Computer Organization

Virtually every computer has six logical components:

Input Unit - obtains data (and programs) from an input device for processing. Keyboard, mouse, CD-ROM drive, diskette drive, scanner, digital camera...

Output Unit - takes information from the computer and places it on an output device - monitor screen, printer, diskette drive, CD-Writer…

Central Processing Unit (CPU) - Coordinates the operation of the other sections of the computer.

Arithmetic & Logical Unit (ALU) - where calculations, relational and logical operations are performed - part of the CPU.

Main Memory Unit - primary memory, primary storage - short-term main storage area for data and executable programs (RAM). Ordered sequence of storage locations called memory cells, each memory cell has a unique address.

Secondary Storage Unit - secondary memory, permanent memory - long term, secondary storage area for data and programs.

Primary memory	***Secondary***
Temporary, volatile	Permanent, non-volatile
Rapid access	Slower access (1/0)
Low capacity	High capacity
High cost	Low cost

1/0 Devices - we use these to communicate with the computer

Computer Networks - allows each computer to access (e.g.) the same large hard disk drive and high-quality printer - LAN.

In order to communicate with the computer we use one of several programming languages.

Types of Computer Languages

First generation - Machine Language

Each type of computer has its own machine language, the only language it can understand. Most machine languages consist of binary codes for both data and instructions. Machine dependent. E.g., to add overtime pay to base pay we would need a series of binary codes such as:

0010 0000 0000 0100
0100 0000 0000 0101
0011 0000 0000 0110

Second generation - Assembly Languages

Use English-like abbreviations to represent the machine-language instructions. Use a translator program called an assembler to convert each instruction from the assembly language word to the machine language binary code. E.g.:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

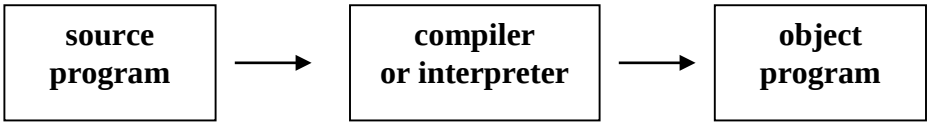
Third generation - Compiler Languages

High-level, machine independent, more English-like, more natural. Each high-level language statement translates to several low-level language statements. Use compilers to translate from the high-level language into machine language. Compilers translate the whole program first, then execute the object program. E.g.,
GROSSPAY = BASEPAY + OVERPAY

A compiler language is a high-level language which is machine-independent.

High-level languages are more English-like, easier to code, more costly to run, less flexible. *e.g.*, FORTRAN, BASIC, COBOL, PL/1, ALGOL, APL, Pascal, SIMSCRIPT, Smalltalk, C, C++, Java.

A compiler is a translator program that transforms high-level program code into a low-level machine-level executable program.



High-level Language Translators	
Compilers	***Interpreters***
Slower translation Entire program Faster execution More efficient execution Good for commercial applications	Faster translation One instruction or macro at a time Slower execution Less efficient execution Good for program development

Programming

When learning to program in any programming language, it's best just to learn the "rules of the game."

Definition -- A ***program*** is a set of instructions in proper sequence, that causes a computer to perform a particular task.

Modern programs are ***projects*** composed of many of individual program modules that have to be linked together in order to be run. In fact most developer systems have their own *integrated development environment* (ide) and programs are developed in phases within the ide:

C++ Program Phases: ...in the C++ programming environment

Edit - Text editor window. Save with .cpp extension

Preprocess - e.g., text replacement, including other library files with the file to be compiled

Compile - to object code

Link - links the object code with the code for the standard library functions referred to in the program. This produces an *executable image* (with no missing pieces). During **Build** process

Load - program is placed in memory

Execute - one instruction at a time

C++ Programming

Every programming language has facilities to:	in C++
1. Read data from some input device	cin >>
2. Write output information onto an output device	cout <<
3. Perform arithmetic operations	+ - * /
4. Perform relational operations	< == >
5. Perform logical operations	! && ||
6. Branch to a non-sequential instruction (w/wo structure)	while
7. Store (and retrieve) data values to (and from) memory	=

Homework

Running a simple C++ program in Microsoft Visual C++

See course notes for a step-by-step guide to working with the MS Visual C++ ide.
[OLD - http://cisnet.baruch.cuny.edu/friedman/cplusplus/hw/hwbigbird.doc]

Some Parts of the Program

```
// hello.cpp
// A First Program in C++
#include <iostream>
Using namespace std;

int main()
{
    cout << "Hello. My name is Big Bird.\n";
    return 0;  //indicates that the program ended
successfully
}
```

(1) Comments - a type of program documentation

```
// indicates that the remainder of the line is a comment

/* comments can also look like this */

/*      also
  like
  this
     */
```

(2) #include <iostream> - a preprocessor directive

Tells the pre-processor to include in the program the contents of the I/O stream header file called iostream.h . This allows us to use standard stream input and output objects like cout (displays to the screen).

As you can see, we need to also code the "using namespace std;" statement. For now, let's just say that this is so that we don't have to use ugly prefixes on cout, like "std::cout".

(3) int main( ) - main function header

Every C++ program has at least one function, called **main**. And there is ONLY ONE **main**. Program execution begins with the first statement in **main**.

