Variables

An individual function consist of nothing more than a set of instructions for manipulating data. At bottom, it is very like a recipe.

The list of ingredients in Charlie's Asparagus corresponds to the data in a function

The directions are like instructions. In particular, they proceed *sequentially* in steps.

Almost all cookbooks list recipes in the format you see at right—and C++ uses the same format. Data must be listed—we would say declared—before the instructions can refer to it.

Charlie's Asparagus

1/2 lb asparagus	1 tbls. soy sauce
2 tbls water or lemon juice	5 drops Sesame oil
1/4 cup mayonnaise	

Directions

Put asparagus spears in a microwavable dish with a tight fitting lid. Add water or lemon juice. Fit lid & microwave on high for about four minutes.

Put mayonnaise in a small bowl. Add other ingredients and mix well. Drizzle onto hot asparagus. Serve

So we have two aspects of a function:

1. Its data
2. Its instructions

Here we focus on the data

Variables are the mechanism C++ uses to store data.

They are similar to the ingredients in a recipe **except that** variables generally change throughout the program whereas ingredients are always the same for a recipe.

That's why they are called variables and not constants. C++ does have constants. We'll talk about them as part of this topic.

The name

variable is taken from Mathematics. To see how it is defined roll the mouse over the highlighted name.

We get two things from these definitions that are fundamental to variables

1. a variable can take on any *value* from a set of values.
2. a variable *holds* data

Type

Point 1 implies that different variables may take on values from different sets. This is the notion of

type .

C++ supports all kinds of different types, for example:

int
: integer variables can hold any integer between -MAX_INT and +MAX_INT where MAX_INT is a constant which depends upon the operating system and compiler you are using.

char
: character variables hold any character in C++'s alphabet. Includes
  - all the upper and lower-case letters (A-Z and a-z)
  - the digits (0-9)
  - standard punction characters (.,<>?":}{} etc.)
  - some special characters ('\n' - a newline, '\t' a tab, etc.)

double
: used for real numbers. Even here the range of values is finite (although it is very large).

The Bin Model of Data

Point 2 implies that the data has to be held *somewhere*. In fact, computer data (and therefore variables) are held in different kinds of computer memory.

Don't worry about the kinds. Just think of a variable as a kind of box or bin.

1. There's one bin for every variable.
2. The bin is sized to fit the type of data exactly.

If you imagine millions of bins stacked up, you have a pretty good idea of what computer memory looks like.

Finding the Right Bin

With millions of bins, how do we find where we put something? Well, we could

1. put a label on the bin .
2. remember the number of the bin we put it in.

The label we put on a bin is called its

name . The number is known as its *address* (just like a street address which let you find the right house on a long street).

Names are for people (programmers). The computer uses the number.

Variable Attributes

If you've done any word processing at all you know there's lots of things that define a font.

There's the name of the font, for example the standard HTML fonts

Arial (sans serif)   Times New Roman (serif)   `Courier New (mono)`

There's the style of the font: normal **bold** *italic* ***bold italic***

We can colour the font: red blue green

Or we can change its size: normal bigger even bigger smaller

And we can do all this good stuff at once: **bigger bold red arial** *smaller italic blue Times New Roman*

name, style, colour and size are all *attributes* of a font.

Variables have attributes as well. so far we have learned about four of them:

type which determines the size of the bin and the kind and range of values it can store.

address which determines the location of the bin in memory.

name which gives us a convenient way to find/refer to the bin.

value which is the actual piece of data contained in the bin.

There are others, but these will do for now.

Declaring Variables

Just as in a recipe, in C++ we must write down a variable before we can use it. This is called a *declaration*.

Declaration Statement

Forms:
Type *Identifier* **;** **|**
Type *Identifier* **=** *Value* **;**

Examples:

```
int i;
int count = 3;
double x = 17.235;
char c = 'A';
```

Interpretation: the compiler sets space aside in memory for two int variables (called `i` and `count`) a double variable called `x` and a char variable called `c`. `count`, `x` & `c` are initialized, `i` is not.

We give each variable declaration a line of its own (although not required by the language it *is* good style).

Lets see some of this in action

simple_variables.cpp

```
/*******  Simple Variables Demonstration ********
    To demonstrate on the teaching machine the four
    fundamental attributes of simple C variables
    name, type, value, location

*******************************************/

#include <iostream.h>   // info from standard library
using namespace std;    // cout is in the std namespace

int main(){
    int i = 110;
    char letter;
    bool flag;
    double pi;

    letter = 'n';
    flag = false;
    pi = 3.14159;


    cout << "i = " << i << '\n';
    cout << "flag = " << flag << '\n';
    cout << "letter = " << letter << '\n';
    cout << "pi = " << pi << '\n';
    return 0;
}
```

Identifiers

A name, produced by the programmer, used for functions and variables.

Syntax:

1. must start with a letter or underscore (_).
2. may contain letters, digits or underscore.
3. case (capitalization) is important.
4. must not be a *reserved word* (e.g., `int`).
5. can be quite long (at least 255 characters are significant).

Choosing Identifiers (Style)

1. Make names meaningful—try to make it clear what they represent.
2. Mix case to separate words, e.g., `CanYouReadThis`.
3. Constants in all capitals, with underscore to separate words, e.g., `MAX_ITEMS`

4. Avoid underscore as the first character (compilers use those to mean something special).
5. Try to be concise—make the identifier just long enough.

Constants

A memory cell whose value never changes (for the duration of the program).

Can be literal value (e.g., 5, `"A message."`), or

referred to by name: a named (symbolic) constant.

Constant declaration: State the type, name and value of a constant:

Constant Declaration

Form:
`const` *Identifier* **=** *Literal* **;**

Example:

```
const int MAX_ITEMS = 5;
```

Interpretation: a memory bin sized to hold an `int` called `MAX_ITEMS` is created. Its value is set to `5` and will not be allowed to change throughout the program

If a constant is to be used more than once use the named constant form

Easier to maintain than a literal

This page last updated on Wednesday, January 14, 2004