Modifying Variables (2.2)

- The contents in variables can "vary" over time (hence the name!).
- Variables can be changed by
  - assigning to them
    - The assignment statement
  - using the increment or decrement operator
  - inputting into them
    - The input statement

Modifying Variables – The Assignment Statement

- An *assignment statement*

  stores a new value in a variable, replacing the previously stored value.

```
identifier = expression;
```


Modifying Variables – The Assignment Statement

EX:

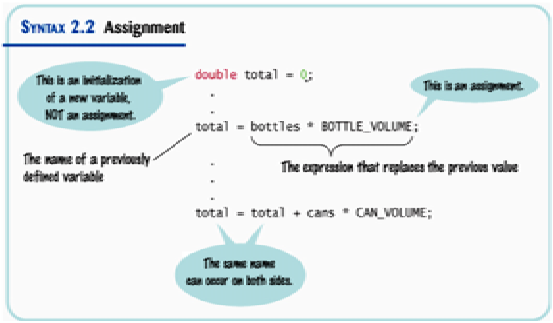
```
cans_per_pack = 8;
```

This assignment statement changes the value stored in `cans_per_pack` to be 8.

The previous value is replaced.

Modifying Variables – The Assignment Statement

Syntax 2.2 Assignment



Modifying Variables – The Assignment Statement

- There is an important difference between a variable definition and an assignment statement:

```
int cans_per_pack = 6; // Variable definition
...
cans_per_pack = 8; // Assignment statement
```

- The first statement is the *definition* of `cans_per_pack`.
- The second statement is an *assignment statement*. An *existing* variable's contents are replaced.

Modifying Variables – The Assignment Statement

- The = in an assignment does ***not*** mean the left hand side is equal to the right hand side as it does in math.
- = is an instruction to do something:
  copy the value of the expression on the right ***into*** the variable on the left.
- Consider what it would mean, mathematically, to state:

```
counter = counter + 1;
```

counter EQUALS counter + 1 ?

assignment operator

$x = x + 1$

$-x \quad -x$

$0 = 1$ ∴ no solution

Modifying Variables – The Assignment Statement

```
counter = 11; // set counter to 11
counter = counter + 1; // increment
```

by 1

Modifying Variables – The Assignment Statement

```
counter = 11; // set counter to 11
counter = counter + 1; // increment
```

1. Look up what is currently in counter (11)

Modifying Variables – The Assignment Statement

```
counter = 11; // set counter to 11
counter = counter + 1; // increment
```

1. Look up what is currently in counter (11)
2. Add 1 to that value (12)

Modifying Variables – The Assignment Statement

```
counter = 11; // set counter to 11
counter = counter + 1; // increment
```

1. Look up what is currently in counter (11)
2. Add 1 to that value (12)
3. *copy* the result of the addition expression *into* the variable on the left, changing counter

```
cout << counter << endl;
```

12 is shown

Modifying Variables – Increment and Decrement

- Changing a variable by adding or subtracting 1 is so common that there is a special shorthand for these:

add 1 subtract 1

The increment and decrement operators.

```
counter++; // add 1 to counter
counter--; // subtract 1 from counter
```

++ increment operator

-- decrement operator

Modifying Variables – Increment and Decrement

C++ was based on C and so it's one better than C, right?

Guess how C++ got its name!

Modifying Variables – Input Statements

- Sometimes the programmer does not know what should be stored in a variable – but the user does.
- The programmer must get the input value from the user
  - Users need to be prompted (how else would they know they need to type something?
  - ① – Prompts are an output statements
- The keyboard needs to be read from
  - ② – This is done with an input statement

↑
cin

Modifying Variables – Input Statements

The input statement

- To read values from the keyboard, you input them from an object called `cin`.
- The << operator denotes the "send to" command.
- `cin >> bottles;` is an *input statement*.

Of course, `bottles` must be defined earlier.

we use cout << ...

but cin >> ....

Modifying Variables – Input Statements

Syntax 2.3 Input Statement

Example:

Don't use endl here.