(4) **{** brackets denote the body of the function **}**

(5) ; statement terminator

Every C++ statement must end with a semicolon.

(6) << stream insertion operator

Expression to the right of the operator is inserted (sent) to the **cout** object (the display screen).

(7) \n newline escape sequence

The backslash is an escape character. The character following it takes on a different meaning. eg,

\t - tab

\a - alert; ring bell

\\ - prints a backslash

\" - prints a double quotation mark

(8) return - exits from the function

In this case control over execution is transferred back to the operating system.

A simple C++ program

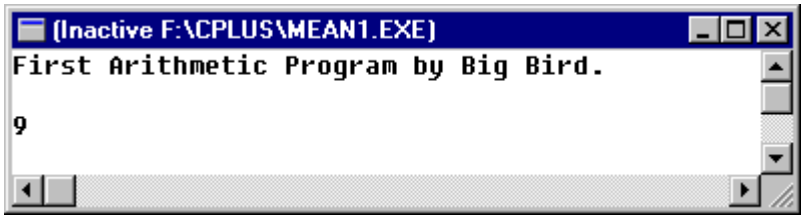
The best approach in learning programming for the first time is to treat it like a game. And, as anyone knows, when you learn a game don't try to understand it just yet – simply learn the "rules of the game" at first. When you have a few simple programs under your belt, you will be able to understand a bit about why the code is the way it is.

This program:

```
//mean1.cpp
// This program calculates the mean of three numbers.

#include <iostream>
using namespace std;                    //need this to drop std::
int main()
{
    cout << "First Arithmetic Program by Big Bird.\n\n";
    cout << (12+5+10)/3;
    return 0;
} //end main
```

produces this output:



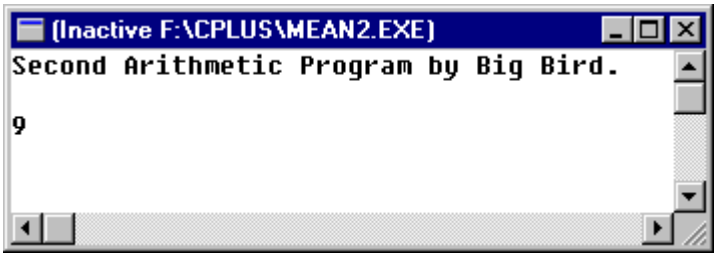
Now, let's try using a variable to store the mean before printing it.

```
//mean2.cpp
// This program calculates the mean of three numbers.
// Big Bird learns about variables.

#include <iostream>
using namespace std;
int main()
{
    float mean;
    cout << "Second Arithmetic Program by Big Bird.\n\n";
    mean =  (12+5+10)/3;
```

```
    cout << mean;
    return 0;
} //end main
```

Output:



Stored Data

Variables must be declared as a certain type, e.g., **int**, **float**, **char**, … This declaration may appear anywhere in the program, as long as it appears before the variable is first used. The declaration creates the object.

For the following declaration,

```
int n;
```

The *name* of the object is n, the *type* [or *class*] is `int`. The object does not yet have a value.

```
n = 66;   //now it has a value
```

or

```
int n=66;  //This declaration also gives a value to n
```

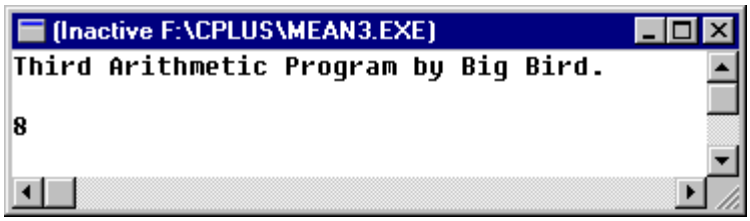
Names (identifiers) - Names should be meaningful to help document your program.

- may use letters, digits, underscores. No spaces.
- may not begin with a digit
- may be any length but better if less than 31 characters long.
- C++ is case sensitive; upper and lower case letters are different.
- no reserved words (e.g., const, void, int, …).
- Avoid using names beginning with _ (underscore) or _ _ (double underscore). The C++ compiler uses names like that for special purposes.

Try a different set of three numbers. Edit the program and run it again.

```
//mean3.cpp
// This program calculates the mean of three numbers.
// Big Bird tries new data.

#include <iostream>
using namespace std;
int main()
{
    float mean;
    cout << "Third Arithmetic Program by Big Bird.\n\n";
    mean =  (11+5+10)/3;
    cout << mean;
    return 0;
} //end main
```

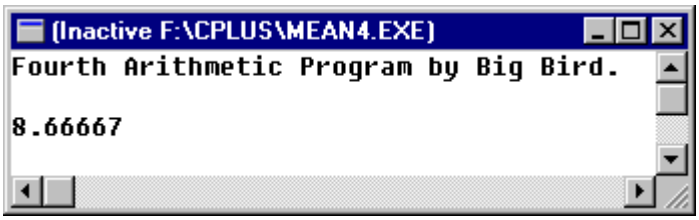


OOPS!! What happened? *[EXPLAIN - syntax (compile-time) error vs. logic (run-time) error]*

```
//mean4.cpp
// This program calculates the mean of three numbers.
// Big Bird learns that expressions have a type.

#include <iostream>
using namespace std;
int main()
{
    float mean;
    cout << "Fourth Arithmetic Program by Big Bird.\n\n";
    mean =  (11+5+10)/3.0;
    cout << mean;
    return 0;
} //end main
```

