# Chapter Two: Fundamental Data Types



## Chapter Goals

- To understand the properties and limitations of integer and floating-point numbers
- To write arithmetic expressions and assignment statements in C++
- To appreciate the importance of comments and good code layout
- To be able to define and initialize variables and constants
- To learn how to read user input and display program output
- To use the standard C++ `string` type to define and manipulate character strings
- To be able to write simple programs that read numbers and text, process the input, and display the results



## Defining Variables (2.1)

- A variable
  - is used to store information (values): the contents of the variable
    - A variable can contain one piece of of information at a time.
  - has an identifier: the name of the variable

    The programmer picks a good name
    - A good name describes the contents of the variable or what the variable will be used for



## Defining Variables

Parking garages store cars.



## Defining Variables

Each parking space is identified
– like a variable's identifier

A each parking space in a garage "contains" a car
– like a variable's current contents.



## Defining Variables

and
each space can contain only *one* car

and
*only* cars, not buses or trucks

## Defining Variables – Type and Initialization

- When creating variables, the programmer specifies the type of information to be stored.
  - (more on types later) e.g. integer or real or string
- Unlike a parking space, a variable is often given an initial value.
  - *Initialization* is putting a value into a variable when the variable is created.
  - Initialization is not required.



## Defining Variables

The following statement defines a variable.

`cans_per_pack` is the variable's name. (identifier)

EX: `int cans_per_pack = 6;`

`int` (type) indicates that the variable `cans_per_pack` will be used to hold integers.

`= 6` (initialization) indicates that the variable `cans_per_pack` will initially contain the value 6.



## Defining Variables

**Syntax 2.1 Variable Definition**

```
int cans_per_pack = 6;
```

Types introduced in this chapter are the number types `int` and `double` and the `string` type

Must obey the rules for valid names

A variable definition ends with a semicolon.

Use a descriptive variable name.

Supplying an initial value is optional, but it is usually a good idea.



## Defining Variables

**Table 1 Variable Definitions in C++**

| Variable Name | Comment |
|---|---|
| `int cans = 6;` | Defines an integer variable and initializes it with 6. |
| `int total = cans + bottles;` | The initial value need not be a constant. (Of course, `cans` and `bottles` must have been previously defined.) |
| ⊘ `bottles = 1;` | **Error:** The type is missing. This statement is not a definition but an assignment of a new value to an existing variable – see Section 2.2. |
| ⊘ `int bottles = "10";` | **Error:** You cannot initialize a number with a string. |
| `int bottles;` | Defines an integer variable without initializing it. This can be a cause for errors – see Common Error 2.2 on page 40. |
| `int cans, bottles;` | Defines two integer variables in a single statement. In this book, we will define each variable in a separate statement. |



## Choosing Variable Names

- When you define a variable, you should pick a name that explains its purpose.
- For example, it is better to use a descriptive name, such as `can_volume`, than a terse name, such as `cv`.



## Choosing Variable Names

In C++, there are a few simple rules for creating variable names:

## Rules for Variable Names

1. Variable names must start with a letter or the underscore (_) character, and the remaining characters must be letters numbers, or underscores. cannot start w/a #
2. You cannot use other symbols such as $ or %. Spaces are not permitted inside names; you can use an underscore instead of a space, as in `can_volume`.
3. Variable names are *case-sensitive*, that is, `CanVolume` and `canvolume` are different names.
   For that reason, it is a good idea to use only lowercase letters in variable names.
4. You cannot use *reserved words* such as `double` or `return` as names; these words are reserved exclusively for their special C++ meanings.



## Rules

| Variable Name | Comment |
|---|---|
| can_volume1 | Variable names consist of letters, numbers, and the underscore character. |
| x | In mathematics, you use short variable names such as x or y. This is legal in C++, but not very common, because it can make programs harder to understand. |
| ⚠ CanVolume | **Caution:** Variable names are case-sensitive. This variable name is different from canvolume. |
| ⊘ 6pack | **Error:** Variable names cannot start with a number. |
| ⊘ can volume | **Error:** Variable names cannot contain spaces. |
| ⊘ double | **Error:** You cannot use a reserved word as a variable name. |
| ⊘ ltr/fl.oz | **Error:** You cannot use symbols such as / or . |



## Number Literals

A number written by a programmer is called a *number literal*.

