

LỜI MỞ ĐẦU

Chào mừng các bạn đến với bài học lập trình C/C++ .

Tôi sẽ là người hướng dẫn (hay là thầy giáo nếu bạn thích 😊) trong suốt các bài học.

Vậy tôi là ai? Tên tôi, hay là nickname của tôi là M@teo21. Tôi đã từng viết rất nhiều bài hướng dẫn khác trên <http://www.siteduzero.com>, tôi cũng chính là người đã tạo nên trang web này.

Và đây không phải là lần đầu tiên tôi viết những bài hướng dẫn cơ bản 😊.
Nhưng khoan hãy nói về tôi đã, hãy nói về chính bản thân các bạn.

Bạn không hề biết tí gì về lập trình.

Cũng không chắc đã biết “lập trình” là gì nhưng... chắc chắn là bạn đang muốn học lập trình đúng không?

Và bạn phải xác định chính xác mục tiêu của mình là: “học lập trình”.

Nhưng lập trình C / C++... Đó nghĩa là gì?

Và có thật sự tốt nếu ta bắt đầu học từ nó?

Và bạn đã biết lập trình trước đó chưa?

Có phải chúng ta có thể làm tất cả mọi thứ trên máy tính với nó?

Nhiệm vụ quan trọng của chương này là trả lời tất cả những câu hỏi đơn giản như thế.

CHƯƠNG I – NHỮNG ĐIỀU CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH C

1) "lập trình" là gì?

- Lập trình là gì ?
- Lập trình bằng ngôn ngữ lập trình nào ?
- Lập trình có khó không ?
- Trắc Nghiệm Kiến Thức.

2) Công cụ cần có để học lập trình

- Những công cụ cần thiết cho lập trình.
- Các bạn có thể chọn... Dev-C++
- Hay là... Visual C++
- Và cái này nữa... Code::Blocks.
- Hoặc Mac... Xcode.

3) Chương trình đầu tiên của bạn

- Console hay là cửa sổ ?
- Những dòng code tối thiểu cần phải có.
- Viết một tin nhắn ra màn hình.
- Những chú thích, vô cùng tiện dụng !

4) Thế giới của những biến số

- Công việc của bộ nhớ.
- Cách khai báo một biến.
- Hiển thị giá trị của biến số.
- Cách gán giá trị vào biến số.

5) Công cụ tính toán

- Những tính toán cơ bản.
- Phương pháp viết rút gọn.
- Thư viện toán học.
- Trắc Nghiệm Kiến Thức.

6) Conditions (điều kiện)

- Condition "if... else".
- Boolean, trung tâm của những condition.
- Condition "switch".
- Ternary : những conditions rút gọn.

7) Loops (vòng lặp)

- Thế nào là vòng lặp.
- Vòng lặp while.
- Vòng lặp do... while.
- Vòng lặp for.

8) Test Program: Hơn hay kém, trò chơi đầu tiên của bạn

- Chuẩn bị và một vài gợi ý.
- Đáp án.
- Ý tưởng cải tiến.

9) Function

- Cách tạo và gọi một function.
- Xem thêm vài ví dụ để hiểu rõ hơn.

Bài 1: Lập trình là gì?

Chúng ta bắt đầu từ một câu hỏi đơn giản nhất có thể. 😊

Nếu bạn đã biết điều này trước đó, tôi vẫn khuyên bạn nên đọc lại nó. Tôi sẽ bắt đầu bài hướng dẫn từ zero, dành một người không biết tí gì về lập trình. 😊

❓ “lập trình” (programmer, program) nghĩa là gì?

Vâng tôi sẽ tránh làm giống như thầy giáo dạy văn của tôi. Tôi sẽ không đưa ra cho bạn nguồn gốc của từ lập trình (programmer, program). Nhưng dù sao đi nữa nó xuất phát từ một từ latin "programmeus". 😊

Nói đơn giản, lập trình nghĩa là tạo nên những "chương trình máy tính". Những chương trình đòi hỏi máy tính thực hiện tất cả công việc.

Máy tính bạn chứa đầy các chương trình ở tất cả mọi thể loại:

- Calculator chính là một chương trình.
- Các công cụ xử lý văn bản của bạn cũng là chương trình.
- Các software dùng để “chat” cũng là chương trình.
- Trò chơi điện tử cũng là chương trình.

Tóm lại, những chương trình ở khắp mọi nơi trên máy tính và cho phép thực hiện bất cứ điều gì.

Bạn có thể tạo ra một chương trình mang tính cách mạng nếu bạn may mắn, hoặc thực hiện một game đánh nhau 3D trên internet. Máy tính của bạn có thể làm tất cả (trừ những việc như làm ra café) 😊



Trò chơi nổi tiếng Half-life 2, được lập trình bằng C++

⚠ Xin lưu ý rằng tôi không nói người ta làm ra trò chơi này hoàn toàn chỉ thông qua việc đánh máy viết code. Ý tôi là chúng ta có thể làm những điều đó, nhưng chắc chắn là ngoài việc gõ phím ra, bạn còn có rất nhiều việc khác để làm.

Bạn sẽ không bắt đầu học lập trình bằng việc tạo ra một game 3D. Chẳng khác nào tự mình kết thúc tất cả 😊. Chúng ta hãy bắt đầu từ những điều cơ bản. Đầu tiên là làm sao *hiển thị lên màn hình một tin nhắn*.

Bạn phải học từng thứ từng thứ một, và từ từ bạn sẽ có khả năng thực hiện những chương trình với độ khó ngày càng gia tăng. Mục đích của toàn bộ bài hướng dẫn này giúp bạn có khả năng xoay sở trên bất kì chương trình nào được viết bằng ngôn ngữ C hay C++

Lập trình bằng ngôn ngữ nào?

Thật sự mà nói, máy tính đúng là một cỗ máy kì lạ: nó chỉ nhận và gửi lại những số 0 và 1.
Ví dụ, nếu dịch câu: “thực hiện phép tính $3 + 5$ ” về ngôn ngữ máy tính, nó sẽ có dạng như sau:
0010110110010011010011110.

(dãy số trên do tôi chế ra đấy 🤖, thật sự thì tôi không giỏi việc dịch ra ngôn ngữ máy tính 😊)

Những số mà bạn thấy ở trên, là ngôn ngữ của máy tính, gọi là **ngôn ngữ nhị phân** (language binary). Máy tính của bạn chỉ hiểu được ngôn ngữ này. Nhưng bạn và tôi, chúng ta hoàn toàn không thể hiểu và học được ngôn ngữ đó. 🤔

Và đây chính là vấn đề đầu tiên của chúng ta:

❓ **Làm cách nào để giao tiếp với máy tính đơn giản hơn việc dùng những số 0 hay 1?**

Máy tính của bạn không nói được tiếng Anh cũng như tiếng Việt. Và không ai quan niệm phải viết một chương trình bằng ngôn ngữ nhị phân, kể cả những nhà lập trình điên nhất cũng không làm chuyện đó. 🤖

Ý tưởng là phải tạo ra một ngôn ngữ mới đơn giản hơn và sau đó nó sẽ được chuyển sang ngôn ngữ nhị phân. Đây là việc của những nhà lập trình chuyên về ngôn ngữ. Và các chương trình trình này đã được tạo ra bởi họ, chúng ta sẽ không cần phải thực hiện lại, thật là may mắn phải không? 😊

Nói đơn giản:

Nếu bạn viết một lệnh bằng ngôn ngữ lập trình nào đó với nội dung:
“thực hiện phép tính $3+5$ ”
thì chương trình dịch sẽ chuyển thành những dạng như sau:
"0010110110010011010011110"

Tôi sẽ lập một biểu đồ để giúp bạn hiểu rõ hơn:



Biểu đồ cực kì đơn giản về việc thi hành một chương trình 🤖

Ở đây tôi chỉ dùng những từ ngữ đơn giản để giải thích, nhưng trong tin học mỗi vấn đề đều có một thuật ngữ riêng.

Suốt các bài học, bạn sẽ phải học không ít những thuật ngữ đó. Điều đó giúp bạn có thể dễ dàng biểu đạt những vấn đề về tin học, hơn nữa, bạn có thể hiểu được ý của một nhà lập trình nào đó mà bạn sẽ trò chuyện sau này. Hẳn là lúc đó, những người xung quanh sẽ nhìn các bạn với ánh mắt khác thường, đó là lý do mà bạn phải chú ý đến việc học những thuật ngữ 😊

Quay lại với biểu đồ ở trên:

Trong ô đầu tiên: “Chương trình được viết bằng ngôn ngữ lập trình đơn giản”. Cụm từ “Ngôn ngữ lập trình đơn giản” còn được gọi là “**ngôn ngữ bậc cao**”. (*high-level programming language*).

Có rất nhiều “cấp bậc” trong ngôn ngữ lập trình. Và ngôn ngữ bậc càng cao càng gần và giống với ngôn ngữ của chúng ta (cũng giống như tiếng Việt hay tiếng Anh). Ngôn ngữ bậc cao giúp ta dễ dàng hơn trong sử dụng, nhưng nó vẫn có một vài thiếu sót mà bạn sẽ thấy về sau. Có rất nhiều ngôn ngữ bậc cao hay thấp trong tin học, trong số đó bạn có thể dùng để lập trình.

Và đây là một vài ví dụ:

- C
- C++
- Java
- Visual Basic
- Delphi
- ...vv...

Nói thêm rằng ở đây tôi không sắp xếp chúng theo “cấp bậc của ngôn ngữ”, vì thế bạn đừng nghĩ rằng ngôn ngữ đầu tiên sử dụng dễ dàng hơn hay ngược lại. 😊 Đó chỉ là một vài ví dụ bất chợt nảy ra trong đầu của tôi.

(Còn rất nhiều ngôn ngữ khác nữa mà tôi không liệt kê ra hết, vì sẽ rất dài nếu ghi hết ra 😊, xin các bạn bỏ qua cho).

Một số ngôn ngữ có bậc cao hơn các ngôn ngữ khác (về mặt lý thuyết thì dễ dàng sử dụng hơn), chúng ta sẽ xem xét điều này sau, đặc biệt là sự khác nhau giữa ngôn ngữ C và C++.

Một từ ngữ khác mà ta phải nắm đó là: **mã nguồn** (code source). Đó đơn giản là phần mã của chương trình được viết bằng ngôn ngữ bậc cao. Và tất cả những mã nguồn đó sẽ được dịch thành ngôn ngữ nhị phân.

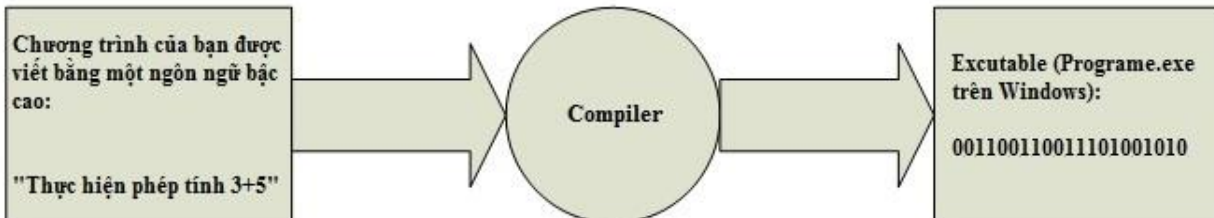
Trong giai đoạn tiếp theo, “chương trình biên dịch” sẽ dịch ngôn ngữ bậc cao đó (C hay C++) sang nhị phân. Chương trình này có tên là **compiler**. Việc biên dịch gọi là **compilation**.

Quan trọng: các ngôn ngữ lập trình bậc cao khác nhau sẽ có các compiler khác nhau. Nói cách khác, chúng ta không thể dịch ngôn ngữ C++ cùng với cách mà ta dịch ngôn ngữ Delphi.

⚠ Về sau bạn sẽ nhận thấy, cùng một ngôn ngữ sẽ có nhiều **compilers** khác nhau (compiler của Microsoft, compiler GNU... tôi sẽ nói về chúng ở những chương sau). Rất may mắn là những compiler đó gần như giống nhau (đôi khi chúng có những khác biệt nhỏ và tôi sẽ chỉ cho bạn).

Cuối cùng, chương trình nhị phân được tạo ra bởi compiler được gọi là: **executable**. Các chương trình này trên Windows có đuôi ".exe" giống như EXEcutable.

