 

**Advanced Placement Computer Science**

[**Shenendehowa HS**](http://www.shenet.org/shen-high-school/)[**mr Hanley**](http://mrhanleyc.com)

**Unit 1: Comp Sys - Numeric String Rep Num Sys**

**Lesson: Integer VS FloatingPoint, Rounding**

***Last Updated:*** *9/11/2017*

Lesson: Parameter Passing Mechanisms

*Last Updated: 100/11001/1100*

**Integer types:**

Integers are stored in a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ inside the computer

Finding the maximum and minimum values for the integer types;

**Floating Point Variables:**

Here’s where some problems will surface!!!!!!!!

Representing floating point values must be of the form:
$$value=signficand\* \frac{1}{2^{n}}$$

For some numbers, this is no problem

For other values, we are in TROUBLE

Take as an example, .5

As mathematical calcs add up, errors can be seen

Round-Off Problems with Computers (Affects C++, Java, ANY Language)

 ==================

 One of the problems with floating point numbers is round-off. Round off errors occur when attempting to represent certain numbers in any number base. For example, 1/3 is not exactly representable in base ten, while 1/10th is easily representable. But since we're dealing with computers, we are specifically in

base two numbers. As opposed to base ten, 1/10th is not exactly representable in base two. For example, the fractional portions of base two are: 1/2 1/4 1/8 1/16 1/32 1/64 1/128 1/256 1/512

The numbers 1/2, 1/4, 1/8, all powers of two, are exactly representable in a computer. But since 1/10 lies between 1/8 and

 1/16, it is not exactly representable using binary notation. So internally the computer has to decide which fractional binary portions to add together to sum close to 1/10. For example:

 1/2 1/4 1/8 1/16 1/32 1/64 1/128 1/256 1/512

 0 0 0 1 1 1 0 0 0

 this adds up to: 0.1093 which is close to 0.1000 but could easily be rounded

 to 1.1 so the computer internal algorithm must try to find another combination of binary fractions which come closer to

0.1000 When it's internal algorithm is satisfied, it will have a number which is CLOSE to 1/10th but not EXACT. This inexactness is known as ROUND-OFF error.

Floating Point Round Off Error

 ------------------------------

 Round off error is especially noticable in the smallest

 floating point data type available: the float. The float data

 type is four bytes in length, and uses these bytes to hold the

 mantissa, exponent, and sign of the number. The following

 program demonstrates that round off error with floating point

 number's occur even with simple assignments:

 void main()

 {

 float number = 123.45;

 cout << number << endl;

 }

 The round off error can be significant when doing multiple or

 iterative calculations, as the following program illustrates:

 void main()

 {

 float anumber = 1.693 / 10.0;

 float original = 1000000.00;

 int i, j;

 for (i=0; i<10; i++)

 {

 original = original \* anumber;

 }

 for (j=0; j<10; j++)

 {

 original = original / anumber;

 }

 cout << original << endl;

 }

 At the end of ten multiplications and divisions, the original

 number is off by a 0.1875. Increasing the size from a 4 byte to

 8 byte real improves things somewhat, as the following code

 illustrates:

 void main()

 {

 double anumber = 1.693 / 10.0;

 double original = 1000000.00;

 int i, j;

 for (i=0; i<10; i++)

 {

 original = original \* anumber;

 }

 for (j=0; j<10; j++)

 {

 original = original / anumber;

 }

 cout << original << endl;

 }

 The difference between the original and calculated original are

 only off by 0.0625 with doubles.