**Implement the following exercises.**

1. Write a class **Time** which represents time. The class should have three fields for hours, minutes and seconds. It should have constructor to initialize the hours, minutes and seconds. A method printTime() to print the current time.

Overload the following operators:

plus operator (+) (add two time objects based on 24 hour clock)  
and < (compare two time objects)

1. Define a class **Set** that stores a finite set of integers. Supply add ( ) and remove ( ) member functions to add and remove set elements. Overload the | and & operators to compute the union and intersection of the set.
2. Complete the following tasks:

a. Design a **Meal** class with two fields—one that holds the name of the entrée, the other that holds a calorie count integer. Include a constructor that sets a Meal’s fields with parameters, or uses default values when no parameters are provided.

b. Include an overloaded insertion operator function that displays a Meal’s values.

c. Include an overloaded extraction operator that prompts a user for an entrée name and calorie count for a meal.

d. Include an overloaded operator+()function that allows you to add two or more Meal

objects. Adding two Meal objects means adding their calorie values and creating a summary Meal object in which you store “Daily Total” in the entrée field.

e. Write a main()function that declares four Meal objects named breakfast, lunch, dinner, and total. Provide values for the breakfast, lunch, and dinner objects.

Include the statement total = breakfast + lunch + dinner; in your program, then display values for the four Meal objects. Save the file as **Meal.cpp**.

f. Write a main()function that declares an array of 21 Meal objects. Allow a user to enter values for 21 Meals for the week. Total these meals and display the calorie total for the end of the week. (*Hint*: You might find it useful to create a constructor for the Meal class.) Save the file as **Meal2.cpp**.

1. Complete the following tasks:

a. Design a **PhoneCall c**lass that holds a phone number to which a call is placed, the length

of the call in minutes, and the rate charged per minute. Overload extraction and insertion

operators for the class.

b. Overload the == operator to compare two PhoneCalls. Consider one PhoneCall to be

equal to another if both calls are placed to the same number.

c. Create a main()function that allows you to enter 10 PhoneCalls into an array. If a

PhoneCall has already been placed to a number, do not allow a second PhoneCall to the

same number. Save the file as **PhoneCall.cpp**.

1. Complete the following tasks:

a. Design a **SoccerPlayer** class that includes three integer fields: a player’s jersey number,

number of goals, and number of assists. Overload extraction and insertion operators

for the class.

b. Include an operator>()function for the class. One SoccerPlayer is considered greater

than another if the sum of goals plus assists is greater.

c. Include overloaded stream operators to read and display the player instance.

d. Include operator =( ) function to create temporary object.

e. Create an array of 11 SoccerPlayers, then use the > operator to find the player who has

the greatest total of goals plus assists. Save the file as **SoccerPlayer.cpp**.

1. Extend the definition of the class **complexType** so that it performs the subtraction and division operations. Overload the operators subtraction and division for this class as member functions. If (a, b) and (c, d ) are complex numbers:

(a, b) - (c, d) = (a - c, b - d ).

If (c, d ) is nonzero:

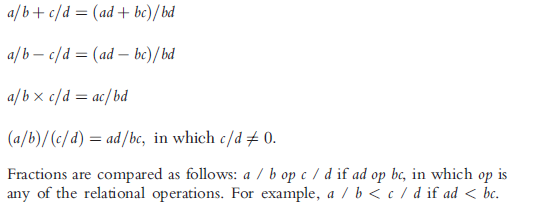
(a, b) / (c, d) = ((ac + bd) / (c2 + d 2), (-ad + bc) / (c2 + d 2)).

Write the definitions of the functions to overload the operators - and / as defined in part a.

Write a test program that tests various operations on the class complexType. Format your answer with two decimal places.

Rewrite the definition of the class complexType so that the arithmetic and relational operators are overloaded as nonmember functions.

1. Rational fractions are of the form a / b, in which a and b are integers and b not equal to 0. In this exercise, by ‘‘fractions’’ we mean rational fractions. Suppose a / b and c / d are fractions. Arithmetic operations on fractions are defined by the following rules:



Design a class—say, **FractionType**—that performs the arithmetic and relational operations on fractions. Overload the arithmetic and relational operators so that the appropriate symbols can be used to perform the operation. Also, overload the stream insertion and stream extraction operators for easy input and output.

