Bern University
of Applied Sciences

C++ Programming I

Exercise-02

Prof. Dr. P. Arnold <patrik.arnold@bfh.ch>
BME – FS2022

1 Introduction

This exercise will focus on using functions in C++.
You will learn the following topics when completing this exercise:

- Repeat the generation of simple CMake-Projects
- Declaring, defining and using functions in C++
- Pass-by-value and pass-by-reference
- Function overloading
- Random Number Generator (C++11) and calculating π

2 Functions

In this section you'll do some easy exercises to repeat the basic usage of functions, namely passing arguments, default values and overloading. Although not needed define the functions in separate header and source files as an exercise for exercise 2.1 and 3. In addition create CMake-Projects with C++11 compiler support and Debug/Release build options for each exercise. Add additional files manually to the project to gain full control over the included project files, e.g:

```
add_executable(${PROJECT_NAME} main.cpp src.cpp hdr.h)
```

2.1 Pass-by-Value & Pass-by-Reference

1. Write a program with a function `add` (e.g. `calculater.cpp/calculater.h`) that takes two `int` parameters, adds them together and returns the sum. The program should ask the user for two numbers, then call the function with the numbers as arguments and display the sum to the user.
   Example usage:
   ```
   Enter two numbers: 33 12
   The sum is 45.
   ```

2. Basically the same as before, but this time, the function that adds the numbers should return `void`, and takes a third, pass-by-reference parameter to put the sum.

3. What's wrong here?

```cpp
#include <iostream>

void doubleNumber(int num){num = num * 2;}

int main()
{
    int num = 35;
    doubleNumber(num);
    std::cout << num; // Should print 70
    return 0;
}
```

(Note! Changing the return type of doubleNumber is not a valid solution.)

2.2 Default Values

Write a **single** function that adds up to four values. It should be callable as follows:

```cpp
addNumbers(1, 2, 3, 4);
addNumbers(1, 2, 3);
addNumbers(1, 2);
addNumbers(1);
addNumbers();
```

2.3 Function Overloading

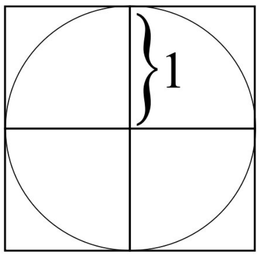
1. Write a function `printType(int intNumber)` that prints the variable type to the console: e.g. `type is int.`

2. Overload the function for the types `double, float and string` and test all the functionality as follows:

```cpp
// Your functions go here

//

int main()
{
    int intNumber = 42;
    printType(intNumber); // -> Type is int

    float floatNumber = 1.666;
    printType(floatNumber); // -> Type is float

    ...
    ..
    .
}
```

3 Train your Skills - Calculate Pi

Using a “Monte Carlo“-like method – that is, a randomized simulation – we can compute a good approximation of π. Consider a circle of radius 1, centred on the origin and circumscribed by a square, like so:



Imagine that the square is a dartboard and that you are tossing darts at it randomly. Throwing a lot of darts, the ratio of darts in the circle to total darts thrown should be the ratio between total darts the area of the circle A_{Circle} and the area of the square A_{Square}. We can use the ratio:

$$\frac{total\,darts}{darts\,in\,circle} = \frac{A_{Square}}{A_{Circle}} \tag{1}$$

r^2

0.25 * pi * r^2

to calculate π! We can simplify the maths by only considering the first quadrant, calculating the ratio of the top right square's area to the area of the single quadrant, *i.e.* `1 ⩾ x,y ⩾ 0`. Adapt the above equation accordingly.
We'll build a function step by step to do all this.

3.1 Setup Random Number Generator

To randomly throw darts to the dartboard we use one of the many new C++11 random functions. We'll use the random generator to simulate the `x` and `y` position of the thrown dart. To generate a random value you can do:

```cpp
#include<iostream>
#include<random>

int main()
{
    // A standard interface to a platform-specific non-deterministic random number generator
    std::random_device rd; // Create Random Device as seed
    std::mt19937 mt(rd()); // Init Mersenne Twister Engine with that seed
    // A continuous random distribution on the range [min, max] with equal probability.
    std::uniform_real_distribution<double> dist(0.0, 1.0);

    // Generate 10 random numbers
    for (int i=0; i<10; ++i)
    {
        std::cout << dist(mt) << "\n";
    }

    return 0;
}
```

3.2 Throwing a Dart

Place your `x` and `y` declarations in a loop to simulate multiple dart throws. Assume you have a variable `n` indicating how many throws to simulate. Maintain a count (declared outside the loop) of how many darts have ended up inside the circle. You can check whether a dart is within a given radius with the Euclidean distance (Use the functionality from the `<cmath>` header, e.g. `sqrt`).

3.3 Throw many Darts

Now use your loop within a π-calculating function. The function should take one argument specifying the number of dart throws to run. It should return the decimal value of pi, using the technique outlined above. Be sure to name your function appropriately. You should get pretty good results for around 5,000,000 dart throws.

4 Submission

Submit your source code (as a zip-file) to Ilias Ex-02 **before the deadline** specified in Ilias.