Note: All the code that you write for the following labs must be uploaded to Github.

C++ Operator Overloading

You can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types (classes) as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list. Therefore, operator functions are the same as normal functions. The only differences are, that the name of an operator function is always the operator keyword followed by the symbol of the operator, and operator functions are called when the corresponding operator is used.

Following is the list of operators which can be overloaded:

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

The following operators cannot be overloaded:

::	.*	.	?:	sizeof

The syntax for overloading an operator is as follows:

- The keyword **operator.**
- An overloaded operator has a return type.
- An overloaded operator has a parameter list.

Thus the declaration is as follows:

return_type operator_keyword operator_symbol (parameters)

1. Implement the following in a new project as follows:

   Within a file titled **Box.h**, write the following code.

```
#pragma once

#include <iostream>
using namespace std;

class Box {
public:

     Box();

     Box(const double newLength, const double newBreadth, const double
newHeight);

     ~Box();

     double GetVolume();

     void SetLength(double len);

     void SetBreadth(double bre);

     void SetHeight(double hei);

     // Overload + operator to add two Box objects.
     Box operator+(const Box& b);

private:
     double length;      // Length of a box
     double breadth;     // Breadth of a box
     double height;      // Height of a box
};
```

Within a file titled **Box.cpp**, write the following code.

```
#pragma once

#include "Box.h"

Box::Box() {
	length = 0.0;
	breadth = 0.0;
	height = 0.0;
}

Box::Box(const double newLength, const double newBreadth, const double
newHeight) {
	length = newLength;
	breadth = newBreadth;
	height = newHeight;
}

Box::~Box() {

}

double Box::GetVolume() {
	return length * breadth * height;
}
void Box::SetLength(double len) {
	length = len;
}
void Box::SetBreadth(double bre) {
	breadth = bre;
}
void  Box::SetHeight(double hei) {

	height = hei;
}

// Overload + operator to add two Box objects.
Box Box::operator+(const Box& b) {
	Box box;

	box.length = this->length + b.length;
	box.breadth = this->breadth + b.breadth;
	box.height = this->height + b.height;
	return box;
}
```

Note use of the **this** pointer. The **this** pointer is a pointer accessible only within the non-static member functions of a class. In other words, the **this** pointer, provides each object with access to its own address in memory. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

The syntax for the this pointer is as follows:

- `this->member-identifier`

For more information on the this pointer see the following link:

- https://learn.microsoft.com/en-us/cpp/cpp/this-pointer?view=msvc-170

Within **main.cpp**, write the following code and run the program.

```
// Main function for the program
int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   Box Box3;                // Declare Box3 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.SetLength(6.0);
   Box1.SetBreadth(7.0);
   Box1.SetHeight(5.0);

   // box 2 specification
   Box2.SetLength(12.0);
   Box2.SetBreadth(13.0);
   Box2.SetHeight(10.0);

   // volume of box 1
   volume = Box1.GetVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.GetVolume();
   cout << "Volume of Box2 : " << volume <<endl;

   // Add two object as follows:
   Box3 = Box1 + Box2;

   // volume of box 3
   volume = Box3.GetVolume();
   cout << "Volume of Box3 : " << volume <<endl;

   return 0;
}
```

2. Implement a new project as follows:
   (a) In file titled **Person.h**, implement the following code.

```
#pragma once

#include <string>

using namespace std;

class Person
{
public:
	Person();
	Person(float newWeight);

	~Person();
	//Overload the add operator
	float operator + (const Person& otherPerson);

private:
	float mWeight;
	string mFirstName;
	int mAge;
};
```

   (b) In a file title **Person.cpp**, provide the definition for the class declared in **2(a)**
   (c) Implement **main.cpp** as follows:

```
int main() {
	Person Jane = Person("Jane", 60.0f);
	Person John = Person("John", 75.0f);

	float totalWeight = Jane + John;

	cout << "Total weight: " << totalWeight << endl;

	return 0;
}
```

(d) Overload the following relational operators using **mFirstName**

- ==
- !=

Then add the following to the main function implemented in **2(c)**

```
if (Jane == John){
      cout << "This is the same person" << endl;
}

if (Jane != John){
      cout << "This is NOT the same person" << endl;
}
```

(e) Overload the following relational operators using **mAge**.

- <
- >

Then add the following to the main function implemented in **2(c)**

```
if (Jane < John){
      cout << "Jane is younger than John" << endl;
}

if (John > Jane){
      cout << "John is older than Jane" << endl;
}
```

C++ Conversion Operators

Conversion operators are used to convert one concrete type to another concrete type or primitive type implicitly. Therefore, C++ provides operators (which can be overloaded) that can convert from on type (for instance a class type), to another type.

The declaration for conversion operators is as follows:

- **operator_keyword conversion_type**()

3. Within the Person class declared in the header file **2(a)**, add the following declaration

```
operator int ();
```

   Within the **Person.cpp** file, define the conversion operator declared above, as follows:

```
Person::operator int()
{
     return mAge;
}
```

   Add the following to the main function implemented in **2(c)**.

```
int johnAge = John;
cout << "John's Age: " << johnAge << endl;
```

4. Within the Person class declared in the header file in **2(a)**, add the appropriate conversion operator declarations and definitions (within Person.cpp) so that you then add the following to the main function implemented in **2(c)**

```
string janeFirstName = Jane;
cout << "Jane's FirstName" << janeFirstName << endl;

float janeWeight = Jane;
cout << "Jane' weight" << janeWeight << endl;
```

References

- https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm
- https://www.tutorialspoint.com/cplusplus/cpp_this_pointer.htm
- https://www.geeksforgeeks.org/operator-overloading-cpp/
- https://www.geeksforgeeks.org/conversion-operators-in-cpp/