Output [that's better!]:



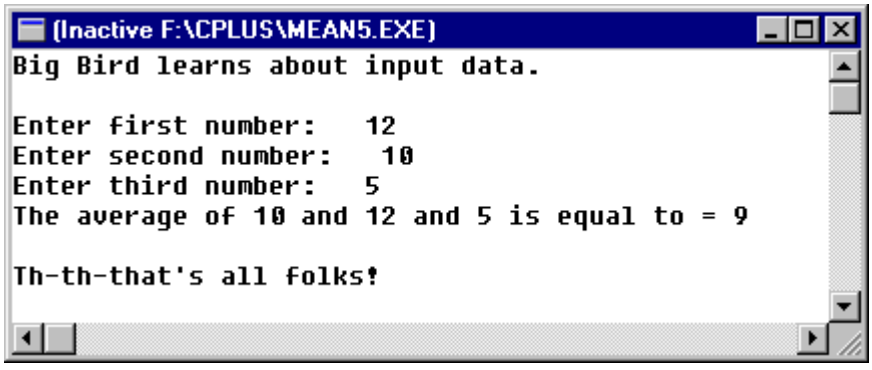
NOW we have a working program that correctly gives us the mean of 11 and 5 and 10…. WOW!! [Uh… How useful is this?]

What if we want to be able to calculate the mean of ANY three numbers? We need to *input* data to the program.

```
//mean5.cpp
// This program calculates the mean of ANY three numbers.
// Big Bird learns about input data.

#include <iostream>
using namespace std;
int main()
{
    float num1, num2, num3, mean;
    cout << "Big Bird learns about input data.\n";
    cout << endl;
    cout << "Enter first number:   ";
                                //prompt user for input
    cin >> num1;
    cout << "Enter second number:   ";
    cin >> num2;
    cout << "Enter third number:   ";
    cin >> num3;
    mean =  (num1+num2+num3)/3.0;
    cout << "The average of " << num1 << " and " << num2 <<
     " and " << num3;
    cout << " is equal to = " << mean << endl <<endl;
    cout << "Th-th-that's all folks!\n";
    return 0;
} //end main
```

Interactive screen session:



Stream Input/Output

```
cout <<         //means "cout gets a value"
cin >> var      //means "cin gives a value to var"
                //note: on execution press enter key to end input
```

<< is the ***stream insertion operator***
>> is the ***stream extraction operator***

The stream insertion operator "knows" how to output different types of data.

Cascading *stream insertion operators* - using multiple stream insertion operators in a single statement. Also known as ***concatenating*** or ***chaining***.

e.g.,

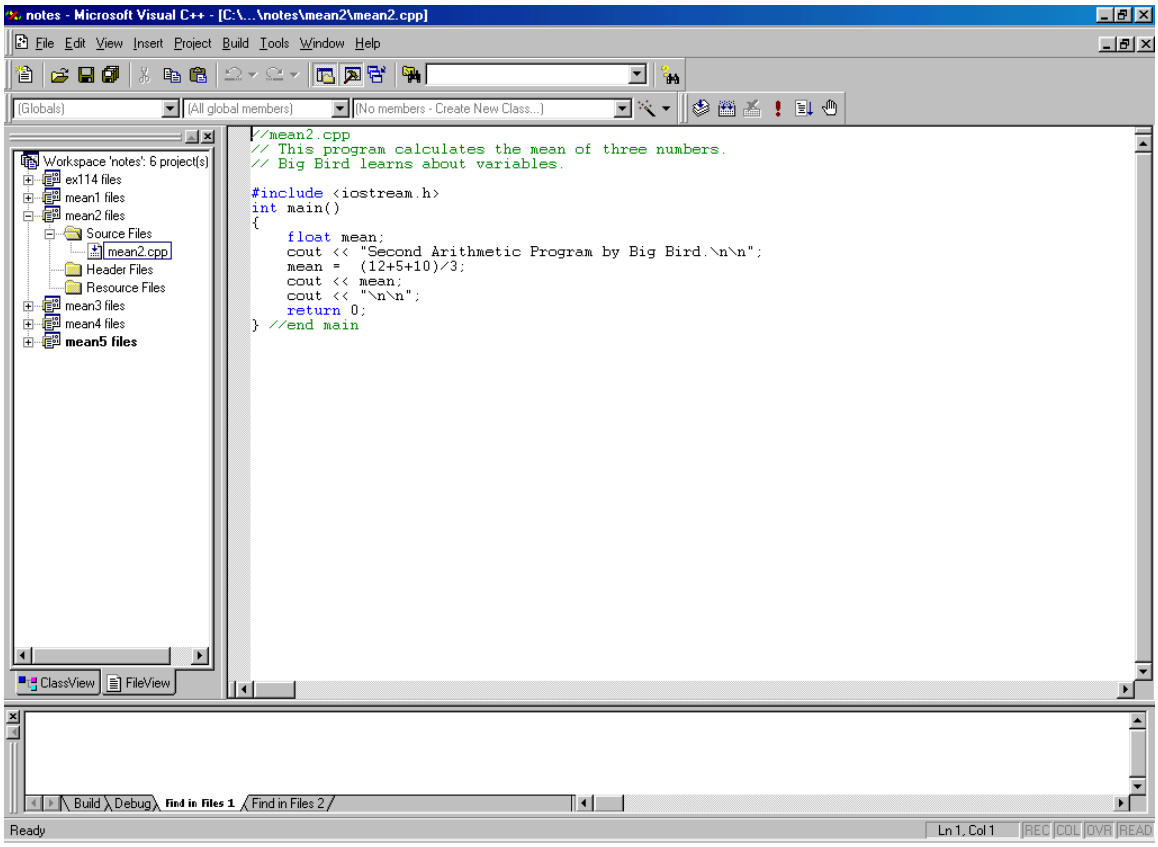
```
cout << "The answer is: " << result << ".\n";
```

