

CS 216

Laboratory 3:

Data Representation

Objective:

To become familiar with 1) the underlying representation of various data types, 2) how to examine these representations in the debugger.

Background:

In lecture we discussed how various data types: integers, characters, floating point numbers, were represented in computers. In this lab we will use the debugger to examine some of these representations.

Reading:

Read Weiss *C++ for Java Programmers* Section 11.6, pp. 224-225, on C++ Primitive arrays. (A handout of this will be provided in class and in lab.)

In-Lab Activities

0. Size of C++ data types

The size of C++ data types is dependent on the underlying hardware on which you are running. A programmer may determine the size of various data types by using the `sizeof()` operator. `sizeof()` returns the size, in bytes, of a given variable or data type. You should use the help index in Visual Studio or on MSDN to look up the `sizeof` operator.

- Write a small C++ program that demonstrates the use of `sizeof()` with the following types: `int`, `unsigned int`, `float`, `char`, `double`, `bool`, `char*`, `double*`, `int*`. Use your program to fill in the first column of table shown below.

Note: `char`, `short`, `int`, and `long` are all *integral types*. Integral types may be either signed or unsigned. Signed types have a different range of values than their unsigned counterpart. (Unless specified as unsigned, integral types are signed.)

- Demonstrate your program to a TA

C++ Type	Size in bytes?	Max value? (in hex or base 10)	Zero is stored as?
<code>int</code>			
<code>unsigned int</code>			
<code>float</code>			
<code>double</code>			
<code>char</code>			Char '0' =
<code>bool</code>		true =	false =
			NULL is stored as?
<code>int*</code>			
<code>char*</code>			
<code>double*</code>			

0. The Limits of Representation

→ Answer the following questions and fill in the last two columns of the table above:

-) What is the maximum value (in hex and base-10) that can be stored in a variable of type: `char`, `int`, `unsigned int`, `float`, `double`?
-) How is the value **zero** stored as a `char`, `int`, `unsigned int`, `float`, `double`? Assign the value zero to several variables of these different types to answer this question.

How is a NULL pointer represented?

-) What do you think will happen when you add 1 to a variable containing the maximum value of a type?

→ Write a program to answer the following questions:

-) What happens when you add 1 to an `int` variable containing the maximum value of a `int`?
-) Does the program halt?
-) What answer do you get?

Note: you can assign hex constants directly in your C++ program: rather than saying `=17`, you can say `= 0x12`. You can also assign them directly in the variable window in the debugger. (In C++, a number beginning with 0 is in octal, e.g. 011 is 9 decimal.)

→ Tell the TA your answers to the questions, and demonstrate your program to the TA.

0. Representation in memory

To complete this checkoff you will need to show the TA that you know how to read the contents of a particular memory address in the *memory debug window*.

Using the memory debug window: You should go to Debug->Windows->Memory while running in debug mode to open up this window. Visual Studio allows you to have up to 4 memory windows open at a time.

Unlike in the watch or variable windows, in the *memory debug window*, the four bytes of a memory word are stored in what seems to be reverse order. That is: `0xD97C34A2` would be viewed in the memory window as: `A2 34 7C D9`, with the least significant byte listed first. This format is known as *little endian* format. Note that you can resize the memory window. The distance between the addresses listed on the left changes as you widen the window to display more bytes horizontally.

Note also that right clicking on this window will allow you to change the format in which memory is displayed (byte, short hex or long hex). The contents of memory remain the same, it is merely redisplayed in a different format. Right clicking on the *watch* or *variable debug windows* will also allow you to toggle between hexadecimal and decimal display.

➔ Do this:

-) Write a C++ program where you consecutively declare variables of these types: `bool`, `char`, `int`, `double`, `int*`, and assign a value to each of them.
-) Put a breakpoint near the end of the program.
-) In the *watch debug window* (typically the one at the lower left) type expressions to examine the addresses of all of these variables (e.g. `&i`)
-) Then for each of these variables, type their addresses in to the Address box in the *memory debug window* and hit return. You may also drag the address from the watch window to the memory window. See the memory window scroll so that the contents of that address are displayed. Identify the variables in memory.
-) Find one of your `int` variables in memory. Right click on one of its bytes in the memory window. Select “Edit value” and change the value in memory to something else. The byte will turn red to show that it has changed. Now go to the watch window to see what the new value of that variable is. Does this make sense? Now edit the value in memory to change the integer from a positive value to a negative value and back again.

➔ Demonstrate the following to a TA:

-) The variables in memory.
-) Changing an `int` value from positive to negative.
-) Anything else interesting you notice (e.g. skipped memory) and your explanation for it.