There are rules for writing literal values:

1. integers do not have decimal points
2. real #s (double) have decimal points OR are written in exponential notation
3. no commas in your #s
4. no fractions w/integers



## Number Literals

| Number | Type | Comment |
|---|---|---|
| 6 | int | An integer has no fractional part. |
| -6 | int | Integers can be negative. |
| 0 | int | Zero is an integer. |
| 0.5 | double | A number with a fractional part has type double. |
| 1.0 | double | An integer with a fractional part .0 has type double. |
| 1E6 $1 \times 10^6$ | double | A number in exponential notation: $1 \times 10^6$ or 1000000. Numbers in exponential notation always have type double. |
| 2.96E-2 | double | Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$ |
| ⊘ 100,000 | | **Error:** Do not use a comma as a decimal separator. |
| ⊘ 3 1/2 | | **Error:** Do not use fractions; use decimal notation: 3.5 |



## Number Ranges – Not Standardized

The C++ Standard does not completely specify the number of bytes or ranges for numeric types.



## Number Ranges – Not Standardized

bit = 0 or 1 **Table 4 Number Types** 8 bits = 1 byte

| Type | Typical Range | Typical Size |
|---|---|---|
| ★ int | −2,147,483,648 . . . 2,147,483,647 (about 2 billion) | 4 bytes |
| unsigned | 0 . . . 4,294,967,295 | 4 bytes |
| short | −32,768 . . . 32,767 | 2 bytes |
| unsigned short | 0 . . . 65,535 | 2 bytes |
| ★ double | The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits | 8 bytes |
| float | The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits | 4 bytes |

## long long – Not Standard C++

Some compiler manufacturers have added other types like:

```
long long
```

integer type

| | | |
|---|---|---|
| long long | -9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807 | 8 bytes |

This type is not in the C++ standard as of this writing.



## Comments

- *Comments* are explanations for human readers of your code (other programmers).
- The compiler ignores comments completely.

Ex:

```
double can_volume = 0.355; // Liters in a 12-ounce can
```

Comment



## Comments

Comments can be written in two styles:

- Single line:

```
double can_volume = 0.355; // Liters in a 12-ounce can
```

The compiler ignores everything after // to the end of line

- Multiline for longer comments:

```
/*
   This program computes the volume (in liters)
   of a six-pack of soda cans.
*/
```



## Notice All the Issues Covered So Far in this Program

Ex:

ch02/volume1.cpp

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

/*
   This program computes the volume (in liters) of a
   six-pack of soda cans.
*/
int main()
{
   int cans_per_pack = 6;
   double can_volume = 0.355; // Liters in a 12-ounce can

   cout << "A sixpack of 12-ounce cans contains "
      << cans_per_pack * can_volume << " liters." << endl;

   return 0;
}
```

always indicate "purpose" as a comment

int * double → double



## Common Errors – Using Undefined variables

You must define a variable before you use it for the first time. For example, the following sequence of statements would not be legal:

```
double can_volume = 12 * liter_per_ounce;
double liter_per_ounce = 0.0296;
```

? ?

Statements are compiled in top to bottom order. When the compiler reaches the first statement, it does not know that `liter_per_ounce` will be defined in the next line, and it reports an error.



## Common Errors – Using Uninitialized Variables

Initializing a variable is not required, but there is always a value in every variable, even uninitialized ones. Some value will be there, the flotsam left over from some previous calculation or simply the random value there when the transistors in RAM were first turned on.

```
int bottles; // Forgot to initialize
int bottle_volume = bottles * 2;// Result is unpredictable
```

What value would be output from the following statement?

```
cout << bottle_volume << endl; // Unpredictable
```