What I have covered in class so far:

Programming languages – High level languages and how to program in them

Syntax, Semantics, runtime and logical errors

Constant, variable. Data types (set of values together with a set of operations)

 A variable should be declared with a name and type before using. First time you place a value in a variable is called initializing. Remember, **it never is zero** when you declare a variable. If you want it to be 0, you must place 0 in the variable.

C++ data types fall into three categories: (1) Simple data type (2) Structured data type (3) Pointers

Memory, address, each cell can contain a programming instruction or data

EBCDIC, ASCII, Unicode

Arithmetic operators and their use.

Order of precedence

* All operations inside of () are evaluated first
* \*, /, and % are at the same level of precedence and are evaluated next
* + and – have the same level of precedence and are evaluated last
* When operators are on the same level
	+ Performed from left to right (associativity)
* 3 \* 7 - 6 + 2 \* 5 / 4 + 6 means

(((3 \* 7) – 6) + ((2 \* 5) / 4 )) + 6

This week we will cover:

How to send output to a file.

#include "stdafx.h"

#include <iostream>

#include <fstream> //for files

using namespace std;

int main () {

 ofstream outfile; //in fsteam use ofstream and creat an instance outfile

 outfile.open ("lab2.txt"); //give outfile a name

 outfile << "Place something in the file.\n";

 outfile.close(); //close the file; if you don't do it some items may be lost

 return 0;

}

fstream, istream, ostream, extraction operators

Reading values into different types

When reading data into a char variable

* + >> skips leading whitespace, finds and stores only the next character
	+ Reading stops after a single character

To read data into an int or double variable

* + >> skips leading whitespace, reads + or - sign (if any), reads the digits (including decimal)
	+ Reading stops on whitespace non-digit character

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

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.

Function (subprogram): set of instructions

* + When activated, it accomplishes a task
	+ main executes when a program is run
	+ Other functions execute only when called
	+ C++ includes a wealth of functions
	+ Predefined functions are organized as a collection of libraries called header files
* To use a predefined function, you need the name of the appropriate header file
	+ You also need to know:
		- Function name
		- Number of parameters required
		- Type of each parameter
		- What the function is going to do

Here is a sample program

// functions builtin.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <math.h>

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[])

{

 cout<<"square root of 90.0 is: " ;

 cout << sqrt(90.0);

 cout <<"\n 2 raised to the 3 is: ";

 cout << pow(2.0, 3.0);

 cout << "\n Absolute value of -88.2 is: " << fabs(-88.2);

 int absolute = abs(-33);

 cout << "\n Absolute value of-33 is: " << absolute<<endl;

 return 0;

}

Program run

square root of 90.0 is: 9.48683

 2 raised to the 3 is: 8

 Absolute value of -88.2 is: 88.2

 Absolute value of-33 is: 33

Press any key to continue . . .

User defined functions – introduction (Much more to come later)

Program 8-1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Program: Function Calls

Written by: Dr. John P. Abraham

Instructional objective: Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

void chorus();

void verse1();

void verse2();

void verse3();

void verse4();

int main()

{

 verse1();

 chorus();

 verse2();

 chorus();

 verse3();

 chorus();

 verse4();

 chorus(); return(0);

}

 Program 2-1 continued:

void chorus()

 {

 cout <<"\nTwinkle, twinkle, little star,";

 cout <<"\nHow I wonder what you are! ";

 cout << "\nUp above the world so high, ";

 cout << "\nLike a diamond in the sky!\n";

 }

void verse1()

{

cout <<"\nWhen the blazing sun is gone," ;

cout <<"\nWhen he nothing shines upon,";

cout <<"\nThen you show your little light,";

cout <<"\nTwinkle, twinkle, all the night. \n";

}

void verse2()

{

cout <<"\nThen the traveler in the dark, ";

cout <<"\nThanks you for your tiny spark, ";

cout <<"\nHe could not see which way to go,";

cout <<"\nIf you did not twinkle so. \n";

}

void verse3()

{

cout <<"\nIn the dark blue sky you keep, ";

cout <<"\nAnd often through my curtains peep, ";

cout <<"\nFor you never shut your eye, ";

cout <<"\nTill the sun is in the sky.\n ";

}

void verse4()

{

cout <<"\nAs your bright and tiny spark, ";

cout <<"\nLights the traveler in the dark,";

cout <<"\nThough I know not what you are, ";

cout <<"\nTwinkle, twinkle, little star. \n";

}

Program Run 8-1

When the blazing sun is gone,

When he nothing shines upon,

Then you show your little light,

Twinkle, twinkle, all the night.

Twinkle, twinkle, little star,

How I wonder what you are!

Up above the world so high, Like a diamond in the sky!

Then the traveler in the dark,

Thanks you for your tiny spark,

He could not see which way to go,

If you did not twinkle so.

Twinkle, twinkle, little star,

How I wonder what you are!

Up above the world so high,

Like a diamond in the sky!

In the dark blue sky you keep,

And often through my curtains peep,

For you never shut your eye,

Till the sun is in the sky.

Twinkle, twinkle, little star,

How I wonder what you are!

Up above the world so high,

Like a diamond in the sky!

As your bright and tiny spark,

Lights the traveler in the dark,

Though I know not what you are,

Twinkle, twinkle, little star.

Twinkle, twinkle, little star,

How I wonder what you are!

Up above the world so high,

Like a diamond in the sky!

When we run this program, the poem is displayed verse1, chorus, verse2, chorus, and so

on. Look at the main: it is not cluttered. In the main, the functions were called. The

function chorus was called four times; all other functions were called only once. The

main function returned a zero to the operating system indicating all is well; all other

procedures returned nothing. Before calling the functions, the prototypes of the

functions must be declared. Please underline the prototypes just above the main in order

for you to remember to do it when you write your programs. The function prototype

contains the following information: (1) name of the function, (2) the type of data returned

by the function, and (3) number, type, and order of parameters passed into the function.