

# Bài 7: SỐ TỰ NHIÊN VÀ SỐ CHẤM ĐỘNG TRONG C++ (INTEGER, FLOATING POINT)

Xem bài học trên website để ủng hộ Kteam: [Số tự nhiên và Số chấm động trong C++ \(Integer, Floating point\)](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

## Dẫn nhập

Ở bài học trước, bạn đã nắm được [BIẾN TRONG C++ \(Variables\)](#), và đã biết nguyên lý hoạt động và một số kinh nghiệm về biến trong C++. Và bài học trước chỉ đề cập cơ bản về biến của một số nguyên.

Trong C++ vẫn còn rất nhiều kiểu dữ liệu khác, bạn sẽ được học 2 loại kiểu dữ liệu mới trong bài học hôm nay: **Số nguyên và Số chấm động trong C++ (Integer, Floating point)**

---

## Nội dung:

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về phần:

- [BIẾN TRONG C++ \(Variables\)](#)

Trong bài này, ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về kiểu dữ liệu cơ bản trong C++
- Kiểu số nguyên (Integer)
- Kiểu số chấm động (Floating point)

## Tổng quan về kiểu dữ liệu cơ bản trong C++

Ở bài học trước **Biến trong C++ (Variables)**, bạn đã biết biến (variable) là tên của một vùng trong bộ nhớ RAM, được sử dụng để lưu trữ thông tin. Bạn có thể gán thông tin cho một biến, và có thể lấy thông tin đó ra để sử dụng. Có rất nhiều loại thông tin (Ví dụ: thông tin dưới dạng số nguyên, số thực, ký tự, ...), và trong C++, các biến cũng có thể lưu những loại thông tin khác nhau thông qua các kiểu dữ liệu khác nhau.

Kích thước của biến phụ thuộc vào kiểu dữ liệu của biến đó và quyết định số lượng thông tin mà biến đó lưu trữ. Khi bạn khai báo một biến, một vùng trong bộ nhớ sẽ dành cho biến đó. Ngày nay, việc khai báo biến với kích thước vài byte không là vấn đề gì, so với độ lớn của bộ nhớ máy tính. Nhưng nếu chương trình của bạn có số lượng biến lên tới hàng triệu, thì việc phải sử dụng biến với kích thước sao cho phù hợp là điều rất quan trọng.

Bảng bên dưới sẽ liệt kê những kiểu dữ liệu cơ bản trong C++. Kích thước kiểu dữ liệu tương ứng bên dưới chỉ **là kích thước nhỏ nhất có thể của kiểu dữ liệu đó**. Trong thực tế, kích thước này **phụ thuộc vào từng compiler và kiến trúc máy tính**.

Category	Type	Minimum Size	Note
<b>boolean</b>	bool	1 byte	
<b>character</b>	char	1 byte	May be signed or unsigned
	wchar_t	2 byte	
	char16_t	2 bytes	C++11 type
	char32_t	4 bytes	C++11 type
<b>integer</b>	short	2 bytes	
	int	2 bytes	Typically 4 bytes on modern architectures
	long	4 bytes	
	long long	8 bytes	C99/C++11 type
<b>floating point</b>	float	4 bytes	
	double	8 bytes	
	long double	8 bytes	

Để xác định kích thước của một kiểu dữ liệu trên một máy tính cụ thể, C++ cung cấp cho bạn toán tử **sizeof**. **Toán tử sizeof** là toán tử một ngôi, **nhận vào một kiểu dữ liệu hoặc một biến**, và **trả về kích thước (byte)** của của kiểu dữ liệu hoặc biến đó.

### Ví dụ:

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

int main()
{
    cout << "bool:\t\t" << sizeof(bool) << " bytes" << endl;
    cout << "char:\t\t" << sizeof(char) << " bytes" << endl;
    cout << "wchar_t:\t" << sizeof(wchar_t) << " bytes" << endl;