More on Creating Projects in MS Visual C++

To create a new project as a subproject, click **File → New → Projects**. Select **Add to Solution**.

Enter a Project Name and Location for the new subproject.

e.g., [need to redo this figure]



This is a good way of managing a lot of small projects (like the ones we do in the first part of this semester), but will be especially useful when we learn to write several larger programs that use the same files.

Data Types - *CLASSES*

-- How a particular set of values is represented in memory, and
-- what operations can be performed on those values.

Predefined data types - part of the C++ language definition.

float, double - real. int - integer. char - single character.
Type char literals use single quotes: 'A' '*' '2'
A string literal is a sequence of characters in double quotes:
"ABCDE" "127" (not the same as **int** 127)
"true" (not the same as bool true)

System-defined types - part of the C++ libraries

Standard I/O stream objects cin, cout defined in iostream library

e.g., string, ofstream - `ofstream cprint ("file.txt");`

User-defined types - e.g., enum type, classes

In C++ a *type* is the same as a *class*.

Declarations

Declarations inform the compiler that it will need to set aside space in memory to hold an object of a particular type (class) with a particular name.

Constant declarations:

Used to associate meaningful names with constants -- items that will never change throughout the execution of the program.

```
const float PI=3.14159;
          //Note: one convention is to use all
          //uppercase letters for constant identifiers
const float METERS_TO_YARDS=1.196;
```

Variable declarations:

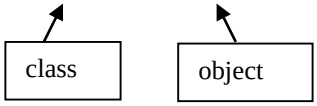
Used to associate identifiers of a given type with memory cells used to store values of this type. - the values stored in the data cells are changeable.

```
char letter;
char letter1, letter2;
float x, y;
```

Object declarations:

Like variables, these are used to associate identifiers of a given type with memory cells used to store values of this type. - the values stored in the data cells are changeable. We use some system-defined classes in the standard C++ class libraries. A class is equivalent to a type; variables can store data values and are called objects.

```
ofstream cprn ("printfile.txt");        //see p. 138 Staugaard
```



The variable's type (or, class) tells the compiler how the variable's values are to be stored and how they may be used. On some computers (DOS PCs) the **int** set of values consists of all integers in the range –32,768 to 32,767. [Why?]

There are nine **int** types:

```
short int     unsigned short int      char
int           unsigned int            signed char
long int      unsigned long int       unsigned char
```

The difference among these 9 types is the range of values they allow. These ranges may depend somewhat on the computer system.

short is the same as **short int**

char type uses 8 bits to store a single character. Is actually a numeric type in that it stores the ASCII (American Standard Code for Information Interchange) code value of the character. Character input is automatically converted; on output the value is converted to the equivalent char first.

```
char          signed char          unsigned char
```

Use unsigned char for a very short bit-string. Depending on the system, char will be equivalent to either signed char or unsigned char.

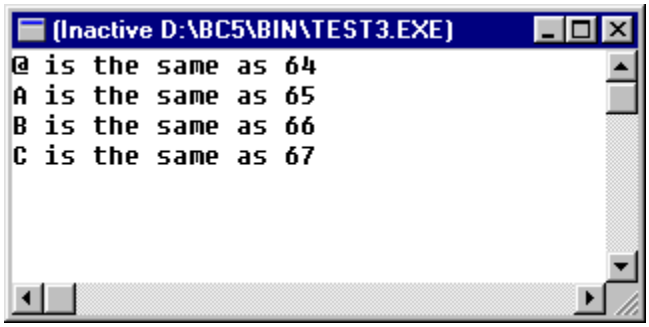
examples using char data type:

```
char c = 54;
char d = 2 * c – 7;
c++ ;
…


…
char c = 64;
cout << c << "  "; //prints '@'
c = c + 1;           //increments c to 65
cout << c << "  "; //prints 'A'
c = c + 1;           //increments c to 66
cout << c << "  "; //prints 'B'
c = c + 1;           //increments c to 67
cout << c << "  "; //prints 'C'
…
```

The int(char) function is a "cast" – In this next example, it converts c from *char* type to *int* type:

```
#include <iostream>
using namespace std;
void main ()
{
   char c = 64;
   cout << c << " is the same as " << int(c) << endl;
   c = c+ 1;
   cout << c << " is the same as " << int(c) << endl;
   c = c+ 1;
   cout << c << " is the same as " << int(c) << endl;
   c = c+ 1;
   cout << c << " is the same as " << int(c) << endl;
   return;
}
```



Typecasting – we can also *cast*, e.g., float(c); int(fl); …

Real number types.