0. Primitive Arrays in C++

- ➔ Read Weiss C++ for Java Programmers section 11.6, pp. 224-225 on C++ Primitive arrays. (a handout of this will be provided in lab) Note how two (or higher) dimensional arrays are stored in row-major order in C++ as opposed to being stored as arrays of arrays in Java.
- ➔ Write a C++ program where you declare a **one** dimensional array of chars and a **one** dimensional array of ints:

```
int IntArray[10];
char CharArray[10];
```

In your program you should also declare a **two** dimensional array of chars and a **two** dimensional array of ints:

```
int IntArray2D[6][5];
char CharArray2D[6][5];
```

Assign *different* values into each element of all four arrays. As in part 3, put a breakpoint in your program after the four arrays have been assigned values. Find the address of the first element of each array, and type that address into the memory window.

- ➔ Show the TAs where the elements of the four arrays are in memory.
- ➔ Assuming that memory is *byte-addressable*, write an expression that will tell you the address of the (i, j)th element of IntArray2D as declared above. (Assume that: $(0 \leq i < 6)$, and $(0 \leq j < 5)$). Note: & here means “the address of”, you may use & in your answer.

$\&(\text{IntArray2D}[i][j]) = \underline{\hspace{15em}}$

0. Floating Point Values

- ➔ Using the Number Representation handout, calculate how the value 3.0, should be represented in **IEEE single precision floating point (32 bits)**. Write out your answer in clearly labeled bits (show your work). You will need to know how to do this on an exam.

- ➔ Verify that your answer is correct by examining the contents of a float variable containing the value 3.0 in the *memory debug window*.

- ➔ Assign this float variable into an integer variable. What is its representation as an int? Verify that your answer is correct by examining the contents of the int variable in the *memory debug window*.

- ➔ Demonstrate your answers and program to a TA.

0. What, done already?

O.k. for more fun and enjoyment and the adulation of your TAs and instructor (no extra credit), examine some of the following:

- Examine the data layout you can piece together by examining addresses of these structures: stl vector, the nodes of a linked list allocated off of the heap.
- What are the sizes of the same data types you examined above on BLUE.UNIX?
- Does BLUE.UNIX use big endian or little endian representation?

Procedure

Pre-lab – *(There was no pre-lab this week other than reading.)*

In-lab

Use one checkoff sheet per person.

- 1. The head TA will assign you a partner for this in-lab activity.
 - Introduce yourself to your partner and find a computer to work on. Go ahead and exchange email addresses to facilitate further communication if needed.
- 1. Work through the steps one a time. At each checkoff, the TA will ask one question of *each* partner, so be sure that both partners understand what is happening.
- 1. Be sure to leave your checkoff sheet in lab.

Post-lab

- 1. You should complete the Radix conversion worksheet on the next page.
- 1. You should answer the postlab question given below.
- 1. Staple the following (in the upper-left corner) in this order:
 - ➔ Your radix conversion worksheet.
 - ➔ A typed or NEATLY written solution to the **postlab question (see below)**. Note that you may discuss postlab questions with your partner or other classmates, but the work you turn in should be your own. A reasonable policy is what we'll call the Gilligan's Island Policy. Specifically, you should a) not take away any written notes from a discussion of the questions and you should b) engage in some mindless activity such as watching an episode of Gilligan's Island for 30 minutes after the discussion. If after that time you can still write up a correct solution, then you probably indeed understand it yourself. In the future, we may have some postlab questions that will be worked on and turned in as groups, but for lab 3 they should be done individually.
- 0. **For this week only: Submit to the cs216 mailbox before 4 PM on Thursday, September 22, 2005.** If you think you will forget to do this, then I would recommend bringing your completed assignment to lecture on Wednesday. The 216 mailbox is located in Olsson Hall, just to the right as you walk in the front door to the computer science department. As you walk in the door you will see a row of mailboxes to your right. The cs216 box is on the bottom row and is on the end furthest away from the front door.

Postlab Question

- 0. Write a recursive function that returns the number of 1's in the binary representation of N . Use the fact that this is equal to the number of 1's in the representation of $N/2$, plus 1, if N is odd. You may assume that N is a non-negative integer stored in two's complement. Please write your function as a valid C++ function that takes the integer N as input. This should be a rather simple function that uses what you've learned about integer representation. If you find you need things like global variables or the pow function to implement this then you are going too far.

UVa Email ID (no aliases please) _____

Name _____ Section _____

Lab 3 - Radix Conversion Worksheet

Convert:

0) $0x\ 4E23$ into Octal

0) 362_{10} into radix 7

0) 110010110110_2 into Decimal

0) $2BG_{19}$ into Decimal

5) Given the following positive binary integer in two's complement,

0111001101010101_2

a) Convert the number to hexadecimal:

b) Negate the number.

CS 216: Program and Data Representation

Lab #3: Data Representation - Check-off Sheet

(One check-off sheet per person)

Date:	Circle: Your 1 Tue 12:15 – 2 PM Section: 2 Tue 7:30 – 9:15 PM
My <u>Email</u> Address: (UVa ID please, no aliases)	My Name:
Partner's <u>Email</u>:	My Partner's Name:

1. Size of C++ data types	
2. The Limits of Representation	
3. Representation in memory	
4. Primitive Arrays	
5. Floating Point Values	

Pledge: