CSIS 123
LABORATORY EXPERIMENTS

The work in this laboratory reinforces the concepts covered in *CSIS 123, Programming Fundamentals.* Each project teaches a new feature of the C++ language.

PROJECT	DESCRIPTION
1	FAMILIARIZATION
2	TEMPERATURE CONVERSIONS
3	FUNCTION APPLICATIONS
4	NUMBER SORT
5	ADDRESS BOOK
6	PERMANENT ADDRESS BOOK

LABORATORY REPORT CONTENTS

Every student will turn in a complete, individually written lab report for each experiment performed in CSIS 123. The content of a formal laboratory report is expected to be as follows. (If you need to deviate from this format, please check with the instructor first.)

a) A COVER PAGE containing:

- Your name
- Your class and semester (CSIS 123 - - Spring 2011)
- Experiment Title
- INSTRUCTOR: TOM WHEELERe) Due Date of report (Week # or date given in class)
- Operational sign-off blank
- Final sign-off blank.

HINT:

Make sure to have a completed cover page when you start each lab. No sign-off can be given without a proper cover page. **Make sure to have a completed cover page when you start each lab**. No sign-off can be given without a proper cover page.

A sign-off will be given only during the scheduled laboratory period. The sign-off verifies that your program works correctly. No credit is given to any report lacking a sign-off. You must present your program listing and demonstrate correct operation to obtain a sign-off. Your name must be at the top of all program listings.

LABORATORY REPORT CONTENTS (CONTINUED)

b) INTRODUCTION - The introduction provides written explanation of what the experiment is designed to do (and what is expected as a learning outcome).

c) THEORY OF OPERATION - This section is a walk-through of the C++ code. The primary intent of this section should be to give an overview of how the program works. Do not document each line of code! Instead, document each major activity your program takes as it completes its task.

d) PROGRAM LISTING - This will be a complete listing of the program. All code must be properly commented: All methods/functions must have a comment header giving the method name, purpose, parameter list, return value, and a description of each parameter and return item value. Each major idea should be commented, rather than individual instructions.

e) CONCLUSION - This section provides a brief explanation of what can be concluded from the activities of the experiment. It is not a summary. It must be based on the information you've worked with during the experiment and it may also comment on topics such as the efficiency/effectiveness of various software methods used to achieve the experimental goals. It is also appropriate to reflect on your own learning in this section.

Important: Remember that *you* are to be the author of all laboratory work; refer to the MCC Academic Integrity policy in the online Student Handbook. The laboratory reports are to be written by *you*. Do not just copy or paraphrase the text from the laboratory instructions, laboratory example, or Internet. Do not share your writing with other students. Do not "loan" your writing to a fellow student to "help" them write a report. The written analysis must reflect *your* ideas and interpretations.

Course Policies

I. Lab Partners: There are *no* lab partners allowed in CSIS 123. Each project and its written report are to be completed individually. You are responsible for adhering to the College's code of academic conduct.

II. Handing Work In: All work should be given directly to the instructor.

III. Late Work: Laboratory reports are accepted up to one week late, with a 5 point deduction in score for each day late. ***Reports more than one week late are assigned a grade of zero***.

IV. Lab Success Hints: The successful student will have worked through the majority of the source code before entering lab. Laboratory time should be used to proper advantage; this is the main time that the instructor will be available to assist in troubleshooting and debugging, and it is the only time that sign-offs will be available. Please plan your activities accordingly.

V. Plagiarism: *Copying the work of another, and claiming it to be your own is* **plagiarism**. This includes (but is not limited to) copying others homework, copying from a lab manual or textbook, or collusion. The minimum penalty for cheating in any form is a grade of zero for the element involved; in some cases, failure of the course and/or expulsion from the College will also result. **All cases of misconduct will be documented and forwarded to the College administration for disciplinary consideration**.