    // C++11, may not be supported by your compiler
    cout << "char16_t:\t" << sizeof(char16_t) << " bytes" << endl;
    cout << "char32_t:\t" << sizeof(char32_t) << " bytes" << endl;

    cout << "short:\t\t" << sizeof(short) << " bytes" << endl;
    cout << "int:\t\t\t" << sizeof(int) << " bytes" << endl;
```

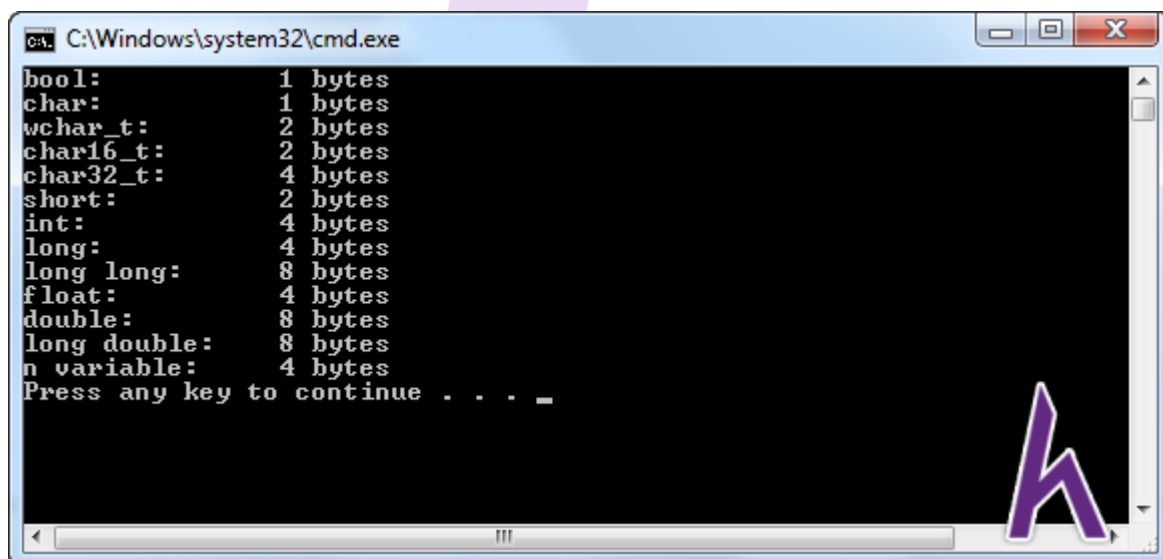
```
cout << "long:\t\t" << sizeof(long) << " bytes" << endl;

// C++11, may not be supported by your compiler
cout << "long long:\t" << sizeof(long long) << " bytes" << endl;

cout << "float:\t\t" << sizeof(float) << " bytes" << endl;
cout << "double:\t\t" << sizeof(double) << " bytes" << endl;
cout << "long double:\t" << sizeof(long double) << " bytes" << endl;

// You can also use the sizeof operator on a variable name
int n;
cout << "n variable:\t" << sizeof(n) << " bytes" << endl;
return 0;
}
```

Chương trình bên trên khi thực thi trên Window 7 x64 (Visual studio 2015) sẽ cho ra kết quả:



```
C:\Windows\system32\cmd.exe
bool:      1 bytes
char:      1 bytes
wchar_t:   2 bytes
char16_t:  2 bytes
char32_t:  4 bytes
short:     2 bytes
int:       4 bytes
long:      4 bytes
long long: 8 bytes
float:     4 bytes
double:    8 bytes
long double: 8 bytes
n variable: 4 bytes
Press any key to continue . . . _
```

Một điều thú vị là toán tử **sizeof** là một trong 3 toán tử không phải là ký hiệu trong C++, 2 toán tử còn lại là new và delete sẽ được giới thiệu trong bài [CẤP PHÁT ĐỘNG \(Dynamic memory allocation\)](#).

## Kiểu số nguyên (Integer)

Số nguyên là các số nguyên dương (1, 2, 3, ...), các số đối (-1, -2, -3, ...) và số 0. C++ có 5 loại số nguyên cơ bản để sử dụng:

Category	Type	Minimum Size	Note
character	char	1 byte	
integer	short	2 bytes	
	int	2 bytes	Typically 4 bytes on modern architectures
	long	4 bytes	
	long long	8 bytes	C99/C++11 type

**Chú ý:** Char là một kiểu dữ liệu đặc biệt, nó vừa là kiểu số nguyên, cũng vừa là kiểu ký tự.

Chi tiết về tính chất ký tự của char sẽ được nói trong phần Character. Ở mục này, bạn tạm thời coi nó là một kiểu số nguyên bình thường.

Sự khác nhau giữa các kiểu số nguyên trên nằm ở kích thước. Kiểu có kích thước lớn sẽ lưu được những số nguyên lớn. Vùng giá trị của một kiểu số nguyên được xác định trên 2 yếu tố: kích thước và dấu của nó.

**Số nguyên có dấu** là những số nguyên dương (1, 2, 3, ...), các số đối (-1, -2, -3, ...) và số 0. Có 2 cách để khai báo một biến số nguyên có dấu:

```
// Khai báo không tường minh, thường được sử dụng
char c;
short s;
int n;
// Hoặc khai báo tường minh, sử dụng từ khóa signed
signed char c;
signed short s;
signed int n;
```

**Số nguyên không dấu** là những số nguyên dương (1, 2, 3, ...) và số 0. Đôi khi chương trình của bạn có những biến không cần lưu trữ các số âm (Ví dụ: chiều

cao, cân nặng, độ dài, chỉ số danh sách, ...). Để khai báo số nguyên không dấu, bạn sử dụng từ khóa unsigned. Ví dụ:

```
// Sử dụng từ khóa unsigned
unsigned char uc;
unsigned short us;
unsigned int un;
```

**Lưu ý:** Một số nguyên không dấu không thể lưu trữ các số âm, nhưng nó có thể lưu trữ số dương lớn hơn gấp 2 lần số nguyên có dấu.

Bên dưới là bảng miền giá trị số nguyên

Type	Size	Range (Min)	Range (Max)
signed (char)	1 byte	-128	127
unsigned (char)	1 byte	0	255
signed (short, int)	2 byte	-32,768	32,767
unsigned (short, int)	2 byte	0	65,535
signed (int, long)	4 byte	-2,147,483,648	2,147,483,647
unsigned (int, long)	4 byte	0	4,294,967,295
signed (long long)	8 byte	-9,223,372,036,854,770,000	9,223,372,036,854,770,000
unsigned (long long)	8 byte	0	18,446,744,073,709,500,000

## Số chấm động (Floating point numbers)

Trong C++, kiểu số chấm động đại diện cho số thực (Ví dụ: 69.9696, 3.14159, 0.00001 ...), dùng để lưu trữ những số rất lớn hoặc rất nhỏ. Cấu trúc lưu trữ

bên trong của số thực được thiết kế theo chuẩn **số chấm động (floating-point) của IEEE**.

**Số chấm động không có** từ khóa **unsigned**. Có 3 kiểu số chấm động khác nhau trong C++: **float**, **double**, **long double**.

Type	Size	Range	
<b>float</b>	4 bytes	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	6-9 significant digits, typically 7
<b>double</b>	8 bytes	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	15-18 significant digits, typically 16
<b>long double</b>	8, 12 or 16 bytes	$\pm 3.36 \times 10^{-4932}$ to $\pm 1.18 \times 10^{4932}$	18-21 significant digits (12 bytes),
			33-36 significant digits (16 bytes)

**Chú ý:** Một số môi trường lập trình đồng nhất kiểu long double với kiểu double nên kiểu này ít được sử dụng trong lập trình ứng dụng.

Cách để định nghĩa một biến số chấm động:

```
// Definitions of floating point numbers
float fVarName;
double dVarName2;
long double ldVarName3;
```

**Chú ý:**

Khi bạn sử dụng một **hằng số dưới dạng một số chấm động**, quy ước số đó **phải có ít nhất 1 chữ số thập phân**, điều này giúp phân biệt số chấm động và số nguyên.

**Ví dụ:**

```
// Initializations of floating point numbers
float fVarName{4.0f}; // 4.0 means floating point (f suffix means float)
double dVarName2{4.0}; // 4.0 means floating point (double by default)
long double dVarName3{4.0L}; // 4.0 means floating point (L suffix means long double)
int nVarName4{4}; // 4 means integer
```

**Chú ý:** Mặc định một hằng số thực sẽ là kiểu **double**. Để có một số thực kiểu **float**, bạn cần **thêm hậu tố 'f'**.

## Ký hiệu khoa học (Scientific notation)

Ký hiệu khoa học là cách xử lý những số rất lớn hoặc rất nhỏ. **Ví dụ:** chu kỳ xoay mặt trăng của Mộc Tinh là 152853.5047 s. Khi đó, bạn có thể viết bằng ký hiệu khoa học là  $1.528535047 \times 10^5$  s. Hay một số khá quen thuộc với bạn như khối lượng của một electron là  $9.1093822 \times 10^{-31}$ . Bên dưới là một số ví dụ khác:

$$24327 = 2.4327 \times 10^4$$

$$7354 = 7.354 \times 10^3$$

$$0,0078 = 7.8 \times 10^{-3}$$

$$0,00069 = 6.9 \times 10^{-4}$$

**Chú ý:** Số mũ sẽ là dương nếu dấu thập phân chuyển sang phải, là âm nếu dấu thập phân chuyển sang trái.

Trong C++, bạn có thể sử dụng ký hiệu khoa học để gán giá trị cho biến số chấm động. Dùng ký hiệu 'e' hoặc 'E' để thay cho 10.



**Ví dụ:**

