



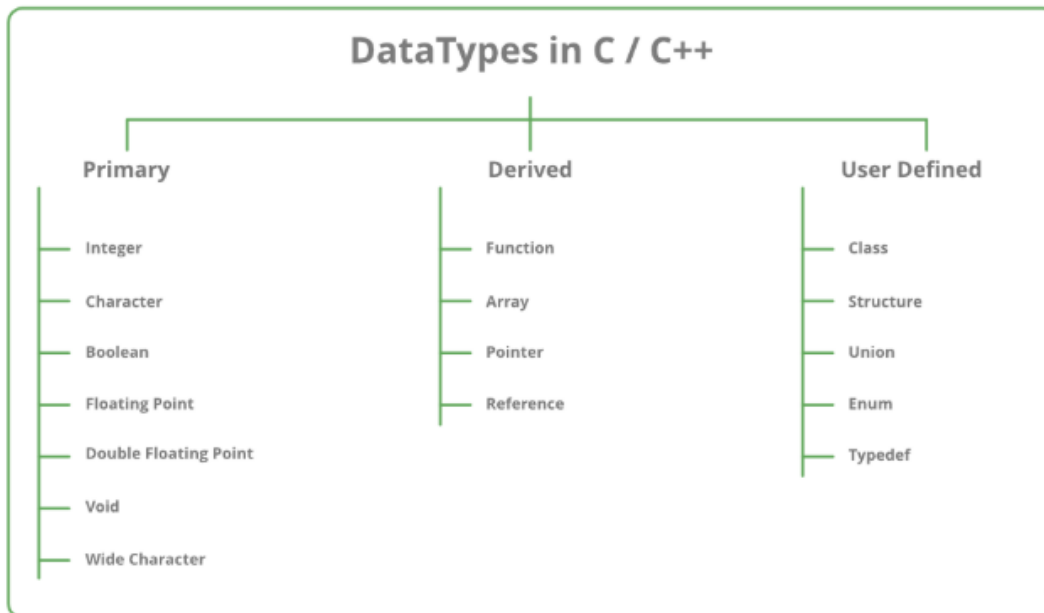
CHI TIẾT

Kiểu dữ liệu trong lập trình C++

(tài liệu tham khảo dành cho học viên)

A - C++ Data Types?

- Tất cả các biến sử dụng kiểu dữ liệu trong khi khai báo để hạn chế loại dữ liệu sẽ được lưu trữ.
- Các kiểu dữ liệu được sử dụng để báo cho các biến số loại dữ liệu mà nó có thể lưu trữ.
- Bất cứ khi nào một biến được định nghĩa trong C ++, trình biên dịch sẽ phân bổ một số bộ nhớ cho biến đó dựa trên kiểu dữ liệu được khai báo.
- Mỗi loại dữ liệu đòi hỏi một lượng bộ nhớ khác nhau.



- Các kiểu dữ liệu trong C ++ chủ yếu được chia thành ba loại:
 - **Primitive** Data Types
 - **Derived** Data Types
 - **User-Defined** Data Types

1. Primitive Data Types?

- Đây là các kiểu dữ liệu được dựng sẵn (build-in) hoặc được xác định trước và có thể được người dùng sử dụng trực tiếp để khai báo các biến.
- ví dụ: **int**, **char**, **float**, **bool**, v.v. Các kiểu dữ liệu nguyên thủy có sẵn trong C++ là:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

2. Derived Data Types?

- Các kiểu dữ liệu được dẫn xuất từ các kiểu dữ liệu nguyên thủy hoặc dựng sẵn được gọi là Derived Data Types. Đây có thể là bốn loại cụ thể là:
 - Function
 - Array
 - Pointer
 - Reference

3. User-Defined Data Types?

- Những kiểu dữ liệu này được xác định bởi chính người dùng.
- Giống như, định nghĩa một lớp trong C++ hoặc một cấu trúc.
- C++ cung cấp các kiểu dữ liệu do người dùng định nghĩa sau:
 - Class
 - Structure
 - Union
 - Enumeration
 - Typedef defined DataType

B - Nội dung chi tiết về kiểu Primitive Data Types?

- **Integer:** Từ khóa được sử dụng cho các loại dữ liệu số nguyên là int. Các số nguyên thường yêu cầu 4 byte không gian bộ nhớ và nằm trong khoảng từ -2147483648 đến 2147483647.
- **Character:** Kiểu dữ liệu ký tự được sử dụng để lưu trữ ký tự. Từ khóa được sử dụng cho kiểu dữ liệu ký tự là char. Các ký tự thường yêu cầu 1 byte dung lượng bộ nhớ và nằm trong khoảng từ -128 đến 127 hoặc 0 đến 255.

- **Boolean:** Kiểu dữ liệu Boolean được sử dụng để lưu trữ các giá trị boolean hoặc logic. Một biến boolean có thể lưu trữ **True** hoặc **False**. Từ khóa được sử dụng cho kiểu dữ liệu boolean là **bool**.
- **Floating Point:** Kiểu dữ liệu Dấu phẩy động được sử dụng để lưu trữ các giá trị dấu phẩy động chính xác hoặc giá trị thập phân. Từ khóa được sử dụng cho kiểu dữ liệu dấu phẩy động là **float**. Các biến float thường yêu cầu 4 byte không gian bộ nhớ.
- **Double Floating Point:** Kiểu dữ liệu Dấu phẩy động kép được sử dụng để lưu trữ giá trị dấu phẩy động chính xác kép hoặc giá trị thập phân. Từ khóa được sử dụng cho kiểu dữ liệu dấu phẩy động kép là **double**. Biến double thường yêu cầu 8 byte không gian bộ nhớ.
- **void:** Void có nghĩa là không có giá trị. void datatype đại diện cho một thực thể vô giá trị. Kiểu dữ liệu trống được sử dụng cho các hàm không trả về giá trị.
- **Wide Character:** Kiểu dữ liệu ký tự rộng cũng là kiểu dữ liệu ký tự nhưng kiểu dữ liệu này có kích thước lớn hơn kiểu dữ liệu 8 bit thông thường. Đại diện bởi **wchar_t**. Nó thường dài 2 hoặc 4 byte.

C - Hiểu thêm về kiểu Wide Character?

- **Wide char** tương tự như kiểu dữ liệu **char**, ngoại trừ **wide char** chiếm hai lần dung lượng và kết quả có thể nhận các giá trị lớn hơn nhiều.
- **Char** có thể lấy **256** giá trị tương ứng với các mục trong bảng ASCII. Nhưng **Wide Char** có thể nhận 65536 giá trị tương ứng với các giá trị UNICODE, là một tiêu chuẩn quốc tế gần đây cho phép mã hóa các ký tự cho hầu hết tất cả các ngôn ngữ và các ký hiệu thường được sử dụng.
 - Giống như kiểu cho các hằng ký tự là char, kiểu cho ký tự rộng là **wchar_t**.
 - Kiểu dữ liệu này chiếm 2 hoặc 4 byte tùy thuộc vào trình biên dịch được sử dụng.
 - Chủ yếu là kiểu dữ liệu **wchar_t** được sử dụng khi các ngôn ngữ quốc tế như tiếng Nhật được sử dụng.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị cách sử dụng **wchar_t**:

```

//IMIC - An example C++ program to demonstrate use of wchar_t
#include <iostream>
using namespace std;

int main()
{
    wchar_t w = L'A';
    cout << "Wide character value:: " << w << endl ;
    cout << "Size of the wide char is:: " << sizeof(w);
    return 0;
}

```

Output:

```


Wide character value:: 65
Size of the wide char is:: 4

```

- **L** là tiền tố cho các ký tự rộng (wide character) và các ký tự chuỗi ký tự rộng (wide-character string) cho trình biên dịch biết rằng char hoặc chuỗi có kiểu char rộng.
- **w** có tiền tố trong các hoạt động như **scanning** (wcin) hoặc **printing** (wcout) trong khi vận hành loại wide-char type.

D - Wide char type array hoặc string?

- Cũng giống như chuỗi mảng kiểu **char**, cũng có thể có chuỗi mảng kiểu **wide-char**. Dưới đây là triển khai C++ để hiển thị chuỗi mảng kiểu **wide-char**:



```

//IMIC - An example C++ program to demonstrate use
// of wchar_t in array
#include <iostream>
using namespace std;

int main()
{
    // char type array string
    char caname[] = "IMIC Technology" ;
    cout << caname << endl ;

    // wide-char type array string
    wchar_t waname[] = L"IMIC Technology" ;
    wcout << waname << endl;

    return 0;
}

```

Output:

```

IMIC Technology
IMIC Technology

```

- Đầu ra **là như nhau**, nhưng sự **khác biệt duy nhất là mảng ký tự rộng sử dụng bộ nhớ gấp đôi** để mã hóa mỗi ký tự.

E - Functions for wide character array strings?

- Hầu hết các hàm cho chuỗi mảng ký tự rộng được xác định trong tệp tiêu đề **cwchar**.
- **wcslen(): size_t wcslen (const wchar_t* wcs);**
- Nó trả về độ dài của chuỗi rộng. Đây là hàm tương đương với **strlen**.
- Dưới đây là một triển khai C ++ đơn giản để chỉ ra cách lấy độ dài của chuỗi ký tự rộng:

```

//IMIC - An example C++ program to demonstrate use
// of wcslen()
#include <iostream>
#include<wchar>
using namespace std;

int main()
{
    // wide-char type array string
    wchar_t waname[] = L"IMIC Technology" ;

    wcout << L"The length of '" << waname
           << L"' is " << wcslen(waname) << endl;


    return 0;
}

```

Output:

```
The length of 'IMIC Technology' is 15
```

- **wcscpy()**: `wchar_t *wcscpy(wchar_t *strDestination, const wchar_t *strSource);`
- **wcscpy()** là viết tắt của Wide-Character String Copy.
- Nó sao chép một chuỗi ký tự rộng được trỏ bởi strSource vào mảng ký tự rộng được trỏ bởi strDestination.
- Đây là hàm tương đương với **strcpy**.
- Dưới đây là một triển khai C++ đơn giản để hiển thị việc sử dụng **wcscpy**:



```
//IMIC - An example C++ program to demonstrate use
// of wcsncpy()
#include <iostream>
#include<wchar>
using namespace std;


int main()
{
    wchar_t waname[] = L"IMIC Technology" ;
    wchar_t wacopy[14];
    wcsncpy(wacopy, waname);
    wcout << L"Original = " << waname
            << L"\nCopy = " << wacopy << endl;

    return 0;
}
```

Output:

```
Original = IMIC Technology
Copy = IMIC Technology
```

- **wcscat():** `wchar_t *wcscat(wchar_t *strDestination, const wchar_t *strSource);`
- `wcscat()` là viết tắt của Ghép chuỗi ký tự rộng (Wide-Character String Concatenation).
- Nối một bản sao của chuỗi rộng `strSource` vào chuỗi rộng `strDestination`.
- Đây là hàm tương đương với **strcat**.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị việc sử dụng **wcscat**:



```
//IMIC - An example C++ program to demonstrate use
// of wcscat()
#include <iostream>
#include<wchar>
using namespace std;

int main()
{
    wchar_t string1[] = L"IMIC Technology" ;
    wchar_t string2[] = L" - keep up technology" ;

    wcscat(string1, string2);

    wcout << L"Concatenated wide string is = "
        << string1 << endl;

    return 0;
}
```

Output:

```
Concatenated wide string is = IMIC Technology - keep up technology
```

- **wcscmp()**: `int wcscmp(const wchar_t* wcs1, const wchar_t* wcs2);`
- `wcscmp()` là viết tắt của So sánh chuỗi ký tự rộng (Wide-Character String Comparison).
- Nó trả về 0 nếu **wcs1 và wcs2 bằng nhau**, trả về giá trị > 0 nếu ký tự rộng đầu tiên không khớp có giá trị lớn hơn trong wcs1 so với wcs2. Và trả về giá trị < 0 nếu ký tự rộng đầu tiên không khớp có giá trị nhỏ hơn trong wcs1 so với wcs2.
- Đây là hàm tương đương với **strcmp**.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị việc sử dụng **wcscmp**:


```

//IMIC - An example C++ program to demonstrate use
// of wcsncmp()
#include <iostream>
#include <wchar>
using namespace std;

int main()
{
    wchar_t string1[] = L"IMIC Technology" ;
    wchar_t string2[] = L"IMIC" ;
    wcout << L"Comparison1 = "
            << wcsncmp(string1, string2) << endl;
    wcout << L"Comparison2 = "
            << wcsncmp(string1, string1) << endl;
    wcout << L"Comparison3 = "
            << wcsncmp(string2, string1) << endl;
    return 0;
}

```





Output:

```

Comparison1 = 1
Comparison2 = 0
Comparison3 = -1

```

- **wcstok()**: wchar_t* wcstok(wchar_t* str, const wchar_t* delim, wchar_t ** ptr);
- wcstok() là viết tắt của mã thông báo chuỗi ký tự rộng (Wide-Character String Tokenize).
- Tìm mã thông báo tiếp theo trong chuỗi rộng kết thúc null được trả bởi str.
- Các ký tự phân cách được xác định bởi chuỗi rộng kết thúc null được chỉ ra bởi dấu phân cách.
- str - con trỏ tới chuỗi rộng kết thúc null để token hóa.
- delim - con trỏ tới chuỗi phân định xác định chuỗi rộng kết thúc null.
- ptr - con trỏ tới một đối tượng có kiểu wchar_t *, được wcstok sử dụng để lưu trữ trạng thái bên trong của nó.
- Đây là ký tự rộng tương đương với **strtok()**.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị việc sử dụng wcstok:

 //IMIC - An example C++ program to demonstrate use
 // of wcstok()
 #include <iostream>
 #include <cwchar>
 using namespace std;
 int main()
 {
 wchar_t string[] = L"IMIC Technology,is,keep up,technology" ;

 wchar_t* internal_state;

 wchar_t delim[] = L"," ;
 wchar_t* token = wcstok(string, delim, &internal_state);





 while (token)
 {
 wcout << token << endl;
 token = wcstok(NULL, delim, &internal_state);
 }

 return 0;
 }

Output:

```
IMIC Technology
is
keep up
technology
```

- **wcsncpy()**: wchar_t* wcsncpy(wchar_t* destination, const wchar_t* source, size_t n);
- Sao chép n ký tự rộng đầu tiên của nguồn tới đích.
- Nếu kết thúc chuỗi rộng nguồn được tìm thấy trước khi n ký tự được sao chép, đích được đệm thêm ký tự rộng null cho đến khi có tổng số n ký tự.
- Đây là hàm tương đương với **strncpy()**.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị việc sử dụng wcsncpy:

```
//IMIC - An example C++ program to demonstrate use
// of wcsncpy()
#include <iostream>
#include<wchar>
using namespace std;

int main()
{
    wchar_t string1[] = L"IMIC Technology Viet Nam";
    wchar_t string2[20];
    wchar_t string3[20];

    wcsncpy(string2, string1, 20);

    // partial copy
    wcsncpy(string3, string2, 5);

    string3[5] = L'\0'; // null character manually added

    wcout << string1 << endl << string2
          << endl << string3 ;

    return 0;
}
```

