

# Software Development for Robotics

## Assignment 1

Miguel Oteo Marín (s3173860), Álvaro Redondo Arroyo (s3245888)

February 2024

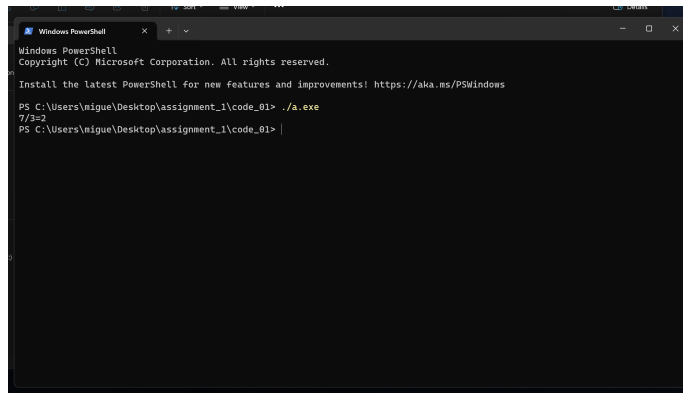
## 1 File structure and build process

### 1.1 Compile and run simple divide program

The code performs the division of two integers, the first one being the numerator stored in variable  $a$  and the second one being the denominator stored in variable  $b$ , then a function is called

```
/**
 * @brief Computes a division of two numbers
 *
 * @param int numerator
 * @param int denominator
 * @return int division
 */
int divide(const int num, const int den)
{
    return num/den;
}
```

This function receives as parameters both variables and computes the division, then returns the value as a int, in this case the solution is stored in variable  $c$ . Finally a printing on the console is called to print the result in the form  $num/den = result$ .



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\unique\Desktop\assignment_1\code_01> ./a.exe
7/3=2
PS C:\Users\unique\Desktop\assignment_1\code_01> |
```

Result from code

## 1.2 Separate into two source files

The main file *main.cpp* contains the main method which is executed when the program is ran. In addition the new header file was included with *include* to reference the function declares in the other file *divide.cpp*.

```
int main(void) // Starts main loop
{
    // Variable declarations
    int a = 7;
    int b = 3;

    // Compute the division
    int c = divide(a, b);

    // Print the result
    cout << a << "/" << b << "=" << c << endl;
    return 0;
}
```

The *divide.cpp* contains the the function together with the header file, included with *include*, for proper linking between the files

```
#include "header.h"

int divide(const int num, const int den) {
    return num/den;
}
```

The last file is the header file which links the declaration and the call of the function which are in different files.

```
#ifndef HEADER_H
#define HEADER_H

int divide(const int, const int);

#endif
```

Regarding the compiling process ...

### 1.3 Compile error

The example shows the compile error of the common mistake of missing a semicolon. It occurs during the compilation phase, where the compiler checks the syntax and semantics of the code. The compiler returns the error expecting a semicolon and the line where the error occurred.

```
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_112> g++ -o program main.cpp divide.cpp
divide.cpp: In function 'int divide(int, int)':
divide.cpp:17:19: error: expected ';' before '}' token
    return num/den
                   ^
}
^
```

Result from a compile error

### 1.4 Link error

One common error in linking could be the missing of the include in one of the .cpp files which causes a linking error. Linking errors in C++ typically occur during the linking phase of the compilation process when the compiler is trying to combine object files and libraries into an executable. These errors often indicate that the compiler can't find the definitions for functions or variables that were declared but not defined in the code. In this case the problem will be when calling a function from another file which was declared in the missing includes.

```
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_112> g++ -o program main.cpp
C:\Strawberry\c\bin\../lib/gcc/x86_64-w64-mingw32/8.3.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:\Users\miguel\AppData
a\Local\Temp\ccncVQn.o:main.cpp:(.text+0x24): undefined reference to 'divide(int, int)'
collect2.exe: error: ld returned 1 exit status
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_112> |
```

Result from a linking error

### 1.5 Runtime error

One common run time error is the division by 0 error. Runtime errors occur when a program is running and something unexpected happens that prevents the normal execution of the program. These errors are often the result of logical mistakes, memory issues, or unforeseen conditions during execution.