Please do not turn in any work that is not your own! If in doubt, ask the instructor. Here are some ways to avoid any problems:

- Don't share your computer files (text, C/C++ source/object, etc) with anyone else.
- Don't share media (flash drives, CDs, net/cloud storage points) with other students; it's too easy to get files mixed up.
- Don't copy answers from a neighbor. If you don't understand how to do it, ask!
- Decline any request from fellow students for a copy of your work. Anybody needing this level of help should ask the instructor.

GOOD DATA PROCESSING PROCEDURES

Computers will be used extensively in this lab. The student can expect to spend many hours creating and updating C++ programs; loss of this data can be disastrous! The following tips will help to minimize the chance of losing a project:

- Make frequent backups. These backups should be in at least two different physical locations. FAILURE TO KEEP A BACKUP OF YOUR DATA IS NOT AN EXCUSEABLE REASON FOR LATE OR MISSING WORK.

- Always keep schoolwork on two different media; even better, e-mail critical work to yourself to maintain an archive copy. (Note that in industry, e-mail will normally not be used to archive code due to its insecure nature.)

- Don't save your data to the hard disk on the workstation, except in an emergency. The hard disks on lab workstations are periodically "cleansed" of any extra information as part of a housekeeping program. (MCC Labs may be set up with "Deep Freeze," which is a program that restores default computer settings each time a user logs off - - so be aware that anything you save to the computer may disappear once you log off the machine.)

- Write your name on removable media (such as USB flash drives). It's easy to accidentally leave these behind in the lab. If your name is marked on the media, it's much easier to return it to you.

- Don't share your computer with others, especially the one you're using to develop code.

- Remember that at some time in the future, one or more of your C++ programs may become dangerous to the general health and well being of your computer. An errant C++ program can easily wipe a hard disk clean, and in some cases, can damage the hardware. Be guided accordingly.

PROGRAM DOCUMENTATION

Program documentation is information that is directed at a technical person who may be required to maintain your code. Later, someone (possibly you!) will most likely have to make corrections or additions to your program. The quality of program documentation (and the basic structuring of your code) largely determines how easy (or difficult) this task will be.

Documentation is generally done in at two forms. For involved projects, the **flowchart** is an excellent method for capturing the train of events inside a program. The flowchart gives a conceptual, "bird's eye" view of what is happening. Flowcharts should avoid specific details such as "increment the R register" unless such detail is necessary to understanding the overall action of the program. A more suitable comment might be "Add $1.00 to account balance," since this directly states what the desired effect is. In other words, flowcharts are more concerned with *what* is happening rather than *how* the machine mechanically achieves it.

The second form of documentation is the program **listing**. The program listing is a detailed explanation of the mechanics behind the execution of a task. It should be *appropriately* and *thoroughly* commented. In C++ programs, blank lines should be inserted to separate parts of a task that are logically separate, and indentation should be used where appropriate. Consider the two pieces of code below:

```
// This poorly-styled code actually works!
int my_function(x,y)
int x,y;
{int alpha,z;alpha=0;
  for(z=1;z<10;z++) {alpha=alpha+particle(x,y*y,z);y--;}
return(alpha);}
```

This code will compile and execute just fine, but it's difficult for humans to understand and maintain. The purpose of this code (and how it fits into the context of the whole program) is impossible to determine by mere inspection. It would be easier to understand if it were written like this:

```
////////////////////////////////////////////////////////////////////
// Function: int my_function(int x,int y)
//
// Inputs: integers x and y.  x is the RMS mass of the particle and y
//         is its current velocity in Kilometers/uS.
// Returns: an integer with an approximation of the decelaration of the
//          particle in Kilometer/uS^2
//
// Errors: There are no error conditions possible for
//
////////////////////////////////////////////////////

int my_function(int x,int y)
{
int alpha,z;
```

Note how the C++ comment character sequence "//" makes a nice "box" for the function header!

```
alpha=0;    // initial acceleration is assumed to be zero.

/* Compute the approximate acceleration using BIG AL's iterative method.
   Reference: CAR AND DRIVER magazine, July 1990, pp 34-35. */

for(z=1;z<10;z++)
      {
      alpha=alpha+particle(x,y*y,z);
      }

return(alpha);
}
```

It's OK to mix C and C++ comment styles, as long as you're consistent. Note how *each algorithm is explained to the reader.*

Consistent use of indents improves readability.

