CS 11 – Homework Assignment #1 – Due Thursday 16 March 2000

Below, I have described two problems that you are to solve using a computer. In class, we discussed the overall strategy: understanding the problem, creating an algorithm, typing your ideas into a C++ program, and then running and testing the program. Every time you write a program you will be strengthening your problem-solving and programming skill. For the moment, these 2 programs are relatively short and simple. They mainly feature straightforward arithmetical operations and I/O. Later assignments will be longer and somewhat more detailed.

Begin **today** on this assignment. If there is anything written here in the specification that you don't understand, you need to ask me **today**. Procrastination is your enemy. On or before the due date, you are to submit a disk containing these two programs, plus a hard copy of each program. The names of the source files should be tripcost.cpp and addtime.cpp. One advantage to finishing early is that you can show me your program to see if there are any glaring mistakes – in time for you to still have the opportunity to correct them before the deadline.

In general, you should always test your programs thoroughly to ensure they work for all possible input data. However, in this assignment you may assume that the user will not enter invalid input (e.g. a negative distance). Style is also an important consideration. Someone reading your program should have no trouble understanding its purpose and structure. Documentation both at the beginning of the source file and within the code will be helpful. Comments I like to see are short paragraphs that describe what is happening in your code. From reading your comments, it should be clear how you solved the problem correctly.

Other notes:
As far as style goes, it is a good idea to have more than just the variables needed for I/O. You will need additional variables to perform intermediate calculations. Also, please use helpful variable names like "num_people" and "total_miles" rather than single letters like "x" and "y". Your program's I/O should be composed in complete sentences so that the user will know what your program is all about. It is also a good idea to have a special welcome message as soon as your program starts, telling the user the name and purpose of the program. In the real world, users of computer programs generally do not look at the source code, so you could repeat some of the information from your introductory documentation. But the welcome message doesn't have to be elaborate.

1. Suppose you are going on a road trip with some friends. Write a program that will compute the average cost of the trip per passenger. You, the driver, will count as one of the passengers for the purpose of calculating this average. For simplicity, the program should only consider the cost of transportation, not lodging or meals. The program should ask the user for the following information:
   a. The total number of people going on the trip.
   b. The total number of miles driven for the round trip.

c. The average fuel economy (MPG or miles per gallon) of your car.
d. The average price for a gallon of gasoline.
e. The total amount of tolls during the trip.

The output from your program will be a single monetary figure, representing each person's share of the total cost. In this program, all variables (except the number of people) should be of type **float**.

2. In this second program, we are interested in the time for a 2-lap race. The user will tell you how long each lap took, in minutes and seconds. The program will output:
   a. The total time, also in minutes and seconds.
   b. The percent of the total time taken up by each lap, shown to the nearest tenth of a percentage point (e.g. 53.2%).

   The format of both laps in the input will be *mm*:*ss*, where *mm* and *ss* are both integers representing minutes and seconds, respectively. You may assume that *ss* is in the range 00-59. The total time output from your program should also be of the format *mm*:*ss*. Note that if the number of seconds is less than 10, you will need to use the setfill function to print a leading zero.

CS 11 – Homework Assignment #2 – Due Friday 24 March 2000

There are 2 questions. For each, write a modular C++ program that solves the overall problem. Please adhere to the elements of good style I outlined in the previous assignment.

1. Do problem #12 on page 130 in the textbook. However, the book did not make clear what the input format is. Your program must ask the user for the following information:
   a. number of adults
   b. number of children
   c. cost of each adult meal
   d. cost of each dessert
   e. room fee
   f. tip percentage: if the user enters 15, this means 15% or .15. But if the user enters 0.15, this means 0.15% or .0015!
   g. deposit

   Note that there is more than one way to break this program into parts (functions). One possibility is to have a function that reads in all the input, a function that performs all the necessary calculations, and a third function that writes the output to the screen. On the other hand, you may wish to separate the meal costs (first 4 inputs) in a separate function by itself. Just make sure your output has the same format as given in the book, although you may add more of an introduction (welcome message) and you may change the name of the catering company.

2. A parking garage charges $0.50 per hour for each hour, or part thereof. For example, if you are parked for 3.3 hours, you get charged $2.00. Write a program that asks for the amount of time that three cars were parked in the lot, and then print the charges of each car, along with the total. As far as modularity is concerned:
   a. One function should handle the input, and have 3 reference parameters corresponding to the times (in hours) that the cars were parked
   b. One function should take a single amount of time, in hours, and return the charge for that customer. This function will not use reference parameters. Hint: to figure out the "part thereof" you should use the ceil() function defined in math.h.
   c. One function should handle the output, which should arrange the data in neat columns like this:

```
Car        Hours     Charge
1           1.5       1.00
2           4.0       2.00
3           6.1       3.50
total      11.6       6.50
```

   You may assume that the hours will always be measured in tenths (not hours and minutes).