① Display a prompt in the console window.
② Define a variable to hold the input value.
③ The program waits for user input, then places the input into the variable.

```
cout << "Enter the number of bottles: ";
int bottles;
cin >> bottles;
```


Good programming practice is to

define variables when you need them

(not at the top of the program necessarily)

Known Values – Constants for Known, Constant Values

- Sometimes the programmer knows certain values just from analyzing the problem, for this kind of information, programmers use the reserved word `const`.
- The reserved word `const` is used to define a constant.
- A `const` is a variable whose contents cannot be changed and must be set when created.
  (Most programmers just call them constants, not variables.)
- Constants are commonly written using capital letters to distinguish them visually from regular variables:

EX:

```
const double BOTTLE_VOLUME = 2;
```

↑ # liters in a 2 liter bottle

Known Values – Constants for Known, Constant Values

Another good reason for using constants:

```
double volume = bottles * 2;
```

What does that 2 mean?

Known Values – Constants for Known, Constant Values

If we use a constant there is no question:

```
double volume = bottles * BOTTLE_VOLUME;
```

Any questions?

Known Values – Constants – No Magic Numbers!

And still another good reason for using constants:

```
double bottleVolume = bottles * 2;
double canVolume = cans * 2;
```

What does *that* 2 mean?

— *WHICH 2?*

That 2 is called a "***magic number***"

(so is that one)

because it would require magic to know what 2 means.

It is not good programming practice to use magic numbers.
Use constants.

Known Values – Constants for Known, Constant Values

And it can get even worse ...

Suppose that the number 2 appears hundreds of times throughout a five-hundred-line program?

Now we need to change the BOTTLE_VOLUME to 2.23 (because we are now using a bottle with a different shape)

How to change ***only*** some of those magic numbers 2's?

Known Values – Constants for Known, Constant Values

Constants to the rescue!

```
const double BOTTLE_VOLUME = 2.23;
const double CAN_VOLUME = 2;

...

double bottleVolume = bottles * BOTTLE_VOLUME;
double canVolume = cans * CAN_VOLUME;
```

(Look no magic numbers!)

The Complete Program for Volumes

ch02/volume2.cpp

```
#include <iostream>

using namespace std;

int main()
{
   const double BOTTLE_VOLUME = 2;
   const double LITER_PER_OUNCE = 0.0296;
   const double CAN_VOLUME = 12 * LITER_PER_OUNCE;

   double total_volume = 0;

   // Read number of bottles
   // Display prompt and get user response
   cout << "Please enter the number of bottles: ";
   int bottles;
   cin >> bottles;
```


The Complete Program for Volumes

```
   // Start the computation of the total volume
   total_volume = bottles * BOTTLE_VOLUME;

   // Read number of cans
   cout << "Please enter the number of cans: ";
   int cans;
   cin >> cans;

   // Update the total volume
   total_volume = total_volume + cans * CAN_VOLUME;

   cout << "Total volume: " << total_volume << endl;

   return 0;
}
```

Compound Assignment Operators

In C++, you can combine arithmetic and assignments.
For example, the statement

```
total += cans * CAN_VOLUME;
```

is a shortcut for

```
total = total + cans * CAN_VOLUME;
```

Similarly,

```
total *= 2;
```

is another way of writing

```
total = total * 2;
```

Many programmers *prefer* using this form of coding.

Arithmetic Operators

C++ has the same arithmetic operators as a calculator:

- `*` for multiplication: `a * b` (not `a * b` or `ab` as in math)
- `/` for division: `a / b` (not + or a fraction bar as in math)
- `+` for addition: `a + b`
- `-` for subtraction: `a - b`

Arithmetic Operators – Precedence

Just as in regular algebraic notation,
* and / have higher precedence
than + and –.

In `a + b / 2`,
the `b / 2` happens first.

Arithmetic Operators – Two Kinds of Division

- If both arguments of / are integers, the remainder is discarded:
  `7 / 3` is `2`, not `2.5`
- but
  `7.0 / 4.0`
  `7 / 4.0`
  `7.0 / 4`
- all yield `1.75`.

Arithmetic Operators – Getting the Remainder

- The % operator computes the remainder of an integer division.
- It is called the ***modulus operator*** (also modulo and mod)

- It has nothing to do with the % key on a calculator

Arithmetic Operators – Getting the Remainder

Time to break open the piggy bank.

You want to determine the value in dollars and cents stored in the piggy bank.
You obtain the dollars through an integer division by 100.
The integer division discards the remainder.
To obtain the remainder, use the % operator:

```
int pennies = 1729;
int dollars = pennies / 100; // Sets dollars to 17
int cents = pennies % 100; // Sets cents to 29
```

(yes, 100 is a magic number)