```
// Initializations of floating point numbers
double dVarName1{69000.0};
double dVarName2{6.9e4};    // 6.9e4 is equal to 69000.0

double dVarName3{0.00069};
double dVarName4{6.9E-4};   // 6.9e-4 is equal to 0.00069
```

## Độ chính xác của số chấm động (Precision)

Số chấm động sẽ bao gồm những số hữu hạn và vô hạn. Đối với số vô hạn, nghĩa là phần thập phân sẽ có chiều dài vô hạn (Ví dụ:  $1/6 = 0.166666666666...$ ,  $\pi = 3.141592653589793...$ ), nhưng bộ nhớ máy tính và kích thước kiểu dữ liệu thì hữu hạn. Nên biến số chấm động chỉ lưu được một độ chính xác nhất định, và phần số còn lại phía sau sẽ bị mất.

Trong C++, khi xuất một số chấm động, **std::cout** mặc định số có 6 chữ số. Những số ngoài phạm vi sẽ bị cắt bỏ và làm tròn lên 1 đơn vị nếu số bị cắt sau nó lớn hơn 5, hoặc số đó có thể được chuyển sang ký hiệu khoa học trong vài trường hợp tùy vào từng compiler. **Ví dụ:**

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

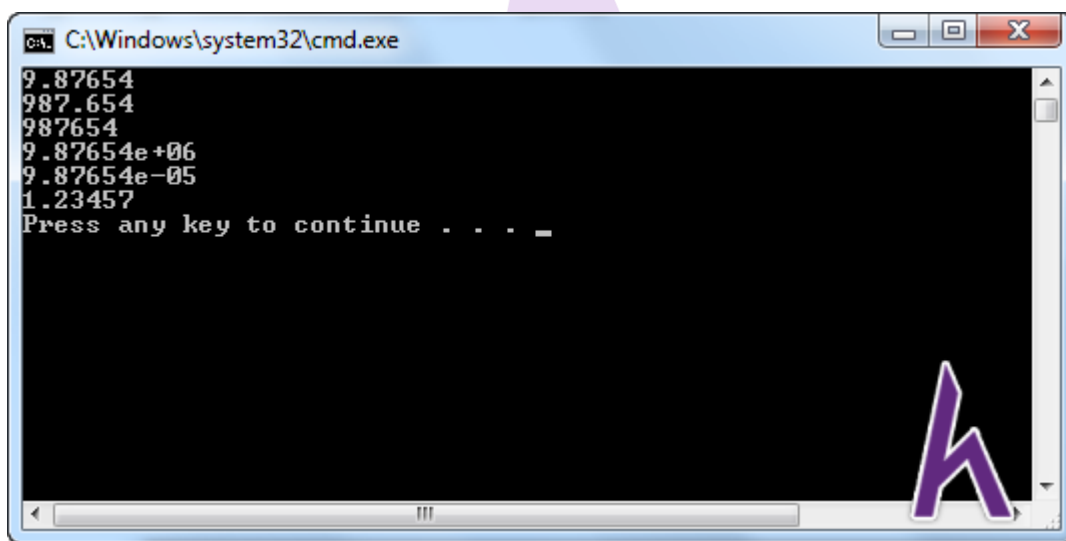
int main()
{
    double d;
    d = 9.87654321;
    cout << d << endl;
    d = 987.654321;
    cout << d << endl;
    d = 987654.321;
    cout << d << endl;
    d = 9876543.21;
```

```

    cout << d << endl;
    d = 0.0000987654321;
    cout << d << endl;
    d = 1.23456789;
    cout << d << endl;
    return 0;
}

