Unit 3, Lesson 3
Math Operations, Assignment Operators, and Arithmetic Operators

Mr. Dave Clausen
La Cañada High School

Assignment Operator

- You have used the assignment operator (=) to initialize variables, so you already know most of what there is to know about the assignment operator
- The **assignment operator** changes the value of the variable to the left of the operator.
- For example:

```
i = 25;     //changes the value of i to 25
```

Code List 3-1

```
#include  <iostream.h>
//iassign.cpp          iassign.txt
//declaring and initializing variables in two steps
int main ()
{
    int i;                  // declare i as an integer
    i=1000;                 // assign the value 1000 to i
    cout << i <<  endl;
    i=25;                   // assign the value 25 to i
    cout << i <<  endl;
    return 0;
 }
```

Code List 3-2

```
#include <iostream.h>
//multint.cpp          multint.txt
//declaring multiple variables in one step
//initializing multiple variables in one step
int main ()
{
    int i, j, k;        //  declare i, j, and k as integers
    i =j=k=10;   //initialize all of the variables to 10
    cout  << i <<  '\n';
    cout  << j <<  '\n';
    cout  << k <<  '\n';
    return 0;
 }
```

Declare & Initialize

- Constants **MUST** be declared and initialized in the same statement.
- Variables **MAY** be declared and initialized in the same statement.
- For example:

```
int number = 10;
double result = 0.0;
char letter = ' ';  // a space
```

Arithmetic Operators

- A specific set of **arithmetic operators** is used to perform calculations in C++
- These arithmetic operators, shown in Table 3-1, may be somewhat familiar to you
- Addition and subtraction and performed with the familiar + and - operators

Table 3-1

The arithmetic operators are used with two operands, as in the examples in Table 3-1

TABLE 3-1
Arithmetic operators

SYMBOL	OPERATION	EXAMPLE	READ AS...
+	Addition	3 + 8	three plus eight
–	Subtraction	7 – 2	seven minus two
*	Multiplication	4 * 9	four times nine
/	Division	6 / 2	six divided by two
%	Modulus	7 % 3	seven modulo three

Using Arithmetic Operators

Arithmetic operators are most often used on the right of an assignment operator as shown in the examples in Table 3-2

TABLE 3-2
Examples of expressions

STATEMENT	RESULT
cost = price + tax;	cost is assigned the value of price plus tax.
owed = total – discount;	owed is assigned the value of total minus discount.
area = l * w;	area is assigned the value of l times w.
one_eighth = 1 / 8;	one_eighth is assigned the value of 1 divided by 8.
r = 5 % 2;	r is assigned the integer remainder of 5 divided by 2 by using the modulus operator.
x = -y;	x is assigned the value of -y.

Assignment Operator vs. The Equal Sign

- The assignment operator (=) functions differently in C++ from the way the equal sign functions in Algebra.
- Look at the following statement:

  x = x + 10;
- This statement is False in Algebra.
- In C++, the old value of x is incremented by 10 and then assigned to the new value of x.

Code List 3-3

```
//  assign.cpp                  assign.txt
#include <iostream.h>
int main ()
{
        int i = 2;                  //declare and initialize in one step
        int j = 3;                  //declare and initialize in one step
        int k = 4;                   //declare and initialize in one step
        int l;
        float a = 0.5;              //declare and initialize in one step
        float b = 3.0;              //declare and initialize in one step
        float c;
        1 = i + 2;
        cout  <<  1  <<  '\n';
        1 = 1- j;
        cout  <<  1  <<  '\n';
        1 = i  *  j  *  k;
        cout  <<  1  <<  '\n';
        1  =  k  /  i;
        cout  <<  1<<  '\n';
       c = b * a;
       cout << c << '\n';
       c = b / a;
      cout << c << '\n';
      getch();
      return 0;
}
```

Arithmetic operators are used to create expressions

Expression	Result of expression
cost = price + tax	cost is assigned the value of price plus tax
owed = total – discount	owed is assigned the value of total minus discount
area = l * w	area is assigned the value of l times w
one_eighth = 1 / 8	one_eighth is assigned the value of 1 divided by 8
r = 5 % 2	r is assigned the integer remainder of 5 divided by 2 by using the modulus operator
x = -y	x is assigned the value of -y

Assignment Statements

- A Method of putting values into memory locations

  variable name = value;

  a variable name = expression;
- Assignment is made from right to left
- Constants can’t be on left side of statement
- Expression is a constant or variable or combination thereof

Assignment Statements

- Values on right side not normally changed
- Variable and expression must be of compatible data types (more later)
- Previous value of variable discarded to make room for the new value
- For now char, int, and double are compatible with each other

Assignment Examples

- score1 = 72.3;
- score2 = 89.4;
- score3 = 95.6;
- average = (score1 + score2 + score3) / 3.0

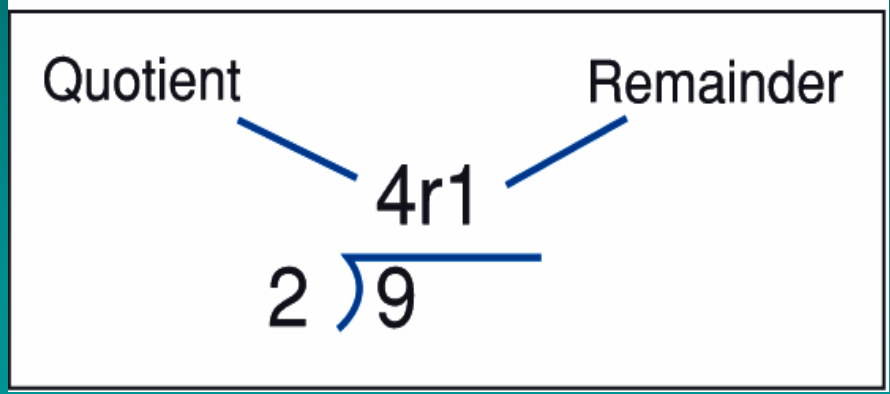
  why not divide by 3 instead of 3.0?

The Modulus operator

- The Modulus Operator, which can be used **only** for integer division, returns the remainder rather than the quotient.
- As shown in figure 3-1, integer division is similar to the way you divide manually

FIGURE 3-1

The division operator and the modulus operator return the quotient and the remainder.



Code List 3-4

```
//  remain.cpp                remain.txt
//  Calculations using assignment statements
#include <iostream.h>
int main ()
{
      int dividend, divisor, quotient, remainder;
      / /  ------------------------Input----------------------------
      cout  <<  “Enter the dividend  “;
      cin  >>  dividend;
      cout  <<  “Enter the divisor  “;
      cin  >>  divisor;

      / /  ----------------------Calculations-----------------------
      quotient = dividend / divisor;
      remainder = dividend % divisor;

      / /  -------------------------Output-----------------------------
      cout  <<  “the quotient is “  <<  quotient;
      cout  <<  “ with a remainder of  “ <<  remainder  <<  ‘\n’;
      return 0;
}
```

Code List 3-5

```
// remain2.cpp                remain2.txt
//calculations in cout statements  - while it can be done, please DON'T DO THIS
#include <iostream.h>

int main()
{
      int dividend, divisor;

      // ---------------------Input------------------------------
      cout << "enter the dividend";
      cin >> dividend;
      cout << "Enter the divisor";
      cin >> divisor;

      // ---------------Calculations in the Output------------Don't Do This For My Class
      cout << "the quotient is " << dividend/divisor;
      cout << "with a remainder of " << dividend % divisor << '\n';
      return 0;
}
```

Incrementing and Decrementing

- Adding or subtracting 1 from a variable is very common in programs. Adding 1 to a variable is called incrementing, and subtracting 1 from a variable is called decrementing.

The ++ and -- Operators

- C++ provides operators for incrementing and decrementing. In C++, you can increment an integer variable using the ++ operator, and decrement using the -- operator, as shown in Table 3-3

TABLE 3-3
Incrementing and decrementing

STATEMENT	EQUIVALENT TO...
counter++;	counter = counter + 1;
counter- -;	counter = counter – 1;

Code List 3-6

```
// inc_dec.cpp                inc_dec.txt
#include <iostream.h>
int main()
{
        int j;          // declare j as int

        j=1;            // initialize j to 1
        cout << "j = " << j << '\n';
        j++;            //increment j
        cout << "j = " << j << '\n';
        j--;            //decrement j
        cout << "j = " << j << '\n';

        return 0;
}
```

Variations of Increment and Decrement

At first glance, the ++ and – operators seem very simple. But there are two ways that each of these operators can be used. The operators can be placed either before or after the variable. The location of the operators affects the way they work.

c++ or ++c

c- - or - - c

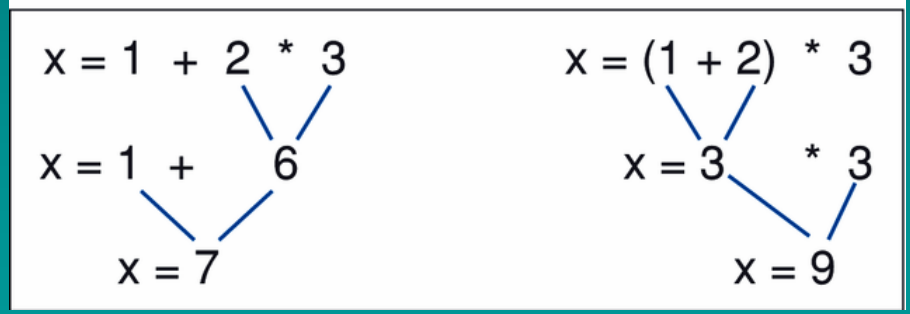
We will use c++ and c- - for this class

Order of Operations

- You may recall from your math classes the rules related to the order in which operations are performed. These rules are called the order of operations. The C++ compiler uses a similar set of rules for its calculations. Calculations are processed in the following order
  - Minus sign used to change sign (-)
  - Multiplication and division (*, /, %)
  - Addition and subtraction (+, -)

•C++ lets you use parentheses to change the order of operations. For example, consider the two statements in Figure 3-2.

FIGURE 3-2
Parentheses can be used to change the order of operations.



Code List 3-7

```
// order.cpp          order.txt
#inlcude  <iostream.h>

int main ()
{
      int answer

      answer = 1 + 2  * 2 + 3;
      cout  <<  answer  <<  '\n';
      answer  =  (1 + 2)  *  (2 + 3);
      cout  <<  answer  <<  '\n';
      answer  =  1 + 2 * (2 + 3);
      cout  <<  answer  <<  '\n';
      answer  =  (1 +2)  *  2 + 3 ;
      cout  <<  answer  <<  '\n';

      return 0;
}
```

Integer Arithmetic

- + Addition
- - Subtraction
- * Multiplication
- / Quotient (Integer Division)
- % Remainder (Modulus)

$$Divisor \overline{)\, Dividend} \;\; \begin{matrix} Quotient \\ + \\ \dfrac{\text{Remainder}}{Divisor} \end{matrix}$$

Integer Order Of Operations

- Expressions within parentheses
  - nested parentheses: from inside out
- * (multiplication), % (modulus), / (division)
  - from left to right
- + (addition), - (subtraction)
  - from left to right

Integer Arithmetic (Examples)

(3-4)*5 =	-5
3 * (-2) =	-6
17 / 3 =	5
17 % 3 =	2
17 / (-3) =	-5
-17 % 7 =	-3
-42+50%17=	-26

Real Number Arithmetic

- Type double:

- + Addition
- - Subtraction
- * Multiplication
- / Division

Real Number
Order Of Operations

- Expressions within parentheses
  - nested parentheses: from inside out
- * (multiplication), / (division)
  - from left to right
- + (addition), - (subtraction)
  - from left to right

Real Number Arithmetic (Examples)

$2.0 * (1.2 - 4.3) = -6.2$

$2.0 * 1.2 - 4.3 = -1.9$

$-12.6 / (3.0 + 3.0) = -2.1$

$3.1 * 2.0 = 6.2$

$-12.6 / 3.0 + 3.0 = -1.2$

Compound Assignments

- "Short hand" notation for frequently used assignments (We will not use these for readability of our programs.)

Short hand	Longer form
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Sample Program

Here is a program that prints data about the cost of three textbooks and calculates the average price of the books:

Books.cpp

Books.txt

Input

- Cin (pronounced see-in)

  gets data from keyboard, the standard input stream

  extractor operator >>

  - obtain input from standard input stream and direct it to a variable (extract from stream to variable)

  inserter operator <<

  - insert data into standard output stream

  EGG ILL

  - Extractor Greater Greater, Inserter Less Less

Input

- Data read in from keyboard must match the type of variable used to store data
- Interactive Input

  enter values from keyboard while the program is running

  cin causes the program to stop and wait for the user to enter data from the keyboard

  prompt the user for the data (user friendly)

Input: Sample Programs

No prompt for any of the data values:

input.cpp
input.txt

One prompt for each data value (preferred)

triples.cpp
triples.txt