Lab 6

Review of Variables, Formatting & Loops

By: Dr. John Abraham, Professor, UTPA

Variables:

We learned that a **variable** is a name assigned to the first byte of the necessary memory to store a value. During compile time the compiler reserves necessary memory for each variable. It may require more than one byte to store a value. The compiler needs to know the size of memory to reserve. This is why a type is always associated with a variable. In a 16-bit computer 2 bytes are used to store an integer, 4 bytes for floats, 8 bytes for double, and 1 byte for character and bool. The size of memory reserved by the compiler you are using can be determined by the operator **sizeof**.

For example, **cout << sizeof(x**); if x is an integer, either a 2 or 4 will be displayed.

Some of these types can be signed or unsigned. Unsigned can hold much larger number. For example, in a signed integer variable the range of values are -32,768 to 32,767. In an unsigned integer this range is from 0 to 65,535. Float and double are real numbers, the difference is in their precision. The major differences between integers and real numbers are as follows.

1. integers are ordinal; an integer has a specific predecessor and successor. They are represented precisely in the computer in binary. Negative numbers are represented in two's complement.
2. Real numbers are not ordinal. They have decimal portions. They are represented in the computer as two portions, mantissa and exponent. Real numbers cannot be represented exactly in the computer; they are approximations only. Therefore, two real numbers cannot be checked for equality. Truncation errors occur when two real numbers are multiplied or divided.

Formatting:

When outputing a real number it is often necessary to indicate the number of decimal digits we want to display. When dealing with dollars there is little reason to go beyond two decimal points. We can use **setprecision(n)** which is a manipulator available from <iomanip>, to indicate the decimal precision we require. For example **cout << setprecision(2) << dollars**; will display dollars with 2 decimal point precision. The last decimal digit will be rounded of. Consider the following program:

Program 6-1

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
        cout << 28 <<" " <<28*1.128 <<endl;
        cout << 123 << " " <<123*11.228 <<endl;
return 0;
}
```

Program Run 6-1:
28 31.584
123 1381.04
Press any key to continue

The columns in the above program do not line up. Now consider the following program and its formatted output. The integers line up. But even though we used the setprecion(2), the real numbers are displayed differently. Obviously for real numbers the setprecision alone is not enough. In Program 6-3, setw, setiosflags and setprecision are added. This seems to have fixed the problem. Here the **ios::fixed causes the real number to be output in fixed point format instead of exponential format.**

Program 6-2

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        cout << setw(5)<<28 <<" " <<setprecision(2) << 28*1.128 <<endl;
        cout << setw(5) <<123 << " " <<setprecision(2) << 123*11.228 <<endl;
return 0;
}
```

Program Run 6-2
28 32
123 1.4e+003
Press any key to continue

Program 6-3.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
        cout << setw(5)<<28
                <<setw(10)<<setiosflags(ios::fixed) <<setprecision(2) <<
28*1.128 <<endl;
        cout << setw(5) <<123
                <<setw(10)<<setiosflags(ios::fixed) <<setprecision(2) <<
123*11.228 <<endl;
return 0;
}
```

Program Run 6-3.

```
   28      31.58
  123   1381.04
Press any key to continue.
```

Neither **setw** nor **width** truncates values. If formatted output exceeds the width, the entire value prints, subject to the stream's precision setting. Both **setw** and **width** affect the following field only. Field width reverts to its default behavior (the necessary width) after one field has been printed. However, the other stream format options remain in effect until changed.

Here are some useful escape characters. Recall that you already used \n for a new line.

\b	move back one space
\f	move to next page
\t	tab over
\\	prints a back slash
\'	prints a single quote.

IOMANIP provides several input and output format manipulators. Here is a summary of the most commonly used ones. An explanation of numeric formatting also is given. This section is obtained from the Visual C++ help.

SETIOSFLAGS

ios::skipws Skip white space on input.

ios::left Left-align values; pad on the right with the fill character.

ios::right Right-align values; pad on the left with the fill character (default alignment).

ios::internal Add fill characters after any leading sign or base indication, but before the value.

ios::dec Format numeric values as base 10 (decimal) (default radix).

ios::oct Format numeric values as base 8 (octal).

ios::hex Format numeric values as base 16 (hexadecimal).

ios::showbase Display numeric constants in a format that can be read by the C++ compiler.

ios::showpoint Show decimal point and trailing zeros for floating-point values.

ios::uppercase Display uppercase A through F for hexadecimal values and E for scientific values.

ios::showpos Show plus signs (+) for positive values.

ios::scientific Display floating-point numbers in scientific format.

ios::fixed Display floating-point numbers in fixed format.

ios::unitbuf Cause **ostream::osfx** to flush the stream after each insertion. By default, **cerr** is unit buffered.

ios::stdio Cause **ostream::osfx** to flush stdout and stderr after each insertion.

More on Loops:

Suppose you already have a program and you want to add a loop around it to make it run many times. Let us modify the program from Lab 3 (the grades program) to add a while loop. First step is to decide which of the variables can be selected as loop control variable. If no variables are available to select, you can add new variable - for example, a char variable - and ask " Do you want to run the program again? ". However this is extra work for the user to answer yes or no each time and is not a preferred method.

In the grades program we asked for three scores, found the average and letter grade and displayed these values. We could choose the first score as the loop control variable. If a negative number was entered for the first score, we can exit the while loop. Remember the steps required for a while loop, (1) decided on a LCV; (2) initialize the LCV; (4) setup the while loop condition; (5) write the body of the loop; and (6) change the value of LCV within the body of the loop.

Program 6-4

