



3. Kiểu dữ liệu

🕒 Thời gian tạo	@August 13, 2023 6:48 PM
📄 Loại bài giảng	Lý thuyết

1. Đặt vấn đề

- Hàng ngày, chúng ta gặp các vấn đề khác nhau. Với mỗi một vấn đề, chúng ta sẽ có cách xử lý khác nhau.
- Tương tự như vậy, máy tính chúng ta sử dụng cũng phải đối mặt với nhiều loại thông tin khác nhau. Mỗi một loại thông tin, máy tính sẽ đưa ra các phương pháp xử lý để cho ra kết quả như mong muốn.
- Như vậy, kiểu dữ liệu ra đời, nhằm xác định rõ loại dữ liệu để từ đó có thể xác định rõ ràng các phương án xử lý dữ liệu tương ứng.

2. Phân loại các kiểu dữ liệu

- Kiểu dữ liệu nguyên thủy (Primitive data types): là những kiểu dữ liệu được chính ngôn ngữ lập trình định nghĩa/quy định sẵn.

- Kiểu dữ liệu tự chọn/tự thiết lập (custom/user-defined data types): là những kiểu dữ liệu do chính chúng ta tạo ra, dựa trên các kiểu dữ liệu có sẵn.

⚠ Bài viết này sẽ tập trung vào kiểu dữ liệu nguyên thủy (sau đây sẽ gọi ngắn gọn là **primitives**). Kiểu dữ liệu user-defined sẽ được đề cập trong những bài viết sau.

3. Các kiểu dữ liệu nguyên thủy (primitives) trong C++

3.1. Các kiểu dữ liệu số nguyên (integer):

Kiểu dữ liệu	Kích thước (byte)	Khoảng giá trị
<code>char</code>	1	<code>-128</code> → <code>127</code> (hoặc $[-2^7, 2^7 - 1]$)
<code>short int</code> (hoặc <code>short</code>)	2	<code>-32768</code> → <code>32767</code> (hoặc $[-2^{15}, 2^{15} - 1]$)
<code>unsigned short int</code> (hoặc <code>unsigned short</code>)	2	<code>0</code> → <code>65535</code> (hoặc $[0, 2^{16} - 1]$)
<code>int</code>	4	$[-2^{31}, 2^{31} - 1]$
<code>unsigned int</code>	4	$[0, 2^{32} - 1]$
<code>long int</code> (hoặc <code>long</code>)	4	(giống <code>int</code>)
<code>unsigned long int</code> (hoặc <code>unsigned long</code>)	4	(giống <code>unsigned int</code>)
<code>long long int</code> (hoặc <code>long long</code>)	8	$[-2^{63}, 2^{63} - 1]$
<code>unsigned long long int</code> (hoặc <code>unsigned long long</code>)	8	$[0, 2^{64} - 1]$

- Từ bảng trên ta có khoảng giá trị chung cho các kiểu dữ liệu số nguyên theo công thức (x là kích thước của kiểu dữ liệu **tính bằng bit**):

$$\begin{cases} \text{signed:} & [-2^{x-1}, 2^{x-1} - 1] \\ \text{unsigned:} & [0, 2^x - 1] \end{cases}$$

- Như vậy, mỗi 1 kiểu dữ liệu số nguyên sẽ có tổng cộng 2^x giá trị.

3.2. Các kiểu dữ liệu số thực:

Kiểu dữ liệu	Kích thước (byte)	Khoảng giá trị
<code>float</code>	4	$[-3.4 \times 10^{38}, 3.4 \times 10^{38}]$
<code>double</code>	8	$[-1.7 \times 10^{308}, 1.7 \times 10^{308}]$
<code>long double</code>	12	$[-1.1 \times 10^{4932}, 1.1 \times 10^{4932}]$

- Kiểu dữ liệu logic:

Kiểu dữ liệu	Kích thước (byte)	Khoảng giá trị
<code>bool</code>	1	Chỉ nhận 2 giá trị: <code>true</code> và <code>false</code> (hoặc <code>1</code> và <code>0</code>)

4. Sử dụng với biến:

 Xem ví dụ code ở bài số 2, phần 3 về biến

- Một số lưu ý:
 - Một khi đã khai báo 1 biến với một kiểu dữ liệu, chỉ được gán/gán lại với giá trị thuộc kiểu dữ liệu đó (hoặc “thuộc phạm vi cho phép”)
 - Không nên gán giá trị nằm ngoài khoảng giá trị của kiểu dữ liệu.

5. Xác định kích thước của kiểu dữ liệu với lệnh `sizeof`

- Cách sử dụng:

```
sizeof(kiểu_dữ_liệu)
```

- Paste đoạn code sau:

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

int main() {
    cout << "Size of char is: " << sizeof(char) << "bytes" << endl;
    cout << "Size of int is: " << sizeof(int) << "bytes" << endl;
    cout << "Size of long long is: " << sizeof(long long) << "bytes" << endl;
    cout << "Size of float is: " << sizeof(float) << "bytes" << endl;
    cout << "Size of bool is: " << sizeof(bool) << "bytes" << endl;
}
```

- Kết quả của đoạn code khi chạy là:

```
Size of char is: 1 bytes
Size of int is: 4 bytes
Size of long long is: 8 bytes
Size of float is: 4 bytes
Size of bool is: 1 bytes
```

⚠ Kết quả trên có thể khác tùy thuộc vào máy tính

6. Thư viện kiểm tra giới hạn giá trị của kiểu dữ liệu: `<climits>`


- Thư viện này cung cấp những biến hằng (constants) biểu thị giá trị lớn nhất hoặc nhỏ nhất của các kiểu dữ liệu số nguyên.
- Cách sử dụng:

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

int main() {
    cout << "Maximum of int is: " << INT_MAX << endl;
    cout << "Minimum of long is: " << LONG_MIN << endl;
}
```

- Tương tự, ta có thể thử với các constant khác nằm trong thư viện này. Click vào đường link này để xem thêm: <https://cplusplus.com/reference/climits/?kw=climits>
- Với kiểu dữ liệu số thực (`float` , `double`), xem thêm thư viện `<cmath>` .

7. Chuyển đổi kiểu dữ liệu (Type conversion)

 *Mục này hiện chỉ đề cập tới kiểu dữ liệu số. Các kiểu dữ liệu khác sẽ được đề cập trong các bài viết khác.*

- Phần lớn ngôn ngữ lập trình luôn đảm bảo các dữ liệu có cùng kiểu dữ liệu trước khi thực hiện các công đoạn xử lý tính toán, ... Do đó việc chuyển đổi kiểu dữ liệu là vô cùng cần thiết.
- Trong ngôn ngữ C++, có 2 hình thức chuyển đổi kiểu dữ liệu:
 - Chuyển đổi kiểu dữ liệu ngầm định (implicit type conversion): bản thân compiler sẽ tự cân nhắc và chuyển đổi kiểu dữ liệu sao cho hợp lý, hợp logic nhất.
 - Chuyển đổi kiểu dữ liệu tường minh (explicit type conversion): lập trình viên sẽ bắt buộc một biến được chuyển sang kiểu dữ liệu mong muốn.

7.1. Ví dụ 1:

- Xét đoạn code sau:

```
// cre: https://www.geeksforgeeks.org/type-conversion-in-c/
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    char y = 'a';

    x = x + y;

    float z = x + 1.0;

    cout << "x = " << x << endl
         << "y = " << y << endl
         << "z = " << z << endl;

    return 0;
}
```

- Kết quả thu được là:

```
x = 107
y = a
z = 108
```

- Giải thích:
 - Trước hết, ta có 2 biến: `x = 10` thuộc kiểu `int`, `y = 'a'` thuộc kiểu `char`.
 - `x = x + y`:

