CS 202 – Assignment #6

Purpose: Learn operator overloading both as member and non-member functions, friend functions, this pointer, copy constructor.
Points: 100

Why we need Operator Overloading in C++?

With the use of Operator Overloading we can make operators that can work for our user defined classes. You can redefine, majority of the C++ operators. We can give a special meaning for the existing operator and we can operate that according to our definition. However, the operator overloading works only for user defined types but not pre-defined types(int, double, char, etc).
An example of operator overloading is that, we can use '+' to add two instance variables/objects of the same class, where it is not possible by default.

Assignment:

In this assignment you will learn to use operator overloading both as member and non-member functions, friend functions, this pointer, copy constructor. This assignment is based on American coins. The coins considered are penny, nickel, dime, quarter, fifty_centpiece, char *name. These coins are the member variables (private) of class coinsBox. We are going to explore most of the operators like arithmetic, assignment operator, increment and decrement operators, relational operators, stream insertion and extraction operators, constructors, and copy constructors.

Video Link:
Supplemental video Link: Assignment6_Operator_Overloading

Base form of the class, not including operators – has the following structure:

coinsBox
-penny: int
-nickel: int
-dime: int
-quarter: int
-fifty_centpiece: int
-name: char *
+setPennyCount(int): void
+setNickelCount(int): void
+setDimeCount(int): void
+setQuarterCount(int): void
+setFiftycentpieceCount(int): void

+getPennyCount() const: int
+getNickelCount() const: int
+getDimeCount() const: int
+getQuarterCount() const: int
+getFiftycentpieceCount() const: int
+getTotalAmountinCent() const: double
+getTotalAmountinDollars() const: double
+coinsBox()
+coinsBox(int, int, int, int, int)

Member Descriptions:
-penny: determines the maximum count of pennies in the coinsBox obj.
-nickel: determines the maximum count of nickels in the coinsBox obj.
-dime: determines the maximum count of dimes in the coinsBox obj.
-quarter: determines the maximum count of quarters in the coinsBox obj.
-fifty_centpiece: determines the maximum count of fifty_centpieces in the coinsBox obj.
-char * name: This pointer is used to give name of instance variable of a class like **coniBox1, coinBox2, coinBoxRes.**

```
char namecoinBox1[] = "coinBox1";
```

void setPennyCount(int): This setter/mutator function is to set the penny count of the coinsBox object or instance variable. All other setters functions definitions are similar.

```
void setNickelCount(int);
void setDimeCount(int);
void setQuaterCount(int);
void setFiftycentpieceCount(int);
```
void setName(string); This function is to set the name of the coinsBox instance variable.

int getPennyCount() const: This getter/accessor function is to get the penny count of the coinsBox object or instance variable. In its function definition, you can simply define return penny. All other getters functions definitions are similar.

```
int getNickelCount() const;
int getDimeCount() const;
int getQuaterCount() const;
int getFiftycentpieceCount() const;
```

double getTotalAmountinCent() const: In this function, you have to convert all the coins to cents individually and sum up all of them. This function should return cent (double). You can use the following formula to get the total cents.
cent = penny + (nickel * 5) + (dime * 10) + (quarter * 25) + (fifty_centpiece * 50);

`double getTotalAmountinDollars() const:` In this function, you have to convert the cents to dollars. You have to call the **`getTotalAmountinCent()`** function to get total cents. This function should return dollars (double). You can use the following formula to get the total dollars.
dollars = (cents Total/ 100);

`coinsBox():` It is a default constructor; Initialize all member variables to 0; name=*new* char[LEN];
`coinsBox(int, int, int, int, int, char _name[LEN]):` It is a constructor with parameters; This is used to pass the arguments to constructors. These arguments help to initialize an object when it is created.
`coinsBox &operator=(const coinsBox &):` This is for assignment operator overloading. It allows the deep copy when you use pointers as member variables in any class.
`~coinsBox():` The destructor is used to avoid memory leaks. Dynamic memory is allocated using *new* char[LEN]. To release the memory before the class instance is destroyed, we use destructor. (use delete operator)

The following operators need to be overloaded for `coinsBox` class in order to implement operations on coins. Remember that for all operations, you have to evaluate the total amount in cents and total amount in dollars. When you overload the operators, it is not on total cents or dollars, instead it is on individual member.

