Contents

[Assessment sheet 1 3](#_Toc71897962)

[Library stdio.h: 3](#_Toc71897963)

[Main 3](#_Toc71897964)

[Variable 3](#_Toc71897965)

[Printf 3](#_Toc71897966)

[Scanf 3](#_Toc71897967)

[If statement 3](#_Toc71897968)

[Clear buffer: 4](#_Toc71897969)

[Increment/decrement 5](#_Toc71897970)

[(D-B) 5](#_Toc71897971)

[Code 5](#_Toc71897972)

[Console 6](#_Toc71897973)

[(A) 6](#_Toc71897974)

[Switch statement 6](#_Toc71897975)

[Code 6](#_Toc71897976)

[Console 7](#_Toc71897977)

[Assessment sheet 2 7](#_Toc71897978)

[Loops: 7](#_Toc71897979)

[(D-B) 8](#_Toc71897980)

[Code: 8](#_Toc71897981)

[Console: 9](#_Toc71897982)

[(A) 9](#_Toc71897983)

[Code: 9](#_Toc71897984)

[Console: 10](#_Toc71897985)

[Assessment sheet 3 10](#_Toc71897986)

[(D-B) 10](#_Toc71897987)

[Library math.h: 10](#_Toc71897988)

[do…while loop 10](#_Toc71897989)

[Code: 11](#_Toc71897990)

[Console: 12](#_Toc71897991)

[(A) 13](#_Toc71897992)

[Code: 13](#_Toc71897993)

[Console: 14](#_Toc71897994)

[Assessment sheet 4 15](#_Toc71897995)

[Library stdlib.h: 15](#_Toc71897996)

[Library ctype.h: 15](#_Toc71897997)

[(D-B) 15](#_Toc71897998)

[Code: 15](#_Toc71897999)

[Console: 17](#_Toc71898000)

[(A) 18](#_Toc71898001)

[Code: 18](#_Toc71898002)

[Console: 19](#_Toc71898003)

[Assessment sheet 5 20](#_Toc71898004)

[Array: 20](#_Toc71898005)

[Terminator: 20](#_Toc71898006)

[sizeOf: 20](#_Toc71898007)

[(D-B) 20](#_Toc71898008)

[Code: 20](#_Toc71898009)

[Console: 22](#_Toc71898010)

[(A) 23](#_Toc71898011)

[Getchar: 23](#_Toc71898012)

[Assessment sheet 6 24](#_Toc71898013)

[(D-B) 24](#_Toc71898014)

[(A) 25](#_Toc71898015)

[Assessment sheet 7 25](#_Toc71898016)

[Functions: 25](#_Toc71898017)

[Global variables: 26](#_Toc71898018)

[(D-B) 26](#_Toc71898019)

[Code: 26](#_Toc71898020)

[Console: 28](#_Toc71898021)

[(A) 28](#_Toc71898022)

[Code: 28](#_Toc71898023)

[Console: 30](#_Toc71898024)

[Assessment sheet 8 30](#_Toc71898025)

# Assessment sheet 1

### Library stdio.h:

When starting a C program, we need to include libraries that already provide macros, type definitions and functions for tasks that we will need. The first and basic one that we will use in this program will be the standard library, stdio.h (standard inputs and outputs).

Main:

Main is one of the already known functions in c, serves as the starting point for program execution, which means, it is where the computer will read the program. As default, the main function should always return 0 in the end.

Variable:

A variable is a storage area identified by type, which determines the size and layout of the variable’s memory, the range of values that can be stored in that memory space and the set of operations that can be applied to the variable. Our program can manipulate the variable value and to make it easier for the user to identify each specific allocated space of memory, we give each variable a different name. A variable are different types of variables:

* Int – (integer) - The most natural size of integer for the machine
* Float- (floating point number) - A single-precision floating point value
* Double – (a bigger Float) - A double-precision floating point value
* Char – (character) A single character from the keyboard
* Void – Represents the absence of type.

Printf:

To print the result of our program or any message to the user we will use the function printf (printf(“%’variabletype’, ‘variablename’ list of variables order).