Quay lại với biểu đồ vừa rồi nhưng thay bằng những thuật ngữ tin học chính xác:



Cùng một biểu đồ nhưng biểu đồ này dùng các thuật ngữ chính xác hơn.

Tại sao chọn học C/C++?

Như tôi đã nói với bạn ở trên, có rất nhiều ngôn ngữ bậc cao. Chúng ta sẽ bắt đầu bằng một trong số đó.

Nhưng mà bạn phải có một sự lựa chọn giữa:

- **Một ngôn ngữ lập trình có bậc rất cao:** dễ dàng sử dụng, "thông dụng", như Visual basic. Nhưng các ngôn ngữ này có rất nhiều khuyết điểm: đầu tiên là phải mua bản quyền để sử dụng, giá thành mắc, và bị nhiều hạn chế. Ví dụ, chương trình mà bạn viết sẽ chỉ chạy được trên Windows, đừng nghĩ đến việc chạy nó trên Linux hay Macintosh! Hay trên hết, bạn không có thể làm tất cả những gì bạn muốn với dạng ngôn ngữ này, và điều đó khiến bạn cảm thấy hạn chế khi sử dụng.
- **Một ngôn ngữ khác ở vị trí tương đối thấp hơn** (nhưng nó không thấp lắm đâu!): có thể nó sẽ hơi khó hơn visual basic, nhưng chắc chắn một điều là với một ngôn ngữ như C (hay C++) sẽ giúp bạn học thêm rất nhiều trong việc lập trình cũng như hiểu thêm cách hoạt động của máy tính. Và sau đó hoàn toàn đủ khả năng học thêm một vài ngôn ngữ khác nếu bạn muốn. Bạn sẽ tự chủ hơn. Mặt khác, ngôn ngữ C và C++ được sử dụng khá rộng rãi. Nó được dùng để lập trình phần lớn các chương trình bạn biết. Cuối cùng, để lập trình trên C hay C++, bạn không cần phải mua bất kì chương trình nào vượt ngoài túi tiền của bạn, vì ngôn ngữ này hoàn toàn miễn phí!

Và đó là lý do thúc đẩy tôi hướng dẫn bạn ngôn ngữ C trước tiên. Tôi không hề nói rằng chúng ta bắt buộc phải bắt đầu từ đây, nhưng lựa chọn này sẽ giúp bạn có được những kiến thức bền vững về lập trình.

Tôi xem như đây chính là ngôn ngữ lập trình đầu tiên của bạn, và bạn không biết tí gì về lập trình trước đó. Cũng có thể, bạn đã biết lập trình rồi, nhưng việc học lại từ cơ bản không ảnh hưởng xấu tí nào phải không 😊

❓ Khoan đã, có một cái mà tôi vẫn chưa hiểu: Tôi sẽ học một ngôn ngữ gọi là "C / C++" hay tôi sẽ học 2 ngôn ngữ khác nhau một là "C" và một là "C++"?

Câu trả lời tốt nhất là bạn sẽ cùng lúc học cả 2 ngôn ngữ. Không phải là bạn phải tăng cường độ làm việc lên hai lần đâu 😊 mà là 2 ngôn ngữ này khá giống nhau. (Khi tôi nói đến 2 ngôn ngữ cùng lúc, tôi sẽ viết "C / C++").

Và chúng ta phải hiểu rõ sự khác nhau giữa C và C++ trước khi bắt đầu:

- Giai đoạn đầu, lúc các máy tính có khối lượng tính bằng tấn và có kích thước to như ngôi nhà, người ta đã sáng tạo ra một ngôn ngữ lập trình gọi là **Algol**.
- Sau nhiều cải tiến, người ta đã tạo ra một ngôn ngữ mới gọi là **CPL**, và chính nó đã phát triển thành **BCPL**, sau đó nó được mang tên là **ngôn ngữ B**. (Các bạn không cần phải nắm tất cả những điều này, tôi viết ra chỉ để có thêm chút đáng về về lịch sử mà thôi 😊).
- Và trong một ngày đẹp trời, người ta đã hoàn tất việc tạo ra một ngôn ngữ mới gọi là ... **ngôn ngữ C**. Qua các sửa đổi, ngôn ngữ này vẫn là một trong những ngôn ngữ được sử dụng nhiều nhất cho đến hôm nay.
- Không lâu sau đó, người ta đề xuất thêm vào ngôn ngữ C một vài thứ để cải tiến, và ngôn ngữ mới này được gọi là **ngôn ngữ C++** hoàn toàn dựa trên nền tảng của C. Ngôn ngữ C++ không có gì khác C ngoại trừ một số cái được thêm vào. (Những cái đó là gì? Chúng ta sẽ thấy về sau).

Có rất nhiều cách để học lập trình. Nhiều người nghĩ rằng tốt hơn hết là học ngay từ “C++”. Điều đó cũng đúng, chúng tương đối giống nhau, C++ chỉ là C được thêm vào “những dấu +”. Ngôn ngữ C không phải là “ngôn ngữ già cỗi bị lãng quên”, ngược lại nó được sử dụng rất nhiều trong thời đại hiện nay. Nó là nền tảng của những hệ điều hành lớn như Unix hay Windows.

Nếu bạn bắt đầu bằng ngôn ngữ C, sau này bạn học ngôn ngữ C++ sẽ nhanh và dễ dàng hơn. Và bạn sẽ không cần phải học lại tất cả những gì đã biết, bạn chỉ cần học những cái được thêm vào ở “C++” (và đây chính là điều tôi cần nói 😊).

⚠️ Có một số hiểu lầm khi có người cho rằng **ngôn ngữ C++ tốt hơn C**, thật ra thì nó chỉ cho phép bạn lập trình theo cách khác. Có thể nói, nó chỉ giúp việc lập trình của ta nhanh hơn và dễ dàng hơn trong việc tổ chức mã nguồn của chương trình.

Nắm vững vấn đề: C và C++ không phải là 2 ngôn ngữ cạnh tranh, đối lập với nhau. Chúng ta đều có thể dùng 2 ngôn ngữ này lập trình những cái tương tự. Chỉ là dùng 2 phương pháp lập trình khác nhau.

Điều thuận lợi hơn là sau này có thể dùng C hay C++ tùy theo ý muốn và mục đích của các bạn.

Lập trình có khó không ?

Đây chính là câu hỏi khiến bạn phải suy nghĩ nhiều đúng không? 😊

Và có phải chúng ta bắt buộc phải là một nhà toán học cực giỏi sau nhiều năm học tập để có thể bắt đầu với việc lập trình?

Câu trả lời khiến bạn yên tâm hơn là **điều đó không đúng**.

Bạn không cần phải có một đẳng cấp toán học thật cao. Những kiến thức bạn cần để bắt đầu học chính là:

- Phép cộng (Tôi hi vọng bạn đã nắm vững nó 😊)
- Phép trừ (híc... híc...)
- Phép nhân (😓...)
- Phép chia (😓)

Tôi hi vọng các bạn đã biết tất cả những phép tính đó. 😊 Và chắc chắn là tôi sẽ giải thích cho bạn trong phần tiếp theo, làm sao máy tính có thể thực hiện các phép tính cơ bản.

Tóm lại, về toán học thì không có gì khó khăn để bạn vượt qua. 😊

Tất cả chỉ phụ thuộc vào chương trình mà bạn muốn thực hiện, nếu đó là chương trình liên quan đến toán học, bạn bắt buộc phải hiểu biết nhiều về toán. Nếu bạn muốn làm một game 3D thì đòi hỏi bạn phải có kiến thức về hình học không gian.

Để học ngôn ngữ “C / C++”, bạn không cần thiết phải có những kiến thức cao cấp nào cả.

❓ Nhưng đâu là cái khó khăn ?

Chúng ta cần phải biết máy tính hoạt động như thế nào để có thể hiểu chúng ta đang làm những gì. Và ở điểm này, hãy yên tâm, tôi sẽ cố gắng hết sức để hướng dẫn cho các bạn.

Một người lập trình cũng cần phải có một vài đặc điểm như sau:

- **Kiên trì:** một chương trình có thể sẽ không chạy tốt trong giai đoạn đầu, vì vậy bạn phải nhẫn nại!
- **Tư duy tốt:** chắc hẳn là bạn không cần phải giỏi lắm về toán, cái bạn cần là suy nghĩ một cách logic.
- **Nhệ nhàn:** Người ta không đánh máy bằng việc gõ búa lên bàn phím. 😓 Điều đó cũng không giúp chương trình của bạn chạy tốt hơn tí nào đâu. 😓

Tóm lại một cách đơn giản, việc học lập trình không đòi hỏi bạn phải thật sự có kiến thức chuyên sâu trong một lĩnh vực nào đó. Một người dốt toán vẫn có thể viết ra một chương trình, cái cần thiết chính là khả năng suy nghĩ của bạn.

Tổng Kết:

- Để tạo ra được một chương trình máy tính, người ta phải viết những chương trình đó dựa trên một loại ngôn ngữ mà máy tính có thể biên dịch lại để hiểu. Người ta gọi đó là “Ngôn ngữ lập trình”.
- Có rất nhiều loại ngôn ngữ máy tính được phân theo nhiều cấp độ. Những ngôn ngữ cao cấp thì dễ sử dụng hơn nhưng chưa chắc sẽ mang lại hiệu quả cao hơn những ngôn ngữ cấp thấp.
- Ngôn ngữ lập trình C mà chúng ta đang được học trong tài liệu này được gọi là ngôn ngữ cấp thấp và nó cũng đang là ngôn ngữ lập trình phổ biến nhất thế giới hiện nay.
- Source code là tập hợp văn bản do bạn viết ra thể hiện ý nghĩa của ngôn ngữ lập trình.
- Compiler là một chương trình biên dịch có khả năng dịch source code sang ngôn ngữ nhị phân, sau đó chuyển thành chương trình Executable (.exe). Chúng ta phải biết rằng trong chương trình nhị phân thì không còn chứa source code.
- Việc lập trình không đòi hỏi bạn phải có một kiến thức chuyên sâu về một lĩnh vực nào đó như Toán học (ngoại trừ một số trường hợp chương trình bạn viết đòi hỏi phải sử dụng những công thức toán học chuyên sâu, vd như các phần mềm mã hóa). Tuy nhiên, một tư duy logic, nhạy bén là điều thật sự cần thiết đối với một lập trình viên.

Chà! Chúng ta kết thúc bài 1 rồi, nhưng các bạn vẫn chưa thấy bất kì dòng code nào giống như chúng ta đã thống nhất trước đó.

Trong chương tiếp theo, các bạn sẽ bắt đầu học cách lập trình với những công cụ đầu tiên. Bạn sẽ được hướng dẫn cài đặt những chương trình cần thiết cho bất kì người học lập trình nào. 😊

TRẮC NGHIỆM KIẾN THỨC.

Phần này sẽ giúp bạn kiểm tra lại kiến thức đã học được, bạn chỉ việc chọn câu trả lời đúng nhất thôi.

- ❓ Những file nào được tạo ra từ việc lập trình ?
- A. Những file *.exe chạy trên Windows
 - B. Những hình ảnh (*.jpg, *.png, *.bmp...)
 - C. Những đoạn videos (*.avi, *.mov...)

- ❓ Giữa C và C++, ngôn ngữ nào cho phép ta lập trình tốt hơn ?
- A. C
 - B. C++
 - C. Cả hai đều mạnh mẽ như nhau

- ❓ Chương trình dịch ngôn ngữ cao cấp thành ngôn ngữ nhị phân gọi là :
- A. Programer
 - B. Brumisateur
 - C. Compiler

- ❓ Ngôn ngữ nào bạn sẽ học trong giai đoạn đầu ?
- A. C
 - B. C++
 - C. Cả hai

Đáp án:

- 1- A
- 2- C
- 3- C
- 4- C

Bài 2: Một vài công cụ cần có để học lập trình

Sau bài mở đầu, chúng ta sẽ bắt đầu đi sâu vào bài học bằng cách trả lời câu hỏi “Cần sử dụng chương trình nào để lập trình?”

Sẽ không có vấn đề gì quá khó khăn trong bài này, chúng ta sẽ dành chút thời gian để tìm hiểu về một số phần mềm mới.