```

Chương trình bên trên khi thực thi trên Window 7 x64 (Visual studio 2015) sẽ cho ra kết quả:



```

C:\Windows\system32\cmd.exe
9.87654
987.654
987654
9.87654e+06
9.87654e-05
1.23457
Press any key to continue . . . _

```

Mặc dù khi xuất một số chấm động, **std::cout** mặc định độ chính xác có 6 chữ số, nhưng bạn vẫn có thể thay đổi được độ chính xác này bằng cách sử dụng hàm **std::setprecision()** thuộc thư viện **<iomanip>**.

```

#include <iostream>
#include <iomanip>      // for std::setprecision()
using namespace std;

int main()
{
    cout << std::setprecision(20);           // Show 20 digits

    float    f{ 9.66666666666666666666f }; // Initializations
    cout << f << endl;

    double   d{ 9.66666666666666666666 };   // Initializations
    cout << d << endl;
}

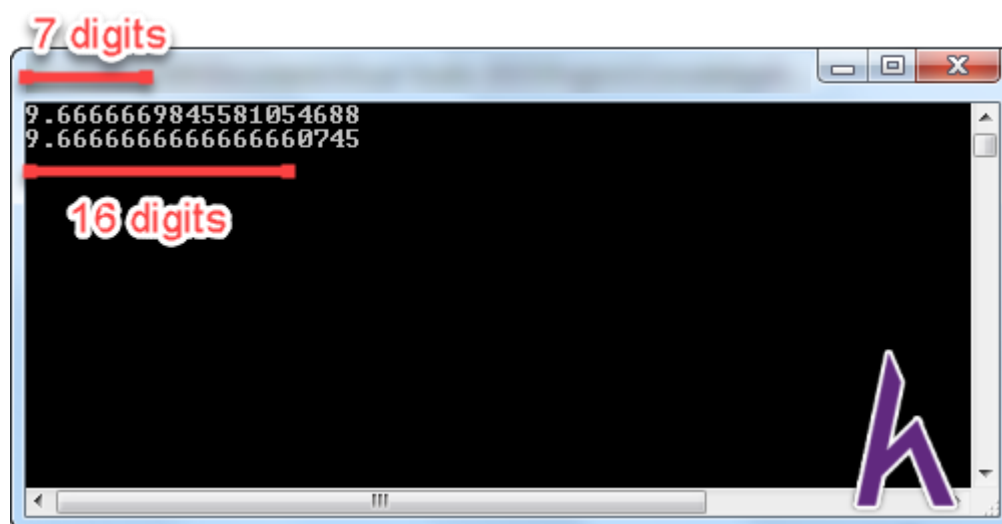
```

```

    return 0;
}

```

Kết quả thu được:



Trong chương trình trên, ta đã thay đổi độ chính xác lên đến 20 chữ số thay vì là 6 chữ số như mặc định. Nhưng dù 2 biến **float** và **double** đều đã hiện đủ 20 chữ số, thì độ chính xác của nó vẫn không đến 20 chữ số.

**Thông thường** số chấm động kiểu **float** có **độ chính xác đơn (single-precision)**, **chính xác đến 7 chữ số**. **Double** có **độ chính xác kép (double-precision)**, **chính xác đến 16 chữ số**. Đó là lý do tại sao chương trình trên lại có những số rác sau khoảng chính xác.

Độ chính xác của số chấm động **không chỉ ảnh hưởng trên phần thập phân**, mà nó có thể ảnh hưởng trên phần nguyên của những số có quá nhiều chữ số.