Introduction to Computer Science with C++ by Lambert, Nance, Naps, PWS 1997.

12. The Caswell Catering and Convention Service has asked you to write a computer program to produce customers' bills. The program should read in the following data.
    a. The number of adults to be served.
    b. The number of children to be served.
    c. The cost per adult meal.
    d. The cost per child's meal (60% of the cost of the adult's meal).
    e. The cost for dessert (same for adults and children).
    f. The room fee (no room fee if catered at the person's home).
    g. A percentage for tip and tax (not applied to the room fee).
    h. Any deposit should be deducted from the bill.

    Write a program and test it using data sets 2, 3, and 4 from the following table:

Set	Child Count	Adult Count	Adult Cost	Dessert Cost	Room Cost	Tip/Tax Rate	Deposit
1	7	23	12.75	1.00	45.00	18%	50.00
2	3	54	13.50	1.25	65.00	19%	40.00
3	15	24	12.00	0.00	45.00	18%	75.00
4	2	71	11.15	1.50	0.00	6%	0.00

Note that data set 1 was used to produce the following sample output.

```
          Caswell Catering and Convention Service
                        Final Bill

Number of adults: 23
Number of children: 7
Cost per adult without dessert:          $    12.75
Cost per child without dessert:          $     7.65
Cost per dessert:                        $     1.00
Room fee:                                $    45.00
Tip and tax rate:                             0.18

Total cost for adult meals:              $   293.25
Total cost for child meals:              $    53.55
Total cost for dessert:                  $    30.00
Total food cost:                         $   376.80

Plus tip and tax:                        $    67.82
Plus room fee:                           $    45.00
Less deposit:                            $    50.00

Balance due:                             $   439.62
```

CS 11 – Homework Assignment #3 – Due Thursday 30 March 2000

Ice Cream Maker

In this assignment you will write a C++ program designed to help someone planning to make some ice cream. The issue here is "how much can I make?" There are five basic ingredients for vanilla ice cream: eggs, sugar, cream, milk and vanilla extract. You need certain amounts of each ingredient to make ice cream properly. For the purpose of this assignment we will assume that in order to make 1 quart of ice cream you need:

2 eggs
6 oz. of sugar
16 oz. of cream
8 oz. of milk
0.3 oz. of vanilla extract

Let's say you want to make as much ice cream as your inventory of ingredients will allow. This means that before you begin the recipe, you need to know how much of each ingredient you have. Because all 5 ingredients are necessary, each one presents a limiting condition on how much ice cream can be made. For example, if you only have 1 egg, then it doesn't matter how much of the other ingredients you have: you can only use up one egg to make ½ a quart of ice cream.

Your program will contain an input function that asks the user enter the amounts of each of the ingredients. The number of eggs will be an integer; the rest of the ingredients will each be a real (float or double) number of ounces. This should be a void function with five reference parameters.

You will also need a function that computes (and tells the user) the maximum amount of ice cream that can be made. This function should also say which of the 5 ingredients was the limiting factor.

The calculations that need to be made may not be obvious, so here is a hint. The assumptions we made about the ingredients above can be expressed as special constants in C++. For example, at the beginning of your program, you can make these declarations:

```
const int eggs_per_quart = 2;
const float sugar_per_quart = 6.0;
const float cream_per_quart = 16.0;
const float milk_per_quart = 8.0;
const float vanilla_per_quart = 0.3;
```

For each ingredient, you need to calculate how much ice cream could be made if you had an unlimited supply of all the rest. Then, you need to figure out which of these five numbers is the smallest. A large part of your code will be devoted to finding the smallest of these 5 numbers. This may seem quite tedious, but actually it can be accomplished using only 10 comparisons. (There is a mathematical reason why this is the case.) In C++, the comparisons will be organized into if-else statements.