Hãy tận hưởng thời gian này bởi vì trong các bài tiếp theo, chúng ta sẽ thực sự học cách lập trình và có lẽ sẽ không có thời gian cho bạn “đánh một giấc” đâu nhé. 😊

Những công cụ cần thiết cho việc lập trình:

Vậy theo bạn, chương trình mà chúng ta đang cần là gì ?

Nếu như các bạn có theo dõi kỹ nội dung bài học trước thì hẳn là bạn phải biết ít nhất 1 cái tên nào đó chứ.

Bạn biết điều tôi đang muốn nói mà ... đúng không ??? 😊

Vâng, đó là trình biên dịch (compiler), một chương trình chuyên dùng để biên dịch ngôn ngữ C/C++ của bạn sang ngôn ngữ nhị phân của máy tính.

Như tôi đã từng nói sơ qua cho các bạn ở bài trước, chúng ta có 1 vài trình biên dịch phổ biến cho ngôn ngữ lập trình C/C++. Việc lựa chọn trình biên dịch nào thật sự không phải là vấn đề quá khó.

Nào, vậy ngoài ra chúng ta còn cần những gì nữa?

Tôi sẽ không để bạn phải thắc mắc thêm, dưới đây là những hành trang tối thiểu cho một lập trình viên:

- Một chương **trình soạn thảo văn bản** (text editor program) để viết mã nguồn (source code) của chương trình. Trên lý thuyết thì để thực hiện việc này chúng ta chỉ cần sử dụng phần mềm “**Notepad**” trong Windows hoặc “**Vi**” trong Linux là đủ. Nhưng sẽ lý tưởng hơn khi bạn sử dụng một trình soạn thảo văn bản có thể tô đậm màu sắc các thành phần trong mã nguồn nhằm giúp bạn xác định rõ ràng hơn khi cần thiết.
- Một chương **trình biên dịch** mã nguồn (compiler) để giúp dịch ngôn ngữ lập trình C/C++ của bạn sang ngôn ngữ nhị phân của máy tính.
- Một chương **trình tìm và sửa lỗi** (debugger) để giúp bạn theo dõi các lỗi trong chương trình của mình. Tin không vui là cho tới bây giờ chúng ta vẫn chưa phát minh ra chức năng “hiệu chỉnh” để sửa chữa những lỗi của chương trình. Điều đó cũng đồng nghĩa với việc nếu bạn đã nắm rõ cách hoạt động của debugger, nó sẽ giúp bạn tìm ra lỗi một cách dễ dàng, chỉ vậy thôi.

Thời gian đầu, nếu bạn là người thích mạo hiểm thì bạn có thể làm việc mà không cần tới debugger nhưng tôi tin chắc rằng không sớm thì muộn bạn cũng sẽ cần đến nó thôi.

Từ những điều trên chúng ta sẽ thấy có 2 trường hợp:


- TH1: Để lập trình, chúng ta sẽ phải dùng 3 chương trình riêng biệt, và đây cũng là cách phức tạp nhất, nhưng sự thật là nó có thể hoạt động. Chẳng hạn như với Linux, nhiều lập trình viên vẫn thích sử dụng 3 chương trình riêng biệt cho công việc lập trình của họ. Tôi sẽ không nói chi tiết về việc này trong bài này mà tôi sẽ chỉ bạn một cách đơn giản hơn.
- TH2: Chúng ta có chương trình nào bao gồm 3 trong 1 không? Tức là một chương trình có chứa 3 thành phần “text editor”, “compiler” và “debugger”. Câu trả lời là có và người ta gọi những chương trình này là IDE.

Chúng ta có một vài IDE phổ biến và sẽ có một chút khó khăn trong thời gian đầu để bạn có thể chọn cho mình một IDE phù hợp. Nhưng có một điều chắc chắn là trong mọi trường hợp, bạn đều có thể lập trình với bất kỳ loại IDE nào.

Chọn IDE phù hợp với bạn:

Sẽ khá thú vị khi giới thiệu cho bạn một vài IDE nổi tiếng nhất mà tôi biết. Dĩ nhiên tất cả đều là miễn phí, hehe. Cá nhân tôi hơi lộn xộn trong việc sử dụng IDE cho mình, tôi có thể dùng mỗi ngày một IDE khác nhau tùy cảm hứng.

- Một trong những IDE ưa thích của tôi đó là Code :: Blocks. Nó hoàn toàn miễn phí và hoạt động được trên hầu hết các hệ điều hành máy tính phổ biến ngày nay. Tôi khuyên bạn nên bắt đầu học lập trình với phần mềm này (thậm chí nó vẫn rất tốt cho tất cả mọi người về sau). Thành này có thể chạy mượt mà trên Windows, Mac và Linux.
- Không thể không nhắc tới một sản phẩm nổi tiếng trên Windows, nó là phần mềm Microsoft Visual C ++. Có rất nhiều phiên bản tính phí (tất nhiên là mắc vcl). Nhưng may thay, có một phiên bản miễn phí tên là Visual C++ Express, thành này thật sự rất ngon lành (nó chỉ khác phiên bản tính phí ở một số điểm nhỏ nhất thôi). Phiên bản miễn phí này cung cấp các chức năng rất đầy đủ và có một bộ module hiệu chỉnh lỗi tuyệt vời (debugging). Em này thì tất nhiên là chỉ chạy được trong môi trường Windows
- Đối với hệ điều hành Mac OS X, các bạn có thể sử dụng một phần mềm tên là Xcode, thường được cung cấp sẵn trên các đĩa cài đặt Mac OS X. IDE này được các lập trình viên làm việc trên hệ điều hành Mac đánh giá rất cao. Và hiển nhiên luôn, nó chỉ chạy được trên Mac OS X.

 Lưu ý một chút với người dùng Linux: Có rất nhiều IDE cho hệ điều hành này, nhưng có vẻ những lập trình viên đã có kinh nghiệm lại thích thú với việc tách biệt mọi thứ ra thay vì sử dụng IDE 3 trong 1, cũng chỉ hơi khó hơn một chút thôi. Trong trường hợp của

chúng ta bây giờ, tôi khuyên các bạn nên cài đặt Code :: Blocks dù bạn đang dùng Linux, chủ yếu là để có thể thuận tiện hơn trong việc theo dõi bài viết của tôi thôi.

❓ Vậy thì cái IDE nào là tốt nhất ?

Tất cả những IDE tôi vừa giới thiệu cho bạn đều có thể giúp bạn lập trình tốt mà không có vấn đề gì. Có thể sẽ có cái này cung cấp nhiều tùy chọn tốt hơn, cái kia lại cho ta trực quan sinh động dễ nhìn, dễ xài hơn. Nhưng trong mọi trường hợp, chương trình mà bạn tạo ra sẽ đều giống nhau dù cho bạn sử dụng IDE nào. Vì vậy đừng quá quan trọng hóa việc lựa chọn IDE tốt nhất.

Nói từ này đến giờ thì tôi quyết định sẽ sử dụng Code :: Blocks. Nếu bạn muốn theo dõi những gì tôi sắp nói dưới đây một cách trực quan sinh động nhất, tôi nghĩ bạn nên cài đặt phần mềm này đầu tiên.

Code :: Blocks (Windows, Mac, Linux)

Code :: Blocks IDE là hoàn toàn miễn phí và có thể cài đặt được trên hệ điều hành Windows, Mac và Linux.

Hiện tại IDE này chỉ mới có phiên bản tiếng Anh thôi, đừng để điều này ngăn cản bạn sử dụng nó.

Hãy nhớ rằng, trong suốt quá trình học lập trình, bạn sẽ còn bắt gặp nhiều tài liệu khác được viết bằng tiếng Anh, và bây giờ chúng ta lại có thêm 1 lý do chính đáng để trau dồi khả năng ngoại ngữ đúng không nào.

Xin mời bạn tải Code :: Blocks về theo link này <http://www.codeblocks.org/downloads/binaries> (hoặc có thể lên google search là ra ngay thôi mà).

- Nếu bạn là người dùng Windows, hãy tải về phiên bản có đuôi “**mingw-setup.exe**”, bởi vì chỉ có phiên bản này mới tích hợp sẵn compiler trong đó, nếu tải những phiên bản khác thì bạn sẽ phải vất vả để biên dịch chương trình đấy.
- Nếu bạn đang dùng Linux thì chỉ việc chọn phiên bản phù hợp với nhà phân phối hệ điều hành của bạn.
- Cuối cùng là với Mac OS X, chỉ đơn giản là tải về phiên bản mới nhất trong danh sách các phiên bản mà bạn nhìn thấy trên website.

⚠️ Hãy cẩn thận tải cho đúng phiên bản cho hệ điều hành Windows của bạn. Bạn có thể xem hình dưới để rõ hơn.



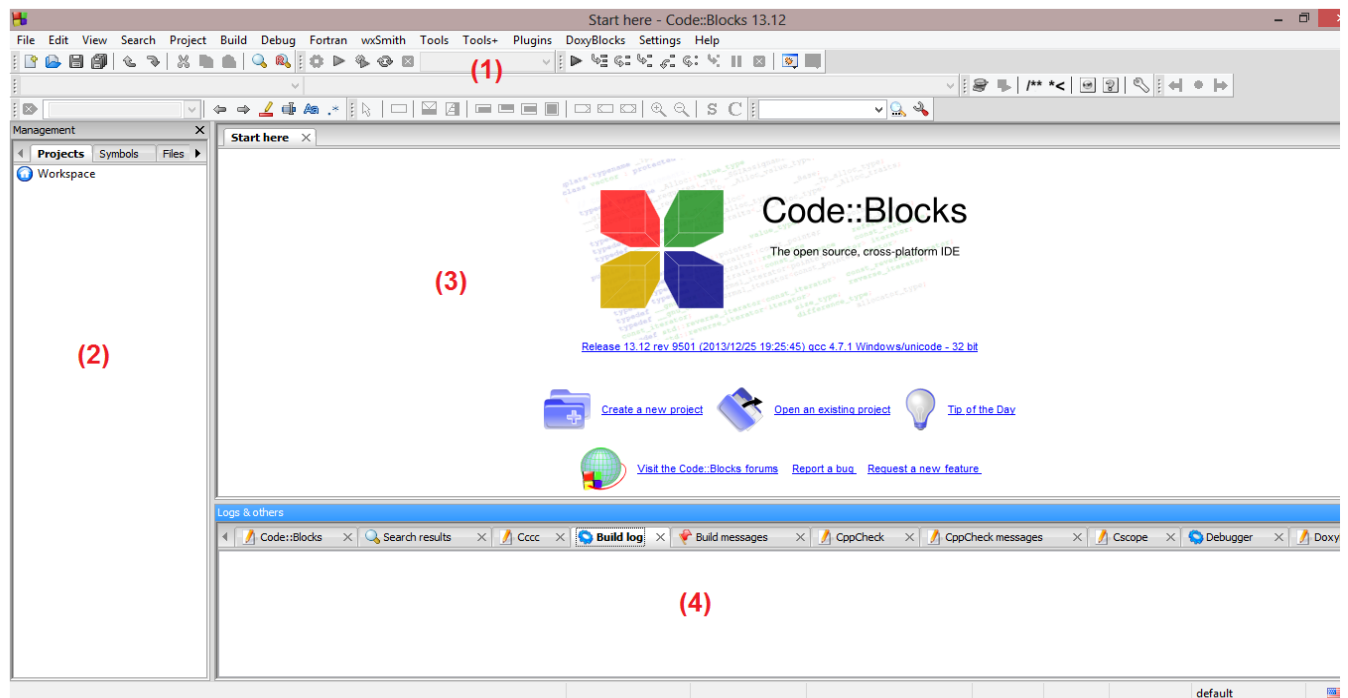
Windows 2000 / XP / Vista / 7:

File	Date	Download from
codeblocks-13.12-setup.exe	27 Dec 2013	BerliOS or Sourceforge.net
codeblocks-13.12mingw-setup.exe	27 Dec 2013	BerliOS or Sourceforge.net
codeblocks-13.12mingw-setup-TDM-GCC-481.exe	27 Dec 2013	BerliOS or Sourceforge.net

NOTE: The codeblocks-13.12mingw-setup.exe file *includes* the GCC compiler and GDB debugger from TDM-GCC (version 4.7.1, 32 bit). The codeblocks-13.12mingw-setup-TDM-GCC-481.exe file includes the TDM-GCC compiler, version 4.8.1, 32 bit. While v4.7.1 is rock-solid (we use it to compile C::B), v4.8.1 is provided for convenience, there are some known bugs with this version related to the compilation of Code::Blocks itself.

