C++基础数据类型

整型（Signed / Unsigned）

中文名称	类型名	字节数	位宽	最小值	最大值
布尔型	`bool`	1	8	`false` (0)	`true` (1)
字符型	`char`	1	8	-128	127
无符号字符型	`unsigned char`	1	8	0	255
短整型	`short`	2	16	-32,768	32,767
无符号短整型	`unsigned short`	2	16	0	65,535
整型	`int`	4	32	-2,147,483,648	2,147,483,647
无符号整型	`unsigned int`	4	32	0	4,294,967,295
长整型	`long`	4/8	32/64	-2,147,483,648 (4字节)	2,147,483,647 (4字节)
				-9,223,372,036,854,775,808 (8)	9,223,372,036,854,775,807 (8)
无符号长整型	`unsigned long`	4/8	32/64	0	4,294,967,295 (4字节)
				0	18,446,744,073,709,551,615 (8)
长长整型	`long long`	8	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
无符号长长整型	`unsigned long long`	8	64	0	18,446,744,073,709,551,615

浮点型

中文名称	类型名	字节数	位宽	最小值（绝对值）	最大值（绝对值）
单精度浮点型	`float`	4	32	1.17549e-38	3.40282e+38
双精度浮点型	`double`	8	64	2.22507e-308	1.79769e+308
长双精度浮点型	`long double`	8/16	64/128	3.3621e-4932 (16字节)	1.18973e+4932 (16字节)

3. 宽字符型

中文名称	类型名	字节数	位宽	说明
宽字符型	`wchar_t`	2/4	16/32	依赖系统（Windows: 2字节）
UTF-16字符型	`char16_t`	2	16	C++11引入
UTF-32字符型	`char32_t`	4	32	C++11引入

注意事项

1. **平台差异**：

- `long`在32位系统通常为4字节，64位Linux中为8字节，Windows中仍为4字节。
- `long double`在x86系统中通常为10或12字节，但实际分配可能为16字节对齐。

2. **标准宏获取极值**：
   - 使用`<climits>`（如`INT_MAX`）和`<cfloat>`（如`FLT_MIN`）获取精确值。
3. **布尔型**：
   - `bool`实际存储为`0`（false）或`1`（true），但占用1字节空间。

以下是用于显示C++基础类型宽度的完整程序，包含字节大小和位宽计算：

代码

sizeof1.cpp

```cpp
#include <iostream>
#include <cwchar>    // 用于wchar_t类型
#include <iostream>
#include <cwchar>    // 用于wchar_t类型
#include <cuchar>    // 用于char16_t/char32_t类型

using namespace std;

int print_type1()
{
    cout << "C++基础类型在当前系统的存储宽度：\n";
    cout << "==================================\n";

    // 布尔类型
    cout << "1. 布尔型:\n";
    cout << "bool: " << sizeof(bool) << " 字节 (" << sizeof(bool)*8 << " 位)\n\n";

    // 字符类型
    cout << "2. 字符类型:\n";
    cout << "char: " << sizeof(char) << " 字节 (" << sizeof(char)*8 << " 位)\n";
    cout << "unsigned char: " << sizeof(unsigned char) << " 字节 (" << sizeof(unsigned char)*8 << " 位)\n\n";

    // 整型家族
    cout << "3. 整型:\n";
    cout << "short: " << sizeof(short) << " 字节 (" << sizeof(short)*8 << " 位)\n";
    cout << "unsigned short: " << sizeof(unsigned short) << " 字节 (" << sizeof(unsigned short)*8 << " 位)\n";
    cout << "int: " << sizeof(int) << " 字节 (" << sizeof(int)*8 << " 位)\n";
    cout << "unsigned int: " << sizeof(unsigned int) << " 字节 (" << sizeof(unsigned int)*8 << " 位)\n";
    cout << "long: " << sizeof(long) << " 字节 (" << sizeof(long)*8 << " 位)\n";
    cout << "unsigned long: " << sizeof(unsigned long) << " 字节 (" << sizeof(unsigned long)*8 << " 位)\n";
    cout << "long long: " << sizeof(long long) << " 字节 (" << sizeof(long long)*8 << " 位)\n";
```

```
    cout << "unsigned long long: " << sizeof(unsigned long long) << " 字节 (" << sizeof(unsigned long long)*8 << " 位)\n\n";

    // 浮点类型
    cout << "4. 浮点型:\n";
    cout << "float: " << sizeof(float) << " 字节 (" << sizeof(float)*8 << " 位)\n";
    cout << "double: " << sizeof(double) << " 字节 (" << sizeof(double)*8 << " 位)\n";
    cout << "long double: " << sizeof(long double) << " 字节 (" << sizeof(long double)*8 << " 位)\n\n";

    // 宽字符类型
    cout << "5. 宽字符型:\n";
    cout << "wchar_t: " << sizeof(wchar_t) << " 字节 (" << sizeof(wchar_t)*8 << " 位)\n";
    cout << "char16_t: " << sizeof(char16_t) << " 字节 (" << sizeof(char16_t)*8 << " 位)\n";
    cout << "char32_t: " << sizeof(char32_t) << " 字节 (" << sizeof(char32_t)*8 << " 位)\n\n";

    // 平台差异说明
    cout << "注意事项：\n";
    cout << "- long类型在Windows中通常为4字节，在Linux/macOS中为8字节\n";
    cout << "- long double的大小可能因编译器而异（常见8/12/16字节）\n";
    cout << "- wchar_t在Windows中为2字节，在Linux中通常为4字节\n";

    return 0;
}

int main()
{
    print_type1();
    return 0;
}
```

