Name: ____ ID: ____________________

Object-Oriented Programming Lab #1 **Jan 20th, 2023**

Introduction to C++

1. Given the following programs:

```
//// Program 1.1
int main()
{
    int number = 0;
    float value = 0
    double bigNumber;
    return 0;
}
```

```
//// Program 1.2
int main()
{
    int number = 0;
    float value = 0
    double bigNumber;
    bigNumber = number + value;
    return 0;
}
```

```
//// Program 1.3
int main()
{
    int number = 0;
    float value = 0;
    double bigNumber;
    bignumber = number + value;
    return 0;
}
```

```
//// Program 1.4
#include <iostream>
#include <string>

int main()
{
    const string ERROR_MESSAGE
        = "bad string!;
    cout << "Hello!\n;
    return 0;
}
```

```
//// Program 1.5
int main()
{
    float firstVal = 0;
    float secondVal = 0;
    float factor;
    float result
        = (firstVal – secondVal
          / factor;
    return 0;
}
```

```
//// Program 1.6
int main()
{
    const double x = 2.0;
    const double y = 3.1415;
    double product;
    x * y = product;
    return 0;
}
```

For each program, you are expected to:

- find syntax errors and how to correct them
- describe what the program is supposed to do in run-time
- if the program is not working as expected, describe the problem and how to correct them
- add informative output to the program to make it more complete as it is possibly missing displaying results to the user
- describe how to evolve the program to improve it in terms of useful features and completeness

Advice: Use the C++ compiler to help catching syntax errors

2. Given the following program:

```cpp
#include <iostream>
#include <string>

int main()
{
    std::cout << "Please enter P1 name: ";
    std::string p1_name;
    std::cin >> p1_name;

    std::cout << "Please enter P2 name: ";
    std::string p2_name;
    std::cin >> p2_name;

    std::cout << "Player 1: " << p1_name << std::endl;
    std::cout << "Player 2: " << p2_name << std::endl;
    return 0;
}
```

2.1) What will the above program do if you type two names (for example, **`"Mike Leo"`**) on a single line when it asks you for input? Predict the behavior before running the program, then try it.

2.2) Change the program so that it draws frame around the name for both players like the example output shown on the right:

*Program Output (for **2.2**)*

```
**********************************
*                *               *
* Player 1: Mike * Player 2: Leo *
*                *               *
**********************************
```

2.3) Change the program so that it draws frame around the name for both players like shown below:

*Output (for **2.3a**)*

```
******************
*                *
* Player 1: Mike *
*                *
******************
*                *
* Player 2: Leo  *
*                *
******************
```

*Output (for **2.3b**)*

```
+----------------+
|                |
| Player 1: Mike |
|                |
+----------------+
|                |
| Player 2: Leo  |
|                |
+----------------+
```

*Output (for **2.3c**)*

```
+================+
|                |
| Player 1: Mike |
|                |
+----------------+
|                |
| Player 2: Leo  |
|                |
+================+
```

3. Write programs to print patterns with varying sizes.
All programs **must take** the pattern size from user and **must not** print trailing spaces before the end of each line.

3.1) The program should print a triangle pattern like shown below:

Output Size = 0 (0 line)	*Output Size = 1*
	`*`
		Output Size = 2
		`*`
		`**`

Output Size = 3
`*`
`**`
`***`
	Output Size = 4
	`*`
	`**`
	`***`
	`****`
		Output Size = 5
		`*`
		`**`
		`***`
		`****`
		`*****`

3.2) The program should print an arrow pattern like shown below:

Output Size = 0 (0 line)	*Output Size = 1*
	`*`
		Output Size = 2
		`*`
		`**`
		`*`

Output Size = 3
`*`
`**`
`***`
`**`
`*`
	Output Size = 4
	`*`
	`**`
	`***`
	`****`
	`***`
	`**`
	`*`
		Output Size = 5
		`*`
		`**`
		`***`
		`****`
		`*****`
		`****`
		`***`
		`**`
		`*`

3.3) The program should print an arrow pattern like shown below:

Output Size = 0 (0 line)	*Output Size = 1*
	`*`
		Output Size = 2
		` *`
		`**`
		` *`

Output Size = 3
`  *`
` **`
`***`
` **`
`  *`
	Output Size = 4
	`   *`
	`  **`
	` ***`
	`****`
	` ***`
	`  **`
	`   *`
		Output Size = 5
		`    *`
		`   **`
		`  ***`
		` ****`
		`*****`
		` ****`
		`  ***`
		`   **`
		`    *`

4. One way to estimate the value of **π** is by using the random number generator. The calculation is done by generating `N` random (x_i, y_i) pairs where each point $p_i = (x_i, y_i)$ is in range `[-1, 1]` (inclusive), then calculate the probability of the point (x_i, y_i) lying inside the unit circle. With a large number `N` and a good random number generator, the probability value will be close to the number of **π / 4**.

- The point $p_i = (x_i, y_i)$ will be inside the unit circle if the distance $d = \sqrt{(x_i^2 + y_i^2)}$ is in range `[-1, 1]`
- When drawing `N` points and found N_i points inside the unit circle, the probability of the point p_i lying inside will be N_i / N
- The estimate value of **π** will be $4 * N_i / N$

Use the following code as the starting point to write a program for estimating the value of **π** by the above method.

```cpp
//// random.hpp
#ifndef MY_RANDOM_HPP
#define MY_RANDOM_HPP

#include <random>

class Rand_double {
public:
    using seed_type = std::random_device::result_type;

    Rand_double(double low, double high): dist{low,high} {}

    // draw an integer number
    double operator()() { return dist(re); }

    // choose new random engine seed
    void seed(seed_type s) { re.seed(s); }
private:
    std::default_random_engine re;
    std::uniform_real_distribution<double> dist;
};

#endif /* MY_RANDOM_HPP */
```

```cpp
//// lab1_3.cpp
#include "random.hpp"

#include <iomanip>
#include <iostream>
#include <vector>

template<typename T_>
inline constexpr
    T_ pi_v{3.141592653589793238462643383279502884L};

inline constexpr double pi = pi_v<double>;

int main()
{
    constexpr double rnd_min = -1.0, rnd_max = 1.0;
    Rand_double rnd{rnd_min, rnd_max};

    std::random_device rd;
    rnd.seed(rd());
    std::cout << std::fixed << std::setprecision(3);

    double x1 = rnd();
    double y1 = rnd();
    std::cout << "Point #1: (" << x1 << ", " << y1 << ")\n";

    double x2 = rnd();
    double y2 = rnd();
    std::cout << "Point #2: (" << x2 << ", " << y2 << ")";
    std::cout << std::endl;
    return 0;
}
```

4.1) Estimate **π** using `N = 100`, record the approximation, the relative error, and the percent error relative to the exact **π**

4.2) Estimate **π** using `N = 10,000`, record the approximation, the relative error, and the percent error relative to the exact **π**

4.3) Estimate **π** using `N = 1,000,000`, record the approximation, the relative error, and the percent error relative to the exact **π**