CS 11 – Homework Assignment #4 – Due Tuesday 11 April 2000

1. The Goldbach conjecture states that every even integer greater than 2 can be written as the sum of two prime numbers. Write a C++ program that asks the user to enter an even integer, and then finds the two prime numbers that add up to this number. You will need to write the following functions within your program:
   (a) input(): returns the value entered by the user. Error checking is required to make sure the input value is even and also greater than 2. If the user makes a mistake, print an error message and get the input value again.
   (b) is_prime(): takes an integer parameter and returns 1 if it is prime, 0 if it is not. Note that a prime number is a number that has exactly two factors, namely 1 and itself.

   Important note: In many cases there is *more than one* pair of prime numbers that sum the input value. Given this possibility, your program should find all such pairs of prime numbers. But to avoid duplicate output, make sure that the first prime number is less than or equal to the second. For example, if the input number is 16, your program should print the following as output:

   16 = 3 + 13
   16 = 5 + 11

   Do not print 13 + 3 or 11 + 5 because these are the same sets of prime numbers but in a different order.

2. Write a C++ program that computes the length of the longest day for a given range of latitudes. You will need to ask the user to enter a minimum latitude (e.g. 30) and a maximum latitude (e.g. 50). For a known latitude, the formula for length of the longest day is:

   hours = 12 + 24/π * asin (tan (deg2rad(latitude)) / tan (deg2rad(66.55)))

   Note that the asin() and tan() functions are defined in math.h, which you need to include in your program just like iostream.h. The number 66.55 refers to the latitude of the sun at the summer solstice.

   You need to write the function deg2rad(). It converts from degrees to radians, but the conversion is quite simple: just mulitply by π/180.0 and then return this value. The number π should be a "const double" declared at the top of your program (near the function prototypes) equal to 3.14159.

   The program should contain an input function that gets the two latitudes from the user. Beware of invalid input. The latitude numbers must be in the range 0..90, and the second value must not be less than the first. If the user makes a mistake, print an error message and try again.

Your program should contain a function find_length() that implements the above formula for calculating hours. Its parameter will be a particular latitude, and the return value will be the number of hours. Note that this function will be called from inside a loop since we are finding the day length for each latitude. In case the latitude is greater than 66, the length of day will be 24 hours (the midnight-sun effect).

You will also need a function that prints the length of the day in the hh:mm:ss format, given a real number of hours as a parameter. Your output should be in the form of a table just like you did for the parking lot program.

CS 11 – Homework Assignment #5 – Due Wednesday 26 April 2000

Creating the Time Class

In this assignment you will practice object-oriented program development first-hand. All the previous programs you wrote were just single files. By contrast, the program you will be working with this time will consist of 3 files, just like the recent examples we have gone over in class. In a nutshell, I am providing you with the third file (main program using the class) and you have to make the first two files (declaring and implementing the class.)

In this assignment you will be creating a "time" class. An object of this class will simply be the time of day, for example 2:15 pm. The attributes and operations of this class will be described below.

The program itself is called "testtime". It will consist of these 3 files: time.h, time.cpp and testtime.cpp. I have attached a copy of testtime.cpp. All this main program does is test all the features of the time class. No modifications to testtime.cpp are necessary, but you do have to create the other two files:

1. Make a file called time.h. This will contain the declaration of a class called "time" containing the following attributes and operations:
   (a) a default constructor
   (b) an initial value constructor that takes 3 parameters: an integer representing the hour, an integer representing the minute, and a character indicating 'a' for am and 'p' for pm.
   (c) a copy constructor
   (d) a member function is_daytime, that takes no parameter, and returns 1 if the time is between 6:00 am and 6:00 pm *inclusive*, and returns 0 otherwise
   (e) the == operator, that returns the value 1 if the two times are equal, and returns 0 otherwise
   (f) the < operator, that returns 1 if the first time is earlier than the second, and returns 0 otherwise
   (g) the = operator
   (h) the >> operator
   (i) the << operator
   (j) data attribute(s) internally representing the time. For simplicity, I recommend you use just a single integer to represent time, either in a military time format, or counting the number of minutes after midnight. Such a design will simplify the implementations of most of the functions.

2. Make a file called time.cpp. This file will contain the implementations of all the functions listed (a) – (i) above.

Hint: In case you get stuck with some of the implementations, look at our other class examples we've gone over recently, especially the card class. The code should be somewhat similar, especially in structure. For example, in the implementation of the = operator, you will always finish with a line saying "return *this". Your >> operator will return an object called "is", and your << operator will return an object called "os", just like we did in the other classes.

```
// testtime.cpp -- main program that uses time class

#include <iostream.h>
#include "time.h"

int main()
{
  time t1;               // invoke default constructor
  time t2(7,30,'a');     // invoke initial-value constructor
  time t3(t2);           // invoke copy constructor

  cout << "Time t1 = " << t1 << endl;
  cout << "Time t2 = " << t2 << endl;
  cout << "Time t3 = " << t3 << endl;

  time t4, t5, t6;

  cout << "Please enter a time, e.g. 2:15 pm:  ";
  cin >> t4;

  t5 = t4;
  cout << "Value of time t5 equals:  " << t5;

  if (t5.is_daytime())
    cout << "  which is during the day.\n";
  else
    cout << "  which is during the night.\n";

  cout << "Please enter another time:  ";
  cin >> t6;

  if (t5 == t6)
    cout << "Your two times are the same.\n";
  else if (t5 < t6)
    cout << "Your first time is earlier than the second.\n";
  else
    cout << "Your first time is later than the second.\n";

  return 0;
}

// Example I/O
//
// Time t1 = 12:00 am
// Time t2 = 7:30 am
// Time t3 = 7:30 am
// Please enter a time, e.g. 2:15 pm:  8:19 pm
// Value of time t5 equals:  8:19 pm  which is during the night.
// Please enter another time:  12:35 pm
// Your first time is later than the second.
```

CS 11 – Homework Assignment #6 – Due Tuesday 9 May 2000

Input File Oddities

In this assignment you will practice some of the techniques of file I/O we discussed in chapter 9. You will write a program that asks the user to enter the name of some input file. If the file cannot be opened, print an error message, and ask the user to try again. Once the file is successfully opened, the program will read the text file, and determine if this file contains certain unusual features. For each unusual feature you find, tell the user on which line it was found. There are 4 unusual features you should be on the lookout for:

1. Blank lines (two consecutive newline '\n' characters)
2. Lines containing more than 80 characters (not including the newline character)
3. Lines containing one or more tab ('\t') characters
4. Lines containing one or more space (' ') characters at the end of the line

If a particular input line contains one or more tab characters, it is not necessary to count how many there are on that line: it is only important whether or not there are any tab characters. The same applies for trailing space characters as well.

It may be more convenient for you to read file input using the get() member function that belongs to the ifstream class. Recall that the >> operator ignores newline, tab and space characters. If you decide instead to use the getline() function, you may assume that the maximum length of an input line is 1000 characters.

The output from your program should look something like this:

```
Line 4 is blank.
Line 6 contains more than 80 characters.
Line 6 contains one or more tabs.
Line 11 contains more than 80 characters.
Line 13 is blank.
Line 14 contains one or more trailing space characters.
Line 15 is blank.
Line 16 contains one or more tabs.
```

CS 11 – Homework Assignment #7 – Due 3pm Wednesday 17 May 2000

Educational Researcher

The purpose of this assignment is to combine your experience with file I/O along with the new techniques of using arrays that we have been talking about lately. You will write a program that reads a text file "97.in" that contains data on public schools in the United States. The program will then interactively ask the user to enter a 5-digit ZIP code. Finally, the program will print a sorted list of all schools that are located in this ZIP code.

You can access the file 97.in as follows. From one of the PCs in the lab, click on the icon "Home on Servercsdept2", then click on the folder "Spring 2000", and then click on "CS11A-OUT". You should see the 97.in file there. However, this is a very big file that will not fit on your disk. For convenience, you may wish to copy it to the desktop so that it will be called C:\windows\desktop\97.in .

As you read the lines of this big text file, there are 3 items you need to keep track of for each school: the name of the school, the ZIP code where it's located, and its total enrolment. Refer to the separate handout to see exactly where (which columns) on the line these data can be found. If my counting is correct, there are 88673 schools listed in this file. The way to keep track of such a huge volume of information is to use arrays. You may either use 3 arrays, one for the names (strings), one for the ZIP codes (longs) and one for the enrolments (ints), or you may choose to have a single array of class objects. If you choose this latter option, this would entail writing an object-oriented program in which each school object will have 3 attributes.

The output is to be sorted in descending order of size of school. The first school listed should be the one with the highest enrolment, and the last school listed should have the smallest enrolment. Each line of output should just show the name of the school and its enrolment. There is no need to print out the ZIP codes of the schools because they will all be the same. For instance, if the user asks for a list of schools in ZIP code 23320, only these schools will be listed and no others. Please make every effort to make your output have a professional appearance: in other words, the names and enrolments should be organized into two columns. Later today I will provide you an executable file in the "CS11A-OUT" folder that you can copy and run to see what my output looks like.