IF UNSURE, USE "codeblocks-13.12mingw-setup.exe"!

Nếu không nắm rõ thì cứ tải phiên bản có đuôi mingw-setup.exe như hình trên:



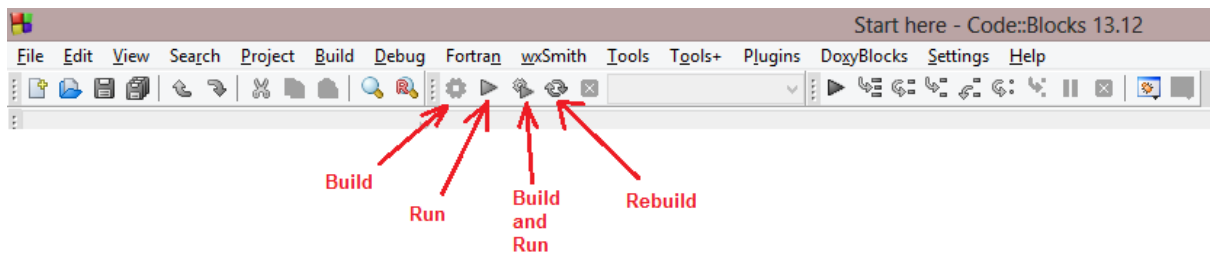
Giao diện chính của Code::Blocks sau khi cài đặt và chạy chương trình.

Quan sát hình trên chúng ta thấy có 4 vùng lớn được đánh số, tôi sẽ giới thiệu sơ cho các bạn về chúng nhé:

1. Thanh công cụ (toolbar): Nó chứa rất nhiều nút chức năng nhưng chỉ một số ít trong đó là được sử dụng thường xuyên. Tôi sẽ nói về những nút này sau.

2. Danh sách các tập tin dự án (list of project files): Khu vực bên trái này hiển thị danh sách các tập tin có chứa mã nguồn (source code) trong chương trình của bạn. Lưu ý rằng hình ảnh này được chụp khi chưa có project nào được tạo, do đó bạn không thấy bất kỳ danh sách tập tin nào được hiển thị.
3. Khu vực chính (main area): Đây chính là chỗ dành cho bạn viết mã nguồn (source code).
4. Khu vực thông báo (notification area): Hay còn được gọi là “death zone” – vùng chết chóc, nơi này sẽ hiển thị lỗi biên dịch nếu mã của bạn có vấn đề, và điều này vẫn thường xuyên xảy ra.

Nào, hãy nhìn vào hình dưới và để ý tới 4 nút đặc biệt trên thanh công cụ. Bạn sẽ thấy các nút chức năng đó theo thứ tự: Build, Run, Build and Run, Rebuild. Tất cả những nút chức năng này sẽ được sử dụng rất thường xuyên:



Tôi sẽ nói cho bạn biết chức năng của những nút này:

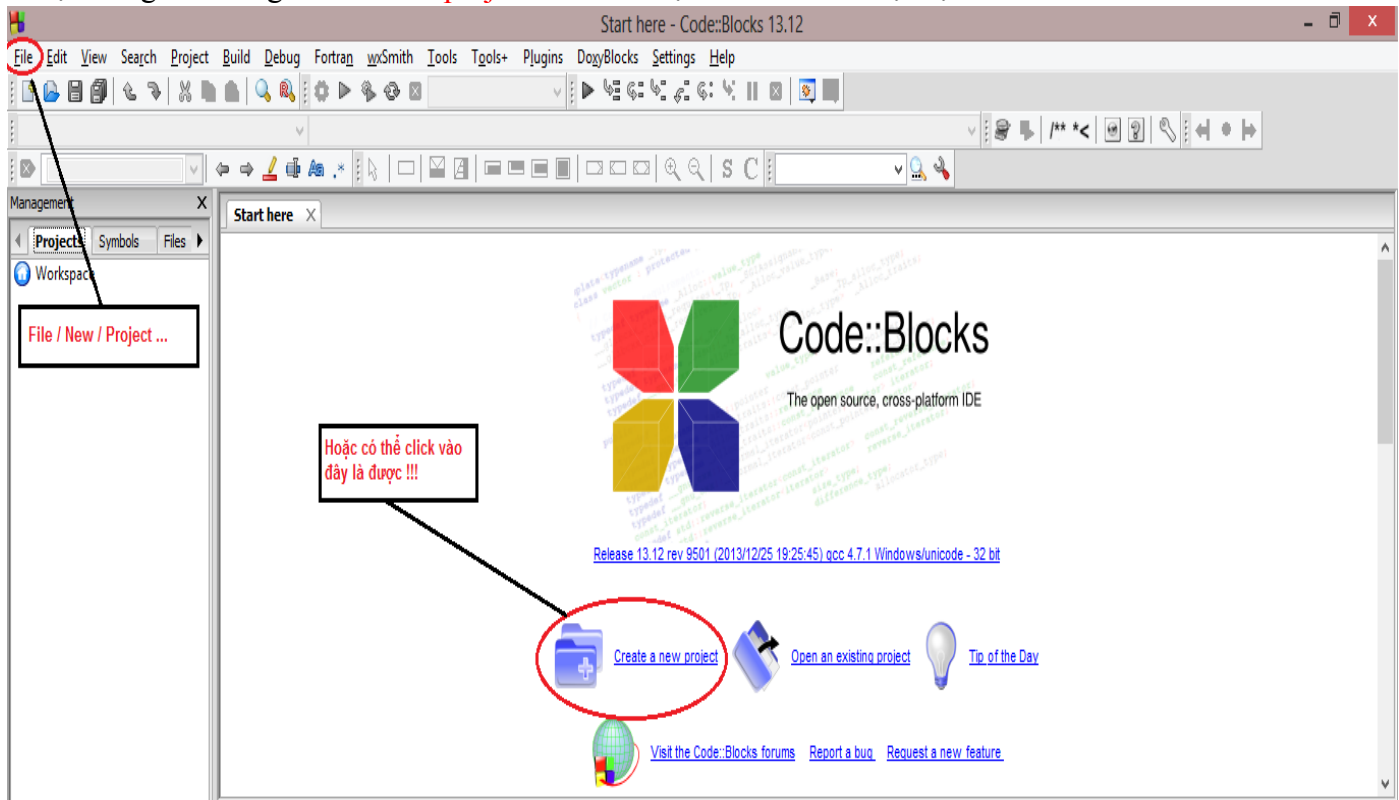
1. **Build:** Hay còn gọi là biên dịch. Nút chức năng này sẽ chuyển tất cả các tập tin chứa mã nguồn trong dự án của bạn đến trình biên dịch để thực thi những tác vụ. Nếu xảy ra lỗi (chắc chắn là sẽ xảy ra ko sớm thì muộn thôi), thực thi sẽ không hoàn thành và bạn sẽ nhìn thấy thông báo ở khu vực bên dưới Code :: Blocks như phần trên đã giới thiệu.
2. **Run:** Hay còn gọi là chạy chương trình. Nút này giúp chương trình của bạn chạy lên sau khi đã được biên dịch, điều này giúp bạn kiểm tra xem chương trình bạn viết hoạt động như thế nào, có đúng như ý bạn muốn. Theo như thứ tự thì bạn sẽ biên dịch trước rồi sau đó chạy chương trình, nhưng có một nút thứ 3 giúp bạn hợp 2 quá trình này lại trong 1 cú click chuột ...
3. **Build and Run:** Chắc hẳn là bạn không cần phải là một thiên tài để hiểu được cái nút thứ 3 này chỉ là một sự giao lưu – kết hợp từ 2 nút đầu tiên. Đây dường như sẽ là nút bạn dùng thường xuyên nhất. Lưu ý rằng, nếu có bất kỳ lỗi nào xảy ra trong quá trình biên dịch, chương trình sẽ không chạy được và tất cả những gì bạn nhận được là một đồng thông báo lỗi ở phía dưới nhé.

4. **Rebuild:** Biên dịch lại. Khi bạn biên dịch chương trình thì thực tế, Code :: Blocks sẽ biên dịch lại những tập tin mà bạn đã thay đổi. Đôi khi ... ý tôi là đôi khi thôi nhé ... bạn sẽ cần Code :: Blocks biên dịch lại tất cả các tập tin. Chúng ta sẽ được biết khi nào chúng ta cần sử dụng chức năng này và cụ thể những gì nó sẽ làm trong những bài tiếp theo. Tại thời điểm hiện tại tôi nghĩ chúng ta không nên nhồi nhét quá nhiều. Tạm thời nút chức năng gần như không cần thiết đối với chúng ta.

⚠ Tôi cũng khuyên các bạn nên tập thói quen sử dụng các phím tắt thay vì nhấp chuột vào các nút chức năng. Việc này sẽ giúp các bạn tiết kiệm được khá nhiều thời gian và nó diễn ra rất thường xuyên. Chẳng hạn như để Build and Run, các bạn chỉ cần bấm F9 là được.

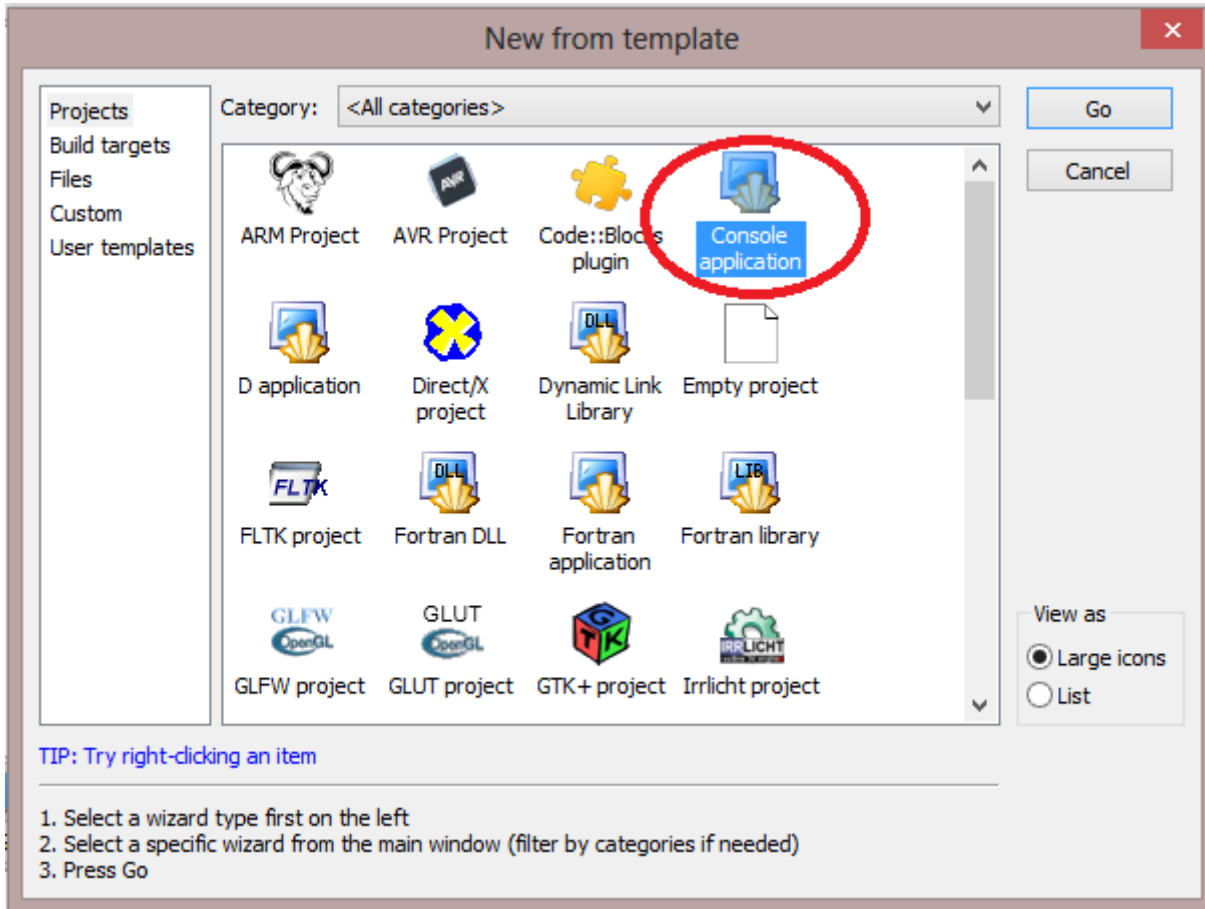
Cách tạo một dự án (Project) mới:

Để tạo một project mới, thật sự rất đơn giản: Bạn chỉ cần chọn **File / New / Project...** hoặc nhấp chuột thẳng vào dòng **Create new project** trên khu vực chính trước mặt bạn.



