

TIN HỌC

MỘT TÀI LIỆU CỦA CLB TIN HỌC NHH

PHẦN 2: C++

<IT / >
NHH IT CLUB



Mục lục

I. Mô tả thuật toán.....	3
0. Định nghĩa.....	4
1. Mô tả thuật toán bằng lời	5
2. Mô tả thuật toán bằng pseudocode (mã giả)	6
3. Mô tả thuật toán bằng flowchart (lưu đồ)	7
II. Giới thiệu về C++	11
1. Sức mạnh C++	11
2. Cài đặt trình soạn thảo và IDE	12
3. Cấu trúc một chương trình C++	13
4. Khai báo namespace	14
III. Viết C++	15
Bài 1. Tầm vực (Scope) và Comment.....	16
Bài 2. Biến và Kiểu dữ liệu	18
Bài 3. Định dạng dữ liệu xuất ra.....	20
Bài 4. Toán tử và ép kiểu dữ liệu	22
Bài 5. Macro	25
Bài 6. Đọc và ghi file cơ bản	27
Bài 7. If – Else và Switch.....	29
Bài 8. Cấu trúc lặp (Iteration)	33
Bài 9. Array.....	39
Bài 10. Chuỗi.....	42
Bài 11. Structure.....	44
Bài 12. Pointer (Con trỏ)	46
Bài 13. Pointer (Con trỏ) (tiếp theo)	49
Bài 14. Hàm (Function)	52

I.

MÔ TẢ THUẬT TOÁN

0. Định nghĩa

Mô tả thuật toán là một công việc **cực kỳ quan trọng** trong quá trình lập trình. Nếu không thể mô tả thuật toán một cách chính xác, bạn sẽ không thể khiến máy tính và chương trình làm việc theo ý mình. Một ví dụ điển hình là việc mô tả thuật toán tìm giai thừa.

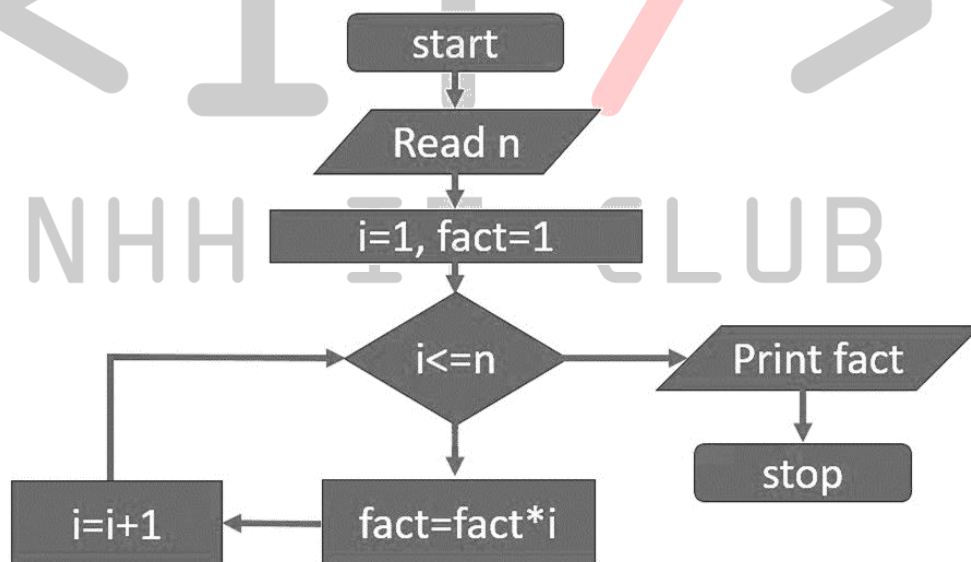
“Giai thừa của 1 số tự nhiên n được định nghĩa là tích của tất cả các số nguyên dương từ 1 đến số n đó.”

Chúng ta có thể hiểu dễ dàng định nghĩa này. Tuy nhiên, máy tính của chúng ta không thông minh đến thế. Việc định nghĩa tất cả các số nguyên dương từ 1 tới n đòi hỏi một thuật toán dựa trên **quy tắc** của dãy số đó:

“Phần tử đứng sau bằng phần tử đứng trước nó cộng 1.”

Từ đó, thuật toán của khái niệm này có thể được mô tả rõ ràng như sau:

“Cho $i = 1$ là một số tự nhiên, $fact$ cũng vậy. $fact$ sau bằng $fact$ trước nhân với i , rồi i tăng 1 đơn vị. Lặp lại đến khi i bằng $n + 1$ thì dừng. $Fact$ là giá trị cần tìm.”

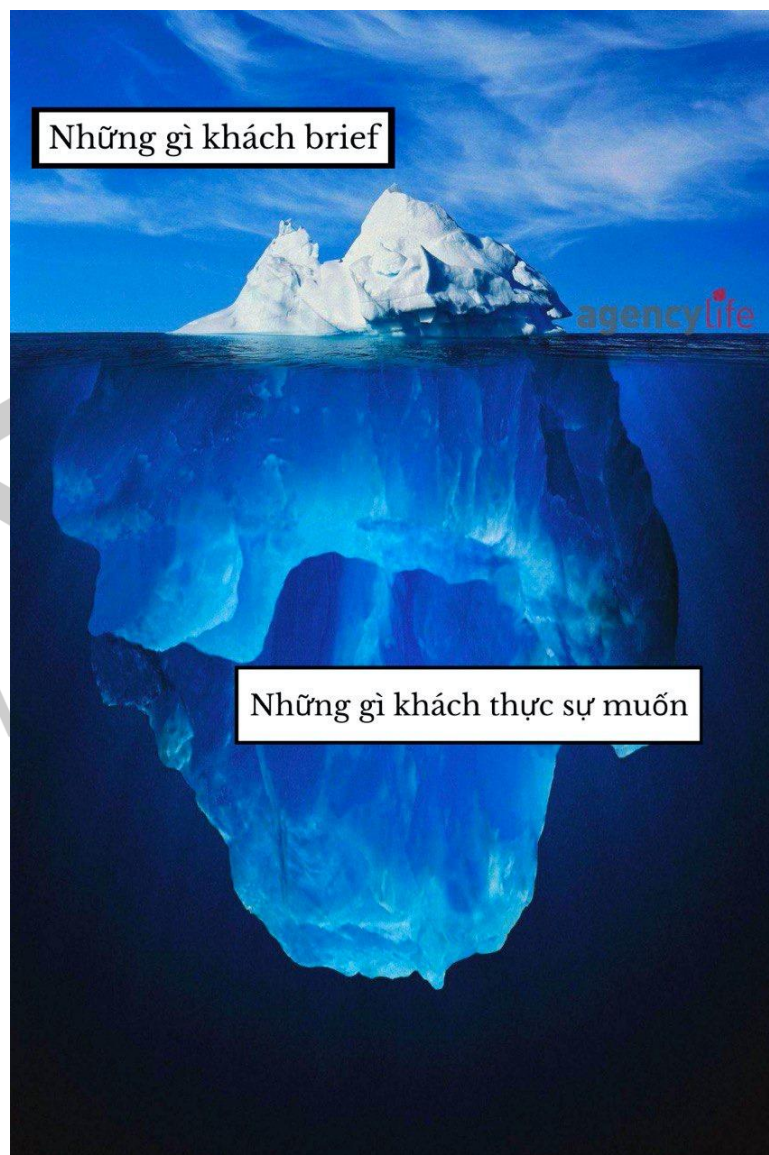


Vậy, mô tả thuật toán cũng giống như **dạy một triệu đứa trẻ làm toán cùng lúc**. Bạn phải dạy cho chúng một cách chính xác cách làm, và chúng sẽ thực hiện cho bạn một triệu phép tính, cùng lúc. Nhưng hãy cẩn thận! Nếu bạn dạy sai, một triệu phép tính ấy sẽ đều sai bét, và đó là lúc chương trình của bạn có bug.

1. Mô tả thuật toán bằng lời

Đây là cách mô tả thuật toán phổ biến nhất. Khách hàng của bạn muốn một phần mềm kiểm tra số lượng hàng hóa bán ra theo giờ? Công ty của bạn cần một chương trình kiểm soát traffic đến máy chủ công ty? Tất cả các nhu cầu của khách hàng đều là mô tả bằng lời cho nhu cầu của họ.

Nói đến đây chưa đủ. Chính xác hơn là lời nói của khách hàng chưa mô tả đủ các công việc cần làm. Việc của bạn là biến chúng thành những mô tả chính xác và đầy đủ hơn, đủ chính xác để có thể biến thành một **mô tả thuật toán bằng lời** hoàn chỉnh.



Đúng... Nó đó các bạn...

Tuy nhiên, việc sử dụng mô tả thuật toán bằng lời để diễn tả thuật toán là một việc cực kỳ nguy hiểm. Bởi dân gian thường nói “Ông nói gà, bà nói vịt”, việc các developer hiểu lầm ý của nhau, hay thậm chí là chính bản thân sau vài đêm thức trắng, cũng quên mất

mình đã viết gì, là chuyện xảy ra như cơm bữa. Chính vì vậy, những phương pháp mô tả thuật toán khác đã ra đời, trong đó có mô tả bằng pseudocode và flowchart.

2. Mô tả thuật toán bằng pseudocode (mã giả)

```
function randomSound()  
Loop: from i = 1 to 100  
    print_number = True  
    If i is even Then  
        If print_number Then  
            Print (i + random())  
        Else  
            PlaySound( rand() )  
        End If  
    Else  
        Print "Odd..."  
    End If  
End (loop)  
End (function)
```

Mã giả

Thoạt nhìn, tất cả mọi người đều có cảm giác code này “rất quen nhưng cũng rất lạ”, không hề giống bất cứ một ngôn ngữ nào mà mình đã từng học. Thực ra, nó không phải bất kỳ một ngôn ngữ nào cả, bởi code này là pseudocode.

Pseudocode là một khái niệm mà, code được biểu diễn theo một quy tắc độc lập với tất cả hoặc hầu hết các ngôn ngữ lập trình khác. Nó có thể được dùng để mô tả chính xác nguyên lý hoạt động của 1 chương trình máy tính hoặc 1 thuật toán.

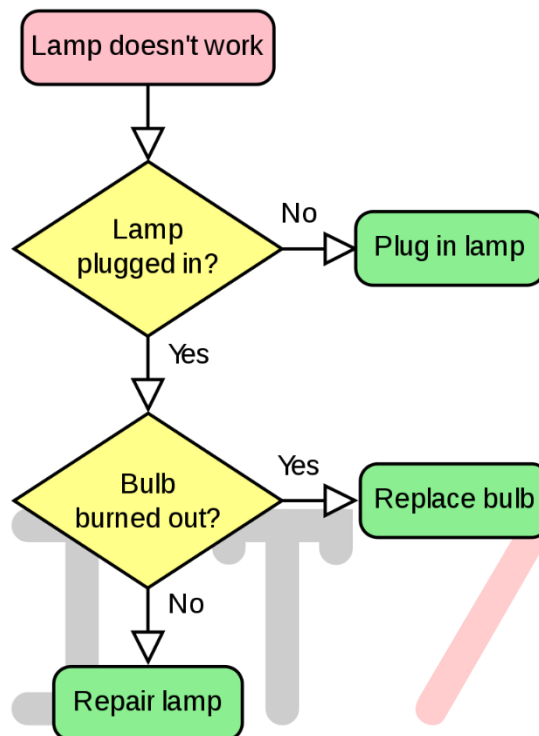
Pseudocode không có bất kỳ một tiêu chuẩn hay quy tắc gì áp dụng chung cả. Người viết có thể thoải mái biểu diễn thuật toán theo cách của riêng mình, miễn là nó rõ ràng và dễ đọc. Trừ phi công ty hoặc tổ chức của bạn có những yêu cầu, quy định riêng, không gì có thể ngăn bạn trong pseudocode.

Việc trình bày pseudocode nên tuân theo những quy ước sau, vì một thế giới code sạch đẹp:

- Viết phỏng theo ngôn ngữ tự nhiên và 1 ngôn ngữ lập trình thống nhất.
- Mặc kệ những tiểu tiết thừa.
- Không cần giải thích những điều hiển nhiên..
- Cố gắng code ngắn gọn

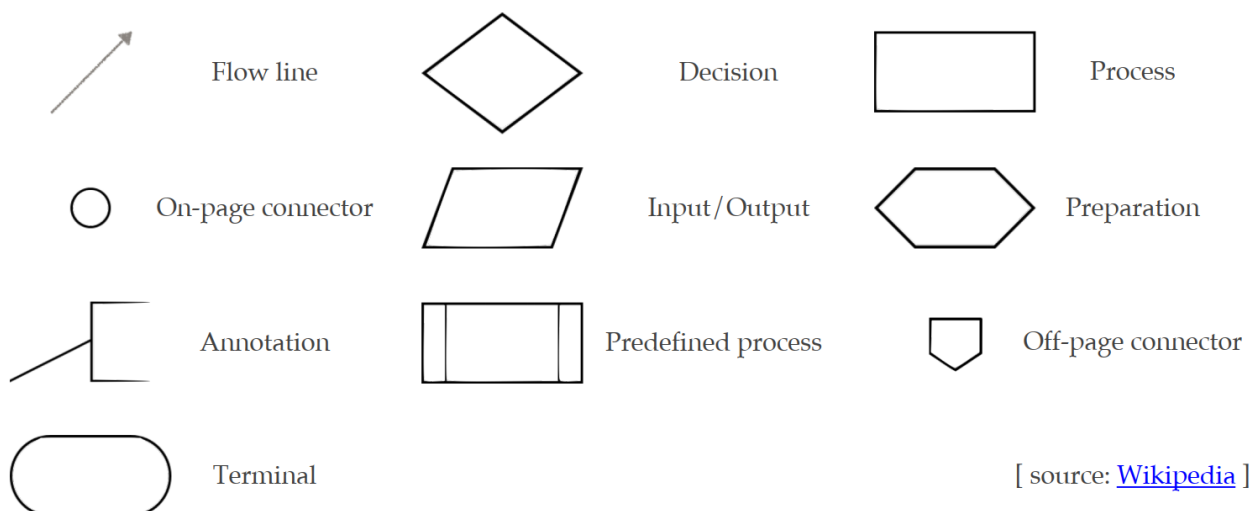
3. Mô tả thuật toán bằng flowchart (lưu đồ)

Flowchart là một loại sơ đồ biểu diễn một thuật toán hoặc một quá trình, biểu hiện các bước công việc dưới dạng các loại hình hộp khác nhau theo thứ tự được biểu diễn bởi các mũi tên. Sơ đồ này có thể thể hiện giải pháp cho vấn đề cần giải quyết từng bước từng bước một.



Đây là 1 flowchart mà mình copy trên Wikipedia, do mình lười vẽ quá

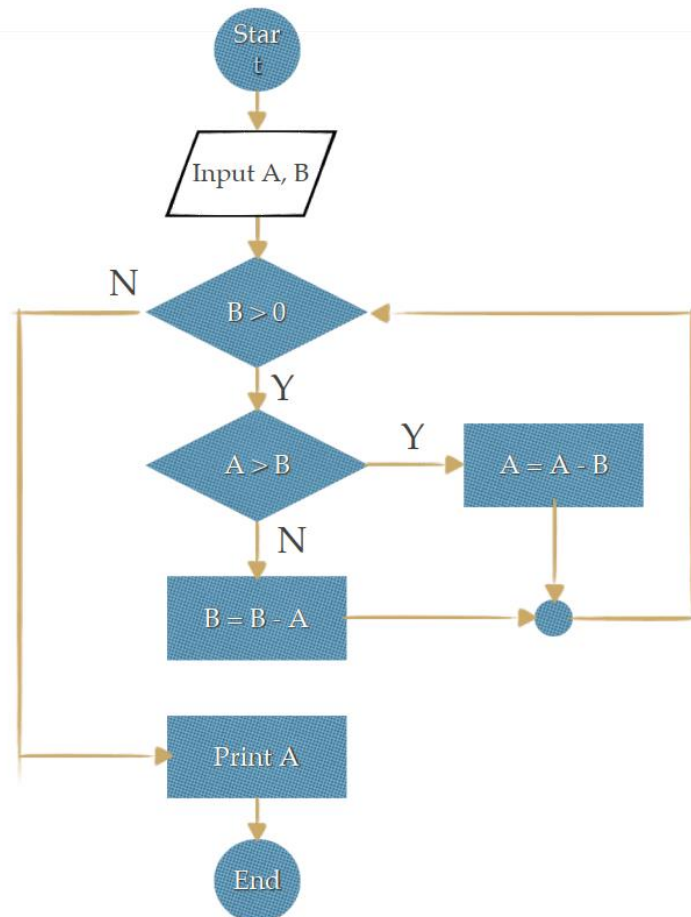
Vẽ flowchart có những quy tắc nhất định. Quan trọng nhất, các hình khối phải tuân theo quy tắc sau đây:



Một câu lệnh if phải được biểu diễn bằng 1 hình thoi và 2 nhánh Yes/No, cũng như một xử lý (process) không thể được biểu diễn bằng 1 hình tròn được. Tất cả phải tuân theo quy tắc.

Ví dụ cho pseudocode và flowchart:

```
INPUT A,B
Loop while B > 0
  If A > B then
    A = A - B
  Else
    B = B - A
  End
End loop
Print A
```



Ghi nhớ

- Một bài toán cần được mô tả chính xác trước khi lập trình.
- Lời, pseudocode và flowchart là 3 cách phổ biến nhất để mô tả thuật toán.



II.

GIỚI THIỆU VỀ C++

C++

/si: pləs pləs/

“C plus plus”

“C cộng cộng”

“C makes it easy to shoot yourself in the foot; C++ makes it harder,
but when you do, it blows away your whole leg.”

- *Bjarne Stroustrup*

1. Sức mạnh C++

Không cần phải bàn cãi khi nói C++ là ngôn ngữ lập trình phổ biến và mạnh mẽ nhất thế giới hiện nay. Nếu như bạn có ý định học 1 ngôn ngữ lập trình, C++ sẽ là lựa chọn lý tưởng. Đây là 1 ngôn ngữ rất hiệu quả để bạn lập trình các phần mềm, ứng dụng cho nhiều nền tảng như: PC, workstations, mainframes, tablets, và điện thoại thông minh. Gần như tất cả các kiểu chương trình đều có thể viết bằng C++, từ drivers, OS, đến hệ thống quản trị hoặc game.



Compiler của C++ cũng cực kỳ phổ biến và đa dạng. Có rất nhiều compiler chạy trên PC, workstations, mainframes được cập nhật thường xuyên, cùng khả năng compile xuyên nền tảng, tức là code có thể được phát triển trên môi trường này và compile để chạy trên môi trường khác.

C++ đi kèm với một Thư viện (Library) Standard rất sâu rộng. Nó tập hợp cực kỳ nhiều các thủ tục (routines) và định nghĩa (definitions) giúp cung cấp các tính năng mà các chương trình thường sử dụng. Điển hình đó là các phép tính số học, xử lý chuỗi, sắp xếp & tìm kiếm, tổ chức và quản lý dữ liệu, và nhập-xuất. Standard Library rộng đến mức mà chúng ta sẽ chỉ “cạo nhẹ bề mặt” của thư viện này trong suốt cả tài liệu.

Chúc các bạn học tốt!

2. Cài đặt trình soạn thảo và IDE

Hiện nay, có vô vàn các trình soạn thảo (Editor) và môi trường thiết kế hợp nhất (IDE). Đối với các bạn mới tìm hiểu về lập trình, các IDE tích hợp đầy đủ Editor và Compiler là thích hợp nhất vì tính tiện dụng. Sau đây là 1 số trình IDE thích hợp nhất cho học sinh THPT:

- Microsoft Visual Studio **RECOMMENDED**



Đây là IDE phổ biến và mạnh mẽ nhất hiện tại của Microsoft. Bản thân tác giả cũng đang sử dụng IDE này. Phiên bản Community hoàn toàn **miễn phí** cung cấp rất nhiều công cụ mạnh như Intellisense, các công cụ theo dõi cho Debugging cũng như tích hợp sẵn nhiều thư viện của Windows. Tốc độ biên dịch cũng cực kỳ cao.

Đây gần như là trình IDE hoàn thiện và mạnh nhất thời điểm hiện tại.

- JetBrains CLion



Phần mềm CLion là một sản phẩm thông minh của JetBrains. CLion là một sự bổ sung thông minh vào nền tảng IDE để phát triển ngôn ngữ C và C++. Được xây dựng là một sản phẩm hàng đầu của IntelliJ Platform, CLion bao gồm rất nhiều tính năng thông minh giúp thúc đẩy sự gia tăng hiệu suất code.

Nhược điểm duy nhất chính là giá thành. Bộ sản phẩm của JetBrains miễn phí với đối tượng học sinh, sinh viên, tuy nhiên nếu là một người đi làm, bạn sẽ phải trả phí cho các phần mềm này. Nhưng mà, tiền nào của đó!

3. Cấu trúc một chương trình C++

```
#include<iostream>      ← preprocessor directives

int main()              ← main() function
{
    std::cout << "Welcome to C++!\n";
    return 0;
}
```

Một chương trình C++ bao gồm các thành phần sau:

- **Preprocessor directive (chỉ thị tiền xử lý):** là tất cả những thứ bắt đầu bằng #
 - ❖ bao gồm: các source kèm theo (thư viện hoặc code khác).
 - ❖ define, undef, v.v.: để định nghĩa constant (hằng), macros.
 - ❖ pragma: những chỉ dẫn đặc biệt cho trình biên dịch (compiler).
- Hàm **main()**: là điểm bắt đầu của chương trình. Nơi tình yêu bắt đầu.
- **Biến toàn cục (global variable):** các biến này có thể truy xuất được ở mọi lớp (class) và hàm (function) trong chương trình.
- Các khai báo và hiện thực hóa các cấu trúc (struct), lớp và hàm.
- **Namespace:** dùng để nhóm các thành phần của 1 module, 1 thư viện hoặc 1 nhóm các thư viện.

Như bạn có thể thấy lệnh **cout** (in ra màn hình) nằm trong namespace **std (standard)** phải được viết đầy đủ như trên. Tuy nhiên, trong trường hợp đa phần các lệnh đều nằm trong namespace này, để tránh lặp đi lặp lại cụm **std::**, chúng ta có thể khai báo namespace mặc định ở đầu chương trình.

4. Khai báo namespace

Xem một namespace như là một ngôn ngữ. Có rất nhiều ngôn ngữ trên thế giới, cũng giống như có rất nhiều namespace trong C++. Giả sử ta có 2 namespace là **tiengAnh** và **tiengViet**.

Khi ta viết 1 từ, ví dụ **hang**, để người nghe hiểu được từ đó thuộc ngôn ngữ nào, ta phải nói rõ ngôn ngữ đó ra. Ví dụ:

tiengAnh::hang	=	treo lên
tiengViet::hang	=	hang động

Tuy nhiên, nếu cả 2 đều là người Việt, hoặc 2 người đã thống nhất với nhau sẽ chỉ nói tiếng Việt, thì để tránh lặp đi lặp lại **tiengViet::**, người ta sẽ khai báo ở đầu cuộc giao tiếp một câu, đó chính là **using namespace tiengViet**.

Kể từ lúc này, mặc định từ hang sẽ được hiểu theo nghĩa tiếng Việt. Người nói nếu muốn đề cập đến hang trong tiếng Anh sẽ phải khai báo rõ ra. Ví dụ:

hang	=	hang động
tiengAnh::hang	=	treo lên
hang	=	hang động

Vậy, đoạn code ở phần 3 sẽ trở thành:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to C++!\n";
    return 0;
}
```

Trong tài liệu này cũng như sau này, đa phần các chương trình đều sẽ sử dụng `#include <iostream>` và `using namespace std` như các thư viện và miền tên chuẩn.

III.

VIẾT C++

Bài 1. Tầm vực (Scope) và Comment

1. Khái niệm tầm vực

Tầm vực (Scope) là vùng hiệu lực của 1 biến, hàm hoặc cấu trúc. Tầm vực trong C++ được giới hạn trong 2 dấu { }.

Ví dụ:

```
{  
    int a = 3;  
    {  
        int b = 2;  
    }  
}
```

Trong trường hợp này, b thuộc scope con của scope chứa a. Lúc này mọi hàm, biến trong scope con chứa b có thể nhìn thấy và truy xuất a, nhưng hàm và biến trong scope chứa a không thể nhìn thấy b.

Nói 1 cách đơn giản hơn, scope con có thể sử dụng dữ liệu từ scope mẹ, nhưng scope mẹ thì không thể sử dụng dữ liệu của scope con.

2. Về biến b

Sau khi chương trình ra khỏi scope con, biến b tự động được giải phóng. Nói cách khác, biến b không còn tồn tại nữa.

3. Lưu ý về đặt trùng tên biến

Giả sử trong scope con có một biến a khác, gán giá trị = 10. Lúc này, khi truy xuất "a", ta sẽ nhận được 10. Vậy khi ra khỏi scope con, "a" bằng mấy?

Rất ngạc nhiên khi "a" trở về bằng 0. Lý do là 2 biến "a" này hoàn toàn khác nhau, chỉ có điều là chúng trùng tên, nên khi gọi trong scope con, chương trình mặc định gọi "a" của scope hiện tại. Mọi tác động lên "a nhỏ" này không làm ảnh hưởng đến "a lớn". Tuy nhiên

trong lập trình, **tuyệt đối phải tránh** những trường hợp đặt trùng tên như thế này. Việc đặt trùng tên rất dễ gây nhầm lẫn cho người đọc code, dẫn đến nhầm lẫn trong quá trình debug khiến chương trình rất khó bảo trì.

4. Comment

Trong quá trình lập trình, việc để lại những comment để các developer khác hiểu ý mình là chuyện bình thường. Comment không làm ảnh hưởng đến quá trình biên dịch code. Trong C++, comment có 2 dạng:

- Comment 1 dòng: nội dung theo sau dấu //

Ví dụ: `cout << abc; // this code do stuffs`

- Comment nhiều dòng: nội dung nằm trong cặp dấu /* và */

Ví dụ: `cout << abc; /* this code do stuffs
and this code do somethings */`

Ghi nhớ

- Scope quyết định không gian tồn tại và truy xuất được của 1 biến.
- Scope con lấn át scope mẹ, tuy nhiên có không gian tồn tại & truy xuất nhỏ hơn.
- Comment không làm ảnh hưởng tới quá trình biên dịch code.

Bài 2. Biến và Kiểu dữ liệu

1. Khái niệm biến

Trong ngôn ngữ lập trình Pascal, chúng ta đã làm quen với nhiều kiểu dữ liệu: integer, real, string, boolean,... Đến với C++ chúng ta sẽ tìm hiểu một số kiểu dữ liệu thông dụng. Nhưng trước đó, chúng ta sẽ cùng xem cách để khai báo 1 biến.

Ở Pascal, việc khai báo biến thường được thực hiện ở đầu chương trình, và biến có thể được truy xuất tại mọi điểm trong thân chương trình. Còn ở C++, biến được khai báo như sau:

```
int a;
```

`int` ở đây là viết tắt của Integer, kiểu dữ liệu số nguyên có kích thước 4 bytes. “a” là tên biến, việc đặt tên biến tuân theo các quy tắc đặt tên biến.

Biến trong C++ có thể được khai báo ở **bất kỳ đâu trong chương trình**, và biến được khai báo ở tầm vực nào thì sẽ chỉ tồn tại ở tầm vực đó.

C++ là một ngôn ngữ **phân biệt Hoa – thường** (Case-sensitive), cho nên biến “Example” sẽ khác hoàn toàn biến “example”.

2. Gán giá trị cho biến

Ở Pascal, việc gán giá trị cho biến được thực hiện thông qua toán tử `:=`, còn trong C++, phép gán thực hiện bởi dấu `=` như sau:

```
int c = 20;  
bool foo = true;
```

3. Các kiểu dữ liệu

C++ gồm 2 kiểu dữ liệu:

- **Kiểu dữ liệu ngầm định:** là những kiểu dữ liệu được định nghĩa sẵn và có thể được dùng để khai báo biến ngay lập tức. Ví dụ: int, char, float, bool,...
- **Kiểu dữ liệu người dùng khai báo:** do người dùng tự định nghĩa. Như là, khai báo 1 class hoặc 1 struct.

Một số kiểu dữ liệu mặc định của C++:

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Kích thước các kiểu dữ liệu:

Type	Typical Size (in bytes)	Typical Range
char	1	-127 to 127 or 0 to 255
unsigned char	1	0 to 255
signed char	1	-127 to 127
int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
signed int	4	-2147483648 to 2147483647
short int	2	-32768 to 32767
unsigned short int	2	0 to 65,535
signed short int	4	-32768 to 32767
long int	4	-2,147,483,648 to 2,147,483,647
signed long int	4	same as long int
unsigned long int	4	0 to 4,294,967,295
float	4	+/- 3.4e +/- 38 (~ 7 digits)
double	8	+/- 1.7e +/- 308 (~ 15 digits)
long double	8 or 12	+/- 1.7e +/- 308 (~ 15 digits)
wchar_t	2 or 4	1 wide character

Trong đó, unsigned là “không dấu” (tức là số dương), short là “ngắn” (ngắn hơn kiểu dữ liệu chuẩn), long là “dài” (dài hơn kiểu dữ liệu chuẩn).

Ghi nhớ

- Biến trong C++ có thể được khai báo ở bất kỳ đâu trong chương trình, và biến được khai báo ở tầm vực nào thì sẽ chỉ tồn tại ở tầm vực đó.
- Trong C++, phép gán thực hiện bởi dấu =.

Bài 3. Định dạng dữ liệu xuất ra

1. Newline và Tab

Đây là 2 trong số những ký tự đặc biệt, có tác dụng tạm ngưng dòng nhập, hoặc nói đơn giản hơn, là để xuống dòng và tab.

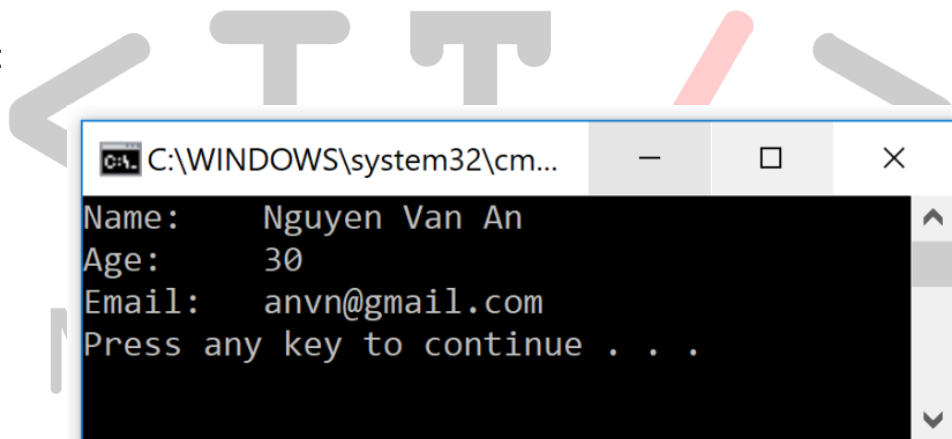
`\n` là xuống dòng (newline)

`\t` là tab

Ví dụ:

```
cout << "Name:\t Nguyen Van An\n";  
cout << "Age:\t 30\n";  
cout << "Email:\t anvn@gmail.com\n";
```

Kết quả:



2. Thư viện <iomanip>

Thư viện <iomanip> (In/Out Manipulators) là một thư viện giúp định dạng dữ liệu xuất ra màn hình. Để sử dụng, chèn thêm `#include <iomanip>` ở phần chỉ thị tiền xử lý.

- **setw(n):** đặt 1 miền rộng n khoảng trống để chứa các kí tự. Mặc định, setw(n) sẽ căn lề phải cho dữ liệu.

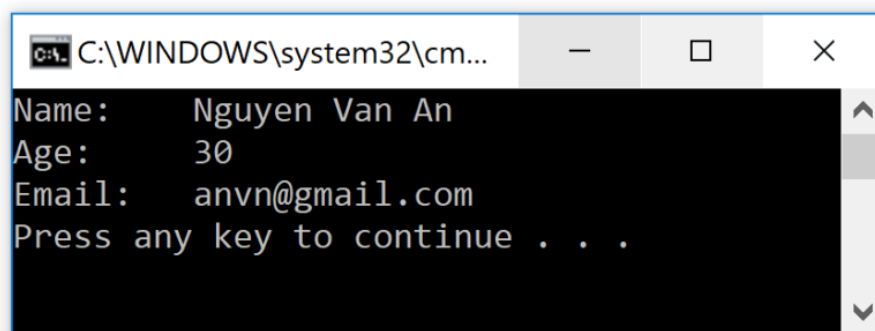
Dùng **left**, **right**, **internal** để chọn phía căn lề theo ý muốn.

Cú pháp: `cout << left << setw(8) << "Text";`

Ví dụ:

```
cout << left << setw(8) << "Name:" << "Nguyen Van  
An\n";  
cout << left << setw(8) << "Age:" << "30\n";  
cout << left << setw(8) << "Email:" <<  
"anvn@gmail.com\n";
```

Kết quả:

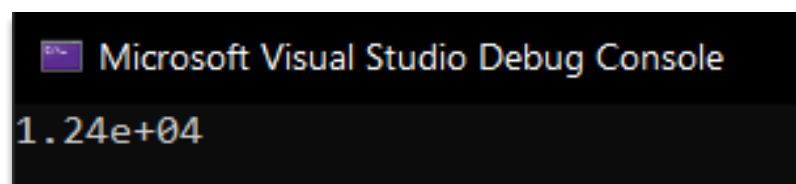


- **setprecision(n):** đặt số lượng chữ số có nghĩa cho số cần hiển thị. Đặt $n = 0$ để reset cài đặt này.

Ví dụ:

```
cout << setprecision(3) << 12356.4 << endl;
```

Kết quả:



Bài 4. Toán tử và ép kiểu dữ liệu

1. Toán tử (Operators)

Toán tử là một khái niệm phổ biến. Nó thường được hiểu như là 1 hàm. Có nhiều loại toán tử: toán tử số học, toán tử bit, toán tử logic và toán tử gán.

- Toán tử số học (Arithmetic operators): +, -, *, /, %
- Toán tử bit (Bitwise operators): ^, ~, &, |, >>, <<
- Toán tử so sánh và logic (Logic operators): !, &&, ||, >, <, ==, !=
- Toán tử gán (Assignment): =

Toán tử số học

Ký hiệu	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia*
%	Modulo

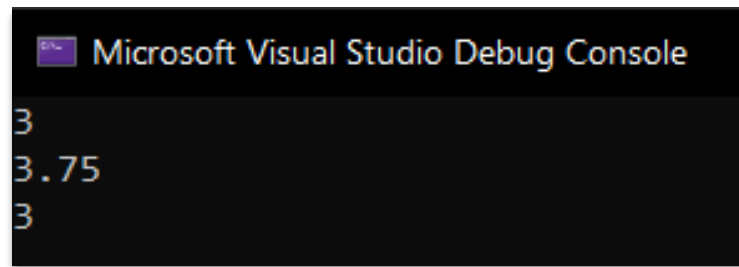
Ví dụ:

```
#include <iostream>
using namespace std;
int main()
{
    cout << 15 / 4 << endl;
    cout << 15 / 4.0 << endl;
    cout << 15 % 4 << endl;
    return 0;
}
```

Tương đương Pascal:

```
15 div 4
15 / 4
15 mod 4
```


Kết quả:



```
Microsoft Visual Studio Debug Console
3
3.75
3
```

Giải thích: phép chia (/) trong C++ nếu làm việc với 2 số nguyên (int) sẽ có tác dụng như phép div trong Pascal. Để giải quyết vấn đề này, chỉ cần để 1 trong 2 toán hạng là số thực (float) thì phép chia sẽ hoạt động như phép chia bình thường.

Toán tử so sánh và logic

Ký hiệu	Ý nghĩa
!	Not
&&	And
	Or
<	Bé hơn
>	Lớn hơn
==	Bằng
!=	Không bằng

Trong số tất cả các toán tử trên, chỉ có **!** là toán tử 1 ngôi (chỉ cần 1 toán hạng), còn lại đều là toán tử 2 ngôi. Các toán tử khi kết hợp với các toán hạng đều trả về kết quả **true** hoặc **false**. Ví dụ:

```
cout << (3 > 5);    // màn hình in ra số 0 (tức là false)
true && true && false // = false
6 != (3*2)          // = false
```

2. Ép kiểu dữ liệu

Là việc chuyển dữ liệu từ kiểu này sang kiểu khác.

Cú pháp: (<target type>) <expression>

Ví dụ:

- `int a = (int)3.9583;`
- `float x = (float)a + 0.5f;`
- `double y = (double)x * (double)a; // y = x * a; is fine`

Thực chất, phép chia thứ 2 ở phần 1 cũng là 1 cách ép kiểu ngầm định, khi mà 4.0 được xác định là float, thì toàn bộ phép tính cũng được ép kiểu về float.

3. Toán tử kết hợp

Operator	Example	Equivalent expression
<code>+=</code>	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>a %= b</code>	<code>a = a % b</code>

4. PreFix và PostFix

Giúp tăng hoặc giảm giá trị của biến 1 đơn vị.

Tăng

prefix: `++a`

postfix: `a++`

Giảm

prefix: `--a`

postfix: `a--`

Prefix sẽ làm thay đổi giá trị **trước khi** trả về giá trị, còn Postfix trả về giá trị cũ trước, **sau đó** mới tăng giá trị, và rồi mới trả về giá trị mới.

Trên thực tế, người ta thường sử dụng Prefix do nó có hiệu suất cao hơn, vì nó không phải trả về giá trị cũ trước khi thay đổi.

Bài 5. Macro

1. Macro định nghĩa

- #define/#undef: là chỉ thị tiền xử lý.
- Mở rộng thông qua 1 dòng code duy nhất.
- Không có dấu “;” ở cuối dòng
- Nếu macro gồm nhiều dòng, cuối mỗi dòng có dấu “\”

Định nghĩa 1 hằng

Cú pháp **#define <identifier> <replacement>**

- #define MAX_LENGTH 50
- #define MY_STRING “This is a constant string”
- #define pi_2 3.14159/2
- #define pi_2 1.570785

Định nghĩa một Macro sẽ làm cho cú pháp được định nghĩa <identifier> mang đúng giá trị của <replacement>

Ví dụ:

```
cout << MAX_LENGTH << endl;
```

Kết quả: màn hình in ra 50.

Hiểu 1 cách đơn giản, Macro sẽ thay thế chính xác và y xì những gì ta định nghĩa vào trong dòng code. Tức là dòng code sau đây sẽ có cùng chức năng với dòng code trên:

```
cout << 50 << endl;
```

Macro không quan tâm thứ bạn định nghĩa là số nguyên, ký tự hay true/false, nó chỉ đơn giản là copy và paste.

2. Macro hàm

Cũng là Macro, nhưng lần này, nó sao chép thông minh hơn.

Cú pháp: **#define tenham(a, b) <replacement with a, b>**

- #define subtract(a, b) a - b

3. Lưu ý khi sử dụng Macro

- Bản chất của Macro là **copy và paste**. Giả sử với định nghĩa hàm sub trên, ta gọi hàm `subtract(8, 5)`. Kết quả trả ra là 3, đúng ✓
- Giờ ta cho nó vào phép tính `3 * subtract(8, 5)`. Cái chúng ta đang mong chờ là `3 * 3 = 9`, nhưng khi thực hiện copy, Macro “bê” y xì mọi thứ vào. Thế là ta có `3 * 8 - 5 = 19`, sai ✗
- Bây giờ ta khắc phục bằng cách định nghĩa lại:

```
#define subtract(a, b) (a - b)
```

Giờ thì `3 * (8 - 5)` nghe có vẻ đúng. Giờ ta gọi phép tính `subtract(8, 2 + 3)`. Cái chúng ta đang mong chờ là `8 - (2 + 3) = 3`. Tuy nhiên, Macro lại một lần nữa copy và paste. Giờ thì phép toán trở thành `8 - 2 + 3 = 9`, lại sai ✗

Chỉ khi ta định nghĩa lại 1 lần nữa:

```
#define subtract(a, b) ((a) - (b))
```

thì Macro của chúng ta mới chính xác.

4. Nếu khó nhằn như thế, vậy dùng Macro để làm gì?

Ưu điểm của Macro là nó không hề chiếm 1 ô nhớ nào trên bộ nhớ máy tính. Tất cả các hoạt động thay thế, điền khuyết đều được diễn ra ở bước tiền xử lý trước khi dịch, do đó, Macro giúp tiết kiệm bộ nhớ rất nhiều.

Một ưu điểm nữa, là giúp hạn chế việc quá tải hàm (function overloading¹). Đại khái là, mỗi hàm chỉ có thể chạy với một kiểu dữ liệu nhất định. Muốn hàm làm việc với kiểu dữ liệu khác, người dùng phải overload hàm đó bằng 1 hàm cùng tên, khác kiểu dữ liệu. Nhưng có rất nhiều kiểu dữ liệu trong C++. Chúng ta không thể overload tất cả các kiểu dữ liệu cho mỗi hàm được. Lúc đó, Macro thể hiện sức mạnh của mình.

Lấy ví dụ

```
#define swap(type, x, y) {type tmp = x; x = y; y = tmp;}
```

Gọi `swap(int, x, y)`, hàm chạy với int. Gọi `swap(char, x, y)`, hàm chạy với char. Gọi `swap(bool, x, y)`, hàm chạy với bool. Không cần overload hàm.

¹ Tránh nhầm với ghi đè (override)
facebook.com/itclub.nhh

Bài 6. Đọc và ghi file cơ bản

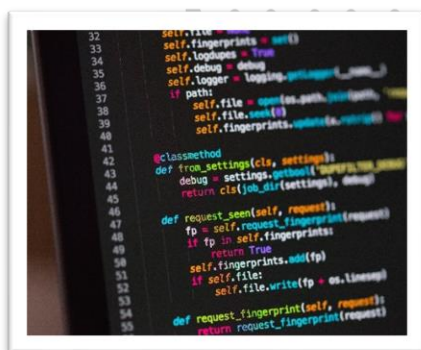
1. Đọc và ghi file

Trong quá trình làm việc, chúng ta sẽ phải đọc và ghi thông tin từ file bên ngoài (thường là file .txt). Các bước để đọc và ghi một file bao gồm:

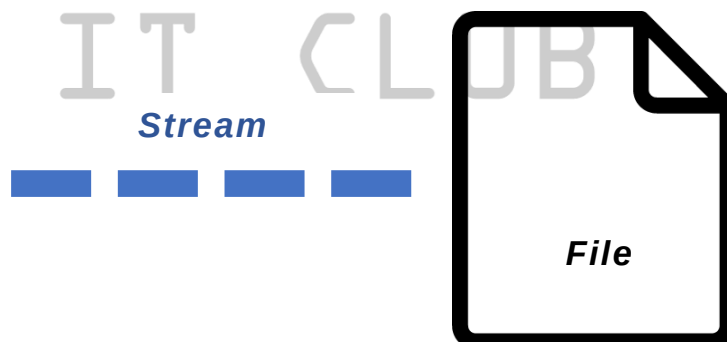
1. Include thư viện <fstream>
2. Tạo dòng nhập xuất (stream) (input, output, cả hai):
 - `ifstream` myfile; (để đọc file)
 - `ofstream` myfile; (để ghi lên file)
 - `fstream` myfile; (để đọc và ghi file)
3. Mở file: `myfile.open("filename");`
4. Đọc và ghi file
5. Đóng file: `myfile.close();`

2. Dòng nhập xuất (stream) là gì?

Stream là 1 khái niệm trừu tượng mô tả 1 thiết bị mà trên đó, việc nhập xuất dữ liệu được thực hiện. Ví dụ, file stream là các *object* trong C++ có nhiệm vụ kết nối và tương tác với file; Một khi stream được dùng để mở 1 file, bất kỳ thao tác nhập, xuất trên stream đều trực tiếp ảnh hưởng đến file đó.



Program



Một số đoạn code mẫu:

1. In nội dung của 1 file text:

```
// print the content of a text file.
```

```
#include <iostream> // std::cout
#include <fstream>   // std::ifstream
using namespace std;
int main() {
    ifstream ifs;
    ifs.open("example.csv", ifstream::in); /*ifstream::in to show the
    stream is for reading, 'if' stands for 'input file'*/
    char c = ifs.get();
    while (ifs.good()) {
        cout << c;
        c = ifs.get();
    }
    ifs.close();
    return 0;
}
```

2. Ghi nội dung vào 1 file text:

```
#include <fstream>
using namespace std; // std::ofstream
int main() {
    ofstream ofs("test.txt", ofstream::out); /*ofstream::out to show
    the stream is for writing, 'of' stands for 'output file'*/
    ofs << "Welcome to NHH IT Club!\nLecture 6: File IO.\n";
    ofs.close();
    return 0;
}
```

Ghi nhớ

- Việc thao tác với file trong C++ được thực hiện thông qua stream.

Bài 7. If – Else và Switch

1. Toán tử so sánh

Operator	Meaning
"=="	Equal to
"<"	Less than
">"	Greater than
"<="	Less than or equal to
">="	Greater than or equal to
"!="	Not equal to

Nếu 2 toán hạng của mỗi toán tử thỏa điều kiện và tính chất, toán tử sẽ trả về **true**. Ngược lại, toán tử trả về **false**. Ví dụ:

```
8 == (3 + 5) : true
2 <= 6      : true
6 != 3*2    : false
```

Lưu ý, toán tử **so sánh bằng** trong C++ là **==** chứ không phải là **=** như Pascal.

2. Toán tử Logic

Operator	Meaning	Behavior
&&	and	If both inputs are true the outcome of the operation is true; otherwise false
	or	If both inputs are false the outcome of the operation is false; otherwise true
!	not	Negates the logical condition

Ví dụ:

```
true && true : true
false || true : true
! true      : false
```

3. Câu lệnh If - Else

Công dụng: thực hiện 1 lệnh hoặc 1 chuỗi lệnh nếu thỏa điều kiện được cho.

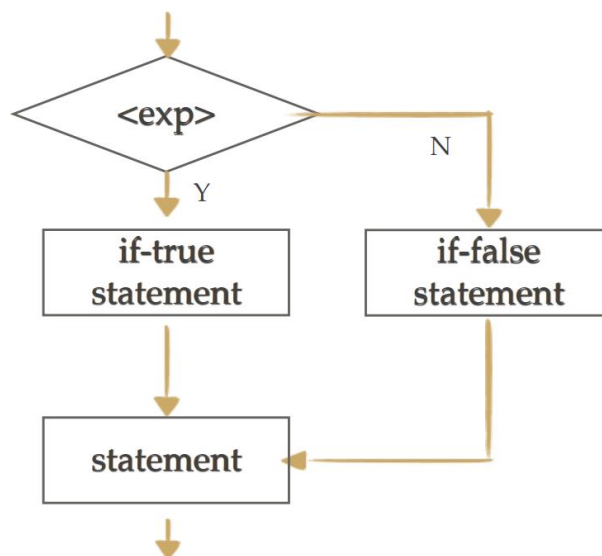
- **if** (<conditional expression>) <if-true statement>;
 else <if-false statement>;
- **if** (<conditional expression>) {
 <if-true statements>
}
 else {
 <if-false statements>
}

Chuỗi lệnh phải nằm trong cặp dấu { }, và xem như nằm trong 1 scope con của scope chứa lệnh If - Else. Một lệnh đơn lẻ thì không cần { }.

If – Else lồng (nested):

```
if (<exp 1>) <statement 1>;
else if (<exp 2>) <statement 2>;
else if (<exp 3>) <statement 3>;
else <statement 4>;
```

Flowchart của câu lệnh **If – Else**:



4. “Toán tử 3 ngôi”

Trong C++, chỉ có 1 toán tử duy nhất có 3 ngôi (3 toán hạng), đó chính là toán tử có điều kiện **?:**:

Về bản chất, toán tử này là 1 lệnh if – else nhưng chỉ áp dụng cho các biểu thức.

Cú pháp:

<expression> ? <if-true expression> : <if-false expression>

Ví dụ:

A == B ? A -= B : B += A;

“Nếu A bằng B, thì A -= B. Nếu không, B += A”

Do chỉ có 1 toán tử duy nhất có 3 ngôi trong C++, nên người ta thường gọi toán tử có điều kiện là toán tử 3 ngôi.

5. Switch

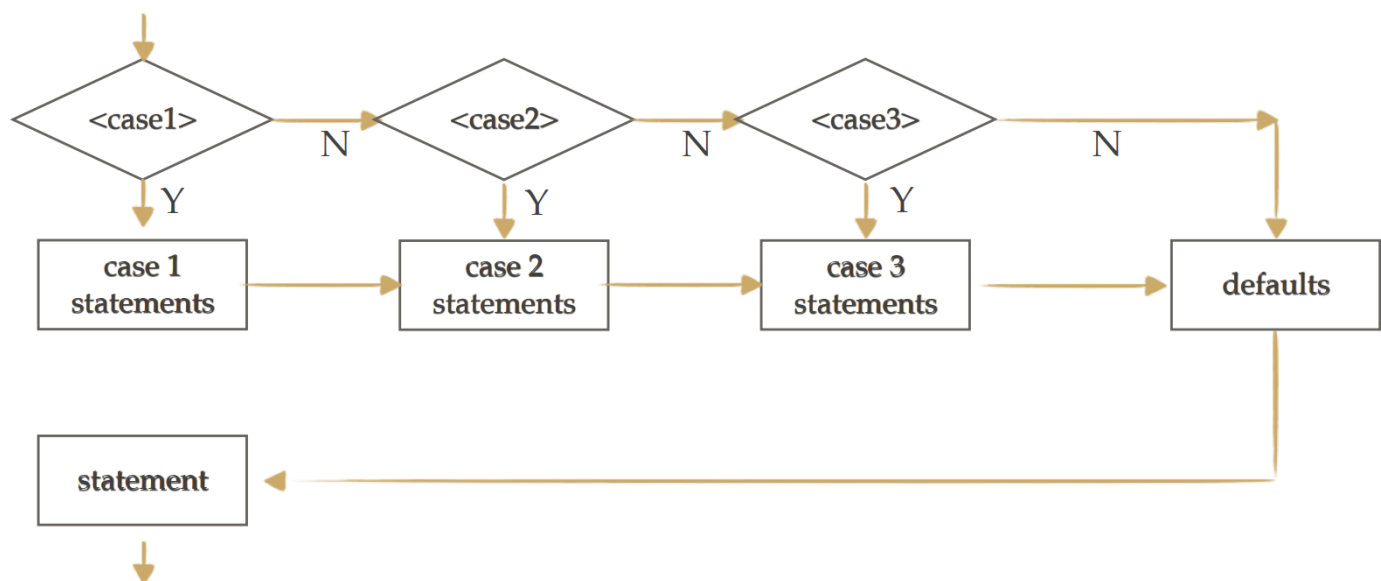
Để tránh các If lồng nhau với cùng kiểu điều kiện, người ta dùng câu lệnh Switch. Switch là 1 cách tiện lợi để viết câu lệnh nhiều trường hợp. Cú pháp như sau:

```
switch(<exp>) {
    case <value 1>: <statements>; break;
    case <value 2>: <statements>; break;
    ...
    case <value N>: <statements>; break;
    default: <statements>; //optional
}
```

Một số đặc điểm của câu lệnh Switch:

- Các **<value>** của nó phải là 1 giá trị xác định.
- Các **<statements>** NÊN được đặt trong 1 scope.
- Phải có **break** cho từng trường hợp. Nếu không, tất cả các statements nằm sau trường hợp thỏa đều sẽ được thực thi (thuật ngữ gọi là **fall-through**)
- Tuy nhiên, fall-through vẫn đôi khi được lợi dụng có chủ đích.

Flowchart cho câu lệnh Switch:



Ví dụ:

```

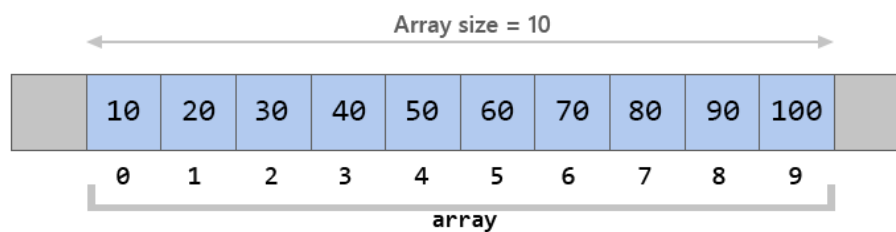
char grade;
switch(grade) {
    case 'A' :
        cout << "Excellent!" << endl;
        break;
    case 'B' :    //intentionally fall-through
    case 'C' :
        cout << "Well done" << endl;
        break;
    case 'D' :
        cout << "You passed" << endl;
        break;
    case 'F' :
        cout << "Better try again" << endl;
        break;
    default :    //optional
        cout << "Invalid grade" << endl;
}
  
```

Bài 8. Cấu trúc lặp (Iteration)

1. Tại sao cần vòng lặp?

Có nhiều nguyên nhân, mà 2 nguyên nhân chính đó là:

1. Chờ đợi một thứ gì xảy ra hoặc một điều kiện nào đó được thỏa.
2. Xử lý trên nhiều (1 chuỗi) đối tượng:
 - List, Array
 - String



2. Vòng lặp While

Cú pháp:

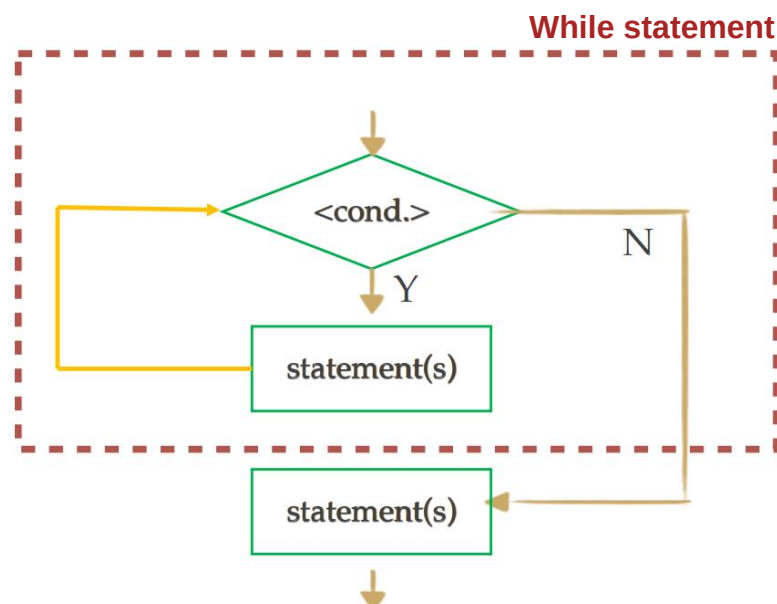
```
while (<condition>) <statement>;
```

hoặc

```
while (<condition>) {
    <statements>;
}
```

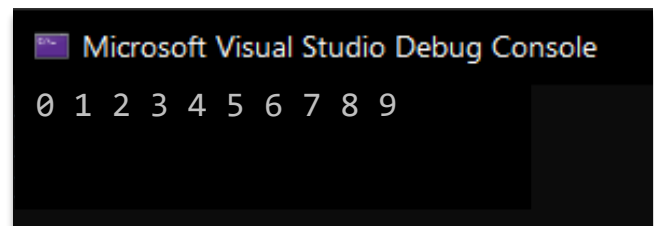
Công dụng: khi điều kiện không còn thỏa, ngừng lặp.

Flowchart của While:



Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    int counter = 0;
    while (counter < 10) {
        cout << counter << " ";
        counter++;
    }
    cout << endl;
    return 0;
}
```

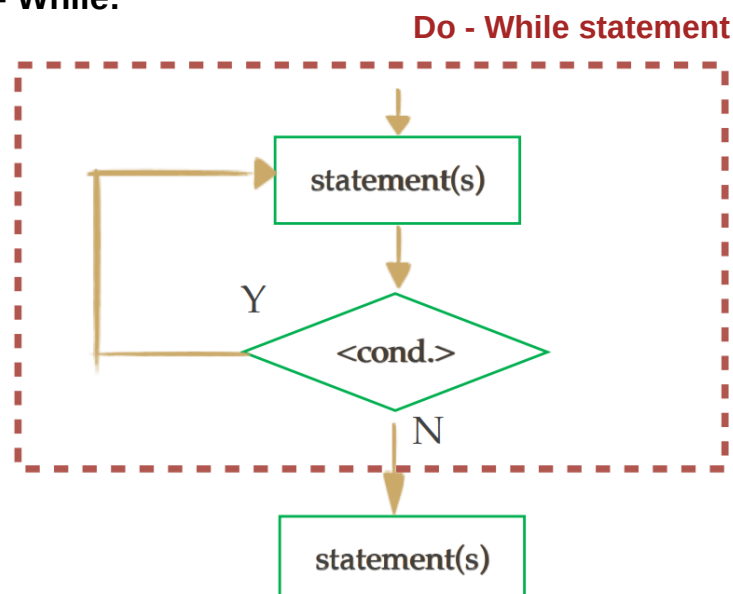


3. Vòng lặp Do – While

Cú pháp: `do <statement> while (<condition>);`
 hoặc `do {`
 `<statements>;`
 `} while (<condition>);`

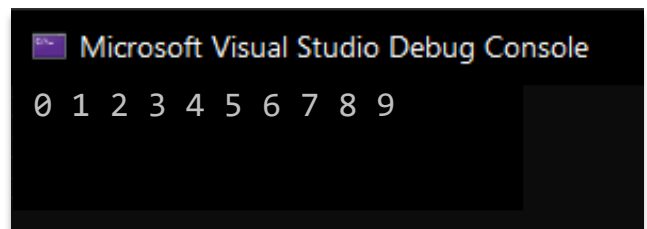
Công dụng: lặp lại cho đến khi điều kiện không còn thỏa. Khác biệt so với While chính là số lần thực hiện statement tối thiểu của While là 0, còn Do – While là 1.

Flowchart của Do - While:



Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    int counter = 0;
    do {
        cout << counter << " ";
        counter++;
    } while (counter < 10);
    cout << endl;
    return 0;
}
```



Lưu ý khi dùng While và Do – While:

- Hãy nhớ khởi trị cho biến trong điều kiện (condition) trước khi vào vòng lặp (ít nhất bạn cũng biết chuyện gì xảy ra khi kiểm tra các điều kiện).
- Không được quên điều kiện dừng, nếu không, sẽ xuất hiện **vòng lặp vô hạn**.
- Cần thận với các biến đếm.

4. Vòng lặp For

For cũng là 1 câu lệnh lặp. Điểm đặc biệt là, nó kết hợp tất cả các yếu tố sau vào trong cùng 1 câu lệnh:

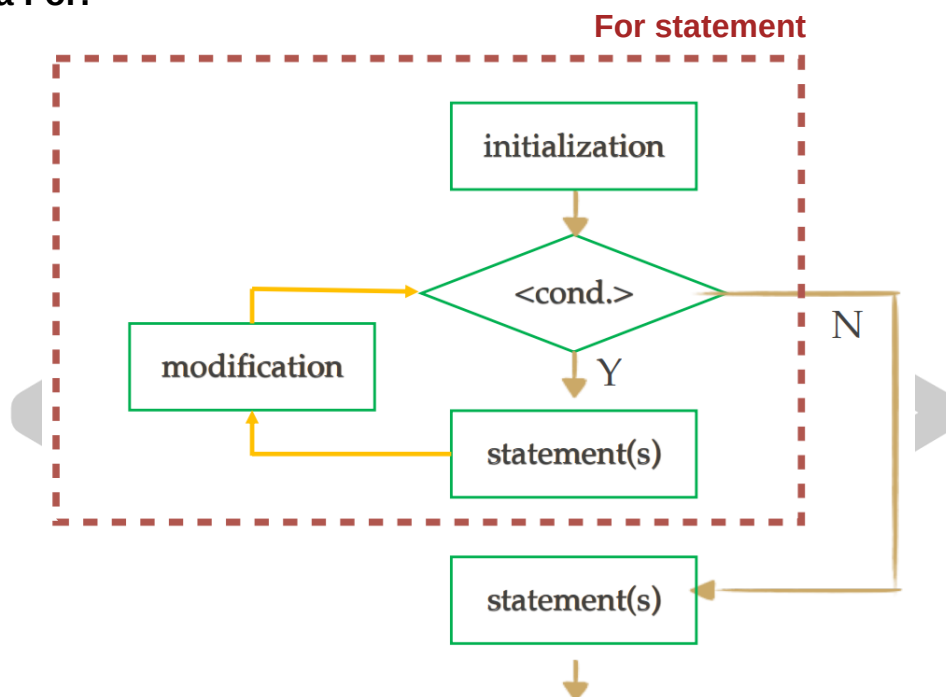
- Khai báo & khởi trị biến đếm.
- Điều chỉnh biến đếm.
- Kiểm tra điều kiện dừng.

Cú pháp:

```
for (<initialization>; <condition>; <modification>) <statement>;
hoặc for (<initialization>; <condition>; <modification>) {
    <statements>;
}
```

Giải thích:

- **Initialization:** đặt giá trị cho biến đếm.
 - ❖ Khai báo 1 hoặc nhiều biến đếm (cùng loại) và khởi trị cùng lúc.
 - ❖ Khởi trị nhiều biến cùng lúc nếu cần.
- **Condition:** Một biểu thức boolean mà được đánh giá lại vào mỗi lần lặp.
- **Modification:** Thay đổi giá trị biến đếm sau mỗi lần lặp.

Flowchart của For:**Ví dụ:**

```

#include <iostream>
using namespace std;
int main() {
    for (int i = 0; i < 10; i++)
        cout << i << " ";
    cout << endl;
    return 0;
}

```



Chú ý: `initialization` và `modification` có thể cùng lúc chứa nhiều dòng lệnh, ngăn cách bởi các dấu phẩy.

```
#include <iostream>
using namespace std;
int main() {
    int i, j;
    for (i = 5, j = 10; i + j < 20; i++, j++) {
        cout << "i + j = " << (i + j) << '\n';
    }
    return 0;
}
```

5. Thoát khỏi vòng lặp

Để thoát ngang (force exit) khỏi vòng lặp mà không cần chờ tới điều kiện dừng, ta có 2 cách phổ biến sau:

break: kết thúc vòng lặp, và thực hiện câu lệnh đầu tiên NGAY SAU vòng lặp.

continue: ép cho vòng lặp tiếp theo diễn ra ngay lập tức.

Ví dụ:

BREAK

```
#include <iostream>
using namespace std;
int main() {
    int count;
    for (count = 1; count <= 10; count++) {
        if (count == 5)
            break;
        cout << count << " ";
    }
    cout << "\nBroke out at count = " << count << endl;
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
1 2 3 4
Broke out at count = 5
```

CONTINUE

```
#include <iostream>
using namespace std;
int main() {
    int count;
    for (count = 1; count <= 10; count++) {
        if (count == 5)
            continue;
        cout << count << " ";
    }
    cout << "\nUsed continue to skip printing 5" << endl;
    return 0;
}
```

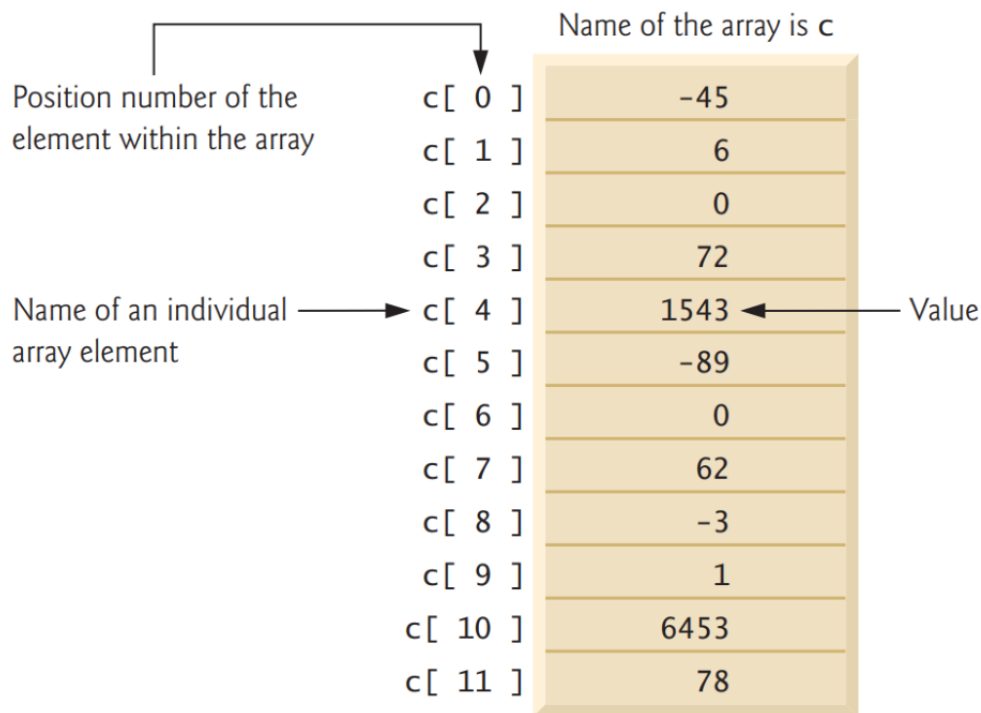
Microsoft Visual Studio Debug Console

1 2 3 4 6 7 8 9 10
Used continue to skip printing 5

Ghi nhớ

- Có 3 loại vòng lặp: While, Do – While và For
- While và Do – While thường dùng khi vòng lặp dựa trên điều kiện đúng/sai, còn For thường dùng cho vòng lặp với quy tắc đã biết trước.

Bài 9. Array



1. Khái niệm và tính chất của Array (Mảng)

Array (Mảng) là 1 trong 3 kiểu dữ liệu có cấu trúc mà ta sẽ học trong tài liệu này. Nó bao gồm 1 dãy kề nhau các ô nhớ chứa cùng 1 kiểu dữ liệu. Chỉ số (index) của một Array chứa n phần tử bắt đầu từ 0, và kết thúc ở $n - 1$.

Array trong C++ không có tính chất nối đuôi, tức là không thể truy xuất phần tử cuối cùng bằng cách lấy phần tử thứ -1 .

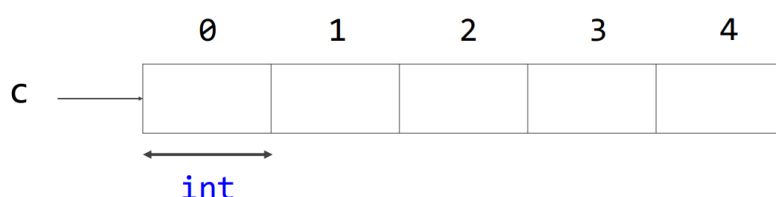
2. Khai báo 1 mảng

Để khai báo 1 mảng, ta sử dụng cú pháp:

```
<type> arrayName [ arraySize ];
```

Với **<type>** là kiểu dữ liệu của các phần tử trong mảng, và **arraySize** là số phần tử có trong mảng.

Ví dụ: `int c[5];`



3. Khởi trị cho mảng

- Từng phần tử một:

Ví dụ: `c[4] = 10;`

- Sử dụng vòng lặp:

Ví dụ: `for (int i = 0; i < 5; i++) c[i] = 0;`

- Khai báo bằng 1 danh sách giá trị:

Ví dụ: `int c[5] = { 10, 20, 30, 40, 50 };`

4. Truy xuất giá trị 1 phần tử trong mảng

Giá trị của bất kỳ một phần tử nào trong mảng cũng có thể được truy xuất như một biến (variable) bất kỳ. Cú pháp là:

`name[index]`

Ví dụ 1:

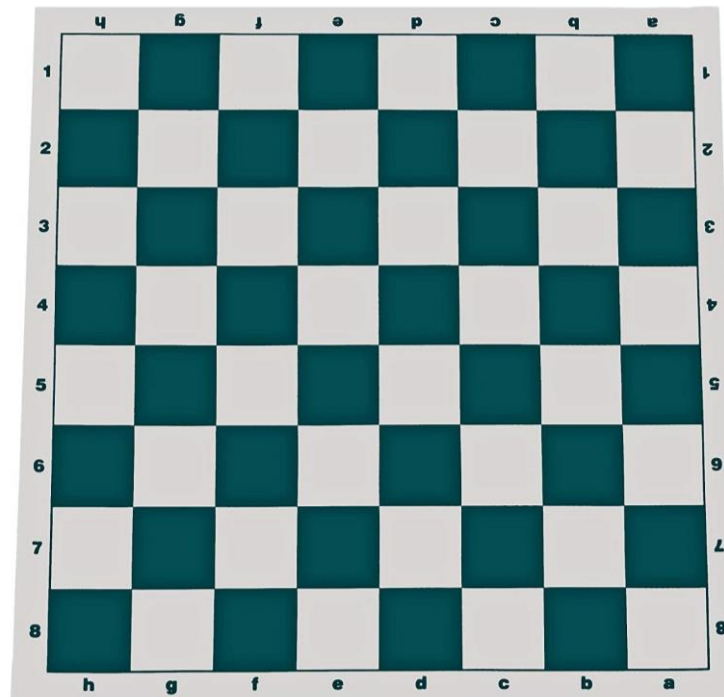
```
int a = 2;
c[0] = a;
c[a] = 75;
b = c[a + 2];
c[c[a]] = c[2] + 5;
```

Ví dụ 2:

```
#include<iostream>
#include<iomanip>
using namespace std;
int main() {
    int c[10];
    for (int i = 0; i < 10; i++) c[i] = 0;
    cout << "Element" << setw(13) << "Value" << endl;
    for (int i = 0; i < 10; i++)
        cout << setw(7) << i << setw(13) << c[i] << endl;
    return 0;
}
```

5. Mảng 2 chiều

Mảng 2 chiều về bản chất là mảng 1 chiều, với mỗi phần tử là một mảng 1 chiều khác.

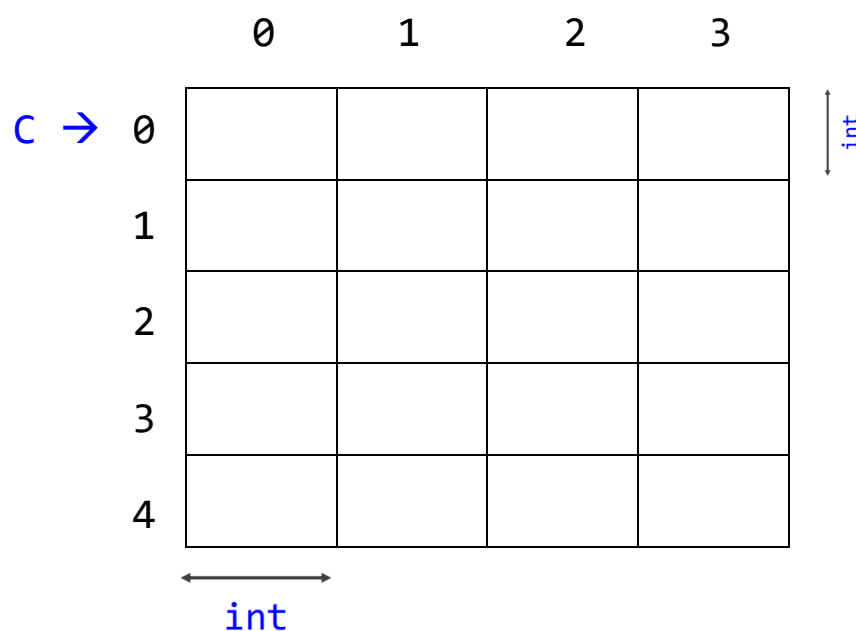


Bàn cờ vua là minh họa cho mảng 2 chiều, với mỗi phần tử trong chiều đầu tiên là một hàng ngang trên bàn cờ. Có cả những mảng 3 chiều, 4 chiều,...

Để khai báo một mảng 2 chiều, ta sử dụng cú pháp:

```
<type> arrayName [dimension1Size][dimension2Size];
```

Ví dụ: `int c[5][4];`



Bài 10. Chuỗi

1. Chuỗi kiểu C (thư viện <string.h>)

Chuỗi (string) là kiểu thứ 2 trong 3 kiểu dữ liệu có cấu trúc mà ta sẽ học trong tài liệu này. Chuỗi kiểu C (ngôn ngữ C) thực chất là 1 mảng (Array) các kí tự kiểu char, với kí tự cuối cùng luôn là kí tự `\0`. Để chứa nội dung "Hello" gồm 5 chữ cái, chuỗi này yêu cầu 1 mảng `char[]` gồm ít nhất 6 ký tự:

	0	1	2	3	4	5
c →	H	e	l	l	o	\0

Khai báo & khởi trị cho chuỗi kiểu C

- Khai báo tường minh kích thước:

Ví dụ: `char c[12] = "Hello World";`

- Khai báo không tường minh kích thước:

Ví dụ: `char c[] = "Hello World";`

Sau này khi học tới **con trỏ (pointer)**, chúng ta sẽ tìm hiểu 1 cách khai báo nữa, là khai báo bằng con trỏ kiểu `char*`.

Mỗi phần tử `char` có thể lấy giá trị số nguyên theo bảng mã ASCII. Ví dụ khi khai báo:

`char c[4] = 79 //is 'O'`

Thì chuỗi C sẽ trở thành: `Hello World`

Về các hàm trong thư viện <string.h>: các bạn có thể tham khảo tại cppreference.com

2. Chuỗi kiểu C++ (thư viện <string>)

Thư viện <string.h> rất mạnh và đầy đủ công cụ để các bạn sử dụng nhằm thao tác với chuỗi. Tuy vậy, việc xử lý mảng kí tự vẫn còn nhiều khó khăn. Một ví dụ điển hình là việc bạn thực hiện nối chuỗi kí tự bằng hàm `strcat`. Bạn luôn phải để tâm đến việc số lượng ô nhớ mà bạn đã cấp phát cho mảng kí tự có đủ để chứa thêm chuỗi kí tự được nối vào không. Hay là khi bạn khai báo một mảng kí tự, bạn cũng phải đặt ra câu hỏi: Liệu bao nhiêu ô nhớ là đủ? Và bạn phải luôn đặt kí tự `'\0'` tại vị trí kết thúc chuỗi kí tự... Quá nhiều thứ khiến bạn phải để tâm.

Bây giờ, chúng tôi sẽ giới thiệu với các bạn về kiểu dữ liệu `string` được định nghĩa trong thư viện `<string>` của ngôn ngữ C++ (các bạn đừng nhầm lẫn giữa thư viện `<string.h>` của ngôn ngữ C với thư viện `<string>` của ngôn ngữ C++, hai thư viện này hoàn toàn riêng biệt).

Lưu ý: Tương tự như cách chúng ta khai báo biến thông thường, kiểu dữ liệu sẽ được dùng trong bài học này là `string` (Các bạn cần include thư viện `<string>` vào trước khi sử dụng). Lớp `string` cũng được đặt trong **namespace** `std` nên dòng lệnh `using namespace std` là cần thiết.

Khai báo & khởi trị cho chuỗi kiểu C++

- Khai báo một string:

Ví dụ: `string stringName;`

- Khởi trị cho string:

Ví dụ: `stringName = "Hello World";`

Truy xuất phần tử trong string

Cũng tương tự như việc truy cập phần tử trong mảng kí tự, chúng ta sử dụng cặp dấu ngoặc vuông và truyền vào một giá trị số nguyên đại diện cho chỉ số của phần tử cần truy xuất.

```
stringName[3]
```

Bên cạnh đó, lớp `string` cũng đã định nghĩa lại rất nhiều phương thức và toán tử, trong đó có toán tử nối chuỗi kí tự `+=` giúp chúng ta tiết kiệm thời gian viết code hơn.

```
string str = "";  
str += "Use \"+=\" operator ";  
str += "to append string";  
// "Use "+=" operator to append string"
```

Bài 11. Structure

Books
<input type="checkbox"/> Title
<input type="checkbox"/> Author
<input type="checkbox"/> Subject
<input type="checkbox"/> Book ID

1. Khái niệm và tính chất của Structure

Structure (sau đây gọi là **struct**) là một **kiểu dữ liệu người dùng tự định nghĩa** (xem lại Bài 2), cho phép người dùng kết hợp nhiều kiểu dữ liệu vào thành 1 kiểu duy nhất.

Struct thường được dùng để thể hiện 1 bản ghi (record).

Struct là kiểu cuối cùng trong 3 kiểu dữ liệu có cấu trúc mà ta sẽ học trong tài liệu này.

2. Định nghĩa một Struct

```
struct structure tag {  
    member definition;  
    member definition;  
    ...  
    member definition;  
};
```

Trong đó, **structure tag** là tên mà bạn đặt cho kiểu dữ liệu này. Các **member definition** là khai báo các thành phần của struct đó.

Ví dụ:

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
};
```


3. Khai báo 1 đối tượng thuộc 1 struct

```
Books book1;
```

4. Truy cập dữ liệu của 1 đối tượng thuộc 1 struct

Dùng toán tử truy xuất thành viên (member accessing operator) .

```
Books Book1;           // Declare Book1
                        // book 1 specification

strcpy_s(Book1.title, "Learn C++ Programming");
strcpy_s(Book1.author, "Chand Miyan");
strcpy_s(Book1.subject, "C++ Programming");
Book1.book_id = 6495407;
```

Ghi nhớ

- Struct là một cách cực kỳ khoa học và tiện lợi để sắp xếp và phân loại các dữ liệu thuộc 1 nhóm vào cùng 1 kiểu dữ liệu duy nhất.
- Để truy cập vào 1 member của 1 struct, ta dùng toán tử truy xuất thành viên “.”

Bài 12. Pointer (Con trỏ)

Chúng ta đã học qua các biến chứa các kiểu dữ liệu khác nhau. Biến kiểu int thì chứa giá trị số nguyên, biến float chứa số thực,... Hôm nay, chúng ta sẽ đến với 1 kiểu biến mới: con trỏ.

1. Địa chỉ

Có lẽ các bạn đã từng nhìn thấy những chuỗi kiểu này:

0x0002F017

Đây chính là địa chỉ của 1 ô nhớ. Mỗi ô nhớ đều có 1 địa chỉ xác định, độc lập với dữ liệu bên trong ô nhớ đó. Giống như 1 căn nhà, nơi mà dữ liệu là nội thất trong căn nhà, và địa chỉ chính là biển số.



Nếu là shipper, họ chắc chắn sẽ rất sợ khi nhìn những địa chỉ như thế này. May cho chúng ta, địa chỉ của 1 ô nhớ là 1 con số duy nhất, ghi bằng **hệ thập lục phân** (hexadecimal - HEX). Một ô nhớ thông thường sẽ bằng **1 byte (= 8 bit)**.

2. Thế nào là con trỏ?

Con trỏ trong C++ cũng chỉ là là **biến**, cũng có thể khai báo, khởi tạo và lưu trữ giá trị và có địa chỉ của riêng nó. Nhưng biến con trỏ không lưu giá trị bình thường, nó là biến trỏ tới 1 địa chỉ khác, tức mang giá trị là 1 địa chỉ.

Chính vì con trỏ mang địa chỉ, nó là 1 biến đặc biệt có thêm những quyền năng mà biến bình thường không có. Nhờ việc nó mang địa chỉ, nó có thể trỏ lung tung trong bộ nhớ. Đây là 1 điểm mạnh nếu ta khai thác tốt nhưng nếu quản lý không tốt thì lại là 1 tai hại.

