Lab 2 - Fundamental Data Types

Your name:	Zach Evans
Your email address:	evansz2@students.wwu.edu
Your student ID number:	W01006113

Lab Objectives

To gain experience with

- integer and floating-point numbers
- writing arithmetic expressions in C++
- appreciating the importance of comments and good code layout
- defining and initializing variables and constants
- recognizing the limitations of the `int` and `double` types and the overflow and roundoff errors that can result
- reading user input and displaying program output
- changing the values of variables through assignment
- using the ANSI C++ standard `string` type to define and manipulate character strings
- writing simple programs that read numbers and text, process the input and display the results

R1. Number Types

Numbers are essential to computing. In C++ there are two fundamental numeric types: integers and floating point numbers. *Integers* have no decimal part, whereas *floating point numbers* do. *Variables* of either type are represented by user specified names. Space in the computer's memory is associated with the name so that the variable acts like a container for the numeric data.

Declaring a variable reserves space in memory and associates a specified name with it. When a variable is declared, it contains whatever value is leftover from the previous use of the memory space. *Initializing* a variable sets it to a specific value. If no specific value is known, it's a good idea to initialize new variable to zero in order to avoid having it contain a random value.

```
type   name        = value;
int    identifier;         /* Variable declared but not initialized */
int    identifier  = 0;    /* Variable declared and initialized to zero */
double identifier  = value; /* Variable declared and initialized to another value*/
```

Are the following integers and floating point numbers properly declared and/or initialized? If not, supply a correction.

`int 3;`	No. "int x = 3;"
`double;`	No. "double x;"
`int = 19;`	No. "int x = 19;"
`float myten =10.23;`	Yes

`double_sum =2.2;`	No. "double sum = 2.2;"
`int that_value 212;`	No. "int that_value = 212;"

When declaring a variable, take a moment to consider by what name it will be known. A valid name is made up of characters from the set:

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`	upper case letters
`abcdefghijklmnopqrstuvwxyz`	lower case letters
`0123456789`	digits
`_`	the underscore character

The only limitation is that a variable name can't begin with a digit. Of course, the name must be unique. Neither you nor the compiler would be able to distinguish between variables with the same name. For some simple cases, using single letters like **`i, j, k`** is enough, but as programs grow larger this becomes less informative. Like the name for a product or service, a variable name should do two things:

1) Be unique
2) Express the purpose

Mostly, this will happen if the name selected is descriptive enough to tell a human reader how a variable is being used, for example `count_messages`, `user_preference`, `customer_name` This will also help you when you are trying to maintain the program later on, since a descriptive name will assist you in remembering exactly what a variable is used for.

Complete the following table of (bad) variable names, better names and descriptions.

Bad Variable Name	Variable Renamed	Description
`int monthly income`	int monthly_income;	An integer to hold the monthly income
int x	int counter;	`integer counter`
`double %sales;`	double sales;	a floating point number for money made from s
double salezz;	double bikeSales;	`double sales of bikes`

R2. Constants

Like variables, any numeric constants used by your program should also have informative names, for example: `const int FLASH_POINT_PAPER = 451;` or `const int BOILING_POINT_WATER = 212`. Using named constants transform statements such as `int y = 451 - x + 212` into the more intelligible `int y = FLASH_POINT_PAPER - x + BOILING_POINT_WATER`.

Give definitions for each of the constant descriptions listed below

const definition	Description
const int DAYS_IN_WEEK = 7;	Number of days in a week
const int WEEKS_IN_YEAR = 52;	Number of weeks in a year

const double MWAGE_PER_HOUR = 9.04;	Minimum wage per hour

P1. Comments

Even carefully named variables are rarely sufficient to fully convey the operation of a program. They do not, for example, provide an explanation of a future feature that could be implemented because of how something is done now, or leave a detailed description of how a variable is computed. You can however place any additional information necessary in a comment. Comments are plain text for humans to read. They are separated from the machine readable code needed by the compiler by either of:

```
/* say whatever you want in here, between the slash-asterisk boundaries
   the compiler will ignore it
   . . .
*/
```

or

```
// say what ever you want from the double slashes to the end of the line . . .
// the compiler will ignore it
// . . .
```