Có 2 cách để tạo 1 Project mới

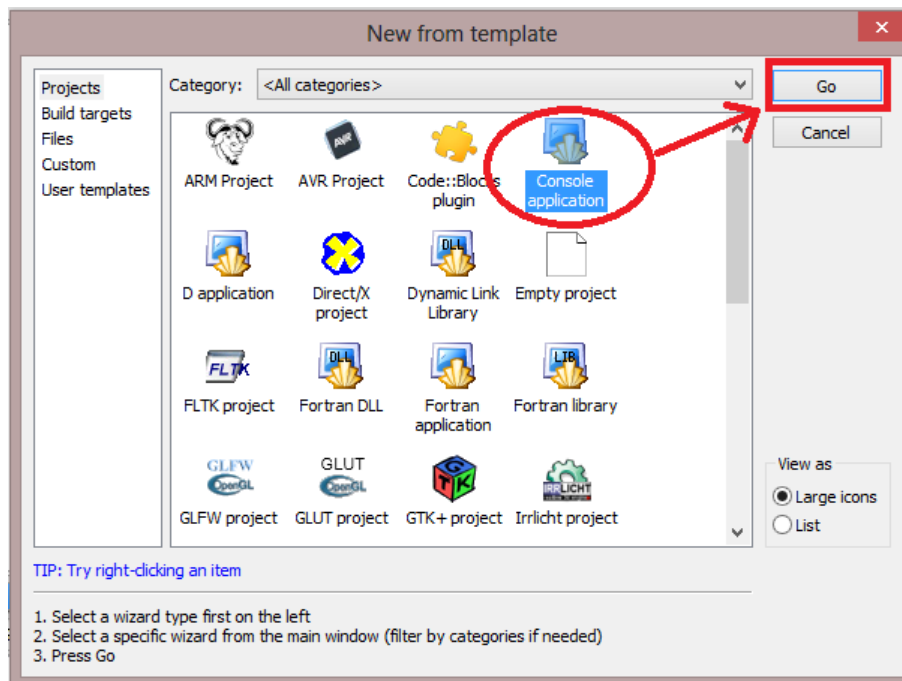
Trong cửa sổ vừa mở ra (xem hình sau), chọn Console application



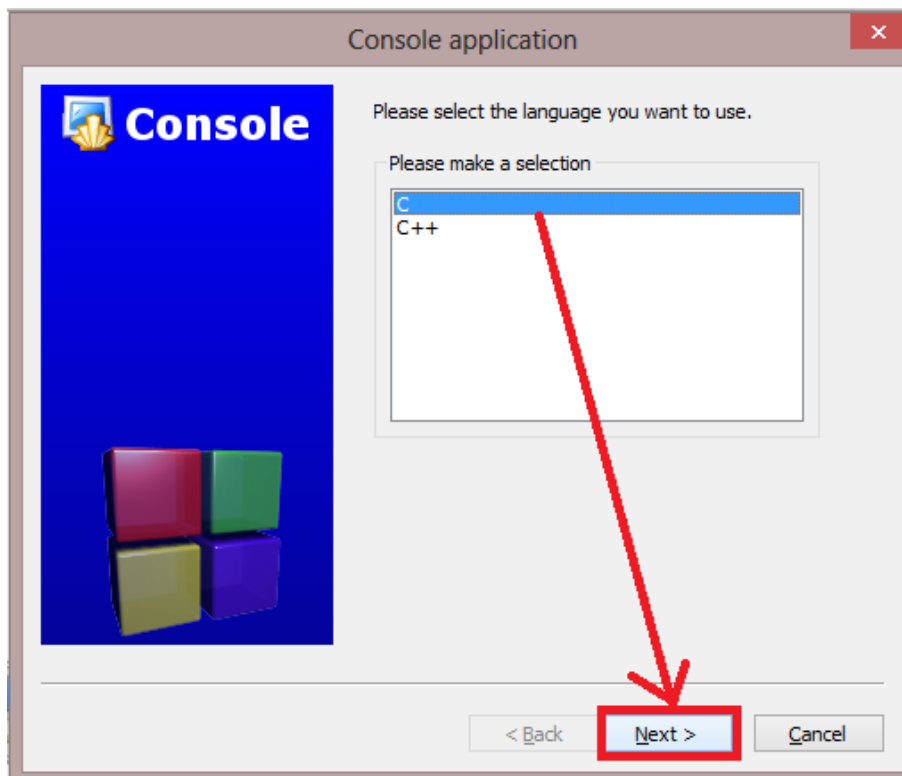
⚠ Như bạn thấy, Code::Blocks đề xuất rất nhiều loại chương trình khác nhau sử dụng các thư viện phổ biến như SDL (2D), OpenGL (3D), Qt và wxWidgets (Windows) ... Hiện tại thì những biểu tượng này chỉ để nhìn cho đẹp thôi chứ chúng vẫn chưa được cài đặt trên máy tính của bạn, bạn nên lướt qua chúng.

Chúng ta sẽ tập trung vào các loại chương trình khác ở các bài học sau này, thời gian này chúng ta sẽ phải nắm vững về “Console” trước, bởi vì thật sự bạn vẫn chưa đủ trình độ để làm việc với các loại chương trình khác đâu.

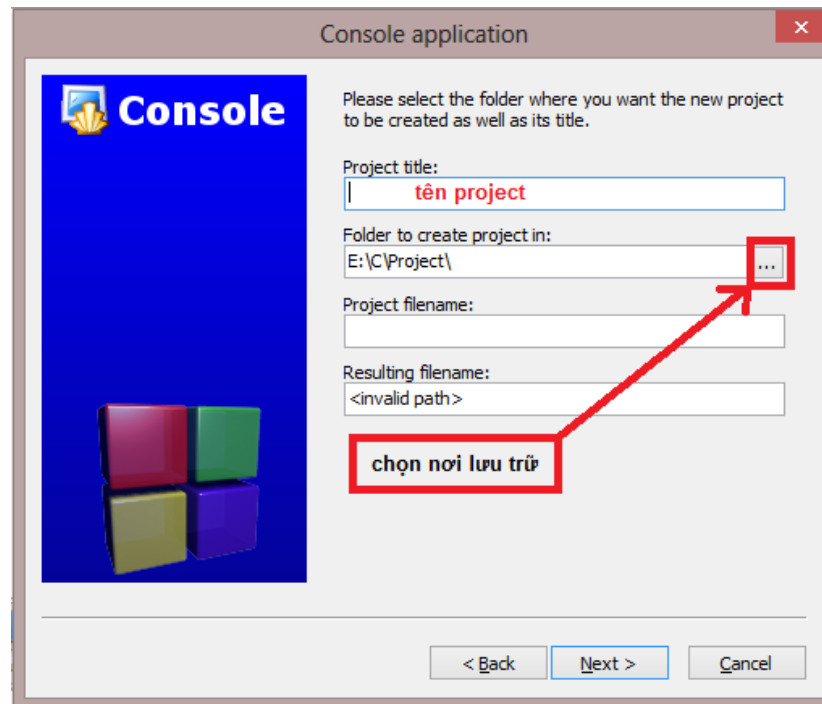
Tiếp theo bấm Go để tạo một project mới.



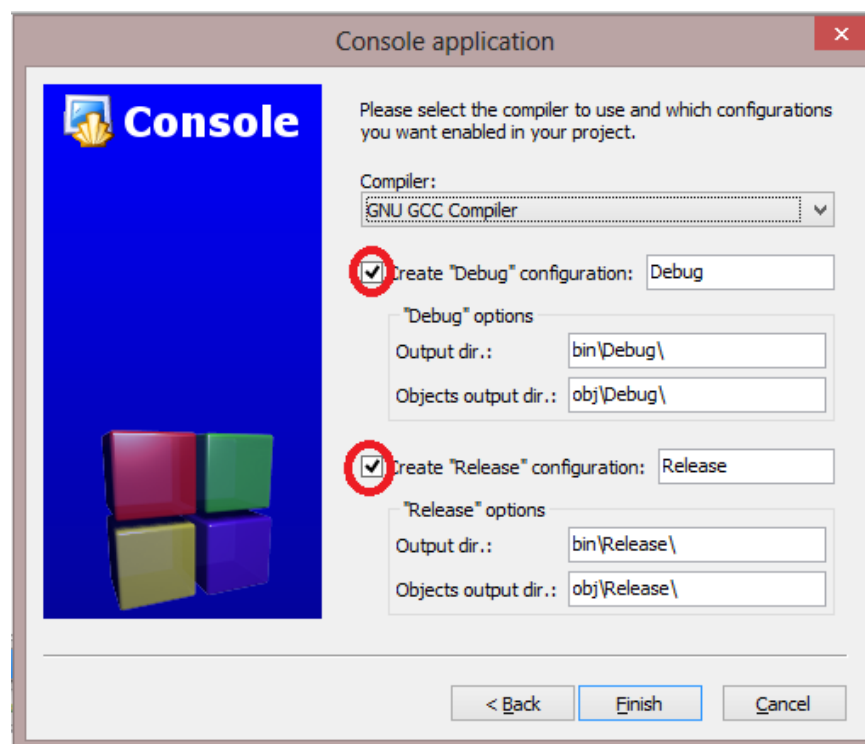
Chương trình sẽ hỏi bạn muốn tạo project cho ngôn ngữ C hay C++. Hãy chọn C và click next.



Chương trình sẽ yêu cầu bạn đặt tên cho project và chọn khu vực lưu trữ nó. Sau đó bấm next.



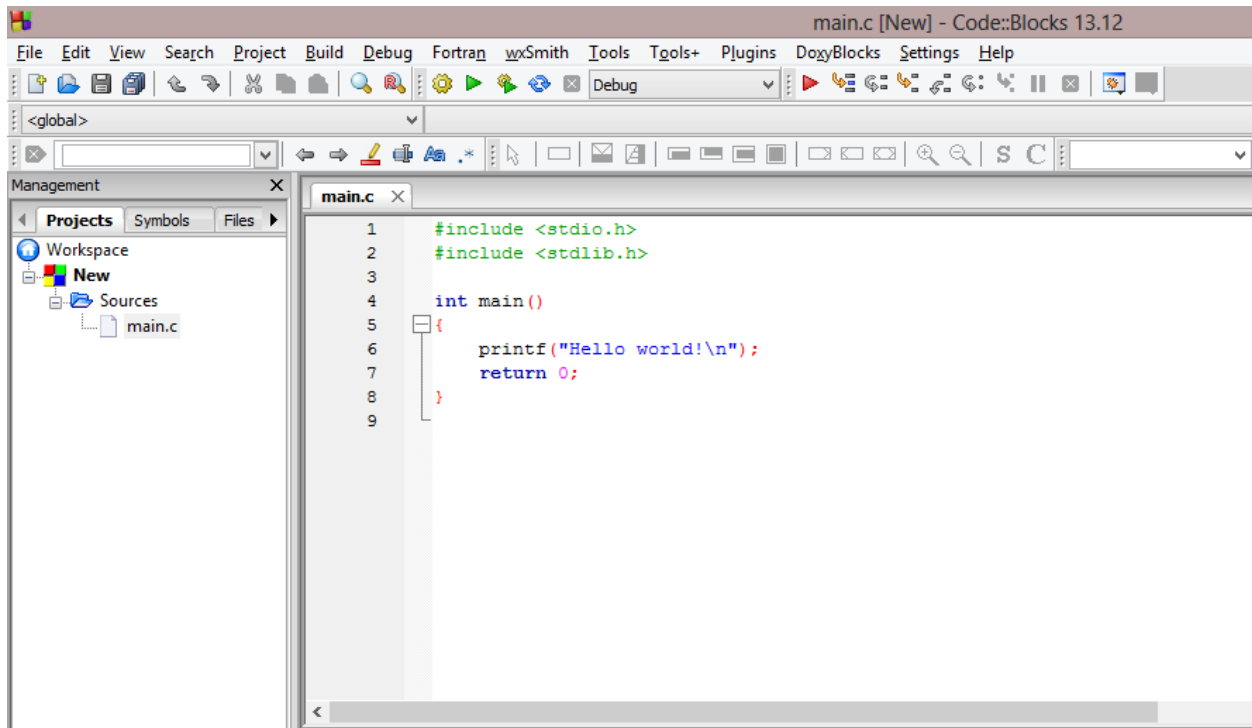
Ở cửa sổ cuối cùng này chúng ta có thể chọn compiler để biên dịch chương trình sau này. Tôi thường chọn compiler tên GNU GCC và để tất cả như mặc định. Đừng quên đánh dấu chọn vào 2 ô chức năng Debug và Release nhé.