```
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_111> g++ -o program main.cpp
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_111> ./program.exe
PS C:\Users\miguel\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_111> |
```

Result from a runtime error

## 2 BMI Calculator

### 2.1 Section a)

```

#include <iostream>
#include "header.h"
using namespace std; // Standard namespace

int main() // Starts main loop
{
    // Declaration of variables
    float weight = 0;
    float height = 0;

    printInfo();

    cout << "Insert your height in m: ";
    cin >> height; // Store console input in variable height

    cout << "Insert your weight in Kg: ";
    cin >> weight; // Store console input in variable weight

    evaluateAndPrintBMI(weight, height);
    return 0;
}

#ifndef HEADER_H
#define HEADER_H

void evaluateAndPrintBMI(const float, const float);
void printInfo();

#endif

```

The following code represents the main function of the program. Two float variables are used to store the height and weight values, which is necessary due to the decimal part of the data being stored. After that, the console asks the user for the respective values by using the console (cout command) and stores the typed value in the variables thanks to the cin command. Finally, the function evaluateAndPrintBMI is called, and the main function ends. It is important to highlight that the values are given to the function as constant, to avoid changes in the variables during the BMI calculation.

## 2.2 Section b)

```

/**
 * @brief Computes the BMI and prints it
 *
 * Computes the BMI using the weight and height of the person and
 * prints the following results based on the BMI
 *
 * BMI TABLE
 * You are underweight: less than 18.5
 * Your weight is normal: between 18.5 and 24.9
 * You are overweight: between 25 and 29.9
 * You are obese: 30 or greater
 *
 * @param float weight The weight of the person in [Kg]

```

```

* @param float height The height of the person in [m]
* @return void
*/
void evaluateAndPrintBMI(const float weight, const float height)
{
    float BMI = weight/pow(height, 2);

    if(BMI < 18.5)
    {
        cout << "You are underweight";
        return;
    }
    if(BMI >= 18.5 && BMI <= 24.9)
    {
        cout << "Your weight is normal";
        return;
    }
    if(BMI >= 25 && BMI <= 29.9)
    {
        cout << "You are overweight";
        return;
    }
    if(BMI > 30)
    {
        cout << "You are obese";
        return;
    }

    cout << "BMI could not be defined";
}

```

The function takes as an input the two constant float values obtained by the main function, and then uses the formula for BMI calculation to obtain a value for the BMI ratio. After that, and if else block is used to check whether the user is underweight, normal weight, overweight, or obese, by comparing the values to predefined constant values that delimit the different BMI ranges. When one of the conditions is met, the function will print on the console the corresponding classification message before ending its execution and returning to the main code.

## 2.3 Section c)

```

/**
* @brief Print BMI table
*
* BMI TABLE
* Underweight: less than 18.5
* Normal: between 18.5 and 24.9
* Overweight: between 25 and 29.9
* Obese: 30 or greater
*
* @return void
*/
void printInfo()
{

```

```

    cout << "BMI-VALUES\n"
           "Underweight: --less-than-18.5\n"
           "Normal: -----between-18.5-and-24.9\n"
           "Overweight: ---between-25-and-29.9\n"
           "Obese: -----30-or-greater\n\n";
}

```

The function `printInfo` allows the user to call it from the main to display on the console the related BMI information that was mentioned before. It will print a single message through the console, explaining the ranges of the BMI classification.

### 3 Classical only: Heart Rates

Using the standardized naming when creating a class file, the file name should be the class name initialized with a capital letter `Human.cpp`. Using encapsulation of the class, data the attributes are declared as private so they are only accessible within the object of the class, below the declaration of the default constructor to declare new objects of the class. After that the declaration of the class methods starting for the getters and setter to access and edit the attributes of the class and follow by the other methods.

```

#include <time.h>
#include <iostream>
using namespace std; // Standard namespace

/**
 * @class Human
 * Defines human properties and methods
 */
class Human
{
    private: // Parameters of the class
        string firstName;
        string lastName;
        int day;
        int month;
        int year;

    public: // Class methods

        // Default constructor
        Human(string fName, string lName, int D, int M, int Y)
        {
            firstName = fName;
            lastName = lName;
            day = D;
            month = M;
            year = Y;
        }

        /**
         * Getters
         */

```

```

string getFirstName()
{
    return firstName;
}

string getFullName()
{
    return firstName + "-" + lastName;
}

std::string getBirthDate()
{
    return to_string(day) + "-" + to_string(month) + "-" +
        to_string(year);
}

