Session 3

Student Name _____

Other Identification _____

Data Storage and Its Implications

The goals of this laboratory session are to:

1. Investigate the storage systems used for three primitive data types.
2. Learn how these storage systems are reflected in the execution of C++ programs.

Your instructor will tell you which of the proposed experiments you are to perform.

In preparation for this laboratory session, you should read Chapter One of *Computer Science: An Overview.*

Data of Type Integer

Recall from Chapter One of *Computer Science: An Overview*, that data items of type integer are normally stored in memory by using two's complement notation and that this imposes a limit on the size of the values that can be represented. For example, with a two's complement system using only four bits for storage, the largest value that can be represented is seven. Of course, if this was the largest integer your machine could represent, it would be of little use. Thus, today's computer systems use more than four bits for integer storage and can store rather large values. But limitations still exist.

In the C++ system, the maximum and minimum integer values are represented by the system defined constants of INT_MAX and INT_MIN, respectively.

Experiment 3.1

In this experiment you will investigate the storage of integer values on your own particular machine.

Step 1. Execute the following program (CP03E01), and record the value of INT_MIN.

```
#include <limits.h>
#include <iostream.h>

void main(void)
{
  cout << "The most negative value of type int \n";
  cout << "represented on this C system is ";
  cout << INT_MIN << endl;
}
```

_____

_____

Step 2. Note that this program contains the statement

```
#include <limits.h>
```

This directive tells the compiler to include the header file limits.h during compilation. This file contains the information required for the compiler to understand references to INT_MIN. Try to compile and execute the program without this statement, and record what happens.

_____

_____

Step 3. Modify the program in Step 1 to find the value for INT_MAX. Record what you learn.

_____

_____

Step 4. Since integers are represented in two's complement notation, the value of INT_MAX and INT_MIN should be one less than some power of two (2^n - 1, for some integer n). Explain why.

Step 5. For what value of n is the value of INT_MAX on your machine equal to 2^n - 1?

Step 6. Use the information collected in Steps 1 through 4 to determine the number of bits used in your machine to represent data of type integer. Explain your answer.

Experiment 3.2

In this experiment you will determine how your computer system responds to programs that try to produce integer values that exceed the limits of INT_MIN and INT_MAX.

Step 1. Execute the following program (CP03E02), and record the results.

```
#include <limits.h>
#include <iostream.h>

void main(void)
{
 int number, i;

 number = INT_MAX - 10;
 i = 0;
 while (i <= 20)
   {
      cout << number << endl;
      number++;
      i++;
    }
}
```

Step 2. Briefly explain the results obtained in Step 1.

Step 3. Modify the program from Step 1 to observe, and record what happens on your system when you try to represent integers smaller than INT_MIN.

Step 4. Explain the results obtained in Step 3.

Data of Type Real

As with the storage of integers, only a finite number of values of type real can be represented in a computer's memory. Other values must be rounded to a value that can be represented. Thus, a value stored as type float is often merely an approximation of the desired value. The header file float.h provides the predefined constants FLT_MIN and FLT_MAX to set the boundaries of representation for floating-point storage.

Experiment 3.3

Step 1. Execute the following program (CP03E03), and record the value of FLT_MAX.

```
#include <float.h>
#include <iostream.h>

void main(void)
{
  cout << "The largest floating-point value that can \n";
  cout << "be represented is " << FLT_MAX << endl;
}
```

Step 2. Modify the program in Step 1 to print the value of FLT_MAX - 10. Compare the value obtained to the value of FLT_MAX, and explain your findings.

Experiment 3.4

Step 1. Execute the following program (CP03E04), and record the results.

```
#include <float.h>
#include <iostream.h>

void main(void)
{
 float x;
 int i;

 x = 1.0;
 i = 0;
 while (i < 1000)              // You may want to alter the
    {                                 // limiting value of this loop.
     cout << "x = " << x << endl;
     x = x + .0001;
     i++;
    }
}
```

Step 2. What would the program in Step 1 produce if all values were represented accurately?

Step 3. Explain the discrepancy between your answers in Steps 1 and 2.

Data of Type Character

In C++, data items of type character are stored as bit patterns according to the ASCII code, which is summarized in Appendix A of *Computer Science: An Overview*. Thus, the letter a is stored as the pattern 01100001.

The following program (CP03E05) illustrates how you use a function in this case the function is toascii, which, given a character, provides the ASCII code that represents that character. The function is called by its name followed by a list, called a parameter list, enclosed in parentheses. The entries in the list are called parameters and are separated by

commas. The toascii function takes only one parameter and it returns an integer as its value.

