PIC 10A

Lecture 3: More About Variables, Arithmetic, Casting, Assignment

Assigning values to variables

Our variables last time did not seem very “variable”. They always had the same value!

Variables stores a value.

This value can be changed.

```
int x=7;

cout << "Value of x is: " << x << "\n";

x = 2; // Note that after a variable has been declared
       // you never put its data type in front!

cout << "Value of x is: " << x << "\n";
```

Example

Another example;

```
int seconds_in_hour = 3600;
int hours_in_day = 24;
int seconds_in_day = seconds_in_hour * hours_in_day;
```

Note how we are using other variables to initialize `seconds_in_day` variable.

The variable whose value is changing is always on the left. The value that it is being set to is defined by the stuff on the right.

Example

```
int x=3;
int y=2;
x=y+5;
```

What is the value of x?

What is the value of y?

```
int x=3;
x=x+2;
```

The x on the right is the new x. The old value of x will be used on the left.

Hence this expression reads like:

```
x=3+2;
```

and value of x will be 5.

Book example

```
#include <iostream>
using namespace std;
int main()
{
    int count;
    double total;

    cout << "How many pennies do you have? ";
    cin  >> count;
    total = count * 0.01;

    cout << "How many nickels do you have? ";
    cin >> count;
    total = count * 0.05 + total;

    cout << "How many quarters do you have? ";
    cin >> count;
    total = count * 0.25 + total;

    cout << "Total value of your coins is: " << total << "\n";

    return 0;
}
```

Some warnings

Make sure your data type matches the variable type:

```
double total;
total = "Hello!"; // Error because a string cannot be
                  //converted to a double
```

This declaration is ok:

```
double total = count * 0.01;
```

This is not:

```
double total = count * 0.05 + total; //Error
```

The second line is an error because you cannot use the variable you are declaring in its definition!

More warnings

We only need to declare a variable once!

```
double decimal=0.8;
double decimal=1.3; // This is wrong!
```

With the above lines you might get a way with a warning, but

```
double decimal=0.8;
int decimal =1; //This is super wrong!
```

You cannot change a variables data type.

Constants

Never “hard code” numbers into your program. If you have an important number that you need to use in your code, assign its value to a variable and then use the variable in place of the number.

If your value always stays the same, use a constant declaration.

Syntax:

```
const type var_name = value;
```

eg.

```
const double PI = 3.14159;
```

Advantages of a constant

If you ever want to change your code and modify the value (Maybe you want more decimal accuracy for PI) of your constant you need to only change one line of code

Once a variable has been declared constant its value cannot be changed. The compiler will give an error if your code tries to change a constant. This helps you catch errors.

Anyone reading your code will understand your constant variable better.

Common assignment shorthand

We have already seen assignments like:

```
x = x + 1;
```

This means new value of variable x will be the old value plus 1.

We say we are incrementing x.

Programmers are lazy (and this types of assignments come up often)

Shorthand:

```
x+=1;
```

You can do

```
x+=12347834;
```

Common assignment shorthand

There are other shortcuts as well

```
x = x - 7;
```

```
x -= 7;
```

or

```
x = x * 8;
```

Can be written

```
x *=8;
```

or even

```
x = x/2;
```

```
x /=2;
```

Common assignment shorthand

Back to the first example:

```
x = x + 1;
```

This is same as

```
x += 1;
```

But this comes up so often that programmers have even shorter notation:

```
x++;
```

or

```
++x;
```

Both increment x by one. There is a subtle difference, but we'll talk about it in a bit.

Common assignment shorthand

Of course there has to be

x--;

and

--x;

These are both equivalent to

x = x – 1;

Before C++ there was a language called C. How do you think C++ got its name?

Difference between ++x and x++

++x; is called pre increment and x++ is called post increment.

If you have two lines

```
++x;
```

and

```
x++;
```

There is no difference. Both do exactly same thing. They increase the value of x by one.

```
int x=0;
x++; // Value of x is now 1
++x; // Value of x is now 2
```

Difference between ++x and x++

The difference only comes out when you combine the increment with another command.

```
int x=0;

cout << ++x;
cout << x;

cout << x++;
cout << x;
```

you have to think of the line `cout << x++;` as two separate lines:

```
cout << x;
x++;
```

When using post increment the incrementing happens after the command.
When using pre increment the incrementing happens before the command.

Arithmatic

Basic operators are:

Symbol	Operation	Example code
+	Addition	x = 7 + 8;
-	Subtraction	x = -2-y;
*	Multiplication	x = 5*y;
/	Division	x=7/8;
%	Mod	x = 6%2;

You CANNOT write (x + z)y for (x + z)*y as in a math class

You CANNOT use ^ for power, e.g. x^3 is NOT allowed, use x*x*x instead or pow function which we meet later.

Evaluating expressions

Arithmetic expressions are always evaluated from left to right using our normal rules for precedence.

Highest precedence

( ) Anything in parenthesis will be evaluated first

Normal precedence

* / % These are all equal weight

Low Precedence

+ -

Examples

```
int x;
x = 3 + 4 * 5; // will perform multiplication first
```

You can use parenthesis

```
x = (3 + 4) * 5;// will perform addition first
```

what happens:

```
int a=9/3*3;
```

Integer division

Division in C++ is bit strange:

```
double r;

r=3/2;  // What is value of r?  It is 1 and NOT 1.5.
```

3.0/2 or 3.0/2.0 or 3/2.0 all evaluate as doubles.

As long as one of the numbers is a double we have a double division

Best have all numbers same type to elimate confusion

```
double r;
r=3.0/2.0; // evaluates as 1.5
```

But note:

```
int a;
a=3.0/2.0;  // a has value 1;

cout << 1/5 * 3; // What is the output?
```

What is % operator?

% is the remainder operator (also called mod operator).

7%3 is 1

8%5 is 3

Important special case:

Any even number % 2 is 0.

Any odd number % 2 is 1.

Casting

C++ is smart it can automatically convert types into other types. This is called implicit casting.

```
int a;
a=5.6; // compiler will give a warning but not an error.
```

Good code generates no warnings when we compile.

What is output of:

```
cout << 5/2; // output is 2
```

Both 5 and 2 are integers so the expression evaluates as an integer division. So if you want a double then you should have:

```
cout << 5/2.0; // output 2.5
```

But what if you have variables?

```
int a=5;
int b=2;
cout << a/b;
```

Casting

```
int a=5;
int b=2;

cout << a/b;
```

we have to change the type.

```
cout << (double)(a)/b;
```

alternatively

```
cout << (double)a/b;
```

What happens if you write

```
cout << (double)(a/b);
```

Note this is only temporary change of data type used in evaluation of an expression. Varaible a is NOT permanently changed to a double. Variables type cannot be permanently changed.

Example

• We want important fixed quantities, like those used for conversions, to be constants.

• Avoid "magic numbers" in your code.

```
const int SECONDS_PER_HOUR = 3600;
```

• Given an integer seconds, determine how many hours have passed (want a whole number) .

```
int hours = seconds / SECONDS_PER_HOUR;
```

• Given an integer seconds, convert to the number of hours (with the decimal) .

```
double hours = (double) seconds / SECONDS_PER_HOUR;
```

Example

A box for donuts holds 12 donuts. Each day the baker packs the donuts into boxes and brings the left over donuts home. He decides to write a C++ program that allows him to calculate for a given amount of donuts how many boxes he will need and how many donuts he will have left over.

```
int amount;
int boxes_needed;
int left_over;

cout << "How many donuts are there?";
cin >> amount;

boxes_needed = amount / 12;
left_over = amount % 12;

cout << "You will need " << boxes_needed << " boxes"
     << " and you will have " << left_over
     << "many donuts left over";
```

cmath library

To do more complicated mathematical equations you need the cmath libra

```
#include <cmath>
```

If you include this file you get to use your favorite functions including:

square root of x - `sqrt(x)`

Raise x to power of y - `pow (x,y)`

Exponential, e to the power of x- `exp(x)`

cos or sin x - `cos(x), sin(x)`

```
double x=0.7;
double y;
y = sqrt(x);
```

Rounding

Easy way to round is to use a little trick

Add 0.5 to the number you are trying to round and then cast the result temporarily as an integer.

```
double x = (int)(y +0.5);
```

Lets see why this works. Consider the following three cases:

```
double y = 9.4;
double x = (int)(y +0.5);
```
9.4 + .5 is 9.9 the cast to integer will truncate it to 9 then the assignment gives x value 9.0

```
double y = 9.7;
double x = (int)(y +0.5);
```
9.7 + .5 is 10.2 the cast to integer will truncate it to 10 then the assignment gives x value 10.0

```
double y = 9.5;
double x = (int)(y +0.5);
```
9.5 + .5 is 10.0 the cast to integer will make it 10 then the assignment gives x value 10.0