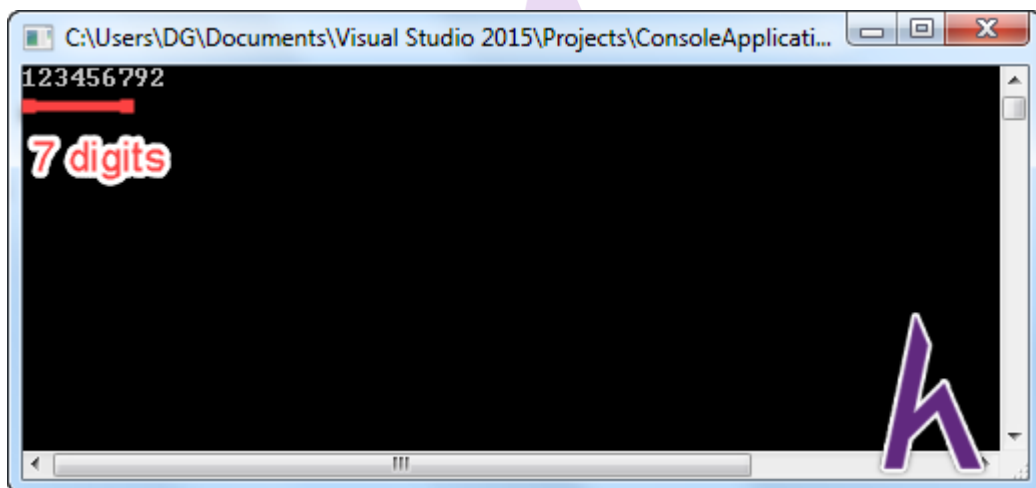
**Ví dụ:**

```
#include <iostream>
#include <iomanip>           // for std::setprecision()
using namespace std;

int main()
{
```

```
float    f{ 123456789.0f };  
  
cout << std::setprecision(9);    // Show 9 digits  
cout << f << endl;  
return 0;  
}
```

Kết quả thu được:



Vì kiểu float có độ chính xác 7 chữ số, nên chương trình đã xuất ra 123.456.792, số này lớn hơn giá trị biến ban đầu rất nhiều. Do đó, bạn nên cẩn thận khi sử dụng kiểu float để lưu trữ những số cần một độ chính xác cao.

**Chú ý:** Bạn nên sử dụng kiểu **double** khi cần lưu trữ một số chấm động, **hạn chế sử dụng float** vì kiểu float có độ chính xác thấp sẽ dẫn tới số không chính xác.

## Lỗi làm tròn số chấm động (Rounding errors)

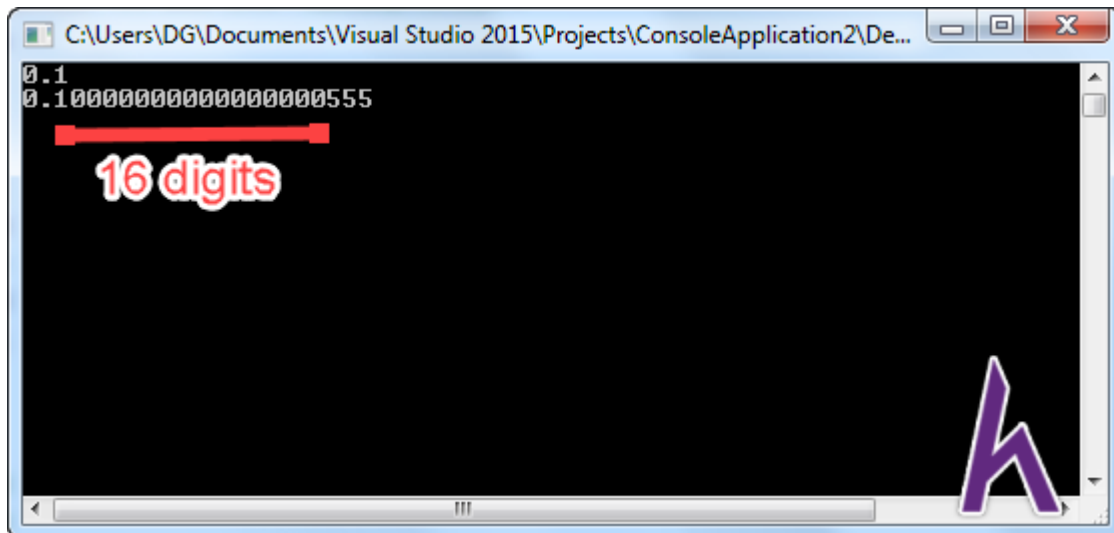
Trong máy tính, số chấm động được lưu dưới hệ nhị phân.