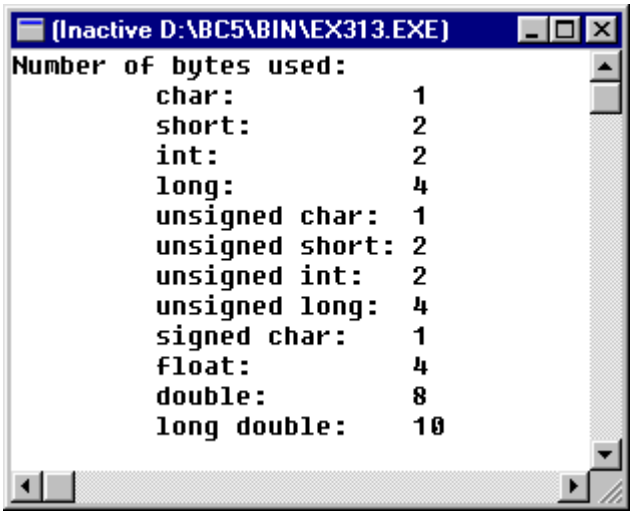
```
float          double                long double
```

On most systems, double uses twice as many bytes as float. In general, float uses 4 bytes, double uses 8 bytes, and long double uses 8, 10, 12, or 16 bytes.

```
//ex313.cpp
// Prints the amount of space each of the 12 fundamental types uses.
// From Hubbard Ex. 3.13, p. 66

#include <iostream>
using namespace std;
void main()
{
   cout << "Number of bytes used:\n ";
   cout << "\t char:           " << sizeof(char) << endl;
   cout << "\t short:          " << sizeof(short) << endl;
   cout << "\t int:            " << sizeof(int) << endl;
   cout << "\t long:           " << sizeof(long) << endl;
   cout << "\t unsigned char:  " << sizeof(unsigned char) << endl;
   cout << "\t unsigned short: " << sizeof(unsigned short) << endl;
   cout << "\t unsigned int:   " << sizeof(unsigned int) << endl;
   cout << "\t unsigned long:  " << sizeof(unsigned long) << endl;
   cout << "\t signed char:    " << sizeof(signed char) << endl;
   cout << "\t float:          " << sizeof(float) << endl;
   cout << "\t double:         " << sizeof(double) << endl;
   cout << "\t long double:    " << sizeof(long double) << endl;

return;
}
```



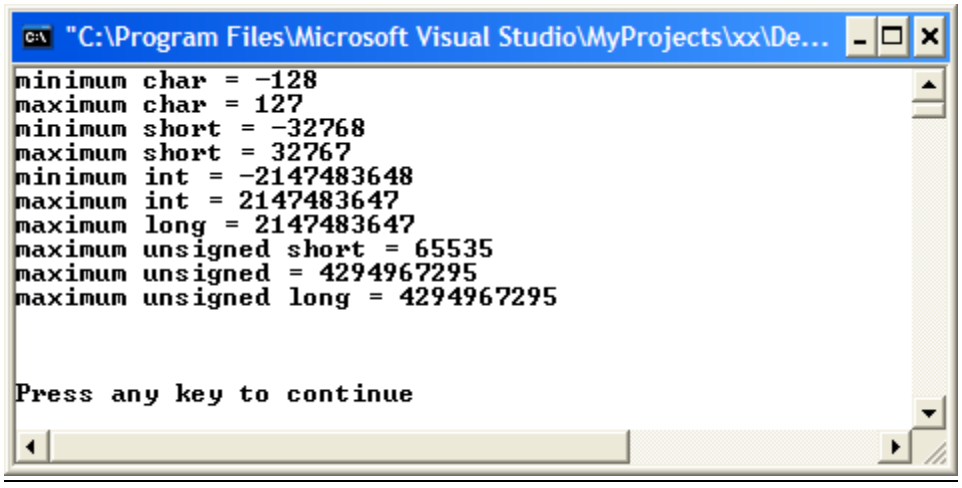
Example:

```
//ex114.cpp
// Prints the constants stored in limits.h
// From Hubbard Ex. 1.14,p.13

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

void main(){
     cout << "minimum char = " << CHAR_MIN << endl;
     cout << "maximum char = " << CHAR_MAX << endl;
     cout << "minimum short = " << SHRT_MIN << endl;
     cout << "maximum short = " << SHRT_MAX << endl;
     cout << "minimum int = " << INT_MIN << endl;
     cout << "maximum int = " << INT_MAX << endl;
     cout << "maximum long = " << LONG_MAX << endl;
     cout << "maximum unsigned short = " << USHRT_MAX << endl;
     cout << "maximum unsigned = " << UINT_MAX << endl;
     cout << "maximum unsigned long = " << ULONG_MAX << endl;
     cout << endl << endl << endl;
     return;
}
```

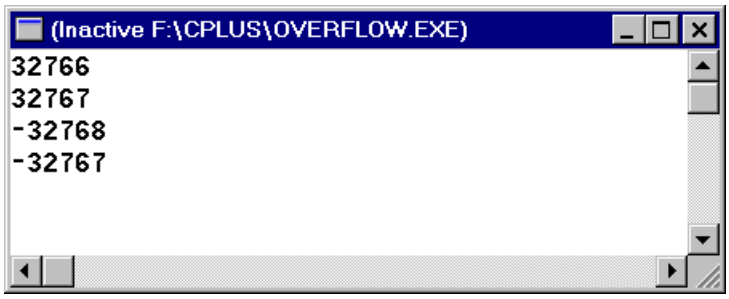
The output:



