Programming 2:
Assignment 1

Jibbe Andringa
s2336219
Group 12

February 10, 2023

1 Assignment 1.1

1.1 Assignment 1.1.1

The given program is used to perform integer devisions. The program defines a function divide that takes two integers, num and den, as arguments and returns the result of num/den. The function main is the entry point of the program, where it declares two integers a and b, and assigns the result of the call to divide to c. Finally, it prints the result to the console using the cout statement, which outputs the values of a, b, and c as a string. When you run the program, it will produce the following output:

```
7/3=2
```

This means that the division of a by b (which are 7 and 3, respectively) is equal to 2, and the program has successfully performed the integer division and printed the result to the console.

1.2 Assignment 1.1.2

Dividing (pun not intended) the program across separate files can be done as follows: first we make a Header file, which contains the definition of the divide function like below.

```
#ifndef ASSIGNMENT1_DIVIDE_H
#define ASSIGNMENT1_DIVIDE_H

int divide(const int num, const int den);

#endif //ASSIGNMENT1_DIVIDE_H
```

Next, the implementation of the divide function is defined in the divide source file by first including the header file and then giving the implementation as follows:

```
#include "divide.hpp"

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

Lastly, the main function remains in the main source file as this is still the main entry of the program. the divide function can be used by including the header file which contains the divide function. The implementation of the main function remains the same:

```
#include <iostream>
#include "divide.hpp"

int main(void) {
    int a = 7;
    int b = 3;
    int c = divide(a, b);
    std::cout << a << "/" << b << "=" << c << std::endl;
    return 0;
}
```

Before this program can be executed, it first needs to be compiled and linked. Compiling the source files will create object (.o) files. This can be done using the following command:

```
$ g++ main.cpp divide.cpp -c
```

Where g++ is the compiler, main.cpp and divide.cpp are the files that need to be compiled and -c is the argument that tells the compiler to only compile. Linking the object files can be done in a similar way. Linking object will give us an executable that will run the program. The command for this is as follows:

```
$ g++ main.o divide.o -o Divide
```

Where main.o and divide.o are the object files that need to be linked, -o is the argument that tells the compiler to link and Divide is the name of the executable.

1.3 Assignment 1.1.3

A compile error can very easily be introduced by removing a semicolon. By removing the semicolon after the declaration int b = 3 and then trying to compile the main.cpp file, the following error is given.

```
main.cpp: In function ‘int main()’:
main.cpp:15:5: error: expected ‘,’ or ‘;’ before ‘int’
   15 |     int c = divide(a, b);
      |     ^~~
main.cpp:16:42: error: ‘c’ was not declared in this scope
   16 |     std::cout << a << "/" << b << "=" << c << std::endl;
      |                                          ^
```

This error is given because the compiler does not know the line should end there, as this is usually denoted by a semicolon. Therefore, the compiler is not able to understand what the function should do, so it is unable to compile the function.

1.4 Assignment 1.1.4

A linker error is not necessarily a fault in the code, but can be introduced by moving the implementation of the divide function elsewhere and removing its connection to the header file by removing the #include"divide.hpp". When compiling the programs, no error is given, but trying to link the objects will give the following error:

```
/usr/bin/ld: /tmp/cc1PZAij.o: in function ‘main’:
main.cpp:(.text+0x25): undefined reference to ‘divide(int, int)’
collect2: error: ld returned 1 exit status
```

This error is given because the compiler cannot find the implementation of the divide function and thus is not able to properly create the executable.

1.5 Assignment 1.1.5

In this specific case a runtime error can be introduced by editing the denominator to be 0. Since you cannot divide by zero the executable will give the following error:

```
Floating point exception (core dumped)
```

This error occurs when you try to do an impossible operation with a floating point number. In this case we try to divide by zero, which is impossible.

2 Assignment 1.2

The goal of Assignment 1.2 was to design and implement a BMI calculator that would take input from the user via the terminal, and then calculate the user's BMI accordingly.

2.1 Design Decisions

- The program uses functions for modularity and readability: The code is divided into several functions that perform specific tasks, such as reading user input, calculating the body mass index (BMI), evaluating and printing the BMI, and printing information about BMI values. This makes the code easier to understand and maintain.
- The use of references as arguments: Functions receive input data via arguments. In this case, the readInput function receives the references to weight and height of the user as arguments. This allows the function to manipulate the user data directly, as a function cannot return multiple values.
- The use of string streams: The input data is first read as a string and then converted to a float using the string stream library. This is done to validate the input data and ensure that only valid numerical data is used in the calculation.
- The use of if-else statements: The evaluateAndPrintBMI function uses if-else statements to determine the category of BMI and print a corresponding message. This allows the function to perform different actions based on the input data.
- The use of setw and left manipulators: The printInfo function uses setw and left manipulators to format the output data. This makes the output look neat and well-aligned.

2.2 Compiling and running the program

Since there is only one source file, there is no need to compile and link the program separately. When the terminal is opened in the Assignment2 file, the following command can be used to make the executable of the program:

```
$ g++ main.cpp -o BMICalculator
```

This will compile the program and create the executable called BMICalculator. This executable can be run using the following command:

```
$ ./BMICalculator
```

3 Assignment 1.3

The goal of Assignment 1.3 was to design and implement a program for calculating a safe heart rate range.

3.1 Design Decisions

- The Date struct takes in the day, month, and year of the user's date of birth, and stores the values in its member variables day, month, and year. The constructor for Date checks if the inputs are valid and if not, it rounds the inputs to the nearest valid date.
- The getCurrentDate function returns the current date by getting the time from the computer's clock and converting it to a tm structure, which can then be used to create a Date object.
- The Human class stores information about a person, including their first name, last name, and date of birth. It has functions to set and get these values, as well as a function to calculate the person's age. The function calculateMaximumHeartRate calculates the maximum heart rate for the person based on their age, and the function calculateTargetHeartRates calculates the minimum and maximum target heart rates for the person and stores them in a map.
- The readInput function prompts the user for their name and date of birth and returns a Human object with the input information.
- Finally, the printInfo function takes in a Human object and displays the person's name, date of birth, age, maximum heart rate, and target heart rates.
- The program creates a Human object by calling the readInput function, and then displays the heart rate information by calling the printInfo function with the Human object as an argument.

3.2 Compiling and running the program

Since there is only one source file, there is no need to compile and link the program separately. When the terminal is opened in the Assignment3 file, the following command can be used to make the executable of the program:

```
$ g++ main.cpp -o HeartRateCalculator
```

This will compile the program and create the executable called HeartRateCalculator. This executable can be run using the following command:

```
$ ./HeartRateCalculator
```

4 Questions

a 1 If you forget to include a header file in a .cpp file, you will get a compiler error. A compiler error occurs when the compiler is unable to understand the code written in a .cpp file because it is missing information that the header file would have provided. In this case, the missing header file results in the compiler not being able to recognize the functions or variables used in the .cpp file, causing the compiler to throw an error.

2 If you omit a .cpp file from the compile command, then the corresponding object file for that .cpp file will not be created. When the linker tries to link all the object files together, it will not be able to find the definitions for any functions that are declared in the missing .cpp file. As a result, the linker will generate an error indicating that the symbols (i.e., function names) are undefined.

b No, this cannot be compiled by including b.h because function A() in a.cpp requires the definition of function B() and vice versa, but including only the declaration of B() in b.h will not provide the complete definition of B() for the compiler. To compile this, you can do the following: Create a header file, say ab.h, that declares both functions A() and B(). Include ab.h in both a.cpp and b.cpp using the #include directive. Compile both a.cpp and b.cpp into object files using the following command: g++ -c a.cpp b.cpp Finally, link the object files into an executable using the following command: g++ a.o b.o -o output This will result in a working executable that contains both functions A() and B().

c You have to include the iostream header in your code because it declares the standard input/output stream objects that are used for reading from and writing to the standard input/output streams.

d In a C++ class, the private and public access specifiers determine the visibility and accessibility of the members (variables and functions) of the class. The private members are only accessible within the class and are not directly accessible from outside the class. This means that the code outside the class cannot access or modify the private members directly. This is useful for encapsulating the implementation details of a class and ensuring that the internal state of the object is only manipulated through the provided public interface. On the other hand, the public members can be accessed from outside the class as well as within the class. This means that the code outside the class can directly access and modify the public members. This is useful for exposing the public interface of a class and allowing users of the class to interact with its objects.

e Data hiding is a mechanism used to encapsulate data within a class and restrict access to it from outside the class. The purpose of data hiding is to protect the internal data and state of an object from being directly accessed or modified by external code. This helps to ensure the integrity and consistency of the object's data, as well as to maintain the modularity and separation of concerns in the program design.

f A program can use the standard library class string without using a using namespace directive by qualifying the string class with its namespace, std. For example: std::string Using a using namespace directive at the start of your code can be problematic because it can cause name collisions and reduce code readability. The directive makes all the names in a namespace available to the code without qualification, which can lead to naming conflicts if two or more namespaces contain classes or functions with the same name. This can make it difficult to understand which class or function is being referred to in the code.

g The output of this code is:
20
10
The reason for this is because the variable a in the main function is shadowing the global variable a defined outside the main function. In the main function, a local variable a with the value of 10 is declared. Then the function function is called, which prints the value of the global variable a, which is 20. Finally, in the main function, the value of the local variable a, which is 10, is printed.