Variables and types

The usefulness of the "Hello World" programs shown in the previous chapter is rather questionable. We had to write several lines of code, compile them, and then execute the resulting program, just to obtain the result of a simple sentence written on the screen. It certainly would have been much faster to type the output sentence ourselves.

前一章中显示的“Hello World”程序的用处是相当值得怀疑的。我们必须编写几行代码，编译它们，然后执行产生的程序，只是为了获得屏幕上写的一个简单句子的结果。我们自己输入输出句子肯定会快得多。

However, programming is not limited only to printing simple texts on the screen. In order to go a little further on and to become able to write programs that perform useful tasks that really save us work, we need to introduce the concept of *variables*.

然而，编程并不仅仅局限于在屏幕上打印简单的文本。为了更进一步，并能够编写执行真正节省我们工作的有用任务的程序，我们需要引入*变量*的概念。

Let's imagine that I ask you to remember the number 5, and then I ask you to also memorize the number 2 at the same time. You have just stored two different values in your memory (5 and 2). Now, if I ask you to add 1 to the first number I said, you should be retaining the numbers 6 (that is 5+1) and 2 in your memory. Then we could, for example, subtract these values and obtain 4 as result.

假设我让你们记住数字5，同时我也让你们记住数字2。你刚刚在你的内存中存储了两个不同的值(5和2)。现在，如果我让你给我说的第一个数字加1，你应该在你的内存中保留数字6(即5+1)和2。然后我们可以，例如，减去这些值，结果是4。

The whole process described above is a simile of what a computer can do with two variables. The same process can be expressed in C++ with the following set of statements:

上面描述的整个过程是计算机处理两个变量的一个比喻。同样的过程可以用c++中的下列语句来表示:

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Obviously, this is a very simple example, since we have only used two small integer values, but consider that your computer can store millions of numbers like these at the same time and conduct sophisticated mathematical operations with them.

显然，这是一个非常简单的例子，因为我们只使用了两个小的整数值，但考虑到您的计算机可以同时存储数百万个这样的数字，并对它们进行复杂的数学运算。

We can now define variable as a portion of memory to store a value.

现在可以将变量定义为存储值的内存的一部分。

Each variable needs a name that identifies it and distinguishes it from the others. For example, in the previous code the variable names were `a`, `b`, and `result`, but we could have called the variables any names we could have come up with, as long as they were valid C++ identifiers.

每个变量都需要一个名称来标识它，并将它与其他变量区分开来。例如，在前面的代码中，变量名是' a '， ' b '和' result '，但是我们可以用任何我们能想到的名字来命名变量，只要它们是有效的c++标识符。

Identifiers

A valid identifier is a sequence of one or more letters, digits, or underscore characters (`_`). Spaces, punctuation marks, and symbols cannot be part of an identifier. In addition, identifiers shall always begin with a letter. They can also begin with an underline character (`_`), but such identifiers are -on most cases- considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

有效的标识符是由一个或多个字母、数字或下划线(*)组成的序列。空格、标点符号和符号不能作为标识符的一部分。此外，标识符必须以字母开头。它们也可以以下划线字符(*)开头，但这种标识符在大多数情况下是启用的——被认为是为特定于编译器的关键字或外部标识符保留的，以及在任何地方包含两个连续下划线字符的标识符。在任何情况下都不能以数字开头。

C++ uses a number of keywords to identify operations and data descriptions; therefore, identifiers created by a programmer cannot match these keywords. The standard reserved keywords that cannot be used for programmer created identifiers are:

c++使用一些关键字来识别操作和数据描述;因此，程序员创建的标识符不能匹配这些关键字。不能用于程序员创建的标识符的标准保留关键字是:

