DUE DATE: **Mon/Tue, March 16/17 2020 during Lab time**

CSE-278: Systems 1

Lab #5

Max Points: 50

You should save/rename this document using the naming convention MUid.docx (example: ahmede.docx).

Objective: The objective of this exercise is to:

0. Review operator overloading
1. Review operator overloading, compiler directive and friend function
2. Gain familiarity with parameter passing in C++

You may discuss the questions with your instructor.

Name: Ben Hilger

Part #0: Review operator overloading

☞ You can verify your solution for this question by copy-pasting the given code and running it in the online C++ documentation at: https://en.cppreference.com/w/cpp/io/cout Run this code

Problem: Trace the operation of the program below and illustrate the output from the program.

```
#include <iostream>

class WordPair {
    friend std::ostream&
    operator<<(std::ostream& os, const WordPair&) {
        os << "WordPair.\n";
        return os;
    }
};
class Phrase {
    friend std::ostream&
    operator<<(std::ostream& os, const Phrase& ph) {
        os << "Phrase.\n";
        os << ph.wp1 << ph.wp2;
        return os;
    }
    WordPair wp1, wp2;
public:
    Phrase operator+(const WordPair& ) const {
        std::cout << "Phrase::operator+ called.\n";
        return *this;
    }
};
int main() {
    WordPair wp;
    Phrase ph;
    std::cout << wp;  // <-- This line prints output
    ph = ph + wp;  // <-- This line prints some output
    std::cout << ph;  // <-- This line prints some output
    return 0;
}
```

Show output from above program here:

```
WordPair.
Phrase::operator+ called.
Phrase.
WordPair.
WordPair.
```

Part #1: Review operator overloading, compiler directive and friend function

```
// PhoneNumber class definition
#ifndef PHONENUMBER_H
#define PHONENUMBER_H
#include <iostream>
#include <string>
class PhoneNumber
{
    friend std::ostream &operator<<( std::ostream &, const PhoneNumber &);
    friend std::istream &operator>>( std::istream &, PhoneNumber &);
    friend bool operator==(const PhoneNumber &, const PhoneNumber &);
    friend bool operator!=(const PhoneNumber &, const PhoneNumber &);
private:
    std::string areaCode;   // 3-digit area code
    std::string exchange;   // 3-digit exchange
    std::string line;       // 3-digit line
};  // end class PhoneNumber

#endif /* PHONENUMBER_H */
```

a. What is the use of the compiler directives?

```
#ifndef PHONENUMBER_H
#define PHONENUMBER_H
#endif
```

#ifndef PHONENUMBER_H: This compiler directive checks to see if the given identifier (PHONENUMBER_H) has been defined. If not, it runs through the code until it’s associated #endif.

#define PHONENUMBER_H: This compiler directive creates an identifier with the specified name (PHONENUMBER_H). This also includes the contents of the files this is defined in.

```
#endif: This compiler directive marks the end of an associated
#if statement. In this instance, it marks the end of defining
PHONENUMBER_H.
```

b. Why we need to use the visibility modifier *friend?*

```
We will need the visibility modifier friend because it allows access to
private instance variable and methods from non-members.
```

Part #2: Understanding parameter passing

In almost all programming languages methods (aka functions) play a central role. Consequently, C++ provides both pass-by-value and pass-by-reference approaches for both primitive data types and objects.

For the following methods, for each parameter, indicate if it is pass-by-value or pass-by-reference. In addition, illustrate example method calls. **Note**: **Prefer to use literal constants (as in: `42` or `"testing"`)** in method calls and only add variables only when needed. The first couple of them have been completed to illustrate an example.

Method signature	Parameter-passing type	Example method call

`doIt(int i, int& j)`	`i`: pass-by value
	`j`: pass-by reference
		`int x = 42;`
		`doIt(10, 42);`

`callMe(std::string str);`	`str`: pass-by value	`callMe("hello?");`
`callMeMaybe(int num);`	`num`: pass-by-value	`callMeMaybe(1);`
`magic(std::string& name, int& age);`	`name`: pass-by-reference `age`: pass-by-reference	`magic("Test");`
`phone(const long data1, const long& data2)`	`data1`: pass-by-value, but can't be changed `data2`: pass-by-reference, but can't be changed	`phone(34, 45);`
`helpdesk(const std::string& problem, std::string& solution)`	`problem`: pass-by-reference, but can't be changed	`Helpdesk("3x", "0");`

	`solution`: pass-by-reference	
`ping(std::string& s, int &i)`	`s`: pass-by-reference `i`: pass-by-reference	`Ping("www.google.com", 10);`

Part #3: Submit to Canvas

Once you have responded to all the questions in this document, save the MS-Word document as a PDF file. Upload the PDF to Canvas. Ensure you actually submit the file.

- No late assignments will be accepted!
- This work is to be done individually
- This MS-Word document (duly filled-in) saved as a PDF document.
- The submission file will be saved with the name ***Lab5_yourMUID.pdf***
- Assignment is due Monday/Tuesday, March 16/17 during Lab time
- On or before the due time, drop the *electronic copy* of your work in the *canvas*

Don't forget to Turn in the file!   Lab5_yourMUID.pdf