Experiment 3.5

Step 1. Execute the following program (CP03E05), and record the results.

```
#include <iostream.h>
#include <ctype.h>
void main(void)
{
  char sym_1, sym_2, sym_3;
  sym_1 = 'a';
  sym_2 = '2';
  sym_3 = 'Z';
  cout << "sym_1 = " << toascii(sym_1) << " in ASCII\n";
  cout << "sym_2 = " << toascii(sym_2) << " in ASCII\n";
  cout << "sym_3 = " << toascii(sym_3) << " in ASCII\n";
}
```

Step 2. Modify the program in Step 1 to find the ASCII code for the character > and the space. Summarize your work and findings below.

Step 3. Modify the program from Step 1 to find the characters immediately before and after the characters a, 2, and Z in the ASCII order. Summarize your modifications, and record your findings below.

Step 4. Note that this program contains the statement

```
#include <ctype.h>
```

This directive tells the compiler to include the header file ctype.h during compilation. This file contains the information required for the compiler to understand references to toascii. Try to compile and execute the program without this statement, and record what happens.

Coercion

In general, coercion refers to the interpretion of data as a type other than that originally intended. A common instance of coercion occurs when adding a numeric value of type int to another value of type float. In this case, the integer value must be converted to a

floating-point representation before the addition can be performed.

In the example mentioned above, C++ performs coercion without an explicit request to do so. In other cases, the programmer must designate that coercion is to take place. In C++ terminology, this is known as casting. Casting is indicated by enclosing the variable containing the value to be coerced in parentheses and preceding it with the name of the new type. Thus, if sym is a variable of type char, then the expression int (sym) would produce the integer value obtained by interpreting the current bit pattern associated with sym as an integer.

In general, relying on coercion is not considered good programming style. The argument is that if data types are chosen correctly, then there should be no need for coercion. On the other hand, there are times when coercion is a logical way of handling a problem. In these cases it is best to document the fact that coercion is taking place. In this light, many software designers insist that casting be used, even in those cases in which C++ would perform the coercion anyway.

Experiment 3.6

Step 1. Execute the following program (CP03E06A), and explain the results.

```
#include <iostream.h>

void main(void)
{
 int x;
 char y;
 float z;

 x = 5;
 y = 'a';
 z = float (y) + float (x);                    // explicit casting
 cout << z << endl;
 z = x;                                        // C++ does coercion
 cout << z << endl;
}
```

Step 2. Execute the following program (CP03E06B), and explain the results. In particular, what coercion is taking place?

```
#include <iostream.h>

void main(void)
{
 int x;
 x = 65;

 while (x < 69)
    {
     cout << char (x) << endl;
     x++;
    }
}
```

Experiment 3.7

Step 1. Predict the results of executing the following program (CP03E07).

```
#include <iostream.h>
#include <ctype.h>

void main(void)
{
 char letter, another_let;

 letter = 'a';
 another_let = 'Z';
 cout << "a = " << toascii(letter) << endl;
 cout << "Z = " << toascii(another_let) << endl;
 letter = letter + 1;
 another_let = another_let - 1;
 cout << letter << " = " << toascii(letter) << endl;
 cout << another_let << " = " << toascii(another_let)<< endl;
}
```

Step 2. Confirm your predictions by actually running the program from Step 1. What happens?

Step 3. Modify the above program, so that the coercion taking place is indicated by casting. Summarize your work below.

Step 4. Rewrite the program above so that it prints the integer (ASCII code) values for all the lowercase letters of the alphabet. Summarize your work below.

Post-Laboratory Problems

3.1. What if you wanted to write a program to manipulate integers lying outside the range of INT_MIN and INT_MAX? The C++ language provides the primitive data type long for such an occasion. A variable of type long can typically hold an integer greater than INT_MAX or less than INT_MIN. Although long has a greater range of representation compared to int, it too has its limitations. Find the values for the system-defined constants LONG_MAX and LONG_MIN.

3.2. In the same way that long serves as an extended integer type (see Problem 3.1), the type double allows for more precision when float is insufficient. Find the limitations for a variable of type double by uncovering the values for the system defined constants DBL_MAX and DBL_MIN.

3.3. Write a program that asks the user for a lowercase letter of the alphabet and then prints its equivalent uppercase letter.

3.4. Write a program that translates natural numbers in the range of 1 to 100 into their binary equivalents.

3.5. Write a program that accepts numbers in base two representation and displays their base ten equivalents.

3.6. Write an encrypting program that converts each character received into the next character in the alphabet. As a special case, convert z into a.

3.7. Not only does C++ support the primitive data types of int, float, and char, but it also supports the integral types of long, short, and unsigned. Find the limits for these additional types and explore their meanings. Are there any standard header files that you need to be aware of? Is there such a thing as an unsigned long int? How about an unsigned char?

3.8. Execute the following program and explain the results.

```
#include <iostream.h>

void main(void)
{
 int i;
 char c;

 c = 'a';
 i = 1000 + c;
 c = i;
 cout << i << " " << c << endl;
}
```