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register,
```

`reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq`

Specific compilers may also have additional specific reserved keywords.

特定的编译器也可能有其他特定的保留关键字。

Very important: The C++ language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the `RESULT` variable is not the same as the `result` variable or the `Result` variable. These are three different identifiers identifiying three different variables.

c++语言是一种“区分大小写”的语言。这意味着一个用大写字母书写的标识符与另一个用小写字母书写的同名标识符是不同的。因此，例如，' RESULT '变量与' result '变量或' Result '变量并不相同。这是三个不同的标识符，用于标识三个不同的变量。

Fundamental data types

The values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones. Our program does not need to know the exact location where a variable is stored; it can simply refer to it by its name. What the program needs to be aware of is the kind of data stored in the variable. It's not the same to store a simple integer as it is to store a letter or a large floating-point number; even though they are all represented using zeros and ones, they are not interpreted in the same way, and in many cases, they don't occupy the same amount of memory.

变量的值以0和1的形式存储在计算机内存中某个未指定的位置。我们的程序不需要知道变量存储的确切位置;你可以直接用它的名字来指代它。程序需要知道的是变量中存储的数据的类型。存储一个简单的整数与存储一个字母或一个大的浮点数是不同的;尽管它们都使用0和1表示，但它们的解释方式并不相同，而且在许多情况下，它们占用的内存数量并不相同。

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

基本数据类型是由语言直接实现的基本类型，语言表示大多数系统本地支持的基本存储单元。主要可分为:

- **Character types:** They can represent a single character, such as `'A'` or `'$'`. The most basic type is `char`, which is a one-byte character. Other types are also provided for wider characters.

  字符类型:它们可以表示单个字符，如" a ' '或" $' '。最基本的类型是' char '，这是一个单字节字符。对于更宽的字符，还提供了其他类型。

- **Numerical integer types:** They can store a whole number value, such as `7` or `1024`. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.

数字整数类型:它们可以存储一个整数值，如' 7 '或' 1024 '。它们有不同的大小，可以是*有符号*也可以是*无符号*，这取决于它们是否支持负值。

- **Floating-point types:** They can represent real values, such as `3.14` or `0.01`, with different levels of precision, depending on which of the three floating-point types is used.

  浮点类型:它们可以表示真实值，如' 3.14 '或' 0.01 '，根据使用的三种浮点类型中的哪一种，具有不同的精度级别。

- **Boolean type:** The boolean type, known in C++ as `bool`, can only represent one of two states, `true` or `false`.

  布尔类型:布尔类型，在c++中称为' bool '，只能表示' true '或' false '两种状态中的一种。

Here is the complete list of fundamental types in C++:

以下是c++中基本类型的完整列表:

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** ***int***	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** ***int***	Not smaller than `int`. At least 32 bits.
	signed **long long** ***int***	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short ***int***	
	unsigned ***int***	
	unsigned long ***int***	
	unsigned long long ***int***	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

* The names of certain integer types can be abbreviated without their `signed` and `int` components - only the part not in italics is required to identify the type, the part in italics is optional. I.e., `*signed* short *int*` can be abbreviated as `signed short`, `short int`, or simply `short`; they all identify the same fundamental type.

某些整数类型的名称可以缩写，不需要它们的signed和int组件——只有非斜体部分需要识别类型，斜体部分是可选的。也就是说，有符号的short int可以缩写为signed short、short int或simply short;它们都识别出相同的基本类型。

Within each of the groups above, the difference between types is only their size (i.e., how much they occupy in memory): the first type in each group is the smallest, and the last is the largest, with each type being at least as large as the one preceding it in the same group. Other than that, the types in a group have the same properties.

在每个上面的团体中,类型之间的区别仅仅是它们的大小(即,他们占用多少内存):每组第一种是最小的,最后是最大的,每种至少一样大的前在同一组。除此之外，组中的类型具有相同的属性。