Example:

coinsBox coinBox1(10, 10, 10, 10, 10, namecoinBox1);

coinsBox coinBox2(5, 5, 5, 5, 5, namecoinBox2);

coinsBox coinBoxRes(0, 0, 0, 0, 0, namecoinBoxRes);

To display/print any object of class CoinsBox, you have to use stream insertion operator(<<). The details of coinBox1 should be as follows:

```
coinBox1: Penny = 10; Nickel = 10; Dime = 10; Quarter = 10; Fifty_centpiece = 10;
Cents = 910; Dollars = 9.1; Address = 0x7ffeedffd768
```

coinBox2 should be as follows:

```
coinBox2: Penny = 5; Nickel = 5; Dime = 5; Quarter = 5; Fifty_centpiece = 5; Cents
= 455; Dollars = 4.55; Address = 0x7ffeedffd748
```

When you overload operator '*':

```
coinsBox operator*(const coinsBox&) const;
```

```
coinBox1: Penny = 10; Nickel = 10; Dime = 10; Quarter = 10; Fifty_centpiece = 10;
Cents = 910; Dollars = 9.1; Address = 0x7ffeedffd768
coinBox2: Penny = 5; Nickel = 5; Dime = 5; Quarter = 5; Fifty_centpiece = 5; Cents
= 455; Dollars = 4.55; Address = 0x7ffeedffd748
MULTIPLICATION: coinBoxRes =  coinBox1 * coinBox2
coinBoxRes: Penny = 50; Nickel = 50; Dime = 50; Quarter = 50; Fifty_centpiece = 50;
Cents = 4550; Dollars = 45.5; Address = 0x7ffeedffd718
```

Note: If you observe the above example the operation os not on total but on individual member.

Things to remember:

- For any operator overloading, if the result is negative, then your program should output zero on every member as well as total functions of cents and dollars.
  `coinBoxRes: Penny = 0; Nickel = 0; Dime = 0; Quarter = 0; Fifty_centpiece = 0; Cents = 0; Dollars = 0`
- When you select an option from Menu (R) `-(Insert Coins to Coinbox1):` The values you enter for penny, nickel, dime, quarter, and fifty_centpiece should be added to the previous coinBox1 values. This approach is similar to (from Menu (S)) `(Insert Coins to Coinbox2)`
- To read data use the friend function:
  ```
  friend istream& operator>>(istream&, coinsBox &);
  ```
- To display/print data use the friend function:
  ```
  friend ostream& operator<<(ostream&, const coinsBox &);
  ```

Function bodies need to be implemented in americanCoinTypeImp.cpp file:

Overload the following operators.

Overload arithmetic operators

- + (Addition) - Adds two numbers represented as `coinsBox` object (coinBox1+coinBox2)
- - (Subtraction) - Subtracts two numbers represented as `coinsBox` object (coinBox1-coinBox2)
- + (Multiplication) - Multiplies two numbers represented as `coinsBox` object (coinBox1*coinBox2)
- -=(Assignment operator) – Subtracts coinBox1 from CoinBox2 and the result is assigned to coinBox2. `(coinBox2 -= coinBox1)`

Overload increment and decrement operators – Use *this pointer for these 4 operators

- x++(post increment/ postfix operator) – It increments all the 5 members by 1 individually. This is post-increment operation.
- ++x(pre increment/ prefix operator) – It increments all the 5 members by 1 individually. This is pre-increment operation.
- x--(post decrement/ postfix operator) – It decrements all the 5 members by 1 individually. This is post-decrement operation.
- --x(pre decrement/ prefix operator) – It decrements all the 5 members by 1 individually. This is pre-decrement operation.

Overload the relational operators

- ==(compare if equal) – Compare two `coinsBox` objects if they are equal. `(compare: if (coinBox1 == coinBox2))`
- !=(compare if not equal)- Compare two `coinsBox` objects if they are not equal. `(compare: if (coinBox1 != coinBox2))`
- <(compare if less than) – Compare if `coinBox1 < coinBox2`
- >(compare if greater than) – Compare if `coinBox1 > coinBox2`
- <=(compare if less than or equal to) – Compare if `coinBox1 <= coinBox2`
- >=(compare if greater than or equal to) – Compare if `coinBox1 >= coinBox2`

Overload the assignment operator

```
coinsBox &operator=(const coinsBox &);
```

Copy the contents of coinBox1 to coinBox2 (`coinBox2 = coinBox1`)

Provided files:

americanCoinType.h
main.cpp
CS202_Assignment6.pdf
sample_output_ast6.pdf
executable file - ast6

You have to complete the implementation file.

Write all the definitions of the functions in americanCoinTypeImp.cpp. The function prototypes are in header file. Do not make any changes to the header file as well as main.cpp file.

Guidelines for submission:

File 1: americanCoinType.h
File 2: main.cpp
File 3: americanCoinTypeImp.cpp - zip the above three files and submit the zip file on Canvas(section you enrolled)

To compile:

```
g++ -g -std=c++11 -Wall -Wpedantic americanCoinType.cpp main.cpp -o output6
```

To check memory leaks:

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20
--track-fds=yes -s ./output6
```

Sample_Output:

Please check the sample output, which is provided in a separate file. Your program should match the sample output. The program has big menu and the menu should be displayed after each and every operation. However, you no need to worry about that, as the main.cpp includes that logic. In sample output, I will exclude the menu display after each operation to cut down the number of pages. But for checking the functionality of program, menu display can be handy.