Scanf:

To have the program to read a value from the user we will use the function scanf (scanf(“%’variabletype’, &’variablename’)) which will take the user input and store it in the variable place.

&:

The ‘&’ operator is used to get the address of a variable which means, when we scan a value, we want to store the value in one variable, and for that we get the variable address by with the ‘&’ (example in scanf explanation above). Basically since a variable is a storage area, we are setting that area value as the value scanned.

If statement:

The if statement evaluates the test expression inside the parenthesis, if the expression is evaluated to true, statements inside the body of ‘if’ are executed and if the test expression is evaluated to false, statements inside the body of ‘if’ are not executed. For more test expressions we can add ‘else if’ and as last option we can add ‘else’ as something that the program will run if all the test expressions are false.

Clear buffer:

When scanning multiple chars, the program can behave in a different way that we want, after we can the first char and press enter, the most probable thing to happen is that the next char scanned will be fulfilled by “enter” instead of the char we wanted. To prevent this to happen we use the following code after we scan:

int ch;

while ((ch = getchar()) != '\n' && ch != EOF);

If condition

Is true

**If (condition)**

If condition

Is false

If condition

Is true

**Else if (condition)**

If condition

Is false

**Else if (condition)**

If condition

Is true

If condition

Is false

**else code**

Increment/decrement operators:

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one. For example: If i=1, i++ will be 2, or i- - will be equal to 0.

(List of e.g.)

a+=1 <=> a = a + 1

b-=1 <=> b = b - 1

c \*= 1 <=> c = c \* 1

c /= 1 <=> c = c / 1

## (D-B)

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, Teams

Description automatically generatedCode:

Console:

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

## (A)

Switch statement:

The switch statement allows us to execute one block among many alternatives. The expression provided in the switch statement will be compared with each block case and if they match, the computer will read that block and end when it reads a break instruction. It works the same way as an if statement but should be easier to read and write.

Graphical user interface, text, application

Description automatically generated with medium confidenceCode:

Console:

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

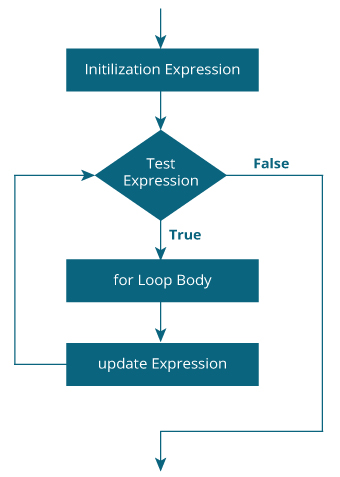
Description automatically generated

# Assessment sheet 2

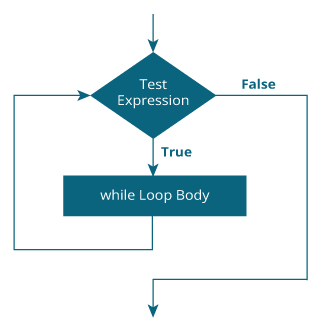
### Loops:

In C we have three types of loops:

1. for loop
   1. The initialization statement is executed only once.
   2. Then, the program evaluates the test expression, if it is evaluated as false the for loop is terminated.
   3. If the test expression is evaluated as true, the statements inside the body of the loop are executed and the update expression is updated.
   4. Step b and c will keep repeating until the test expression is false, terminating the loop.



1. while loop
   1. The while loop evaluates the test expression inside the parenthesis.
   2. If the test expression is true, the statements inside the body of while loop are executed. Then, the test expressions are evaluated again.
   3. The process will continue until it is evaluated as false, ending the loop.



## (D-B)

Code:

Graphical user interface, text, application

Description automatically generated

Console:

Text

Description automatically generatedGraphical user interface, text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

## (A)

Text

Description automatically generated with medium confidenceCode:

Console:

Text

Description automatically generatedGraphical user interface, text

Description automatically generatedText

Description automatically generatedGraphical user interface, text

Description automatically generated

# Assessment sheet 3

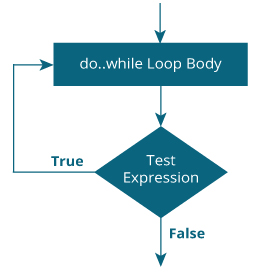
## (D-B)

### Library math.h:

This library gives us instant access to some more mathematical functions and values, the one we will be using is the value of PI (M\_PI). All the functions available in this library take double as an argument and return double as the result.

### do…while loop

* 1. The body of do…while loop is executed once. Only then, the test expression is evaluated.
  2. If the test expression is true, the body of the loop is executed again, and the test expression is evaluated.
  3. The process will continue until it is evaluated as false, ending the loop.



Graphical user interface, text, application

Description automatically generatedCode:

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedConsole:

## (A)

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedCode:

Graphical user interface, text, application, email

Description automatically generated

Console:

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

# Assessment sheet 4

### Library stdlib.h:

We used stdlib.h to get the absolute value from an operation. This header includes four variables and different macros and functions.

The variables:

size\_t : This is the unsigned integral type and is the result of the sizeof keyword.

wchar\_t: This is an integer type of the size of a wide character constant.

div\_t: This is the structure returned by the div function.

idiv\_t: This is the structure returned by the ldiv function.

### Library ctype.h:

We used ctype library to get all scanned chars in capitals with the function “toupper”. This library is useful for testing and mapping characters.

## (D-B)

Graphical user interface, text, application, email

Description automatically generatedCode:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceConsole:

## 

## (A)

Text

Description automatically generated with medium confidenceGraphical user interface, text, application, email

Description automatically generatedCode:

Graphical user interface, text, application, email

Description automatically generated

I used “toupper” function so that all scanned chars (letters) would become uppercase, in this way I did not need to use an or condition in If statements.

Text

Description automatically generatedText

Description automatically generatedConsole:

# Assessment sheet 5

### Array:

To store multiple values in one variable we use arrays, which the compiler will allocate memory after initializing the array and save that memory space for that array elements.

To initialize an array we can either declare the array size doing for example ‘int array[5]’ , or we can declare its elements and the compiler will automatically know the space needed for the array. (int array [] = {1, 3, 5, 6}, the array will be size 4).

### Terminator:

In cases that we have big arrays, and we do not know the size of it, we can add what it is called a terminator, in the first position after the last position used of the array, we had a value that would not be used in the array, in an all-positive value array we could add a -1. In this case when doing loops, we can look to end the loop when the loop matches the value -1.

e.g.:

int array[ ] = { 3, 5, 1, 6, -1};

while(array[count] != -1){

(…code…)