Cuối cùng chỉ cần bấm Finish là xong.

Code::Blocks đã giúp bạn tạo một dự án mới với một chút mã nguồn (source code) được viết sẵn trong đó. Nhìn vào phía bên trái màn hình ở khu vực hiển thị danh sách tập tin mã nguồn. Để hiển thị các tập tin cần thiết. Ít nhất bạn sẽ thấy 1 tập tin đó là main.c

Trong tập tin này bạn sẽ thấy được vài dòng code mặc định được viết sẵn khi nhấp chuột trái 2 lần vào nó.



Vậy là các bước giới thiệu về IDE Code::Blocks đã xong và bạn đã biết cách tạo project cho riêng mình rồi đúng không.

Tạm thời tôi xin phép không hướng dẫn cách cài đặt 2 IDE còn lại (Visual Studio Express và Xcode). Nhưng tôi nghĩ các bạn hoàn toàn dư sức thực hiện những thao tác này đúng không. Còn về link tải phần mềm thì chỉ cần nhờ Google là được đúng không nào.

Tổng kết

- Để bắt đầu học lập trình bạn cần có tối thiểu 3 công cụ: Trình soạn thảo văn bản (text editor), trình biên dịch (compiler) cùng với một trình tìm và sửa lỗi (debugger).
- Các bạn có thể cài những chương trình trên riêng biệt hoặc cài một chương trình bao gồm 3 trong 1, gọi là IDE.
- Code::Blocks, Visual Studio Express, Xcode là những IDE rất phổ biến và thích hợp cho các bạn trong thời điểm hiện tại.

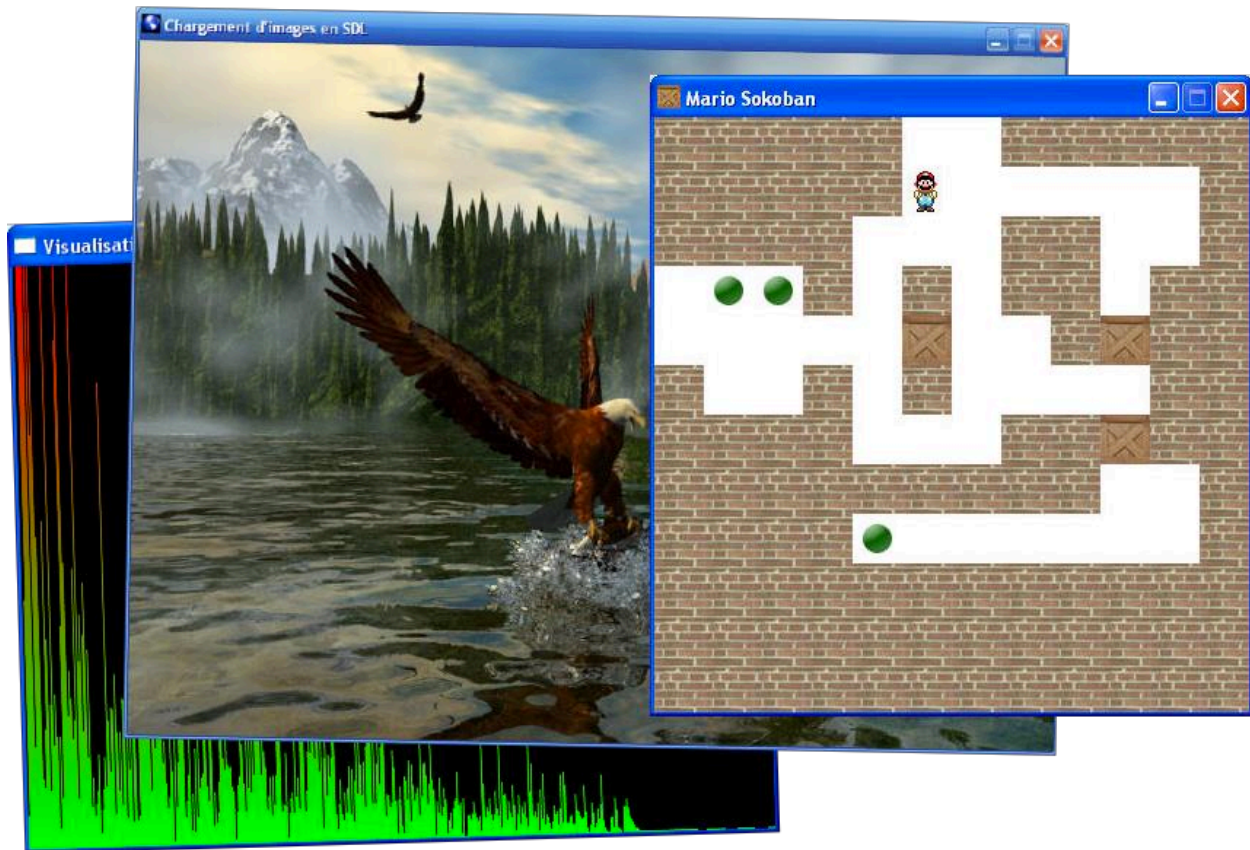
Bài 3: Chương trình đầu tiên của bạn

Chúng ta đã chuẩn bị xong sân chơi, chúng ta sẽ bắt đầu cuộc chơi ngay bây giờ, bạn đang cảm thấy thế nào?

Mục đích của phần hướng dẫn này giúp bạn có thể tạo ra **chương trình đầu tiên** cho chính mình!

Chương trình đầu tiên của bạn:

- Console hay cửa sổ ?
- Đoạn mã tối thiểu
- Viết một tin nhắn lên màn hình
- Những chú thích, khá tiện dụng !
- TRẮC NGHIỆM KIẾN THỨC.



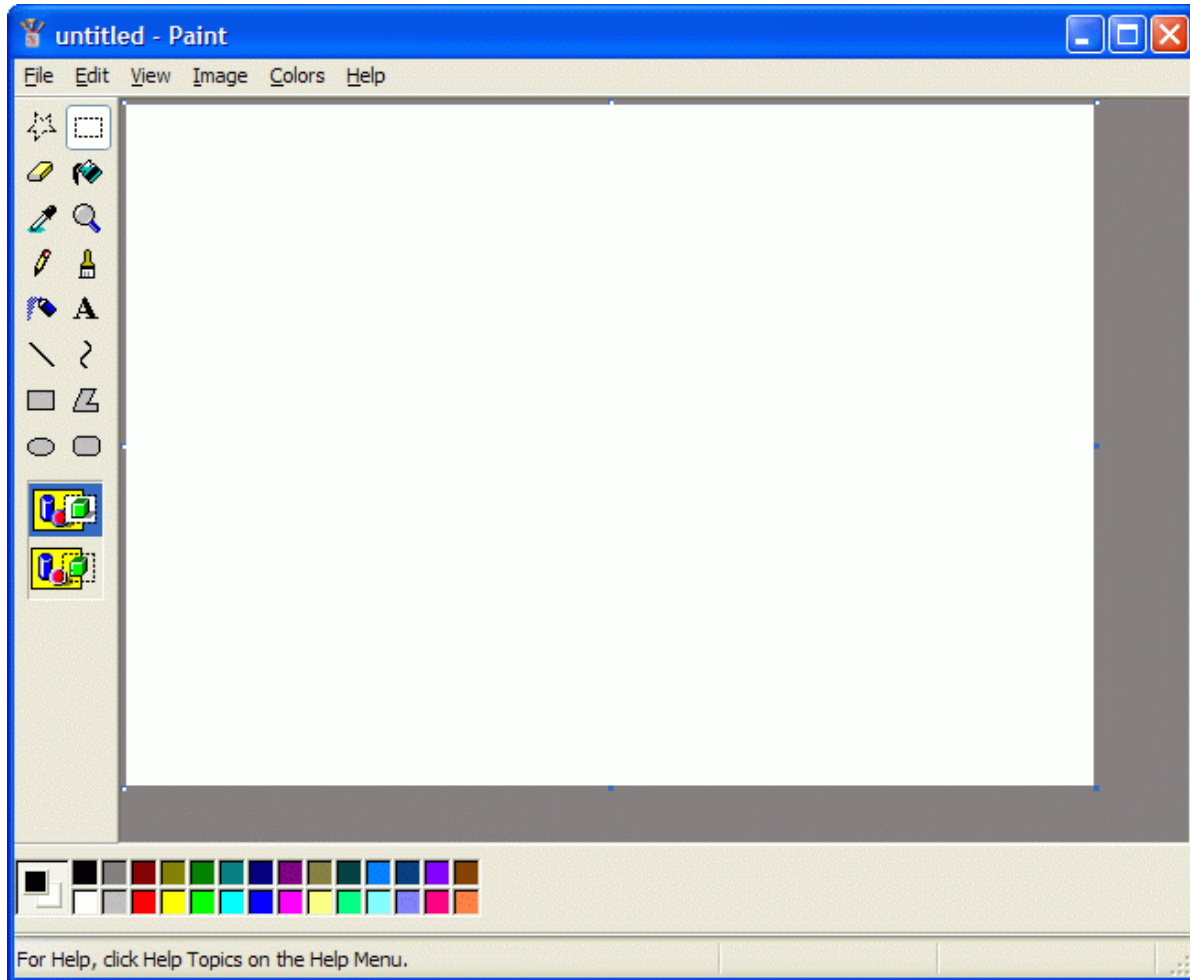
Console hay là cửa sổ?

Có 2 loại chương trình :

- Những chương trình dạng cửa sổ.
- Những chương trình dạng console.

Những chương trình dạng cửa sổ:

Tôi nghĩ rằng các bạn đã biết cái này, lấy một ví dụ điển hình:



Chương trình paint

Đó là một chương trình dạng cửa sổ, các bạn rất muốn tạo ra những chương trình như thế này đúng không?

Với C, chúng ta hoàn toàn có khả năng làm được. Nhưng các bạn chưa đủ sức tạo ra chúng vào lúc này. 😊

Tốt hơn là ta bắt đầu với việc tạo ra một chương trình dạng console.

? Nhưng chương trình dạng cửa sổ có giống với những chương trình dạng console không?

Những chương trình dưới dạng console:

Console chính là những chương trình xuất hiện đầu tiên trên thế giới. Vào thời kì đó, máy tính chỉ có khả năng tạo ra những dòng chữ đen và trắng và không đủ mạnh để hiển thị những cửa sổ nhiều màu sắc và hiệu ứng như bạn thấy hiện nay.

Sau đó, Windows đã cho ra đời máy tính có khả năng chạy những chương trình dạng cửa sổ. Vì vậy mà sản phẩm của họ được dùng rộng rãi, khiến phần lớn người sử dụng quên mất sự tồn tại của console.