Note that every instruction in a program listing isn't necessarily commented. Machine instructions, in and of themselves, are generally self-evident because of the use of mnemonics. The documenting comments therefore serve to illuminate individual ideas or concepts that are important for understanding *how* the machine is accomplishing the task at hand.

Every function you define should have a *header* comment. This important field describes what the function does, what inputs it uses, and what outputs it produces. It should be written as if you were explaining how to use the function to another programmer.

Appropriate placing of comments, program elements, and use of indentation all help to make source code more easily understood.

PROJECT DESCRIPTIONS

For a program to be accepted, it must meet the minimum specifications for each project. Extra functionality, as a function of your own creativity, is encouraged, as long as it does not detract from the original purpose of the project (or make it difficult to evaluate or analyze.)

In most cases, the *inputs* and *outputs* of the program will be specified. Your program should have the exact inputs and outputs that we specify, otherwise it becomes difficult for us to check your program's operation.

PROJECT 1: FAMILIARIZATION

This is an exercise to make sure you can use the editor and compiler. Below is a "C" program. Use the editor to place it into a file called MyFirst.cpp, and then compile it into a working program.

P.S: (Possible Scuttlebutt): There may be some errors in this program. Please fix them before compiling!

```
*/  This is my first C++ program.  It will print the

    message "Hello, World" and then print the sum and

    product of two and five. /*



#incude <stdio.h>

//////////////////////////////////////////////////////
// main() -- entry point for the MyFirst application
//
//////////////////////////////////////////////////////

void main()
{
int a,b;

a=5;

b==2


pantf("\nHello, World!\n\n);

printf("The sum of two and five is %d \n\n", a+b );

printf("The product of two and five is %d \n\n," a*b );

// End of program number 1
//
//
```

PROJECT 2: TEMPERATURE CONVERSIONS

This program will produce a conversion chart to the user's specification for the three major units of temperature measurement, Celsius, Kelvin, and Fahrenheit.

The program will begin by asking the user for the range of temperatures (starting and ending temperatures) in degrees Fahrenheit. It will then produce a neatly formatted table giving the Fahrenheit temperature and equivalent Celsius and Kelvin temperatures for each point in the list. The table must look like this:

```
Starting Temperature: 74 F            Ending Temperature: 80 F

       Degrees F       Degrees C        Degrees K
       ----------------------------------------
       74                23.333          296.483
       75                23.888          297.039
       76                24.444          297.594
       77                25.000          298.150
       78                25.555          298.706
       79                26.111          299.261
       80                26.667          299.817
```

Formatted PRINTF statements will be used to force the decimal points to line up within the table, and three digits will be displayed to the right of the decimal points on the C and K columns. The F column will be computed and displayed as an INTEGER result with 1-degree F "steps" between values.

PROGRAM INPUTS: Starting and ending temperatures, degrees F

PROGRAM OUTPUT: Correctly formatted table as above.

IMPORTANT INFORMATION:

- °C = (°F-32)*(5./9) , °K = °C + 273.15

- You should include the statement "#include <math.h>" at the start of your program so that the compiler knows that the math functions are double-precision.

PROJECT 3: FUNCTION APPLICATIONS

By dividing a program's functionality into subroutines or functions, it becomes much easier to program tasks. Often an algorithm will appear terribly complicated until it's broken down into easier-to-digest parts. Your task in this project is to devise a program that accomplishes a task using C++ functions.

1. The choice of task is totally up to you. Pick something that's interesting to you if possible, and get instructor approval before proceeding.