- cộng giá trị của `y` vào biến `x`, và gán lại vào biến `x`.
 - Do hai biến `x` và `y` khác kiểu dữ liệu nên phải chuyển đổi kiểu dữ liệu một trong hai biến để thực hiện phép cộng và phép gán (`=`).
 - Trong trường hợp này, `y` sẽ được đổi từ `char` sang `int` theo bảng mã ASCII.
- o `float z = x + 1.0` :
- cộng giá trị của `x` với `1.0` và gán vào biến `z`.
 - Giống như ở trên, cần phải chuyển đổi kiểu dữ liệu của một trong hai biến `x` và `z`.
 - Ở đây, biến `x` sẽ được đổi từ kiểu `int` sang `float`.



Ở đây, chuyển đổi ngầm định được sử dụng, dựa trên các phép tính toán, phép gán.

7.2. Quy tắc chuyển đổi kiểu dữ liệu ngầm định:

- Với phép gán (`a = b`): các biến, giá trị sẽ được chuyển đổi sang kiểu dữ liệu của biến nằm trái phép gán (bên trái dấu `=`).
- Với các phép còn lại: dựa trên kiểu dữ liệu nào có khoảng giá trị rộng hơn. Ví dụ: các biến số nguyên sẽ được đổi sang số thực, các biến kiểu `int` sẽ được đổi sang kiểu `long long`.

7.3. Ví dụ 2:

- Xét đoạn code sau:

```
// cre: https://www.programiz.com/cpp-programming/type-conversion
#include <iostream>

using namespace std;

int main() {
```

```

double num_double = 3.56;
cout << "num_double = " << num_double << endl;

int num_int1 = (int)num_double;
cout << "num_int1  = " << num_int1 << endl;

int num_int2 = int(num_double);
cout << "num_int2  = " << num_int2 << endl;

return 0;
}

```

- Giải thích:

- `int num_int1 = (int)num_double` : biến `num_double` chuyển từ kiểu `double` (số thực) → `int` (số nguyên), bằng cách giữ phần nguyên và bỏ phần thập phân. Ví thể: `3.56` → `3`.
- `int num_int2 = int(num_double)` : giống như trên nhưng viết theo cách gọi hàm (sẽ được nói thêm trong bài viết về hàm).



Ở đây, chuyển đổi tương minh được sử dụng, bằng việc nêu rõ kiểu dữ liệu cần chuyển sang là `int` cho biến `num_double` đang thuộc kiểu `double`.

7.4. Cách chuyển (ép) kiểu dữ liệu tương minh:

- Dựa vào đoạn code trên, có cách để ép kiểu dữ liệu tương minh:

```

// cách 1:
(<kiểu_dữ_liệu>) <giá_trị_hoặc_biến>

// cách 2:
<kiểu_dữ_liệu>(<giá_trị_hoặc_biến>)

```


7.5. Lưu ý về ép kiểu dữ liệu tường minh:

- Việc ép dữ liệu từ kiểu dữ liệu có khoảng giá trị lớn hơn sang kiểu dữ liệu có khoảng giá trị nhỏ hơn có thể dẫn đến thất thoát dữ liệu, và kết quả không mong muốn:

```
#include <iostream>

using namespace std;

int main() {
    int num = 500;
    cout << (char) num << endl; // kết quả sẽ là một ký tự ngẫu nhiên
}
```



Điều này thực chất giống như việc gán một giá trị nằm ngoài khoảng giá trị cho phép cho một biến.

- Tuy nhiên, trong trường hợp ép kiểu dữ liệu từ số thực → số nguyên, phần thập phân bị loại bỏ và giữ lại phần nguyên. Sự thất thoát dữ liệu này là có thể chấp nhận được, hoặc là cần thiết trong nhiều trường hợp. Ví dụ: thuật toán kiểm tra số nguyên tố.

7.6. Kết luận:

- Có thể nói, việc ép kiểu dữ liệu là vô cùng cần thiết cho những công đoạn tính toán. Lập trình viên nên rèn luyện về logic liên quan đến các kiểu dữ liệu vì nó là cơ bản của các phép toán.