Operator Overloading

Last updated 3/30/20

Operator Overloading

- Motivation
  - Can we ask the questions?

    with objects circle1 and circle2 instances of Class Circle,

    objects box2 and box 5 instances of Class Box

    objects savings and checking instances of Class Account

    - Is circle1 == circle2
    - Is box2 < box5
    - what is savings – checking
    - Could I say box2 + circle1

Operator Overloading

- C++ allows operators to be overloaded when used with objects
  - All standard C++ operators can be overloaded except

    ?: . .* :: sizeof

  - Create a function to define what the operator should cause to happen
    - The code can call the function as usual
    - The code can use the standard operator symbol instead of calling the function (compiler interprets what is meant)

assuming an overloaded + function in a Class called Box with objects box1, box2 and box3

```
box3 = box1.operator+(box2);   // call the operator+ function for box1 with argument box2
box3 = box1 + box2;            // compiler recognized the overloaded operator + and implements
                               // the box1 operator+ function
```

Operator Overloading

- Operator overloading syntax

  return-type operator$_{\text{symbol}}$(arg list);

  where the arg list is appropriate for the operator (unary, binary, …)

  Account operator+(const Account & rhs);

  notes: operator+(..) looks just like setWidth(…)

  const … & prevents the operation from harming the object passed to the function

  rhs refers to the right hand side of the operator as a reminder (could be called anything)

Operator

```cpp
/*
 * Account.h
 *
 *  Created on: Mar 28, 2019
 *      Author: johnsontimoj
 */

#ifndef ACCOUNT_H_
#define ACCOUNT_H_

class Account{
private:
double savings;
double checking;

public:
Account(void);
Account(double s, double c);
void setSavings(double s);
void setChecking(double c);
double getSavings(void);
double getChecking(void);
double calcBalance(void) const;

Account operator+(const Account & rhs);
Account operator-(const Account & rhs);
bool operator==(const Account & rhs);
bool operator<(const Account & rhs);
bool operator>(const Account & rhs);

};

#endif /* ACCOUNT_H_ */
```

```cpp
/*
 * Account.cpp
 *
 *  Created on: Mar 28, 2019
 *      Author: johnsontimoj
 */

#include "Account.h"

Account::Account(void){
 savings = 0;
 checking = 0;
}
.  .  .

double Account::getChecking(void){
  return checking;
}
double Account::calcBalance(void) const{
  return savings + checking;
}

Account Account::operator+(const Account & rhs){
  Account tmpacct;
  tmpacct.savings = savings + rhs.savings;
  tmpacct.checking = checking + rhs.checking;
  return tmpacct;
}
Account Account::operator-(const Account & rhs){
  Account tmpacct;
  tmpacct.savings = savings - rhs.savings;
  tmpacct.checking = checking - rhs.checking;
  return tmpacct;
}
bool Account::operator==(const Account & rhs){
  if((savings == rhs.savings) && (checking == rhs.checking))
    return true;
  else
    return false;
}
bool Account::operator>(const Account & rhs){
  if(this->calcBalance() > rhs.calcBalance())
    return true;
  else
    return false;
}
bool Account::operator<(const Account & rhs){
  if(this->calcBalance() < rhs.calcBalance())
    return true;
  else
    return false;
}
```

```cpp
/*
 * operator_overloading.cpp
 *
 *  Created on: Mar 28, 2019
 *      Author: johnsontimoj
 */


#include "Account.h"
#include <iostream>
using namespace std;

int main(void){

  Account act1;
  Account act2(100,100);
  Account act3;

  cout << "act1 == act2 " << (act1 == act2) << endl;

  act3 = act2 + act1;
  cout << "act3 == act2 " << (act3 == act2) << endl;

  act3 = act3 - act2;
  cout << "act3 == act2 " << (act3 == act2) << endl;

  cout << act2.calcBalance() << endl;
  cout << (act2 > act1) << endl;
  cout << (act2 < act1) << endl;

  return 0;
}
```

```
act1 == act2 0
act3 == act2 1
act3 == act2 0
200
1
0
```

Operator Overloading

- Unary operator overloading

  - Consider a Class Circle with member variable radius

  - Overload the prefix ++ operator (returns the new value)

```
Circle operator++();              // declaration – no parameters (.h)

Circle Circle::operator++(){      // definition (.cpp)
   ++radius;
   return *this;
}
```

Operator Overloading

- Unary operator overloading
  - Consider a Class Circle with member variable radius
  - Overload the postfix -- operator (returns the original value)

```
Circle operator--(double foo);              // declaration w/ dummy parameter (.h)
                                            // dummy parameter causes compiler to
                                            // use this fn when called in postfix notation

Circle Circle::operator--(double foo){      // definition (.cpp)
   Circle tmpcir = *this;                   // save a copy of original
   radius--;                                // modify original
   return tmpcir;                           // return the copy
}
```

Operator Overloading

- Assignment operator overloading
  - obj2 = obj1 does a variable by variable copy
  - What if one of the member variables is a pointer
    - The copy will leave 2 objects pointing to the same memory location
  - Support a = b = c = d;
    - Return type is a reference to a new object

```
Circle & operator=(const Circle & rhs);              // declaration (.h)


Circle & Circle::operator=(const Circle & rhs){  // definition (.cpp)
   if(&rhs != this){                                  // don't copy if same, e.g. a=a
      this->setRadius(rhs.getRadius());
   }
   return *this;
}
```