2. Some program ideas that are suitable for this project include:

   a) Build a simulator that rolls a pair of dice (using the `rnd()` function to generate the random numbers), and allows the user to play a simple game based on the dice roll or rolls.

   b) Simulate the Tower of Hanoi problem (see the description in the textbook on Page 261).

   c) Devise a game simulation that depends on chance such as Craps or Blackjack. Keep track of the user's bank account and winnings throughout the run.

   d) Devise a simple Computer Aided Instruction (CAI) application as described in the textbook on Page 265. Your CAI application should tell the user the percentage of correct answers at the end of the program run.

   e) Devise a application that plays a simulated game of football, baseball, or another sport of your choice, and gives a play-by-play rundown on how each team is scoring.

PROGRAM INPUTS: Determined by the task.

PROGRAM OUTPUT: Determined by the task.

DESIGN CRITERIA: The program must use a minimum of four different functions to achieve its task. The program structure must utilize looping and decision making based on the game state and user's inputs. (In other words, a program that simply performs a single calculation and terminates is not sufficient.)

Be careful to keep the task as simple as possible when thinking about what you'll simulate!

PROJECT 4: NUMBER SORT

In this exercise, you will write a program that takes a list of 10 floating point numbers, and sorts them into low-high or high-low order (end-user choice). The program will then print the sorted list, and report the AVERAGE, MAXIMUM, and MINIMUM values encountered in the list.

DESIGN CRITERIA: You must use a `double` array in this program to hold the numbers; this array may not be global in scope. **You must use three functions**; one to INPUT the values into the array, one to SORT the array (in desired order), and a third to OUTPUT the sorted array. The array must be passed to the SORT, INPUT, and OUTPUT functions by use of a pointer. This is a requirement for sign-off. The SORT function you write should use a *bubble-sort* algorithm.

PROGRAM INPUT: The ten numbers to be sorted.

PROGRAM OUTPUT: The sorted list of elements, and the **average**, **maximum**, and **minimum** element values.

TIP: If you look closely at the output of your sort routine, you'll see an easy way of extracting the minimum and maximum values from the list.

PROJECT 5: ADDRESS BOOK

In this exercise you'll design a simple address book application using C++ structures to store the data. The book must hold 100 (or more) entries. The following commands must be supported:

Add an entry to the address book
Browse the address book
Delete an entry from the address book
Search for an entry in book by any field of your choice

When it is launched, the program will present a menu with the above choices, and will also display the number of entries in the book. The minimum fields for each entry are as follows:

Name (Up to 64 characters)
Address (Up to 128 characters)
Telephone Number (Format checked by your program; (xxx)-xxx-xxxx)
Comment (Up to 128 characters)

DESIGN CRITERIA: The program must store the data in a C++ structure array. This array may be a global variable on your program, or it may be dynamically allocated in memory. Each major component in the program must be contained in a separate function. Your `main()` function may contain a maximum of only 10 C++ statements. The program is not required to retain results between runs.

PROGRAM INPUT: Data and commands from the end user.

PROGRAM OUTPUT: Menus and information requested by the user.

TIPS:

- Use character arrays for all fields.
- Use `gets()` for all character-string input so that spaces are ignored.
- The program menu should be in a function called by `main()`, with each item on the menu implemented as a sub function.
- Remember that `strcmp()` is case-sensitive. You will need to write a function to convert all strings to upper case, or use `strcmpi()` instead. The library function `strncmp()` can be used to look for partial matches during search, if desired.

PROJECT 6: REFINED ADDRESS BOOK

In this final project you will continue development of the address book application you developed in Project 5 to provide additional capabilities:

- The program will gain the ability to store its data to a disk file (serialization), so that the data is not lost when the program ends.
- The program will also gain full sorting capability by exploiting the C++ qsort() library function.

The addition of these two features will make the program an actual useful application, instead of just a laboratory curiosity.

DESIGN CRITERIA: The program will use binary file I/O to directly write the data in the structure array to the disk. To perform the sorting, the `qsort()` library routine will be utilized.

PROGRAM INPUT: Data and commands from the end user.

PROGRAM OUTPUT: Menus and information requested by the user.