```
/*****************************************
Accept three grades, find the average
and display the letter grade.
Teaching objective - multiple alternatives
By Dr. John Abraham
Created for 1370 students
*****************************************/

#include <iostream>
#include <iomanip>
using namespace std;

float findave (int, int, int);
char getLetterGrade(float);
void Message(int, int, int, float, char);

int main ()
{
  int one, two, three;
  float average;
  char grade;
  cout << "This program will calculate letter grade given three scores for any number of
students.";
  cout << "\n\nEnter the first score or a negative number to quit-> "; cin >> one;

while (one >0)
{
cout << "Enter the second score-> "; cin >> two;
cout << "Enter the third score-> "; cin >> three;
average= findave (one, two, three);
grade = getLetterGrade(average);
Message (one, two, three, average, grade);
cout << "\n\nEnter the first score or a negative number to quit-> "; cin >> one;
}
return 0;
}

float findave(int one, int two, int three)
{
float a;

a = (float(one+two+three) / float(3.0));
//convert numbers to float to avoid warning
return (a);
```

```
}

char getLetterGrade (float average)

{
char grade;
if (average >=90) grade = 'A';
else if (average>= 80) grade = 'B';
else if (average >=70) grade ='C';
else if (average >= 60) grade ='D';
else grade ='F';
return (grade);
}


void Message (int one, int two, int three, float average, char grade)
{
   cout << "Three grades are: " <<one <<setw(4)<<two <<setw(4)<<three <<endl;
cout << "The average is : " <<average <<endl;
   cout << "\nThe letter grade is : "<< grade << endl;


   switch (grade)
   {
   case 'A' : cout << "Very impressive grade indeed!\n";break;
   case 'B' : cout << "A solid performance, congratulations!\n"; break;
   case 'C' : cout << "C++ is a tough course, but YOU MADE IT!\n";break;
   case 'D' : cout << "Made it eh? \n";break;
   case 'F' : cout << "Don't give up. Try keeping up with all the homework!\n";
   }

}
```

Program Run 6-4

```
This program will calculate letter grade given three scores for any number of students.

Enter the first score or a negative number to quit-> 90
Enter the second score-> 88
Enter the third score-> 93
Three grades are: 90 88 93
The average is : 90.3333

The letter grade is : A
Very impressive grade indeed!
```

```
Enter the first score or a negative number to quit-> 78
Enter the second score-> 66
Enter the third score-> 77
Three grades are: 78 66 77
The average is : 73.6667

The letter grade is : C
C++ is a tough course, but YOU MADE IT!


Enter the first score or a negative number to quit-> -1
```

Assignment:
Write a program to find the average, and the smallest and largest of numbers from a series of positive numbers entered.
Here is the program run:

```
Enter a number or enter a negative number to quit-->88
Enter a number or enter a negative number to quit-->89
Enter a number or enter a negative number to quit-->78
Enter a number or enter a negative number to quit-->62
Enter a number or enter a negative number to quit-->99
Enter a number or enter a negative number to quit-->12
Enter a number or enter a negative number to quit-->45
Enter a number or enter a negative number to quit-->77
Enter a number or enter a negative number to quit-->-1
Average :  68
Largest number entered is: 99
Smallest number entered is: 12
```