

C++ 数据类型

使用编程语言进行编程时，需要用到各种变量来存储各种信息。变量保留的是它所存储的值的内存位置。这意味着，当您创建一个变量时，就会在内存中保留一些空间。

您可能需要存储各种数据类型（比如字符型、宽字符型、整型、浮点型、双浮点型、布尔型等）的信息，操作系统会根据变量的数据类型，来分配内存和决定在保留内存中存储什么。

基本的内置类型

C++ 为程序员提供了种类丰富的内置数据类型和用户自定义的数据类型。下表列出了七种基本的 C++ 数据类型：

类型	关键字
布尔型	bool
字符型	char
整型	int
浮点型	float
双浮点型	double
无类型	void
宽字符型	wchar_t

其实 wchar_t 是这样来的：

```
typedef wchar_t short int;
```

所以 wchar_t 实际上的空间是和 short int 一样。

一些基本类型可以使用一个或多个类型修饰符进行修饰：

- signed
- unsigned
- short
- long

下表显示了各种变量类型在内存中存储值时需要占用的内存，以及该类型的变量所能存储的最大值和最小值。

注意：不同系统会有所差异。

类型	位	范围
----	---	----

char	1 个字节	-128 到 127 或者 0 到 255
unsigned char	1 个字节	0 到 255
signed char	1 个字节	-128 到 127
int	4 个字节	-2147483648 到 2147483647
unsigned int	4 个字节	0 到 4294967295
signed int	4 个字节	-2147483648 到 2147483647
short int	2 个字节	-32768 到 32767
unsigned short int	2 个字节	0 到 65,535
signed short int	2 个字节	-32768 到 32767
long int	8 个字节	-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807
signed long int	8 个字节	-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807
unsigned long int	8 个字节	0 to 18,446,744,073,709,551,615
float	4 个字节	+/- 3.4e +/- 38 (~7 个数字)
double	8 个字节	+/- 1.7e +/- 308 (~15 个数字)
long double	16 个字节	+/- 1.7e +/- 308 (~15 个数字)
wchar_t	2 或 4 个字节	1 个宽字符

从上表可得知，变量的大小会根据编译器和所使用的电脑而有所不同。

下面实例会输出您电脑上各种数据类型的大小。

实例

```
#include<iostream>
#include<string>
#include <limits>
using namespace std;
int main()
{
cout << "type: \t\t" << "*****size*****" << endl;
cout << "bool: \t\t" << "所占字节数: " << sizeof(bool);
cout << "\t\t最大值: " << (numeric_limits<bool>::max)();
cout << "\t\t\t最小值: " << (numeric_limits<bool>::min)() << endl;
cout << "char: \t\t" << "所占字节数: " << sizeof(char);
cout << "\t\t最大值: " << (numeric_limits<char>::max)();
cout << "\t\t\t最小值: " << (numeric_limits<char>::min)() << endl;
```

```

cout << "signed char: \t" << "所占字节数: " << sizeof(signed char);
cout << "\t最大值: " << (numeric_limits<signed char>::max)();
cout << "\t\t最小值: " << (numeric_limits<signed char>::min)() << endl;
cout << "unsigned char: \t" << "所占字节数: " << sizeof(unsigned char);
cout << "\t最大值: " << (numeric_limits<unsigned char>::max)();
cout << "\t\t最小值: " << (numeric_limits<unsigned char>::min)() << endl;
cout << "wchar_t: \t" << "所占字节数: " << sizeof(wchar_t);
cout << "\t最大值: " << (numeric_limits<wchar_t>::max)();
cout << "\t\t最小值: " << (numeric_limits<wchar_t>::min)() << endl;
cout << "short: \t\t" << "所占字节数: " << sizeof(short);
cout << "\t最大值: " << (numeric_limits<short>::max)();
cout << "\t\t最小值: " << (numeric_limits<short>::min)() << endl;
cout << "int: \t\t" << "所占字节数: " << sizeof(int);
cout << "\t最大值: " << (numeric_limits<int>::max)();
cout << "\t最小值: " << (numeric_limits<int>::min)() << endl;
cout << "unsigned: \t" << "所占字节数: " << sizeof(unsigned);
cout << "\t最大值: " << (numeric_limits<unsigned>::max)();
cout << "\t最小值: " << (numeric_limits<unsigned>::min)() << endl;
cout << "long: \t\t" << "所占字节数: " << sizeof(long);
cout << "\t最大值: " << (numeric_limits<long>::max)();
cout << "\t最小值: " << (numeric_limits<long>::min)() << endl;
cout << "unsigned long: \t" << "所占字节数: " << sizeof(unsigned long);
cout << "\t最大值: " << (numeric_limits<unsigned long>::max)();
cout << "\t最小值: " << (numeric_limits<unsigned long>::min)() << endl;
cout << "double: \t" << "所占字节数: " << sizeof(double);
cout << "\t最大值: " << (numeric_limits<double>::max)();
cout << "\t最小值: " << (numeric_limits<double>::min)() << endl;
cout << "long double: \t" << "所占字节数: " << sizeof(long double);
cout << "\t最大值: " << (numeric_limits<long double>::max)();
cout << "\t最小值: " << (numeric_limits<long double>::min)() << endl;
cout << "float: \t\t" << "所占字节数: " << sizeof(float);
cout << "\t最大值: " << (numeric_limits<float>::max)();
cout << "\t最小值: " << (numeric_limits<float>::min)() << endl;
cout << "size_t: \t" << "所占字节数: " << sizeof(size_t);
cout << "\t最大值: " << (numeric_limits<size_t>::max)();
cout << "\t最小值: " << (numeric_limits<size_t>::min)() << endl;
cout << "string: \t" << "所占字节数: " << sizeof(string) << endl;
// << "\t最大值: " << (numeric_limits<string>::max)() << "\t最小值: " << (numeric_limits<string>::min)() << endl;
cout << "type: \t\t" << "*****size*****" << endl;
return 0;
}

