**Reinforcement of Operators**

Create a project using the Person.h and PersonOperatorTesting.cpp file, then answer the question below. You will find the code below.

1. Look at the .cpp file. Notice the *//Comparing using > comment*. Below it is an if statement that uses the > operator. What two items are being compared? What is being returned as a result of the comparison? **The hm(Homer) and bt(Bart) objects are being compared. The return is a bool of whether the current/left item is larger than the other/right object.**
2. Locate the same operator in your Person.h file. Compare the parameter and return type to your answers in question 1? What are the similarities? **The specific data being compared is the amount values of each. These are being compared as the current object vs. the passed in object.**
3. Look back in the .cpp file. Notice the *//Comparing using == comment*. Below it is an if statement that uses the == operator. What two items are being compared? **The hm(Homer) object and the mg(Marge) object are being compared and seeing if they are equal.**
4. Add the operator== into the Person.h file. Test your program to make sure it works correctly.
5. Add the ability to use the following operators to the Person.h class >=, <, <=, !=. Add a few more if statements in the .cpp file to test the operators.
6. What if we wanted to compare a person to a number. Add the following if statement to the .cpp file.

if (bt < 21) {

cout << bt << " too young for Moes" << endl;

}

1. Examine that if statement. What type of value is to the right? Is the item to the left, the current object? What type of value would need to be returned? **The value to the right is an int object, while the left is a Person object. This would try to use the overloaded < operator but there isn’t one that has those parameters. This would still return a bool as it needs to be a condition.**
2. Code the < operator in the Person class. Run the program to ensure it works.
3. Add operators for >, <=, >=, ==, != to be able to compare a person to an int. Add a few if statements to ensure that the operators work.
4. Add the following if statement to the .cpp file. Does the code run? **No**

if (21 > bt) {

cout << bt << " less than 21" << endl;

}

1. Examine the operator, what is to the right? What is being returned? **The object to the right is a Person object that is attempting to be compared to a native wrapper int value. This doesn’t work because int has no overload that takes a Person object. It should return a bool comparing the int to the amount member variable.**
2. Since the object is not to the left, we would code this operator as a friend function. The item to the left of the operator will become the first parameter, the item to the right will become the second. Code the friend function, then run the program to test the solution.
3. Add friend functions for >=, <=, <, ==, !=

**Copy Constructor**

Today we will observe the importance of a deep and shallow copy using pointers. To observe this difference, we will be using the LinkedList.h file from the previous chapter. Create a project using the given code and your LinkedList.h. Then answer the questions.

1. Look at the main under the comment //List 1. How many values does LinkedList one have added to it? **A single value, ‘1’**
2. Look at //List 2. What is it set to initially? How many values are added? **List 2 is initially set as list 1, but another one is added, value ‘42’**
3. Are any other values added to list one? **No**
4. Run the program and observe the two lists. How did output differ from what was added to list one? **The ‘42’ value was added to both lists, the one list and the two list.**
5. This is called a Shallow Copy. A shallow copy is when the pointers are used as private variables are copying the same memory location.
6. To fix the problem, we will make a deep copy. A deep copy occurs when you use a Copy Constructor and operator=. Add the following constructor to the LinkedList class.

LinkedList(const LinkedList<Type> &lst) {

auto temp = lst.front;

front = nullptr;

back = nullptr;

for (int i = 0; i < lst.count; i++) {

insert(temp->data);

temp = temp->next;

}

}

1. The same thing can happen if you do not have an operator=. Add an operator= to your LinkedList class. Be sure to delete any existing data in the class beforehand. Test your code by uncommenting the code following the //Operator Equals Test If you coded the operator= correctly, then only the second list will have the value 60. The first will not.

#include "LinkedList.h"

using namespace std;

int main() {

//List 1

LinkedList<int> one;

one.insert(1);

//List 2

LinkedList<int> two = one;

two.insert(42);

//Operator Equals Test

//two = one;

//two.insert(60);

//Display both Lists

cout << one << endl;

cout << two << endl;

}

**Person.h**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

Person(string name, int amount) : name(name), amount(amount) {}

bool operator >(const Person &g);

int getAmount() const;

friend ostream& operator <<(ostream &out, const Person &g);

private:

string name;

int amount;

};

bool Person::operator >(const Person &g) {

return amount > g.amount;

}

int Person::getAmount() const {

return amount;

}

ostream& operator <<(ostream &out, const Person &g) {

out << g.name << ": " << g.amount;

return out;

}

**OperatorTesting.cpp**

#include <iostream>

#include "Person.h"

using namespace std;

int main() {

Person bt("Bart", 10);

Person hm("Homer", 42);

Person mg("Marge", 42);

//Comparing using >

if (hm > bt) {

cout << "Older: " << hm << endl;

cout << "Younger: " << bt << endl;

}

//Comparing using ==

if(hm == mg){

cout << "Same Age: " << endl;

cout << hm << endl;

cout << mg << endl;

}

}