**Ví dụ:** ta có phân số  $1/10$ , = **0.1** trong hệ thập phân = **0.000110011(0011)...** trong hệ nhị phân (lặp vô hạn). Ta thấy số 0.1 chuyển sang hệ nhị phân sẽ lặp vô hạn, nhưng độ chính xác của số chấm động là hữu hạn. Dẫn đến việc nó **không thể được biểu diễn một cách chính xác** như một giá trị nhị phân hữu hạn. Xét ví dụ:

```
#include <iostream>
#include <iomanip>      // for std::setprecision()
using namespace std;

int main()
{
    double d{0.1};
    cout << d << endl;           // use default cout precision of
6
    cout << std::setprecision(20); // show 20 digits
    cout << d << endl;
    return 0;
}
```

Kết quả chương trình:



Trong chương trình trên, ta có một biến `double d{0.1}`. Khi output với độ chính xác mặc định `std::setprecision(6)`, ta nhận được chính xác 0.1. Nhưng khi output với `std::setprecision(20)`, kết quả lại lớn hơn 0.1.

Kết quả cho thấy khi gán số 0.1 cho một biến số chấm động, biến đó sẽ không hoàn toàn bằng 0.1. **Đó gọi là lỗi làm tròn số chấm động.**

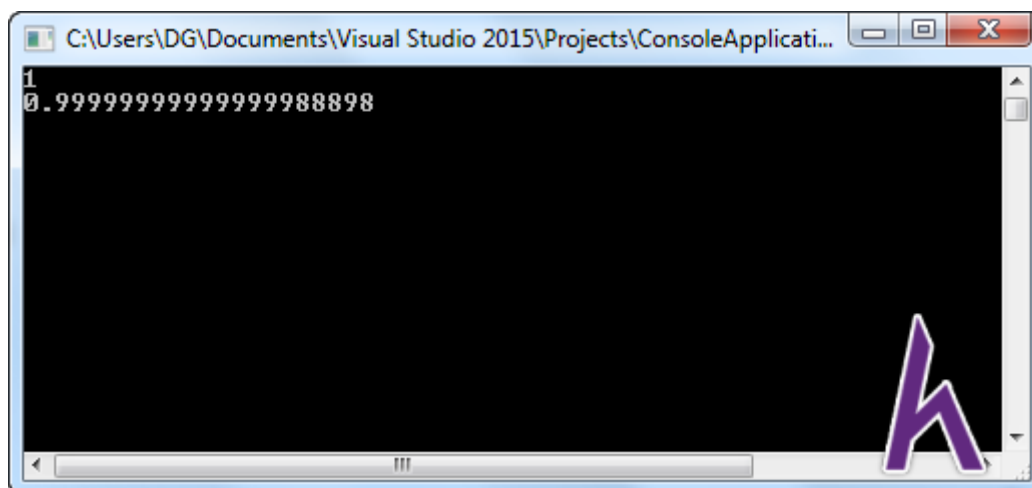
### Xét tiếp ví dụ:

```
#include <iostream>
#include <iomanip>      // for std::setprecision()
using namespace std;

int main()
{
    double d1{ 1.0 };
    double d2{ 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 };

    cout << std::setprecision(20);    // show 20 digits
    cout << d1 << endl;
    cout << d2 << endl;
    return 0;
}
```

Kết quả chương trình:



Trong chương trình trên, trong toán học thì 2 biến  $d1 = d2$ , nhưng trong lập trình biến  $d1 > d2$  vì lỗi làm tròn số dấu chấm động.

Tương tự, bạn hãy thử với trường hợp  $0.1 + 0.7 = 0.8$  ?

**Chú ý:** Không bao giờ so sánh hai giá trị dấu chấm động bằng nhau hay không. Hầu như luôn luôn có sự khác biệt nhỏ giữa hai số chấm động. Cách phổ biến để so sánh 2 số chấm động là tính khoảng cách giữa 2 số đó, nếu khoảng cách đó là rất nhỏ thì ta cho là bằng nhau. Giá trị dùng để so sánh với khoảng cách đó thường được gọi là **epsilon**. Điều này sẽ được giải thích rõ hơn trong bài [Câu điều kiện If trong C++ \(If statements\)](#).

## Kết luận

Qua bài học này, bạn đã nắm được kiểu [Số nguyên và Số chấm động trong C++ \(Integer, Floating point\)](#), và đã biết được những kinh nghiệm cũng như những lỗi thường gặp khi sử dụng nó.

Trong bài học tiếp theo, mình sẽ giới thiệu cho các bạn một kiểu dữ liệu khá quan trọng trong lập trình: [KIỂU KÝ TỰ TRONG C++ \(Data types\)](#).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.