Note in the panel above that other than `char` (which has a size of exactly one byte), none of the fundamental types has a standard size specified (but a minimum size, at most). Therefore, the type is not required (and in many cases is not) exactly this minimum size. This does not mean that these types are of an undetermined size, but that there is no standard size across all compilers and machines; each compiler implementation may specify the sizes for these types that fit the best the architecture where the program is going to run. This rather generic size specification for types gives the C++ language a lot of flexibility to be adapted to work optimally in all kinds of platforms, both present and future.

请注意，在上面的面板中，除了' char '(它的大小正好是一个字节)，没有任何基本类型指定了标准大小(最多是最小大小)。因此，该类型并不是必需的(在许多情况下也不是)。这并不意味着这些类型的大小是不确定的，但在所有编译器和机器上没有标准的大小;每个编译器实现都可以为这些类型指定最适合程序将要运行的体系结构的大小。这种相当通用的类型大小规范为c++语言提供了很大的灵活性，使其能够在当前和未来的各种平台中最优地工作。

Type sizes above are expressed in bits; the more bits a type has, the more distinct values it can represent, but at the same time, also consumes more space in memory:

上面的类型大小是用位表示的;一个类型的位数越多，它可以表示的不同的值就越多，但同时也会消耗更多的内存空间:

Size	Unique representable values	Notes
8-bit	`256`	= 2^8
16-bit	`65 536`	= 2^16
32-bit	`4 294 967 296`	= 2^32 (~4 billion)
64-bit	`18 446 744 073 709 551 616`	= 2^64 (~18 billion billion)

For integer types, having more representable values means that the range of values they can represent is greater; for example, a 16-bit unsigned integer would be able to represent 65536 distinct values in the range 0 to 65535, while its signed counterpart would be able to represent, on most cases, values between -32768 and 32767. Note that the range of positive values is approximately halved in signed types compared to unsigned types, due to the fact that one of the 16 bits is used for the sign; this is a relatively modest difference in range, and seldom justifies the use of unsigned types based purely on the range of positive values they can represent.

对于整数类型，具有更多可表示的值意味着它们可以表示的值范围更大;例如，一个16位无符号整数将能够表示0到65535范围内的65536个不同的值，而在大多数情况下，对应的带符号整数将能够表示-32768到32767之间的值。请注意，与无符号类型相比，有符号类型的正数范围大约减半，这是由于16位中的一位被用于符号;这在范围上是一个相对较小的差异，并且很少证明使用纯基于它们可以表示的正值范围的无符号类型是正确的。

For floating-point types, the size affects their precision, by having more or less bits for their significant and exponent.

对于浮点类型，大小会影响它们的精度，因为其有效位数和指数位数的位数不同。

If the size or precision of the type is not a concern, then `char`, `int`, and `double` are typically selected to represent characters, integers, and floating-point values, respectively. The other types in their respective groups are only used in very particular cases.

如果不关心类型的大小或精度，则通常选择' char '、' int '和' double '分别表示字符、整数和浮点值。它们各自组中的其他类型只在非常特殊的情况下使用。

The properties of fundamental types in a particular system and compiler implementation can be obtained by using the numeric_limits classes (see standard header `<limits>`). If for some reason, types of specific sizes are needed, the library defines certain fixed-size type aliases in header `<cstdint>`.

特定系统和编译器实现中的基本类型的属性可以通过使用numeric_limits类获得(参见标准头文件' ')。如果出于某种原因，需要特定大小的类型，标准库在头文件' '中定义了特定大小固定的类型别名。

The types described above (characters, integers, floating-point, and boolean) are collectively known as arithmetic types. But two additional fundamental types exist: `void`, which identifies the lack of type; and the type `nullptr`, which is a special type of pointer. Both types will be discussed further in a coming chapter about pointers.

上面描述的类型(字符、整数、浮点数和布尔值)统称为算术类型。但是还有另外两种基本类型:“void”，表示缺乏类型; 类型' nullptr '是一种特殊类型的指针。这两种类型将在关于指针的下一章中进一步讨论。

C++ supports a wide variety of types based on the fundamental types discussed above; these other types are known as *compound data types*, and are one of the main strengths of the C++ language. We will also see them in more detail in future chapters.

c++支持基于上面讨论的基本类型的各种类型;其他类型被称为*复合数据类型*，并且是c++语言的主要优点之一。我们也将在以后的章节中更详细地看到它们。

Declaration of variables

C++ is a strongly-typed language, and requires every variable to be declared with its type before its first use. This informs the compiler the size to reserve in memory for the variable and how to interpret its value. The syntax to declare a new variable in C++ is straightforward: we simply write the type followed by the variable name (i.e., its identifier). For example:

c++是一种强类型语言，要求每个变量在第一次使用之前都要声明其类型。这将通知编译器在内存中为变量保留的大小以及如何解释它的值。c++中声明新变量的语法很简单:只需要在变量名(即它的标识符)后写上类型即可。例如:

```
int a;
float mynumber;
```

These are two valid declarations of variables. The first one declares a variable of type `int` with the identifier `a`. The second one declares a variable of type `float` with the identifier `mynumber`. Once declared, the variables `a` and `mynumber` can be used within the rest of their scope in the program.

If declaring more than one variable of the same type, they can all be declared in a single statement by separating their identifiers with commas. For example:

这是两个有效的变量声明。第一个声明了一个具有标识符' a '的类型' int '变量。第二个声明了一个类型为' float '的变量，其标识符为' mynumber '。一旦声明，变量' a '和' mynumber '就可以在程序中它们的其余作用域内使用。

如果声明多个相同类型的变量，则可以通过用逗号分隔它们的标识符在单个语句中声明它们。例如:

```
int a, b, c;
```

This declares three variables (`a`, `b` and `c`), all of them of type `int`, and has exactly the same meaning as:

这声明了三个变量(' a '， ' b '和' c ')，它们的类型都是' int '，其含义完全相同:

```
int a;
int b;
int c;
```

To see what variable declarations look like in action within a program, let's have a look at the entire C++ code of the example about your mental memory proposed at the beginning of this chapter:

要了解在程序中变量声明的作用，让我们看一看本章开始提到的关于心理记忆的例子的整个c++代码:

```
// operating with variables

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

```
int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Don't be worried if something else than the variable declarations themselves look a bit strange to you. Most of it will be explained in more detail in coming chapters.

如果除了变量声明本身之外，其他东西对您来说有点奇怪，也不要担心。大部分内容将在接下来的章节中进行更详细的解释。

Initialization of variables

When the variables in the example above are declared, they have an undetermined value until they are assigned a value for the first time. But it is possible for a variable to have a specific value from the moment it is declared. This is called the *initialization* of the variable.

当上述示例中的变量被声明时，在第一次为它们赋值之前，它们都有一个未确定的值。但是，变量有可能从声明的那一刻起就具有特定的值。这被称为变量的初始化。

In C++, there are three ways to initialize variables. They are all equivalent and are reminiscent of the evolution of the language over the years:

在c++中，有三种初始化变量的方法。它们都是等价的，让人想起了语言多年来的演变:

The first one, known as *c-like initialization* (because it is inherited from the C language), consists of appending an equal sign followed by the value to which the variable is initialized:

第一个被称为* C -like initialization*(因为它是从C语言继承而来的)，它包括在变量初始化的值后面附加一个等号:

```
type identifier = initial_value;
```

For example, to declare a variable of type `int` called `x` and initialize it to a value of zero from the same moment it is declared, we can write:

例如，要声明一个名为' x '的类型为' int '的变量，并在声明它的同时将它初始化为0，可以这样写:

```
int x = 0;
```

A second method, known as *constructor initialization* (introduced by the C++ language), encloses the initial value between parentheses (`()`):

第二种方法称为*构造函数初始化*(由c++语言引入)，将初始值括在圆括号('()')之间:

```
type identifier (initial_value);
```

For example:

```
int x (0);
```

Finally, a third method, known as *uniform initialization*, similar to the above, but using curly braces (`{}`) instead of parentheses (this was introduced by the revision of the C++ standard, in 2011):

最后是第三种方法，称为*统一初始化*，类似于上面的方法，但使用花括号('{}')而不是圆括号(这是在2011年c++标准的修订版中引入的):

```
type identifier {initial_value};
```

For example:

```
int x {0};
```

All three ways of initializing variables are valid and equivalent in C++.

在c++中，这三种初始化变量的方法都是有效且等价的。

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;               // initial value: 5
  int b(3);              // initial value: 3
  int c{2};              // initial value: 2
  int result;            // initial value undetermined
```

```
    a = a + b;
    result = a - c;
    cout << result;

    return 0;
}
```