From both design and documentation perspectives, it is a good idea to get in the habit of commenting your code as early as possible. For example, provide a brief description of what the program coins.cpp from Section 2.2 does.

```
#include <iostream>
using namespace std;
int main()
/* your work goes here--use // or /* as appropriate */
```

```
/* coins.cpp asks for the amount of each coin the user has, then takes the
input and prints out the total value of the coins */
```

```
{  cout << "How many pennies do you have? ";
   int pennies;
   cin >> pennies;

   cout << "How many nickels do you have? ";
   int nickels;
   cin >> nickels;

   cout << "How many dimes do you have? ";
   int dimes;
   cin >> dimes;

   cout << "How many quarters do you have? ";
   int quarters;
   cin >> quarters;

   double total = pennies * 0.01 + nickels * 0.05
       + dimes * 0.10 + quarters * 0.25;
       /* total value of the coins */

   cout << "Total value = " << total << "\n";

   return 0;
}
```

Now, extend coins.cpp to prompt for and accept the number of Golden Dollar coins, placing the finished code here. Add a comment to show when you modified the code, and what change you made.

```
#include <iostream>
using namespace std;
int main()

{  cout << "How many pennies do you have? ";
    int pennies;
    cin >> pennies;

    cout << "How many nickels do you have? ";
    int nickels;
    cin >> nickels;

    cout << "How many dimes do you have? ";
    int dimes;
    cin >> dimes;

    cout << "How many quarters do you have? ";
    int quarters;
    cin >> quarters;

    cout << "How many dollar coins do you have? ";
    int dollarCoins;
    cin >> dollarCoins;

    double total = pennies * 0.01 + nickels * 0.05
    + dimes * 0.10 + quarters * 0.25 + dollarCoins;
    /* total value of the coins */

    cout << "Total value = " << total << "\n";

    return 0;
}
```

R3. Input and Output

C++ defines two standard streams that can be used for text mode input and output, respectively `cin` and `cout`. Operator >> and << are used to direct input and output to and from the stream.

In coins.cpp, for example, you used `cin >> pennies;` to get user input. The expression `cout << "Total value = ";` displays the message `"Total value ="`. It was followed by `<< total` , which places the contents of the variable total into the `cout` stream.

Items in the input stream are separated by *white space* - spaces, tabs and newlines. For example, two numbers may be input as

```
3   4
```

or

```
3
4
```

If there is an error, the stream goes into a failed state rather than plodding ahead with bad data. For example, if the user has typed

```
l0
```

(with a letter l) instead of

```
10
```

then the stream fails when trying to read a number.

Do the following lines of input data work properly? Why or why not?

```
User Input: 24 25 26

int first, second, third;
cin >> first >> second >> third;
```

```
Yes, because the input is three integers separated by whitespace.
```

```
User Input: 24.4 25.5 26

double fourth, fifth, sixth;
cin >> fourth >> fifth >> sixth;
```

```
Yes, because the input is three doubles separated by whitespace (even though
the third number has no decimal place, it is still a double)
```

```
User Input: 24.4 25.5 26.6

int seventh, eighth, ninth;
cin >> seventh >> eighth >> ninth;
```

```
No, because the input is three doubles and the program is expecting three
integers.
```

```
User Input: 23.4 24.5

double tenth, eleventh, twelfth;
cin >> tenth >> eleventh >> twelfth;
```

```
The input would be read, but the program would still run until a third double
was put in.
```

R4. Assignment

There are several primary ways to modify the contents of a variable in C++.

- `cin >>` *variable*; the '>>' operator places the contents of the input stream into the variable
- *variable* `= 3;` the '=' operator, called the assignment operator, takes whatever combination of values are on its right side and puts them into a variable on its left side.
- *variable*++; the '++' operator increments, or adds 1 to the variable to which it is appended. Similarly, the '--' operator will decrement an integer variable by 1.