```

本实例使用了 **endl**，这将在每一行后插入一个换行符，<< 运算符用于向屏幕传多个值。我们也使用 **sizeof()** 函数来获取各种数据类型的大小。

当上面的代码被编译和执行时，它会产生以下的结果，结果会根据所使用的计算机而有所不同：

```

type:          *****size*****
bool:          所占字节数: 1      最大值: 1      最小值: 0
char:          所占字节数: 1      最大值:          最小值: ?
signed char:   所占字节数: 1      最大值:          最小值: ?
unsigned char: 所占字节数: 1      最大值: ?      最小值:

```

wchar_t:	所占字节数: 4	最大值: 2147483647	最小值: -2147483648
short:	所占字节数: 2	最大值: 32767	最小值: -32768
int:	所占字节数: 4	最大值: 2147483647	最小值: -2147483648
unsigned:	所占字节数: 4	最大值: 4294967295	最小值: 0
long:	所占字节数: 8	最大值: 9223372036854775807	最小值: -9223372036854775808
unsigned long:	所占字节数: 8	最大值: 18446744073709551615	最小值: 0
double:	所占字节数: 8	最大值: 1.79769e+308	最小值: 2.22507e-308
long double:	所占字节数: 16	最大值: 1.18973e+4932	最小值: 3.3621e-4932
float:	所占字节数: 4	最大值: 3.40282e+38	最小值: 1.17549e-38
size_t:	所占字节数: 8	最大值: 18446744073709551615	最小值: 0
string:	所占字节数: 24		
type:	*****size*****		

typedef 声明

您可以使用 **typedef** 为一个已有的类型取一个新的名字。下面是使用 typedef 定义一个新类型的语法：

```
typedef type newname;
```

例如，下面的语句会告诉编译器，feet 是 int 的另一个名称：

```
typedef int feet;
```

现在，下面的声明是完全合法的，它创建了一个整型变量 distance：

```
feet distance;
```

枚举类型

枚举类型(enumeration)是C++中的一种派生数据类型，它是由用户定义的若干枚举常量的集合。

如果一个变量只有几种可能的值，可以定义为枚举(enumeration)类型。所谓"枚举"是指将变量的值一一列举出来，变量的值只能在列举出来的值的范围内。

创建枚举，需要使用关键字 **enum**。枚举类型的一般形式为：

```
enum 枚举名{  
    标识符[=整型常数],  
    标识符[=整型常数],  
    ...  
    标识符[=整型常数]  
} 枚举变量;
```

如果枚举没有初始化, 即省掉"=整型常数"时, 则从第一个标识符开始。

例如，下面的代码定义了一个颜色枚举，变量 `c` 的类型为 `color`。最后，`c` 被赋值为 "blue"。

```
enum color { red, green, blue } c;  
c = blue;
```

默认情况下，第一个名称的值为 0，第二个名称的值为 1，第三个名称的值为 2，以此类推。但是，您也可以给名称赋予一个特殊的值，只需要添加一个初始值即可。例如，在下面的枚举中，**green** 的值为 5。

```
enum color { red, green=5, blue };
```

在这里，**blue** 的值为 6，因为默认情况下，每个名称都会比它前面一个名称大 1，但 `red` 的值依然为 0。

[← C++ 注释](#)[C++ 变量类型 →](#)**8 篇笔记** **写笔记**