/**
 * Setters
 */
void setFirstName(string fName)
{
    firstName = fName;
}

void setLastName(string lName)
{
    lastName = lName;
}

/**
 * Methods
 */

/**
 * @brief Inserts the birth date of the patient
 *
 * Checks if the day and month are within the correct
 * logical range, days (From 1 to 31) and month
 * (From 1 to 12), if the conditions are met the new date
 * is stored if not the operation is ignored
 *
 * @param D day of the new birth date
 * @param M month of the new birth date
 * @param Y year of the new birth date
 * @return void
 */
void setBirthDate(int D, int M, int Y)
{
    // Check if days and months are valid
    if (D > 0 && D < 31 && M > 0 && M < 13)
    {
        day = D;
        month = M;
        year = Y;
    }
}

```

```

/**
 * @brief Compute the age in years
 *
 * Checks the computer's current time and computes the
 * difference between the birth date year and the current
 * year to know the approximate age in years.
 *
 * @return int The age in years
 */
int getAge()
{
    // Get the current time of the computer
    time_t currentTime = time(NULL);
    struct tm *timeNow = localtime(&currentTime);

    // Extract the year from it
    int currentYear = timeNow->tm_year + 1900;

    // Compute the difference
    return currentYear - year;
}

/**
 * @brief Compute the maximum BPMs of the patient
 *
 * Computes the maximum BPMs of the patient by
 * subtracting the current age of the patient in year to
 * the value of 220
 *
 * @return int The maximum BPMs of the patient
 */
int calculateMaximumHeartRate()
{
    const int maxRate = 220;
    return maxRate - getAge();
}

/**
 * @brief Computes the target BPMs
 *
 * Computes the target BPMs based on the maximum BPM of the
 * patient, then returns the minimum and maximum values of
 * the range
 *
 * @return pair The min and max value of the target BPMs
 */
pair<float, float> calculateTargetHeartRates()
{
    int maxRate = calculateMaximumHeartRate();
    return make_pair(maxRate*0.5, maxRate*0.85);
}
};

```



## 4 Questions

### 4.1 Assuming that we compile and link all file with one single command (Section B.3)

If you forgot to "include" a header file in a .cpp file, you would likely get a compiler error. Header files contain declarations that inform the compiler about the existence of functions, classes, or variables defined in other files. If you forget to include a necessary header file, the compiler won't know about these declarations, leading to a compilation error. The error message will typically indicate an undeclared identifier or something similar.

If you omitted one .cpp file from the compile command you would get a linker error. Each .cpp file is compiled independently into an object file (.o or .obj), and these object files are then linked together to create the final executable. If you omit one .cpp file from the compilation command, the linker will not find the necessary functions or symbols during the linking stage. This results in a linker error, indicating unresolved external symbols or functions. The error message will typically state that a particular function or symbol is defined but not found.

### 4.2 Given file a.cpp containing function A() in which B() is called and file b.cpp containing function B() in which A() is called.

When you have a circular dependency between two functions or classes in different files, you typically use forward declarations and header files to resolve the issue. If you have a.cpp calling function B() from b.cpp, and b.cpp calling function A() from a.cpp, you'll need to use a forward declaration to inform the compiler about the existence of the functions before they are defined.

### 4.3 Why do you have to include iostream in your code?

The library iostream is the standard input output stream, this means it will allow the program to access the input and output of the computer.

### 4.4 What is the difference between private and public access specifiers in a C++ class?

Private attributes in a class can only be access within the class object itself but will not be accessible from outside the object. To access private attributes values then the getter methods must be implemented. If the variables want to be edited then the setter methods must be implemented.

#### 4.5 How can a program use the standard library class string without using a using namespace directive? What is problematic about putting a using namespace directive at the start of your code?

To use the standard library class string without using a using namespace directive, you can qualify the string class with its namespace, which is std. This involves using the scope resolution operator (::) to specify that string is part of the std namespace. The problem with putting a using namespace directive at the start of your code is that it brings all the names from the specified namespace into the current scope. This can lead to naming conflicts and ambiguity if multiple namespaces define entities with the same name.

#### 4.6 Give the output of the code shown in Listing 1.2 and explain.

The reason behind the change in value of the variable a is related to the way it is initialized, at the beginning the variable is set as a global value of 20. Then inside the main() it is changed to a value of 10. Although, since it has been changed inside the function main() the new value of 10 is only true in the main() function and not in the function function() which still has the value 20 from the global declaration. This causes that when the first print is called in function() the printed value is the global one. When then the print is called in main() the value is 10.

```
PS C:\Users\maigue\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_140> g++ -o program main.cpp
PS C:\Users\maigue\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_140> ./program.exe
20
10
PS C:\Users\maigue\OneDrive\Documents\GitRepos\Software-Development\assignment_1\code_140> |
```

Result from running