3. Khai báo con trỏ

- Cách trực tiếp: **RECOMMENDED**

```
<type> * <identifier>;
```

Ví dụ: `int * a;` hoặc `int* a;` hoặc `int *a;`

- Dùng từ khóa `typedef`:

```
typedef <type>* <alias_type>;
```

Ví dụ: `typedef int* intPointer;`
`intPointer a;`

Khi khai báo con trỏ, cần xác định rõ kiểu dữ liệu mà con trỏ trỏ tới.

Ví dụ: `int* a;`

Cú pháp `int*` cho thấy, kiểu dữ liệu của ô nhớ mà con trỏ này đang trỏ tới phải là `int`. Nếu trỏ tới `char` thì phải là `char*`, `bool` là `bool*`,...

Kí tự `*` không bắt buộc phải đứng sát `type` hay là `identifier`. Miễn là nó đứng giữa 2 thành phần này thì chương trình sẽ hiểu.

Con trỏ là `a`, không phải `*a`. Kiểu của con trỏ là `int*`, không phải là `int`.

4. Khởi trị cho con trỏ

Để có được địa chỉ 1 ô nhớ hoặc 1 biến, ta sử dụng toán tử địa chỉ `&`

Sau đó, lấy địa chỉ này đem gán vào con trỏ.

```
int var = 20; // actual variable declaration.
int *ip; // pointer variable

ip = &var;
```

5. Dereferencing Operator (toán tử gián tiếp)

Dereferencing operator hoạt động trên một biến con trỏ và trả về giá trị của ô nhớ mà con trỏ trỏ tới. Đó gọi là “dereferencing” một con trỏ. Đây là một toán tử 1 ngôi.

```
ip // = 0x0002F017 something, i made it up
*ip // = 20
```

Lưu ý cực mạnh!! Không được nhầm lẫn Dereferencing operator với khai báo biến con trỏ. Mặc dù dùng chung kí tự *, nhưng một cái được sử dụng trong lúc khai báo, một cái được sử dụng trong quá trình truy xuất và gán giá trị con trỏ. Tuyệt đối không nhầm lẫn!

6. Con trỏ trống (NULL và nullptr)

Đôi khi, vì 1 mục đích nào đó mà chúng ta tạo ra một con trỏ nhưng chưa sử dụng. Lúc này, ta phải đặt cho con trỏ trở về 1 giá trị “trống” (NULL).

Về **NULL** và **nullptr**, cả 2 đều có nghĩa là “trống” và đều có thể được dùng để gán cho 1 con trỏ. Tuy nhiên, **NULL** chỉ đơn giản là 1 **Macro** được defined bằng 0, còn **nullptr** thì đặc tả 1 con trỏ mang giá trị NULL. Cách gán “trống” cho 1 con trỏ như sau:

```
int* ptr = NULL;  
int* ptr = nullptr;
```

Tuy nhiên, chúng tôi khuyên bạn sử dụng **nullptr** để tránh những lỗi không đáng có. Nếu bạn đã học xong về **Nạp chồng hàm (Function overloading)**, hãy qua lại đây và xem ví dụ này:

```
void functionTest(int n);  
void functionTest(void* data);  
  
functionTest(NULL); //hàm nào sẽ được gọi ?
```

Mặc dù có vẻ như hàm thứ 2 sẽ được gọi vì tham số truyền vào là con trỏ nhưng trên thực tế thì hàm đầu tiên lại được gọi. Vấn đề ở đây bắt nguồn từ việc **NULL** chính là 0, mà 0 là một **số nguyên** (integer type), do đó trình biên dịch sẽ ưu tiên lựa chọn hàm đầu tiên.

Trên thực tế làm việc thì rất ít gặp lỗi kiểu này, nhưng một khi nó xảy ra thì sẽ rất khó hiểu và tốn thời gian debug để tìm ra nó. Để giải quyết vấn đề này, ta thay đổi số **NULL** bằng **nullptr**.

```
functionTest(nullptr);
```

Ghi nhớ

- Con trỏ chứa dữ liệu là địa chỉ ô nhớ.
- Tránh nhầm lẫn giữa Dereferencing operator và khai báo biến con trỏ.
- Sử dụng **nullptr** để tránh lỗi không đáng có.

Bài 13. Pointer (Con trỏ) (tiếp theo)

7. Sử dụng const với con trỏ

Để tránh vô tình làm “hỏng” hoặc thay đổi dữ liệu, người ta sử dụng từ khóa **const** để quy định một đối tượng là “hằng” (constant – không thay đổi được). Có 4 loại con trỏ:

- Con trỏ **thường** trỏ tới dữ liệu **thường** (Nonconstant Pointer to Nonconstant Data)
- Con trỏ **thường** trỏ tới dữ liệu **hằng** (Nonconstant Pointer to Constant Data)
- Con trỏ **hằng** trỏ tới dữ liệu **thường** (Constant Pointer to Nonconstant Data)
- Con trỏ **hằng** trỏ tới dữ liệu **hằng** (Constant Pointer to Constant Data)

a. Con trỏ thường trỏ tới dữ liệu thường

- Dữ liệu **có thể** được thay đổi qua dereferenced pointer.
- Con trỏ **có thể** được chỉnh sửa để trỏ tới ô nhớ khác.

```
int a = 5;
int b = 9;
int *ptr = &a;
*ptr = 6; // OK
ptr = &b; // OK
```

b. Con trỏ thường trỏ tới dữ liệu hằng (Con trỏ hằng)

- Dữ liệu **không thể** được thay đổi qua dereferenced pointer.
- Con trỏ **có thể** được chỉnh sửa để trỏ tới ô nhớ khác.

```
const int a = 5;
int b = 9;
const int* ptr = &a;
*ptr = 6; // Error
ptr = &b; // OK
```

c. Con trỏ hằng trỏ tới dữ liệu thường (Hằng con trỏ)

- Dữ liệu **có thể** được thay đổi qua dereferenced pointer.
- Con trỏ **không thể** được chỉnh sửa để trỏ tới ô nhớ khác.

```
int a = 5;
int b = 9;
int* const ptr = &a;
*ptr = 6; // OK
ptr = &b; // Error
```

d. Con trỏ hằng trỏ tới dữ liệu hằng (Hằng con trỏ tới hằng)

- Dữ liệu **không thể** được thay đổi qua dereferenced pointer.
- Con trỏ **không thể** được chỉnh sửa để trỏ tới ô nhớ khác.

```
const int a = 5;
const int b = 9;
int* const ptr = &a;
*ptr = 6; // Error
ptr = &b; // Error
```

8. Toán tử số học trên con trỏ

Bốn toán tử số học có thể dùng trên con trỏ bao gồm: ++, --, +, -

```
// trỏ tới ô nhớ kế tiếp
ptr++;
ptr = ptr + 1;
// trỏ tới ô nhớ trước đó
ptr--;
ptr = ptr - 1;
```

9. Con trỏ và Array

Array và Pointer có mối quan hệ rất mật thiết trong C++ và dường như luôn có thể sử dụng thay thế cho nhau.

```
int var[3] = { 10, 100, 200 };
int *ptr;
//C++ cho phép chứa địa chỉ array vào trong pointer.
ptr = var;
```

Tuy nhiên, một Array có thể xem tương đương với 1 con trỏ hằng (Constant pointer):

```
var++; // This is incorrect syntax.
```

Địa chỉ của 1 Array thực chất là địa chỉ của phần tử đầu tiên trong Array (var[0]), nên khi lấy địa chỉ liền kề:

```
ptr = var + 1;
```

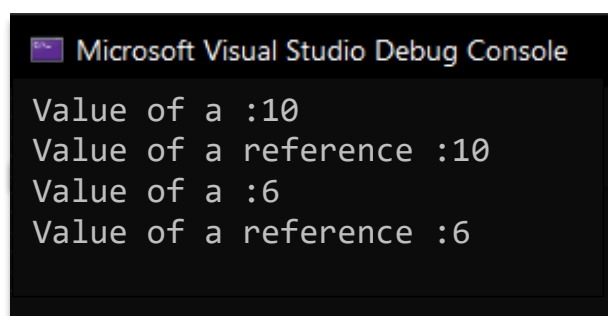
thì con trỏ ptr giờ sẽ trỏ đến phần tử thứ 2 của ô nhớ (var[1]).

10. Tham khảo (Reference)

Một biến tham khảo được xem như là 1 “bí danh”: tức là 1 tên khác cho 1 biến đã tồn tại. Reference thường được dùng cho danh sách đối số của hàm (Function argument list) và giá trị trả về của hàm.

Ví dụ:

```
#include<iostream>
using namespace std;
int main() {
    int a = 10;
    int& b = a;
    cout << "Value of a :" << a << endl;
    cout << "Value of a reference :" << b << endl;
    a = 6;
    cout << "Value of a :" << a << endl;
    cout << "Value of a reference :" << b << endl;
}
```



```
Microsoft Visual Studio Debug Console
Value of a :10
Value of a reference :10
Value of a :6
Value of a reference :6
```

Bài 14. Hàm (Function)

1. Định nghĩa:

Hàm là một nhóm các lệnh được đặt chung dưới 1 cái tên, và có thể được gọi tại một điểm nào đó trong thân chương trình.



Tất cả các hàm trong C++ (trừ hàm main) phải có:

- **Khai báo:** Đây là cách mà hàm sẽ được gọi.
- **Hiện thực hóa/Định nghĩa hàm:** Đây là những lệnh hàm sẽ thực hiện khi được gọi.

2. Khai báo hàm

Một hàm sẽ được khai báo với cú pháp:

```
returnVariableType functionName(para1, para2,..., paraN);
```

Trong đó:

- **returnVariableType** là kiểu dữ liệu mà hàm trả về. Đó là lý do tại sao mà hàm **main** lại có kiểu là **int**, do hàm trả về 0 (return 0) nên kiểu phải là **int**. Nếu không có giá trị trả về, hàm có kiểu là **void**.
- **functionName** là tên hàm. Ví dụ: **int HamViDu()**.
- (para1, para2,..., paraN) là danh sách đối số (parameter list). Mỗi đối số phải bao gồm kiểu dữ liệu và tên biến, ví dụ: (**float** doiso1, **int** doiso2). Trường hợp không có đối số thì để trống: ()

3. Hiện thực hóa hàm

Cú pháp hiện thực hóa như sau:

```
returnVariableType functionName(para1, para2,..., paraN)
{
    statement(s);
}
```


4. Nạp chồng hàm (Function overloading)

Một hàm đã được nạp chồng là một hàm có một hoặc nhiều hàm khác cùng tên nhưng khác danh sách đối số.

C++ lựa chọn hàm có danh sách đối số trùng với kiểu của các đối số truyền vào, bằng cách xem xét số lượng, loại và thứ tự các đối số khi được gọi.

```
//Declare

float add(float a, float b);
int add(int a, int b);
double add(int a, double b);

//Define

float add(float a, float b) {
return a + b;
}

int add(int a, int b) {
return a + b;
}

double add(int a, double b) {
return (double)a + b;
}
```

Nếu gọi `add(3, 4)`, hàm thứ hai sẽ được gọi. Nếu gọi `add(3.7, 6)`, hàm thứ nhất sẽ được gọi. Vân vân...







TÀI LIỆU TIN HỌC – PHẦN 2: C++

Biên soạn bởi CLB Tin học THPT Nguyễn Hữu Huân.

Không sao chép dưới mọi hình thức mà không có sự cho phép.

Thực hiện năm 2020.

