        }

    }

}

**MANIPULATE**

**EX 1 (Track Student Grades)**



You are given an array of student grades (integers from 0 to 20).

*Your task is to write a program that uses functions to analyze and manipulate those grades.*

**Q1** - Implement the function **countAboveThreshold** as follows:

| FUNCTION NAME | countAboveThreshold | |
| --- | --- | --- |
| FUNCTION DESCRIPTION | Returns how many grades are strictly above the given threshold | |
| PARAMETERS | int [] | The list of grades |
| int | The size of this array |
| int | The threshold |
| RETURN | int | How many grades are strictly above the given threshold |

*EXAMPLES*

| PARAMETERS | RETURN |
| --- | --- |
| {9, 8, 19, 13, 15}  5  12 | 3 (3 scores are above 12) |
| {14}  1  14 | 0 (0 scores are above 14) |

**#include <stdio.h>**

**int countAboveThreshold(int listOfGrade[], int sizeOfTheArr, int threshold){**

**int theGradeAbove;**

**for(int i = 0; i < sizeOfTheArr; i++){**

**if (listOfGrade[i] > threshold){**

**theGradeAbove++;**

**}**

**}**

**return theGradeAbove;**

**}**

**int main() {**

**int gradeAbove, listOfGrade[] = {9, 8, 19, 13, 15};**

**gradeAbove = countAboveThreshold(listOfGrade, 5, 12);**

**printf("%d\n", gradeAbove);**

**return 0;**

**}**

**Q2** - Implement the function **addBonus** as follows:

| FUNCTION NAME | addBonus | |
| --- | --- | --- |
| FUNCTION DESCRIPTION | Add a bonus to every grade  *Warning : the final grade cannot be above 20* | |
| PARAMETERS | int [] | The list of grades |
| int | The size of this array |
| Int | The bonus |
| RETURN | void | *Nothing. Changes are made to the array passed by reference* |

*EXAMPLES*

| PARAMETERS | UPDATED GRADES |
| --- | --- |
| { 9, 8, 19, 13, 15 }  5  2 | { 11, 10, 20, 15, 17 } |
| {20, 20 , 20}  3  5 | {20, 20 , 20} |

#include <stdio.h>

void addBonus(int listOfGrade[], int sizeOfTheArr, int theBonus){

for(int i = 0; i < sizeOfTheArr; i++){

listOfGrade[i] += theBonus;

if(listOfGrade[i] > 20){

listOfGrade[i] = 20;

}

}

}

int main(){

int theListOfGrade[5] = { 9, 8, 19, 13, 15 };

addBonus(theListOfGrade, 5, 2);

for(int i = 0; i < 5; i++){

printf("%d ", theListOfGrade[i]);

}

return 0;

}

**Q3** - Implement the function **computeAverage** as follows:

| FUNCTION NAME | computeAverage | |
| --- | --- | --- |
| FUNCTION DESCRIPTION | Calculates and returns the average grade. | |
| PARAMETERS | int [] | The list of grades |
| int | The size of this array |
| RETURN | int | The average grade |

*EXAMPLES*

| PARAMETERS | UPDATED GRADES |
| --- | --- |
| { 10, 12, 10, 12 }  4 | 11 |
| {20, 20 , 20}  3 | 20 |

#include <stdio.h>

int computeAvg(int listOfGrade[], int sizeOfTheArr){

int avgGrade = 0;

for (int i = 0; i < sizeOfTheArr; i++){

avgGrade += listOfGrade[i];

}

return avgGrade/sizeOfTheArr;

}

int main(){

int theListOfGrade[4] = { 10, 12, 10, 12 }, avgGrade;

avgGrade = computeAvg(theListOfGrade, 4);

printf("%d\n", avgGrade);

return 0;

}