During compilation, values on both sides of the >> and = operators are checked to insure that the data received matches the data type of the variable to which it will be assigned. Are the right and left hand sides of the assignment operators of compatible type in the following statements?

`int d=4;`	Yes

`double a; int c = c * (6 + a);`	Yes
`double q = "3";`	No, because "3" is a string literal.

What error does your compiler give in these cases?:

Statement	Error message
`kq+2 = 19;`	Use of unidentified identifier "kq"
`PI = 3;`	Use of unidentified identifier "PI"
xz = 20++;	

Because a variable of any given type represents a fixed amount of space in memory, there is a limit to how large a value it can store.

Size limitations are particularly evident with integers. On some computers, integers can only be about 32,000, on others, they can go up to a little more than 2,000,000,000. What is the largest value your compiler will accept for a signed integer? What is the most negative allowed value?

```
2147483647
-2147483648
```

Floating point numbers don't round to the nearest integer value. Can you explain the values your compiler assigns to `f` and `g` in:

```
double f = pow(10.0, 20.0) - 53;
double g = f;
g = g - 53;
cout << f << "\n" << g << "\n";
```

```
The compiler gives both the value of "1e+20" because the precision isn't high
enough to note the difference of 53.
```

R5. Arithmetic

Recalling what you've learned about integers and floating point values, what value is assigned to Z by each of the following?

`int Z = 9 + 2;`	11
`int Z = 9 - 2;`	7
`int Z = 5 * 2;`	10
`int Z = 8 / 4;`	2
`int Z = 9 / 4;`	2
`double Z = 9 / 4;`	4.5

`int Z = 222 / 300;`	0
`double Z = 222.0 / 300.0;`	.74
`int Z = 222 / 300.00;`	0
`double Z = pow(3,6);`	729

Translate the following algebraic expressions into C++ :

y = x + 3/4 - 2	double y = x + (3/4) - 2;
$y = x^2 + 6x - 2$	double y =(x*x)+(6*x)-2;
$y = \frac{x}{1-x}$	double y = x/(1-x);

Strings

Many programs manipulate not just numbers but text. C++ stores text in string variables. They are declared and assigned in a manner similar to a numeric variable, and the sequence may have any length, including being uninitialized or having 0 characters. For example,

```
string name = "Jane Doe";
```

assigns the 8 characters enclosed in quotes to the variable `name`.

```
 string temp = name;
```

assigns the content of variable `name` to variable `temp`

```
cout << name;
```

displays the contents of variable `name` on the standard output stream.

R6. Concatenation

Given:

```
   string first_name = "Kenneth";
   string last_name = "Cole";
   string name;
```

What value will `name` contain after

```
      name = first_name + last_name;
```

```
KennethCole
```

How could the preceding be revised to give a better result?

```
name = first_name + " " + last_name;
```

R7. Substrings

The primary difference between strings and numeric data types is that any portion of the characters in a string variable is itself a string variable. Such a portion is called a *substring*.

What will be the resulting substring in the following examples? If invalid, give a corrected version of the call to `substr`.

`string name = "Robert Sourchie";` `string first_name = name.substr(2, 4);`	bert
`string name = "Robert Sourchie"'` `string last_name = name.substr(7, 4);`	Sour
`string name = "Pamela Sourchie";` `string name2 = name.substr(1, 11);`	amela Sourc

P2. Extracting Substrings

Using `substr` and concatenation, give a program containing a sequence of commands that will extract characters from `input_string = "Four score and seven years ago our fathers"` to make `output_string = "carefree and no tears"`. Then print `output_string`.

```
#include <iostream>
using namespace std;
int main()

{
    string input_string = "Four score and seven years ago our fathers";
    string output_string = input_string.substr(6,1) + input_string.substr(11,
1) + input_string.substr(8,2) + input_string.substr(35,1) +
input_string.substr(33,1) + input_string.substr(9,1) + input_string.substr(9,1)
+ input_string.substr(4,1) + input_string.substr(11, 4) +
input_string.substr(12,1) + input_string.substr(1,1) + input_string.substr(4,1)
+ input_string.substr(37,1) + input_string.substr(22,4);
    cout << output_string;
}
```