More on run-time errors: Overflow

```
//overflow.cpp
// Hubbard ex. 1.21. Tests for overflow.

#include <iostream>
#include <limits>
using namespace std;
void main()
{
   short n= SHRT_MAX - 1;
   cout << n++ << endl;
   cout << n++ << endl;
   cout << n++ << endl;
   cout << n++ << endl;
   return;   //successful termination
} //end main
```



Using Strings

The string type is part of the ANSI standard C++. Must include the ***string*** header file, not string.h:

```
#include <string>
```

and may also require:

```
using namespace std;
```

variable declaration:

```
string name = "Jack";     // or
string name;              // initialized as " "
```

use:

```
name = "Jill";
```

entering from keyboard:

```
cout << "Please enter your name:  ";

cin  >> name;
```

Since the space character ends an input stream, this will only enter a first name. For example, if I type Linda Weiser Friedman, the variable name will only contain "Linda" .

One way to get around this problem, the function getline (cin, name, '#') will read into the variable ***name*** all characters (including whitespace) until the # delimiter, and save it in ***name*** without saving the delimiter. You may use any delimiter character you wish.

Another good solution is to simply create two variables, ***firstName*** and ***lastName***.

Arithmetic Operators

- Addition +
- Subtraction –
- Multiplication *
- Division / (is integer division if operators are integers)
- Modulus % (remainder)

e.g., if the value of

$$b + c - \frac{de}{f}$$

is to be assigned to variable x, it is coded:

```
x = b + c - d * e / f;
```

Parentheses may be used. These are evaluated first. e.g., `x = (b + c - d)* e / f;` is evaluated as:

$$\frac{(b + c - d)e}{f}$$

order of operations

( )
* / %
+ -
left to right

ex. `z= p*r%q+w/x-y;` Order of operations?

```
Z   =   P   *   R   %   Q   +   W   /   X   -   Y   ;
    6       1       2       4       3       5
```

Exercise: For each of the following arithmetic expressions, construct the equivalent C++ expression.

$\frac{a+b}{c-d}$ _____

$\frac{a+b}{c^2}$ _____

$d - \frac{a+b}{4c}$ _____

$\frac{b^2 - 4ac}{2a}$ _____

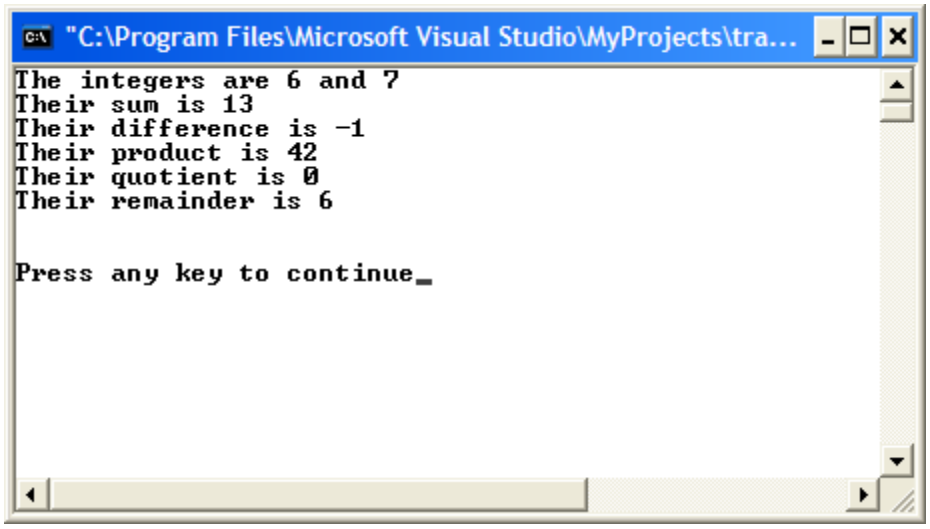
$\frac{a}{b + \frac{c}{d-a}}$ _____

A Simple Program Using Stored Data and Arithmetic

```
// from Schaum's ex 1.26 p. 27
// Prints the sum, difference, etc. of given integers.
#include <iostream>
using namespace std;

int main(){
      int m = 6, n = 7;
      cout << "The integers are " << m << " and " << n << endl;
      cout << "Their sum is " << (m+n) << endl;
      cout << "Their difference is " << (m-n) << endl;
      cout << "Their product is " << (m*n) << endl;
      cout << "Their quotient is " << (m/n) << endl;
      cout << "Their remainder is " << (m%n) << endl << endl <<
endl;
      return 0;
}
```

Trace this program.



Assignment

c = c + 3; same as c +=3;
The += operation adds the value of the expression of the right to the value of the variable on the left and stores the result in the variable on the left.

In general, <var> = <var> op <exp>; can be written as: <var> op = <exp>;

Examples:
c –= 4; same as c = c – 4;
c *= 5; same as c = c *5;
c /= 6; same as c = c /6;

Can we reverse the order of the double operator? e.g.: c =– 4;
No. This simply is the same as the assignment c = –4;

Unary Increment/Decrement Operators

a++; same as a = a + 1; same as ++a;
a––; same as a = a – 1; same as ––a;

a++ postincrement
++a preincrement
a–– postdecrement
––a predecrement

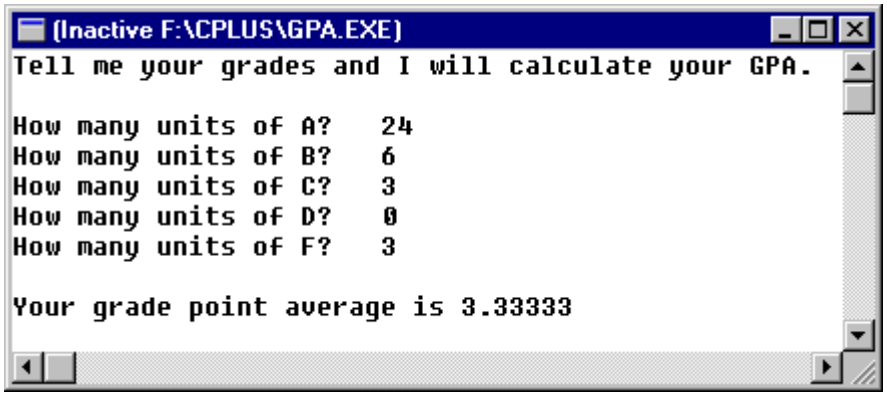
Example:

```
int c;
c = 5;
cout << c << endl;              //prints 5
cout << c++ << endl;            //prints 5 (then increments)
cout << c << endl << endl;      //prints 6
c = 5;
cout << c << endl;              //prints 5
cout << ++c << endl;            //prints 6 (after incrementing)
cout << c << endl;              //prints 6
```

Grade Point Average

```
//gpa.cpp
// This program will calculate your grade point average.

#include <iostream>
using namespace std;
int main()
{
    int A, B, C, D, F;
    float sum, GPA;
    cout << "Tell me your grades and I will calculate your GPA.";
    cout << endl << endl;
    cout << "How many units of A?   ";
    cin >> A;
    cout << "How many units of B?   ";
    cin >> B;
    cout << "How many units of C?   ";
    cin >> C;
    cout << "How many units of D?   ";
    cin >> D;
    cout << "How many units of F?   ";
    cin >> F;
    sum = A + B + C + D + F;
    GPA =  (4*A + 3*B + 2*C + D)/sum;
    cout << endl;
    cout << "Your grade point average is " << GPA <<endl;
    return 0;
} //end main
```

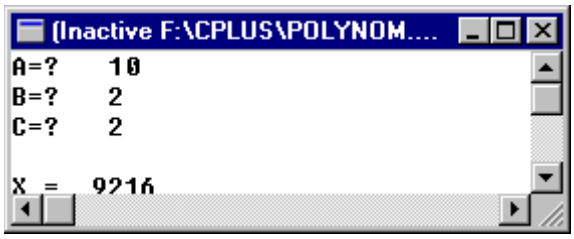


A Polynomial Program

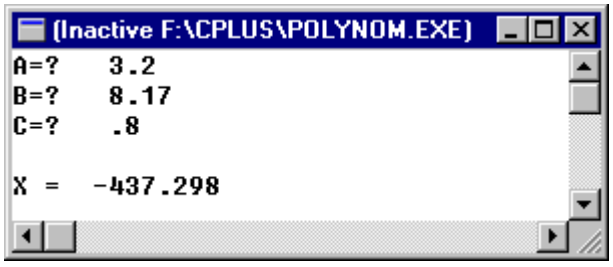
```
//polynom.cpp
// Polynomial Program

#include <iostream>
using namespace std;
int main()
{
    float A, B, C, X;
    cout << "A=?   ";
    cin >> A;
    cout << "B=?   ";
    cin >> B;
    cout << "C=?   ";
    cin >> C;
    X = 5*A*A*B*C*C*C + 8*A*B*B*C*C - 4*B*B*B*C;
    cout << endl;
    cout << "X =  " << X <<endl;
    return 0;
} //end main
```

Run #1:



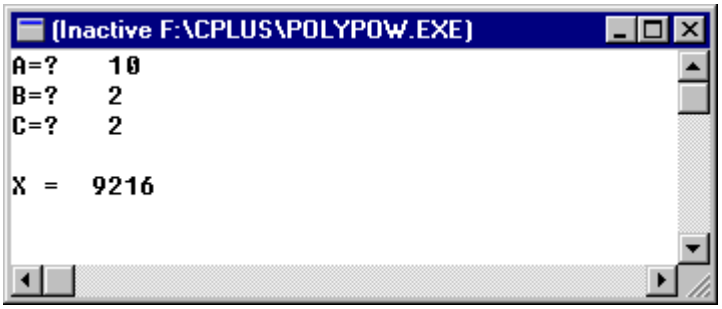
Run #2:



C++ has no arithmetic operator for exponentiation. There is, however, a power function that can do it for us. Use the ***math*** header file.

```
//polypow.cpp
// Polynomial Program
// Using the pow() power function

#include <iostream>
#include <math>
using namespace std;
int main()
{
   float A, B, C, X;
   cout << "A=?   ";
   cin >> A;
   cout << "B=?   ";
   cin >> B;
   cout << "C=?   ";
   cin >> C;
   X = 5*pow(A,2)*B*pow(C,3) + 8*A*pow(B,2)*pow(C,2) - 4*pow(B,3)*C;
   cout << endl;
   cout << "X =  " << X <<endl;
   return 0;
} //end main
```