Và tôi chắc là bạn đang muốn biết console là gì phải không? 😊

Tôi có một tin rất mới cho bạn đây! **console vẫn tồn tại!** Linux đã giữ lại sở thích sử dụng console. Và đây là hình dạng của console trên Linux:

```
2.2.5_appli.html    3.2.7.css          3.6.8.html
2.2.5.css           3.2.8.css          3.6.9.html
2.2.6_appli.html    3.2.9_appli.html   ancres.html
2.2.6.css           3.2.9.css          base.php
2.3.10_appli.html   3.3.10.html        cible_formulaire.php
2.3.10.css          3.3.11.css         cible.html
2.3.11_appli.html   3.3.12.css         design1.css
2.3.11.css          3.3.13_appli.html  erreur_paragraphe.html
2.3.12.css          3.3.13.css         essai2.css
2.3.13.html         3.3.14_appli.html  essai.css
2.3.14.css          3.3.14.css         images
2.3.15.html         3.3.15.css         tests_design.html
2.3.16.css          3.3.1.css          traitement.php
2.3.17.css          3.3.2.html
2.3.18.css          3.3.3.css
[root@ns1 exemples]# cd ..
[root@ns1 xhtml-css]# ls
anins          css.php          images          pseudoformats.php
annexes        design.php       images.php      qcm.php
autres         exemples        index.php      tableaux.php
boites_partiel.php formatage_partiel.php intro.php      texte.php
boites_partie2.php formatage_partie2.php liens.php      xhtml.php
conclusion.php  formulaires.php listes.php
[root@ns1 xhtml-css]#
```

Một ví dụ về console trong Linux

Đó là console và những đặc điểm cần chú ý là:

- Console ngày nay không chỉ hiển thị trắng và đen.
- Console không được những người mới sử dụng chào đón lắm.
- Console là một công cụ mạnh mẽ nếu như chúng ta biết cách sử dụng.

Viết một chương trình dạng console đơn giản và lý tưởng hơn cho những người mới học lập trình (sẽ không hề đơn giản nếu bắt đầu học bằng cách tạo ra một chương trình dạng cửa sổ)
Ghi thêm rằng, console ngày nay đã được cải tiến rất nhiều: hiển thị được nhiều màu sắc, và bạn có thể đặt một hình ảnh nào đó lên nền của console. Và đây là hình ảnh một console đã được tạo dựng khá hoành tráng trên HĐH linux 😊



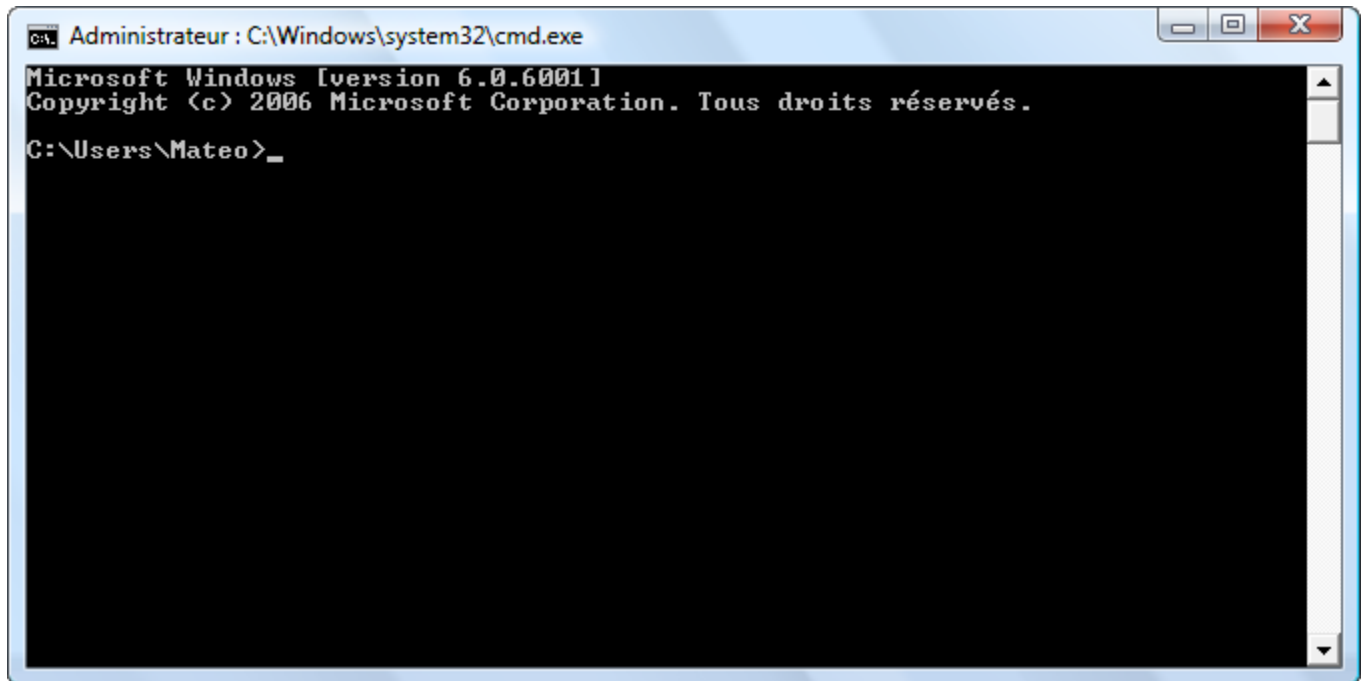
```
[ Wed Aug 17 03:55:04 ]
@ ~ ]--> cd .config
[ Wed Aug 17 03:55:26 ]
@ ~/.config ]--> ls
chromium  geeqie      hotkeys      nitrogen     Thunar       volumeicon
dconf     gnome-mplayer leafpad       openbox      tint2        xfce4
deadbeef  gpview       lxterminal   synapse      transmission xms2
dmenu     gtk-2.0      nautilus     terminator    Trolltech.conf

[ Wed Aug 17 03:55:30 ]
@ ~/.config ]--> cd openbox
[ Wed Aug 17 03:55:34 ]
@ ~/.config/openbox ]--> ls
autostart.sh  menu.xml  pipemenu     rc.xml  rc.xml.bak  scripts
[ Wed Aug 17 03:55:40 ]
@ ~/.config/openbox ]--> sudo leafpad rc.xml
Password:
[ Wed Aug 17 03:58:48 ]
@ ~/.config/openbox ]--> cd
[ Wed Aug 17 03:58:56 ]
@ ~ ]--> cd Images
[ Wed Aug 17 03:58:59 ]
@ ~/Images ]--> scrot -d 3 screenshot.png
Taking shot in 3.. 2.. 1.. 0.
[ Wed Aug 17 03:59:20 ]
@ ~/Images ]--> scrot -d 3 screenshot.png
Taking shot in 3.. 2.. 1.. 0.
```

Hê hê.. khá kinh dị 🐼

? Trên hệ điều hành Windows có console hay không?

Có nhưng nó đã bị giấu đi, ta có thể nói như thế. 😊
Bạn có thể gọi nó bằng cách vào Start => run => nhập "cmd".
Và đây chính là console của Windows, thật kì diệu:



Console trên Windows

Nếu bạn đang sử dụng Windows, chương trình đầu tiên bạn sắp tạo ra sẽ tương tự như thế.
Với việc bắt đầu từ console, bạn sẽ học được những kiến thức lập trình nền tảng cần thiết để có thể tạo ra những chương trình dạng cửa sổ về sau nên đừng nản chí nhé!

Những dòng code tối thiểu cần phải có.

Trên bất kỳ công cụ lập trình nào, chúng ta đều phải viết ra ít nhất một đoạn code, tuy rằng chúng không thực hiện điều gì nhưng đó là điều bắt buộc.
Đó là đoạn code tối thiểu mà ta sắp sửa tìm hiểu ngay sau đây. Hầu hết các chương trình viết bằng ngôn ngữ C đều phải sử dụng.

Tôi sẽ sử dụng IDE (Integrated Development Environment) **Code::Blocks** để hướng dẫn bạn.
Điều bạn cần làm sau khi mở **Code::Blocks** là tạo một project mới như tôi đã hướng dẫn ở bài trước (vào menu chọn *File / New / Project...*, chọn *Console Application* và chọn **ngôn ngữ C**).

Code::Blocks đã tạo sẵn một đoạn mã tối thiểu mà chúng ta cần:

C Code:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```



Cần ghi chú là có một dòng trắng ở cuối đoạn code. Được thực hiện bằng cách nhấn phím "ENTER" sau dấu " } ". Mọi tập tin C bình thường đều phải kết thúc bằng một dòng trắng và cũng không có gì nghiêm trọng nếu bạn không thực hiện nó, chỉ là compiler có thể sẽ hiển thị một thông tin warning để thông báo.

Ghi chú thứ 2 là dòng

```
int main ( )
```

cũng có thể được viết thành:

```
int main (int argc, char *argv[ ])
```

Cả hai cách viết đều đúng, nhưng cách viết thứ 2 thông dụng hơn rất nhiều. Tôi sẽ sử dụng cách viết này ở những bài hướng dẫn kế tiếp. Hiện giờ, bạn có sử dụng cách viết nào cũng không quan trọng vì ta vẫn chưa có đủ kiến thức để hiểu được ý nghĩa và cách hoạt động của chúng.

Nếu bạn đang sử dụng một IDE khác, hãy copy đoạn code ở trên vào file main.c

Hãy lưu lại. Tôi biết là chúng ta vẫn chưa làm gì cả, nhưng hãy lưu lại, đây là một thói quen tốt cần tập. Bình thường bạn chỉ dùng duy nhất một file source "main.c" (những file còn lại là file project được tạo bởi IDE của bạn).

Ý nghĩa đoạn mã tối thiểu ở trên:

Đoạn code đó với bạn thật rắc rối nhưng với tôi đó là đoạn code hiển thị một tin nhắn lên màn hình.

Chúng ta bắt đầu học cách đọc và hiểu chúng. 😊

Bắt đầu từ 2 dòng đầu tiên, chúng có vẻ giống nhau:

C Code:

```
#include <stdio.h>
#include <stdlib.h>
```

Đây chính là những dòng đặc biệt thường thấy ở đầu những file source và dễ dàng nhận biết vì nó bắt đầu từ dấu “#”. Ta gọi chúng là **preprocessor directives (những chỉ thị tiền xử lý)** vì nó sẽ được đọc bằng một chương trình gọi là preprocessor (chương trình tiền xử lý), chương trình này sẽ chạy đầu tiên khi ta thực hiện compilation.

Chúng ta đã thấy hình vẽ đơn giản về compilation ở chương trước. Nhưng quá trình đó thực sự không hề dễ dàng như vậy, có rất nhiều thứ diễn ra trong đó. Tôi sẽ nói sau này, tại thời điểm hiện tại, các bạn chỉ cần biết cách viết những dòng đầu tiên vào file của bạn là đủ.

❓ Nhưng những dòng đó nghĩa là gì? Tôi rất muốn biết điều đó!

Từ “include” tiếng Anh có nghĩa là đặt vào, bao gồm. Nó cho phép thêm vào project một số file. Những file này sẽ được sử dụng trong quá trình compilation.

Ở đây có 2 dòng, vậy là sẽ có 2 file được thêm vào. Những file này có tên là *stdio.h* và *stdlib.h*. Đó là những file đã tồn tại trước đó trong source và luôn sẵn sàng khi bạn gọi ra. Chúng ta thường gọi nó là **thư viện** (library). Và những file này chứa những đoạn code được viết sẵn cho phép hiển thị một đoạn văn lên màn hình.

⚠️ **Ghi chú:** Thư viện tiếng anh là “library”. Bạn hãy nắm vững nghĩa dịch chính xác của nó. Tôi nghĩ việt nam mình chỉ gọi là thư viện thôi nhỉ?

Nếu không có những file thư viện đó, ta không thể nào ghi được một đoạn văn lên màn hình.

Về nguyên tắc, máy tính của bạn sẽ không hiểu gì cả.

Tóm lại, 2 dòng đầu tiên đó cho phép ta ghi một tin nhắn lên màn hình "dễ dàng". 😊

C Code:

```
int main ()  
{  
    printf("Hello world!\n");  
    return 0;  
}
```

Cái mà bạn thấy ở trên, người ta gọi đó là một **function**. Một chương trình C hầu như cấu tạo bởi các function, Tại thời điểm này, chương trình của chúng ta chỉ có một function duy nhất.

Một function cho phép chúng ta tập hợp lại các lệnh cho máy tính, những lệnh này cho phép ta thực hiện chính xác một điều gì đó. Ví dụ, ta có thể viết một function “mở_một_tập_tin” trong đó chứa đựng những chỉ dẫn về cách mở một tập tin cho máy tính.