P3. Reading string input

The function `getline(cin, s)` read a single input line into the string `s`. Write a program that uses `getline` to read input of the following exact form:

```
FNAME first name
MNAME middle name
LNAME last name
```

```
ADDRESS a street name and house number
CITY a city name
STATE a state abbreviation
ZIP a zip code
ORDER_DATE the date of the order
TOTAL_ITEMS number of items (an integer)
ORDER_TOTAL total amount of order (type double)
SHIP_DATE   date of shipment
```

Then print out the top part of a shipment manifest in the following form:

```
John Q. Public
11801 Jones Valley Ct.
Pleasant Hills, MN 66504
=====================================================================x
Order Date: 10-Oct-2001    Total Items: 10    Order Total: $100.00
Ship Date: 20-Oct-2001
```

```
#include <iostream>
using namespace std;
int main()

{

    string FNAME, MNAME, LNAME, ADDRESS, CITY, STATE, ZIP, ORDER_DATE, SHIP_DATE;
    int TOTAL_ITEMS;
    double ORDER_TOTAL;

    getline(cin, FNAME);
    getline(cin, MNAME);
    getline(cin, LNAME);
    getline(cin, ADDRESS);
    getline(cin, CITY);
    getline(cin, STATE);
    getline(cin, ZIP);
    getline(cin, ORDER_DATE);
    getline(cin, TOTAL_ITEMS);
    getline(cin, ORDER_TOTAL);
    getline(cin, SHIP_DATE);


    cout << FNAME << " " + MNAME << " " << LNAME << "\n" << ADDRESS << "\n" <<
CITY << ", " << STATE << " " << ZIP << "\n" <<
"=====================================================================x\nOrder
Date: " << ORDER_DATE << "    Total Items: " << TOTAL_ITEMS << "    Order Total:
$" << ORDER_TOTAL << "\nShip Date: " << SHIP_DATE;

}
```

P4. Formatting numbers

You use the `setprecision`, `setw` and `fixed` manipulators defined in the `<iomanip>` header to control the formatting of numbers. Write a program that calculates the interest and principal payments on a simple loan. Your program should query the user for a percentage annual interest rate, an initial balance and monthly payment, then print out the following table:

```
Month Balance           Interest        Payment
    1 $ 10050.00        $  100.50       $  500.00
    2 $ 9650.5          $   96.51       $  500.00
    3 $ 9247.01         $   92.47       $  500.00
    4 $ 8839.48         $   88.39       $  500.00
```

Use `setw` to set each column width, and use `fixed` and `setprecision` to properly show dollars and cents.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()

{
    double interestRate;
    cout << "Enter annual interest rate (as a decimal): ";
    cin >> interestRate;

    double initialBalance;
    cout << "Enter initial balance: ";
    cin >> initialBalance;

    double monthlyPayment;
    cout << "Enter monthly payment: ";
    cin >> monthlyPayment;

    double interest = initialBalance * interestRate;


    cout << setw(6) << "Month ";
    cout << setw(15) << left << "Balance";
    cout << setw(15) << left << "Interest";
    cout << setw(15) << left << "Payment" << "\n";

    /* I don't believe we've learned for loops yet, but I didn't like how it looked with just a
bunch of copy/pasted lines */

    for(int i = 1; i < 5; i++)
    {
        cout << setw(6) << left << i;
        cout << "$ " << setprecision(2) << fixed << setw(13) << initialBalance;
        cout << "$ " << setprecision(2) << fixed << setw(13) << interest;
        cout << "$ " << setprecision(2) << fixed << setw(13) << monthlyPayment << "\n";
        initialBalance = initialBalance  + interest - monthlyPayment;
        interest = initialBalance * interestRate;
    }
}
```

Don't forget to save this page as a pdf document and then up load it to moodle when you're finished.