Some Built-in Functions in the Math Library - Use the <math.h> header file

		Returned value
abs(a)	absolute value of ***a***	same data type as argument
pow(a1,a2)	a1 raised to the power of a2	data type of argument a1
sqrt(a)	square root of a	same data type as argument
sin(a)	sine of a (a in radians)	double
cos(a)	cosine	double
tan(a)	tangent	double
log(a)	natural logarithm of a	double
log10(a)	base 10 log of a	double
exp(a)	e raised to the power of a	double

[see p. 179 in the Staugaard text for more]

To Output a Report to a Text File

In the program below, the bold items are added in order to produce a report file, that can be printed.

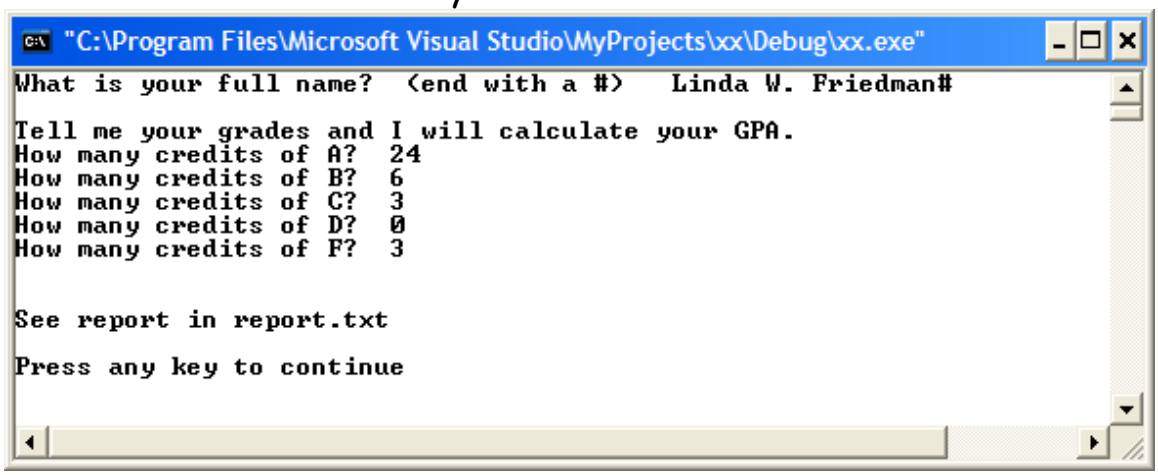
The GPA program:

```
//gpa.cpp
// This program will calculate the user's grade point average

#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main(){
      int a, b, c, d, f;
      float sum, gpa;
      //File report.txt contains the report
      ofstream cprn ("report.txt");
      string name;
      cprn << "School of Hard Knocks\n"
            << "Grade Point Average Program\n\n";
      cout << "What is your full name?  (end with a #)   ";
      getline (cin, name, '#');
      cout << "\nTell me your grades and I will calculate your GPA.\n";
      cout << "How many credits of A?  ";
      cin >> a;
      cout << "How many credits of B?  ";
      cin >> b;
      cout << "How many credits of C?  ";
      cin >> c;
      cout << "How many credits of D?  ";
      cin >> d;
      cout << "How many credits of F?  ";
      cin >> f;
      cprn << "\n\n\nResults\n\n";
      cprn << name << ", these are the grades you entered: \n";
      cprn << "\nGrade \t #credits\n";
      cprn << "\nA \t " << a;
      cprn << "\nB \t " << b;
      cprn << "\nC \t " << c;
      cprn << "\nD \t " << d;
      cprn << "\nF \t " << f;
      sum = a + b + c + d + f;
      gpa = a*4.0 + b*3 + c*2 + d;
      gpa = gpa/sum;
      cprn << "\n\nYour grade point average is " << gpa << endl;
      cout << "\n\nSee report in report.txt\n\n";
      return 0;
}
```

The interactive data entry screen:



This is the textfile report.txt:

```
School of Hard Knocks
Grade Point Average Program



Results

Linda W. Friedman, these are the grades you entered:

Grade       #credits

A      24
B      6
C      3
D      0
F      3

Your grade point average is 3.33333
```

The advantage of sending output to a file for printing is:

- We only need the UGLY console window for interactive screen sessions (like data entry)

- We can produce program output that is larger than the console window, without copying and pasting one screen at a time

- We can produce program output that is formatted – both vertically and horizontally – and so is easier to read and to understand.

Problem: Where is report.txt?

Ans: The systems automatically places it in the "active" file, that is, wherever the project producing it is stored. It will probably be easier for you to find your files if you declare file objects with a path as well as a file name, for example

Instead of:

```
ofstream printfile ("report.txt");
```

it is a good idea to use something like the following so the system will create the report.txt file in the directory of your choosing (*reports*):

```
ofstream printfile ("C:\Documents and
Settings\LWFriedman\My Documents\reports\report.txt");
```

EXERCISE:
There is a problem with the string literal in this example. Why won't it work? Try it.

[HINT: Review 'escape characters']