使用说明：

1. 编译执行：

```
g++ -o type_sizes type_sizes.cpp -std=c++11
./type_sizes
```

2. 输出示例（macOS系统）：

```
C++基础类型在当前系统的存储宽度：
==================================
1. 布尔型：
   bool:           1 字节 (8 位)

2. 字符类型：
   char:           1 字节 (8 位)
```

```
   unsigned char: 1 字节 (8 位)

3. 整型:
   short:          2 字节 (16 位)
   unsigned short: 2 字节 (16 位)
   int:            4 字节 (32 位)
   unsigned int:   4 字节 (32 位)
   long:           8 字节 (64 位)
   unsigned long:  8 字节 (64 位)
   long long:      8 字节 (64 位)
   unsigned l-l:   8 字节 (64 位)

4. 浮点型:
   float:        4 字节 (32 位)
   double:       8 字节 (64 位)
   long double:  16 字节 (128 位)

5. 宽字符型:
   wchar_t:      4 字节 (32 位)
   char16_t:     2 字节 (16 位)
   char32_t:     4 字节 (32 位)

注意事项:
- long类型在Windows中通常为4字节，在Linux/macOS中为8字节
- long double的大小可能因编译器而异（常见8/12/16字节）
- wchar_t在Windows中为2字节，在Linux中通常为4字节
```

3. 输出示例（Windows系统）：

```
C++基础类型在当前系统的存储宽度:
==================================
1. 布尔型:
   bool:           1 字节 (8 位)

2. 字符类型:
   char:           1 字节 (8 位)
   unsigned char: 1 字节 (8 位)

3. 整型:
   short:          2 字节 (16 位)
   unsigned short: 2 字节 (16 位)
   int:            4 字节 (32 位)
   unsigned int:   4 字节 (32 位)
   long:           4 字节 (32 位)
   unsigned long:  4 字节 (32 位)
   long long:      8 字节 (64 位)
   unsigned l-l:   8 字节 (64 位)

4. 浮点型:
   float:        4 字节 (32 位)
   double:       8 字节 (64 位)
```

```
   long double:  16 字节 (128 位)

5. 宽字符型:
   wchar_t:      2 字节 (16 位)
   char16_t:     2 字节 (16 位)
   char32_t:     4 字节 (32 位)

注意事项:
- long类型在Windows中通常为4字节，在Linux/macOS中为8字节
- long double的大小可能因编译器而异（常见8/12/16字节）
- wchar_t在Windows中为2字节，在Linux中通常为4字节
```

4. 关键特性:
   - 使用`sizeof`运算符获取类型字节数
   - 通过`字节数*8`自动计算位宽
   - 明确标注平台差异说明
   - 支持C++11新增的char16_t/char32_t类型
   - 结构化输出便于阅读

该程序能直观展示当前编译环境下各基础类型的实际存储大小，特别适合用于验证不同平台/编译器的类型实现差异。

sizeof2.cpp

```cpp
#include <iostream>
#include <cwchar>    // 用于wchar_t类型
#include <iostream>
#include <cwchar>    // 用于wchar_t类型
#include <cuchar>    // 用于char16_t/char32_t类型

using namespace std;

typedef struct {
    const char *name;
    int size;
} s_type;

s_type type_array[]={
    {"bool", sizeof(bool)},
    {"char", sizeof(char)},
    {"unsigned char", sizeof(unsigned char)},
    {"short", sizeof(short)},
    {"unsigned short", sizeof(unsigned short)},
    {"int", sizeof(int)},
    {"unsigned int", sizeof(unsigned int)},
    {"long", sizeof(long)},
    {"unsigned long", sizeof(unsigned long)},
    {"long long", sizeof(long long)},
    {"unsigned long long", sizeof(unsigned long long)},
    {"float", sizeof(float)},
    {"double", sizeof(double)},
```

```
    {"long double", sizeof(long double)},
    {"wchar_t", sizeof(wchar_t)},
    {"char16_t", sizeof(char16_t)},
    {"char32_t", sizeof(char32_t)},
};

int print_type2()
{
    for (int i = 0; i < sizeof(type_array) / sizeof(s_type); i++){
        cout << type_array[i].name << ": " << type_array[i].size << "字节" << endl;
    }
    return 0;
}

int main()
{
    print_type2();
    return 0;
}
```