Write a C++ program that, using the class FractionType, performs operations on fractions.

Among other things, test the following: Suppose x, y, and z are objects

of type fractionType. If the input is 2/3, the statement:

cin >> x;

should store 2/3 in x. The statement:

cout << x + y << endl;

should output the value of x + y in fraction form. The statement:

z = x + y;

should store the sum of x and y in z in fraction form. Your answer need

not be in the lowest terms.

8. Complete the following tasks:

a. Create a **PhoneBook** class. Fields include first name, last name, area code, and phone number.

b. Include an extraction operator that prompts the user for values for each field. Also include an insertion operator that displays the values of each field.

c. Overload the operator[]()function to change a PhoneBook object’s phone number (but

not area code).

d. Overload the operator()()function to change the area code and phone number.

e. Write a main()function in which you declare an array of five PhoneBook objects and assign data to each. Using a loop, display numbers 1 through 5, along with each object’s data.

Prompt the user to select a number 1 through 5 to modify a PhoneBook entry. When the user has chosen an entry, ask whether the user wants to alter an entire entry, alter the entire phone number including the area code, or alter just the phone number and not the area code.

Accept new data values accordingly. If the user wants to modify an entire entry, create a temporary object, assign values, and use the built-in = operator to assign the temporary object to the correct location in the array. If the user wants to change the area code and phone number, or change the phone number only, prompt for values, then use either the [ ] or () operator to assign the new values to the proper existing object within the array. After the update has taken effect, redisplay the five PhoneBook entries. Save the file as PhoneBook.cpp.

1. You have been developing a **BankAccount** class for Parkville Bank that contains several fields and functions.

a. Add arithmetic operators that operate as follows:

» The += operator takes a double parameter, which can represent a deposit (or credit)

to be added to the BankAccount balance. (If the parameter is negative, it will represent

a withdrawal or debit from the account.)

» The + operator takes an integer parameter that increases the account number, returning

a new BankAccount object with the new account number.

b. Include < and > operators that determine whether one account is less than or greater than another. These comparisons are based on balances. Include an == operator that makes a comparison based on account numbers.

c. Include extraction and insertion operators for the class.

d. If you do not already have one, create a default constructor that assigns 0 to all the class fields.

e. Write a main()function that declares an array of five BankAccount objects. (Use the default constructor, so all the data fields will be initialized to 0s.) For each account, prompt the user for a series of transactions. The user enters a positive number for each deposit, a negative number for each withdrawal, and a 0 to quit entering transactions for an account. After the user has entered transactions for all five accounts, display the resulting details for each BankAccount. Save the file as BankAccountPartE.cpp.

f. Alter the main()function so that the user is prompted for starting account data instead

of accepting the default 0 values. Save the file as BankAccountPartF.cpp.

g. Modify the main()function so that no two accounts are allowed with the same account number. Reprompt the user for a new account number when a duplicate occurs. Save the file as BankAccountPartG.cpp.

h. Modify the main()function to display the data for the accounts with the highest and lowest balances after all the data has been entered. Save the file as BankAccountPartH.cpp.

**10. Friend function**

Create two classes. The first, named **Sale,** holds data for a sales transaction. Its private data members include the day of the month, amount of the sale, and the salesperson’s ID number. The second class, named **Salesperson**, holds data for a salesperson, and its private data members include each salesperson’s ID number and last name. Each class includes a constructor to which you can pass the field values.

a. Create a friend function named display()that is a friend of both classes and displays the date of sale, the amount, and the salesperson ID and name. Write a short main()demonstration program to test your classes and friend function. Save the file as Sales.cpp.

b. Add a function to both the Sale and Salesperson classes that returns the private salesperson ID number. Write a main()function that contains an array of five

Salesperson objects and store appropriate data in it. Then, continue to prompt the user for Sale data until the user enters an appropriate sentinel value. For each Sale transaction entered, determine whether the salesperson’s ID number is valid. Either display an error message, or use the friend display()function to display all the data. Save the file as Sales2.cpp.