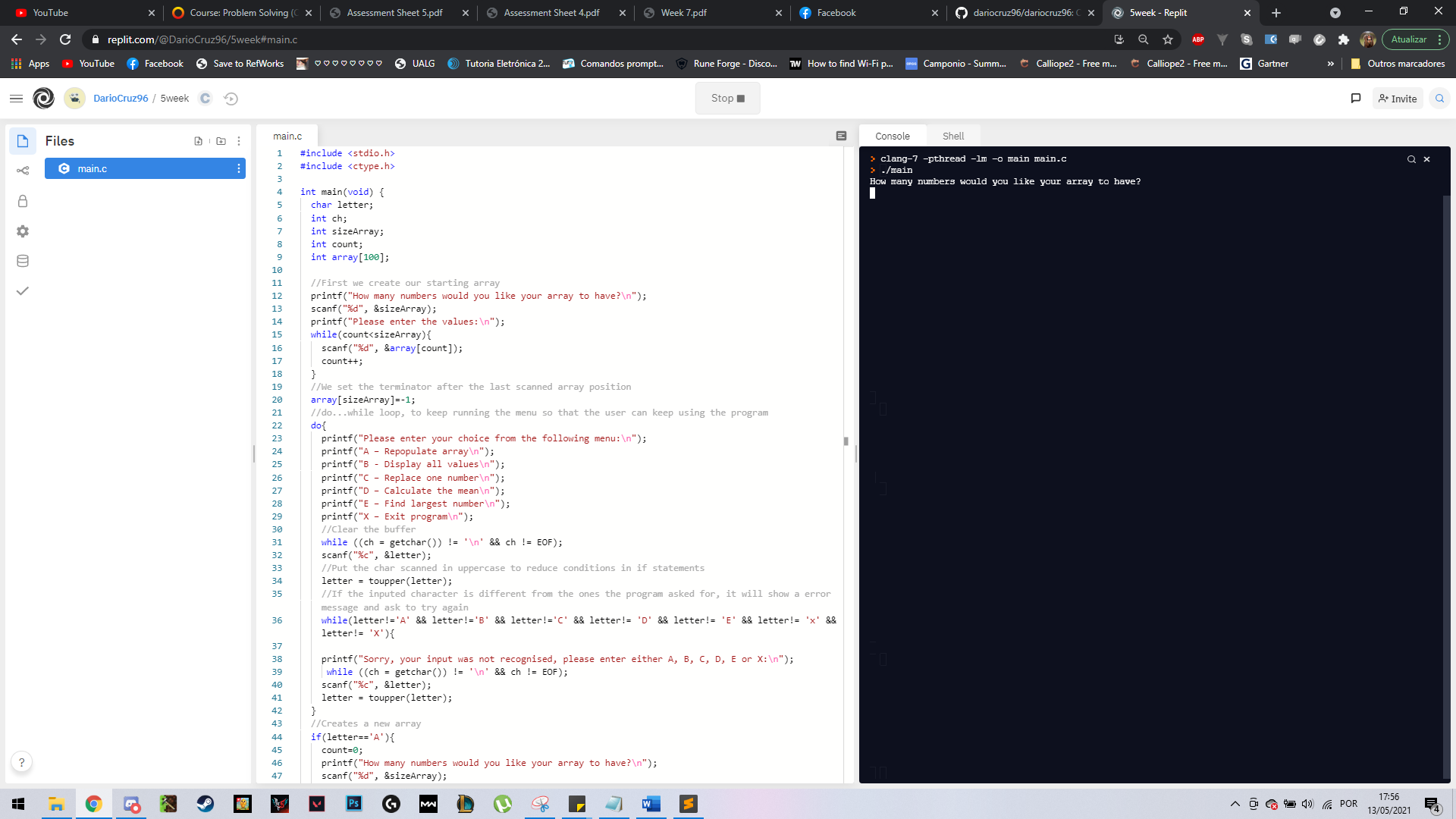
}

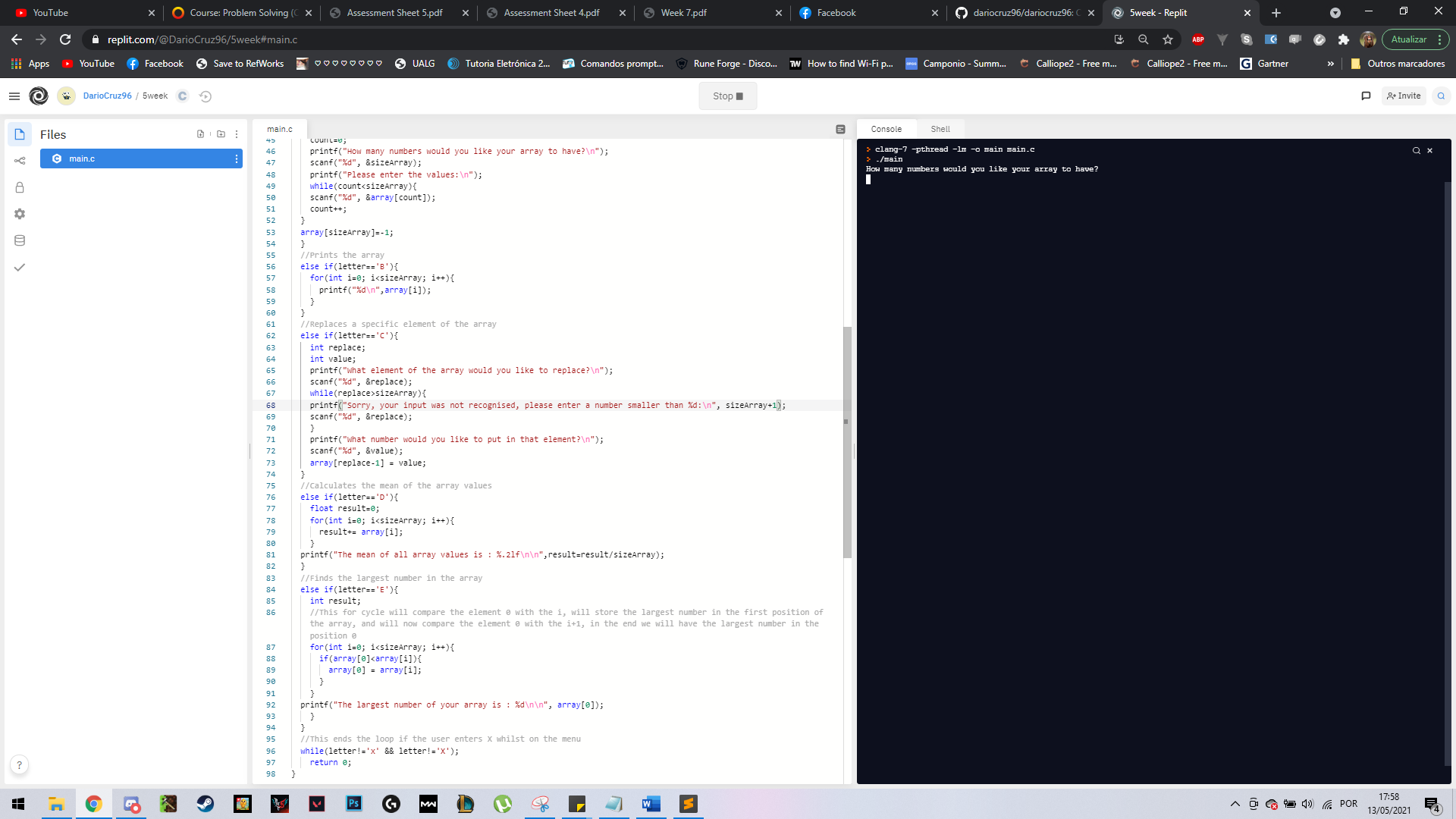
### sizeOf:

This function is used to get the size of the operant. It can be applied to all data (integers, floats, doubles,etc)

## (D-B)

Code:





Console:

Text

Description automatically generated

Text

Description automatically generated

## (A)

Different approach: Instead of using a terminator, we can check for the number of elements of the array by doing:

n = sizeof(array);

printf(“The size of the array is %d”, n/sizeof(int));

In this program we asked the user the number of elements that they wanted to have in the array, and with that we got the number of elements of the array. But if didn’t ask the user for that, and instead, we just asked him for the elements that he wanted to have inside the array, with that expression we could get the size of the array and we could use that as a limit in the loops.

### Getchar:

This function returns the character entered in the console by the user. Instead of doing a scanf and store the char value we can just use getchar and store the value to unassigned “name”.

Text

Description automatically generatedText

Description automatically generatedWe can see in the code that instead of having to clear the buffer and have a scanf we can just have a getchar instead.

# Assessment sheet 6

In this program we used a temporary variable, we called it “temp” and we used it to switch array elements.

## (D-B)

Graphical user interface, application

Description automatically generated

## Graphical user interface, text, application Description automatically generated(A)

As a different way to sort an array of integers I choose the insertion sort algorithm, which follow the next steps:

1. We assume the first element as sorted, taking the second element of the array and store it separately in “key”. Then we compare the “key” with the first element and if the first element is greater than “key”, the “key” is placed before the first element.
2. We compare the third element with the elements on the left placing it behind the element smaller than itself. If there is no smaller element, then we just place it in the beginning of the array.
3. Repeat step 2 until all array is sorted.

# Assessment sheet 7

### Functions:

A function is a block of code that runs using parameters (variables scanned or obtained while the program in running) to do computing and usually returns some value in the end even though it could return nothing and show some outputs instead. Usually, it is used to organize code and to avoid having redundant lines of coding, since we can just keep calling the same function when needed instead of having to write all over again. Besides main which is the only function that automatically runs in the compiler, all other functions outside main will have to be called in main, so the program will not have to run all functions unless we need them.