补充：

`int c{2};`

在使用MAC或者linux编译的时候，要附加c++11标准，才可以编译通过

`macOS: clang++ xxx.cpp --std c++11`

`linux: c++ xxx.cpp -std=c++11`

Type deduction: auto and decltype

When a new variable is initialized, the compiler can figure out what the type of the variable is automatically by the initializer. For this, it suffices to use `auto` as the type specifier for the variable:

当一个新变量被初始化时，编译器可以通过初始化器自动找出变量的类型。为此，使用' auto '作为变量的类型说明符就足够了:

```
int foo = 0;
auto bar = foo;  // the same as: int bar = foo;
```

Here, `bar` is declared as having an `auto` type; therefore, the type of `bar` is the type of the value used to initialize it: in this case it uses the type of `foo`, which is `int`.

这里，' bar '被声明为具有' auto '类型;因此，' bar '的类型就是用于初始化它的值的类型:在本例中，它使用' foo '的类型，即' int '。

Variables that are not initialized can also make use of type deduction with the `decltype` specifier:

未初始化的变量也可以使用' decltype '说明符进行类型推断:

```
int foo = 0;
decltype(foo) bar;  // the same as: int bar;
```

Here, `bar` is declared as having the same type as `foo`.

这里，' bar '被声明为与' foo '具有相同的类型。

`auto` and `decltype` are powerful features recently added to the language. But the type deduction features they introduce are meant to be used either when the type cannot be obtained by other means or when using it improves code readability. The two examples above were likely neither of these use cases. In fact they probably decreased readability, since, when reading the code, one has to search for the type of `foo` to actually know the type of `bar`.

' auto '和' decltype '是最近添加到语言中的强大特性。但是，它们引入的类型推断特性，是在无法通过其他方式获得类型或使用该类型提高代码可读性时使用的。上面的两个例子可能都不是这些用例。事实上，它们可能会降低可读性，因为，当阅读代码时，你必须搜索' foo '的类型来实际知道' bar '的类型。

Introduction to strings

Fundamental types represent the most basic types handled by the machines where the code may run. But one of the major strengths of the C++ language is its rich set of compound types, of which the fundamental types are mere building blocks.

基本类型表示代码可能运行的机器所处理的最基本类型。但是c++语言的一个主要优点是它丰富的复合类型集，其中的基本类型仅仅是构建块。

An example of compound type is the `string` class. Variables of this type are able to store sequences of characters, such as words or sentences. A very useful feature!

复合类型的一个例子是' string '类。这种类型的变量能够存储字符序列，如单词或句子。一个非常有用的特性!

A first difference with fundamental data types is that in order to declare and use objects (variables) of this type, the program needs to include the header where the type is defined within the standard library (header `<string>`):

基本数据类型的第一个区别是，为了声明和使用这种类型的对象(变量)，程序需要包含标准库中定义该类型的头文件(头文件' '):

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

As you can see in the previous example, strings can be initialized with any valid string literal, just like numerical type variables can be initialized to any valid numerical literal. As with fundamental types, all initialization formats are valid with strings:

正如您在前面的示例中所看到的，可以用任何有效的字符串文字初始化字符串，就像数字类型变量可以初始化为任何有效的数字文字一样。和基本类型一样，所有初始化格式对字符串都有效:

```
string mystring = "This is a string";
string mystring ("This is a string");
string mystring {"This is a string"};
```

Strings can also perform all the other basic operations that fundamental data types can, like being declared without an initial value and change its value during execution:

字符串也可以执行基本数据类型可以执行的所有其他基本操作，比如在声明时没有初始值，并在执行过程中更改其值:

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

Note: inserting the `endl` manipulator **end**s the **l**ine (printing a newline character and flushing the stream).

注意:插入' endl '操纵符在结尾处会进行换行(打印换行符并刷新流)。

The string class is a *compound type*. As you can see in the example above, *compound types* are used in the same way as *fundamental types*: the same syntax is used to declare variables and to initialize them.

string类是*复合类型*。正如你在上面的例子中所看到的，*复合类型*的使用方式与*基本类型*相同:使用相同的语法来声明变量和初始化它们。

For more details on standard C++ strings, see the string class reference

有关标准c++字符串的更多细节，请参阅string类参考。