Output:

```
IMIC Technology Viet Nam
IMIC Technology Viet
IMIC
```

- **wcsstr()**: const wchar_t* wcsstr (const wchar_t* wcs1, const wchar_t* wcs2);
- Trả về một con trỏ đến lần xuất hiện đầu tiên của wcs2 trong wcs1.
- Nó trả về một con trỏ null nếu wcs2 không phải là một phần của wcs1.
- Ở đây, wcs1 là chuỗi ký tự rộng được quét và wcs2 chứa chuỗi phù hợp.
- Đây là hàm tương đương với **strstr()**.
- Dưới đây là một triển khai C ++ đơn giản để hiển thị việc sử dụng wcsstr:

```

//IMIC - An example C++ program to demonstrate use
// of wcsstr()
#include <iostream>
#include<cwchar>
using namespace std;

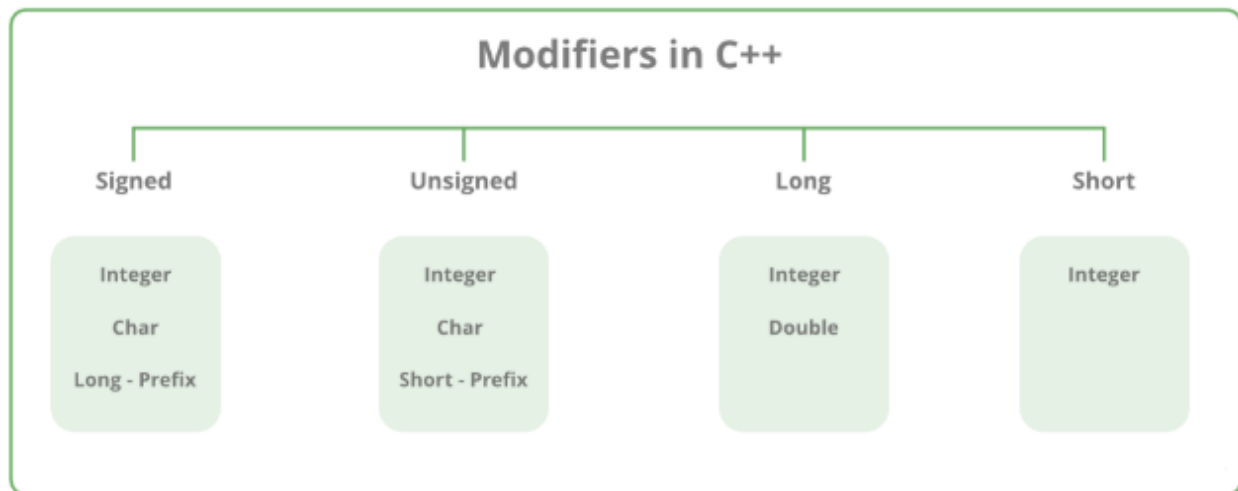
int main()
{
    wchar_t string1[] = L"IMIC Technology Viet Nam";
    wchar_t* string2 = wcsstr(string1, L"Nam");
    wcsncpy(string2, L"Soft", 2);
    wcout << string1 << endl;
    return 0;
}

```

Output:

IMIC Technology Viet Som


F. Datatype Modifiers?



- Data type modifiers có sẵn trong C++ là:
 - Signed
 - Unsigned
 - Short
 - Long

| DATA TYPE | SIZE (IN BYTES) | RANGE |
|------------------------|-----------------|---------------------------------|
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 | 0 to 4,294,967,295 |
| long long int | 8 | $-(2^{63})$ to $(2^{63})-1$ |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 | |
| double | 8 | |
| long double | 12 | |
| wchar_t | 2 or 4 | 1 wide character |

- **Lưu ý:** Giá trị trên có thể thay đổi từ trình biên dịch này sang trình biên dịch khác.
- Chúng ta có thể hiển thị kích thước của tất cả các loại dữ liệu bằng cách sử dụng hàm **sizeof()** và chuyển từ khóa của kiểu dữ liệu làm đối số cho hàm này như dưới đây:



```
//IMIC - C++ program to sizes of data types
#include<iostream>
using namespace std;

int main()
{
    cout << "Size of char : " << sizeof(char)
        << " byte" << endl;
    cout << "Size of int : " << sizeof(int)
        << " bytes" << endl;
    cout << "Size of short int : " << sizeof(short int)
        << " bytes" << endl;
    cout << "Size of long int : " << sizeof(long int)
        << " bytes" << endl;
    cout << "Size of signed long int : " << sizeof(signed long int)
        << " bytes" << endl;
    cout << "Size of unsigned long int : " << sizeof(unsigned long int)
        << " bytes" << endl;
    cout << "Size of float : " << sizeof(float)
        << " bytes" << endl;
    cout << "Size of double : " << sizeof(double)
        << " bytes" << endl;
    cout << "Size of wchar_t : " << sizeof(wchar_t)
        << " bytes" << endl;

    return 0;
}
```

Output:

```
Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
Size of unsigned long int : 8 bytes
Size of float : 4 bytes
Size of double : 8 bytes
Size of wchar_t : 4 bytes
```

--- HẾT ---