Global variables:

Usually, we declare variables inside the function and use it as we need and after running the function the variable will not exist in the rest of the program, but if we want to use the same variable all along the program, we can declare it outside the functions and use it when we need it. We can set a value when starting the variable, but the predominant value will be the one set inside the function.

## (D-B)

Graphical user interface, text, application, email

Description automatically generatedCode:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Console:

Text

Description automatically generated

## (A)

Code:

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Console:

Text

Description automatically generated

# Assessment sheet 8

I started this Problem Solving course with just some basic of C from self-learning over the past two years so me and C already met before. My expectations were to learn how to have variables interacting with each other and how to develop coding that could help me to solve programming problems.

Since I already knew some basics, I got familiar when we started talking about variables (integers, doubles, floats, char) even though I thought that it could get my mind in circles, following the online lessons with the examples given by Darren step by step, seeing the doubts of my colleges and my questions answered with live coding examples helped me a lot to know where should I focus my attention when getting variables together, and I would say that something that looks simple but seems to me really important is giving the variable a name that reminds us on what are we going to use that variable for ( in the first assessment we used variables mileage and cost) and we can organize our logic really quickly because those names just came naturally when we needed them. One more thing that I think every programmer should do and was asked in every assessment was to explain the code, we could either explain it in the portfolio or comment directly in the code, so, I started having comments at the same time I was coding and in the end, I had a coding file with a explained instructions basically, because I could just take a look at a line of code and instead of reading a full “printf” it would say right above it: “//This line prints the area of the circle”(assessment 3 – menu option B).

Once we got to loops I felt comfortable with it, I had already worked on simple coding with the weekly tasks and the assessment 1 and after doing the weekly tasks with loops and reading the notes, I got it in my head that it was just repeating simple coding lines and that’s what a loops does, it just repeats as many times as we want a part of our code and with loops I was able to reduce the size of assessment 2 from maybe 90 lines to 36 and I can see this in a big scale, if we say we reduced 60% of coding lines, in 900 lines we could only have 300 and so on.

Besides what we were taught in class and asked for in the assessments, we had the option to go further and do self-research for a better grade which I thought I could challenge myself and go for it, and I would say it really paid off. At first it can really get confusing with so many online places to search for and so much information that I got confused in the beginning, I started by searching different approaches to if statements which I found switch statements and having two different ways of programming the same thing allowed me to learn different approaches to solving a problem and I was able to see the possibilities for the future where I can use the one that suits the best of my needs.

Then we started arrays. Going from using variables to arrays got a bit confusing because we used different variables to store different values and now, we wanted to store different values in just one variable. On the assessment 5 we worked with arrays and I would say that from my experience this assessment really helped me to better understand arrays because the problem consisted in repopulate the array, display all values, replace one number, and calculate the mean, so these four options made me work in:

* Creating a new array having to give values to each element of the array
* Creating a loop so show me all the elements that I got in the array
* Selecting a specific element on the array and change its value
* Interacting with all the elements in the array, adding them all into one temporary variable and used that to calculate what we want (the mean in this case)

I had a few minutes of brainstorming before figuring it out on how to repopulate the array but ended up deciding that I could just go by going from the first element of the array and defining values for each and in the end the array had been repopulated. In the end of these assessment, I felt confident to work with arrays in the future (like an array studentClass where the students are organized by age) has I can see how having a variable with different values can be useful instead of different values every time.

In the end but not the least, we reached functions which I think it felt right to be in the end and should have all our focus. Until that moment, if we wanted to do a longer program but within the same focus, we for sure would be doing the same loops and the same algorithms all over again and that would not be just time consuming but memory consuming as well, and the functions came to solve that. Creating a function outside the main and having the algorithm ready to use whenever we just by calling it in main it is a time saver, and the code will look much cleaner.

After all the coding I feel more confident to work on harder problems and I can easily read the code and identify where the errors are, which by experience I would say they always are in the smallest places (always look for ‘;’!) and I think that the better way to improve our weaknesses is practicing and see other persons logic when coding.