Lợi ích là, một khi function đã được viết ra, bạn không cần phải nói thêm gì nữa cả. Máy tính sẽ biết làm việc đó bằng cách nào. 😊

Vẫn còn quá sớm để chúng ta tìm hiểu chi tiết về những thành phần cấu tạo nên một function. Chúng ta chỉ xem xét những phần chính của nó. Ở câu đầu tiên, chữ thứ hai (**main**) là tên của function. Theo nguyên tắc, main là một tên đặc biệt, nó chỉ dùng để đặt cho function chính của chương trình, và lúc nào chương trình cũng sẽ bắt đầu từ function main.

Một function luôn có mở đầu và kết thúc, giới hạn bởi những dấu { và }. Tất cả function main của chúng ta đều nằm trong đó. Nếu bạn đã theo kịp những gì tôi đã nói, thì function main của chúng ta gồm 2 dòng:

C Code:

```
printf("Hello world!\n");  
return 0;
```

Ta gọi những dòng nằm trong một function là các **instruction**. (Hãy nắm vững những từ ngữ này 😊).

(instruction: chỉ thị, chỉ dẫn, câu lệnh)

Mỗi một instruction là một lệnh dành cho máy tính, và nó yêu cầu máy tính phải thực hiện chính xác một hành động gì đó.

Như tôi đã nói với bạn, công việc của những người lập trình là động não để viết những instruction, và khi bạn đã thành thục, bạn sẽ có thể tạo ra những function như function “mở_một_tập_tin” hay function “nhân_vật_đi_tới” trong một game nào đó. 😊

Một chương trình không gì khác hơn là tạo nên một dãy các instruction: instruction “hãy làm cái này” instruction “hãy làm cái kia”... Bạn ra những lệnh đã được sắp đặt và máy tính sẽ thực hiện các lệnh đó.



Quan trọng: Tất cả các instruction đều kết thúc bằng một dấu chấm phẩy “;”. Hay nói khác hơn đó là đặc điểm nhận biết một instruction. Nếu bạn quên chúng, chương trình của bạn sẽ không dịch được.

Dòng đầu tiên:

C Code:

```
printf("Hello world!\n");
```

Yêu cầu máy tính hiển thị lên màn hình "Hello world!". Khi chương trình bạn chạy đến dòng này, nó sẽ hiển thị tin nhắn ra màn hình, sau đó chuyển sang instruction kế tiếp.

C Code:

```
return 0;
```

Có nghĩa là đã kết thúc, 😊 dòng này biểu thị rằng ta đã đến giai đoạn kết thúc function main và yêu cầu gửi giá trị 0.

❓ **Vậy thì tại sao chương trình phải trở về số 0?**

Trên thực tế, mỗi chương trình khi kết thúc sẽ gửi về một giá trị, ví dụ như để nói rằng tất cả hoạt động tốt (0= tất cả hoạt động tốt, những số khác có nghĩa là “error”). Hầu như những giá trị này không hề được sử dụng, nhưng thực tế nó vẫn tồn tại.

Chương trình của bạn cũng có thể chạy khi không có **return 0**; nhưng sẽ chính xác và đúng hơn nếu ta thêm vào. 😊

Vậy là! Chúng ta đã tìm hiểu một ít về cách hoạt động của đoạn mã tối thiểu trên.

Hẳn là các bạn vẫn còn một số nghi vấn khác vì chúng ta đã không tìm hiểu sâu lắm. Nhưng bạn hãy yên tâm, tất cả những câu hỏi sẽ từng tí từng tí một được giải đáp. Tôi không muốn giải thích cho bạn tất cả trong một lần, nếu không đầu óc bạn sẽ hoàn toàn rối bèm, tôi đảm bảo. 😊

Đến giờ, bạn vẫn theo kịp tôi đúng không? Bạn không cần thiết phải cố gắng đọc hết một mạch đâu. Hãy nghỉ ngơi và sau đó làm việc với tinh thần minh mẫn nhất.

Tất cả những gì tôi vừa hướng dẫn cho bạn đều là nền tảng, còn nếu bạn cảm thấy không có vấn đề gì thì ta tiếp tục.

Tôi sẽ vẽ cho bạn lại một biểu đồ tổng hợp với những từ ngữ ta vừa học:

```
#include <stdio.h>
#include <stdlib.h> } Preprocessor Directives

int main()
{
    printf("Hello world!\n");
    return 0; } Instructions } Function
```

Test chương trình

Nhanh thôi, bạn chỉ cần biên dịch chương trình rồi chạy. (Nhấn vào nút Build & Run trong Code::Blocks).

Nếu bạn vẫn chưa lưu file lại, Code::Blocks sẽ yêu cầu bạn save file lại, hãy thực hiện điều đó.



Nếu compilation không thực hiện được và bạn có lỗi dạng “My-program - Release” uses an invalid compiler. Skipping... Nothing to be done ...” Điều đó có nghĩa là bạn đã tải và sử dụng phiên bản Code::Blocks không có **mingw** (compiler). Hãy quay về site Code::Blocks tải về phiên bản có mingw.

Và đây là chương trình đầu tiên của bạn:

Chương trình đầu tiên của bạn!

Chương trình hiển thị "Hello world!" (dòng thứ nhất).

Những dòng kế tiếp được tạo ra bởi Code::Blocks và giải thích rằng chương trình đã được chạy trong khoảng thời gian 0.021s kể từ lúc bắt đầu.

Sau đó Code::Blocks yêu cầu bạn nhấn vào một phím bất kì để đóng cửa sổ lại. Chương trình của bạn sẽ dừng lại.

Vâng, tôi biết rằng cái đó chẳng có nghĩa gì cả, giống như một trò đùa nhưng đó là tất cả những gì bạn vừa học được. 😊

Nhưng dù sao, đó cũng là chương trình đầu tiên của bạn, hãy nhớ lại cảm giác đó, có thể nó sẽ theo bạn suốt cả đời đây. 😊

Không phải vậy sao ?...

Trước khi bạn cho tôi thấy vẻ mặt của bạn lúc này, tôi xin phép chúng ta bước sang phần tiếp theo, không chậm trễ. 😊

Viết một tin nhắn lên màn hình

Kể từ bây giờ, chúng ta sẽ tự viết code của mình vào chương trình.

Nhiệm vụ của các bạn là hiển thị tin nhắn “Xin chào” lên màn hình.

Giống như trước đó console sẽ mở ra. Tin nhắn “Xin chào” sẽ xuất hiện trong đó.

❓ **Làm cách nào để viết một tin nhắn lên màn hình?**

Việc này khá đơn giản. Nếu bạn sử dụng lại đoạn code ở trên, bạn chỉ cần thay "Hello world!" bằng "Xin chào" trong câu có chứa printf.

Tôi đã nói printf là một **instruction**. Nó ra lệnh cho máy tính: “*Hãy hiển thị cho tôi một tin nhắn lên màn hình*”.

Cần biết thêm rằng **printf** là một function đã được viết bởi những lập trình viên đi trước.

❓ **function này ở đâu ? Tôi chỉ thấy tồn tại mỗi function main mà thôi !**

Bạn có nhớ hai dòng này chứ ?

C Code:

```
#include <stdio.h>
#include <stdlib.h>
```

Tôi đã nói với bạn rằng nó cho phép ta thêm vào chương trình những thư viện. Và những thư viện đó chứa đầy những function đã được viết sẵn bên trong. `stdio.h` chứa đựng những function cho phép hiển thị một cái gì đó lên màn hình (ví dụ như function `printf`), nhưng nó đòi hỏi người sử dụng phải đánh ra một cái gì đó (đây là những function mà ta sẽ thấy sau này).

Máy tính, chào bạn đi!

Trong function `main`, chúng ta gọi function `printf`.

Để gọi một function rất đơn giản: ta chỉ cần ghi ra tên của nó, kế tiếp là mở ngoặc đóng ngoặc "()", và một dấu chấm phẩy ";".

```
printf ( );
```

Nhưng công việc của bạn vẫn chưa xong đâu. Chúng ta phải cho function `printf` một tin nhắn để hiển thị. Hãy mở ngoặc "()" sau `printf`. Trong đó, mở ngoặc kép "\" . Cuối cùng đánh điều gì bạn cần máy tính hiển thị bên trong.

C Code:

```
printf ("Xin chào");
```

Tôi hi vọng rằng bạn không quên mất dấu chấm phẩy ";" ở cuối cùng, tôi nhắc lại là nó rất quan trọng! Nó cho phép máy tính hiểu rằng instruction của ta kết thúc ở đây.

Và đây là code source mà bạn phải có được:

C Code:

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
    printf ("Xin chào");
    system ("PAUSE");
    return 0;
}
```

Chúng ta có 3 instruction yêu cầu máy tính thực hiện:

1. Hiện thị “Xin chào” lên màn hình.
2. Đưa chương trình vào giai đoạn nghỉ, hiện thị tin nhắn *"Press any key to continue"* và chờ đợi cho đến khi ta đánh thêm 1 phím bất kì lên bàn phím để chuyển sang instruction tiếp theo.
3. Function *main* kết thúc, trả về 0. Chương trình kết thúc.

❓ Việc đưa chương trình vào trạng thái nghỉ có ý nghĩa như thế nào? Chúng ta có được phép xóa đi câu lệnh `system("PAUSE")` hay không?

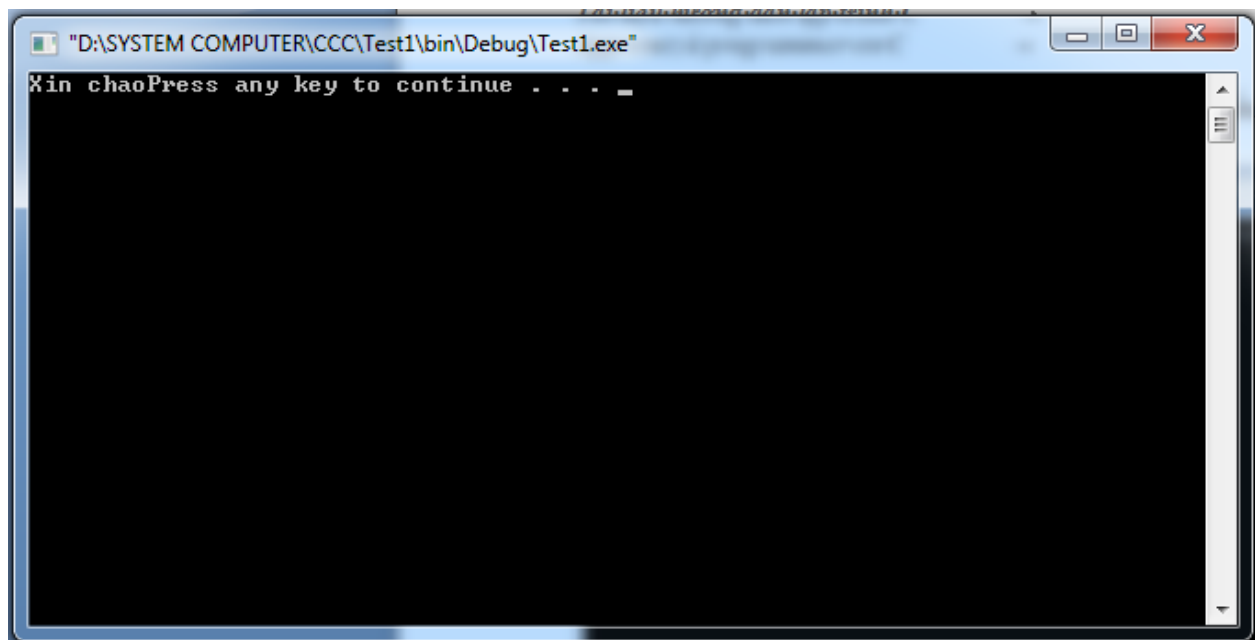
Có chứ, 😊 chắc chắn là bạn có thể. Hãy thử chạy chương trình không có instruction này và bạn sẽ thấy.

Chương trình sẽ không dừng lại. Nói rõ hơn là, máy tính sẽ hiện thị tin nhắn “Xin chào” và tắt chương trình. Cửa sổ của console sẽ hiện ra và biến mất với vận tốc ánh sáng, bạn sẽ không có đủ thời gian để nhận ra điều gì.

Thật ngu ngốc, phải không? 😏

Ghi thêm là, với một số IDE, như là tôi đã nói trước đó, nó sẽ tự động dừng lại ở cuối chương trình. Trong trường hợp đó instruction `system("PAUSE")` coi như vô dụng, bạn có thể xóa nó đi.

Và chúng ta hãy test chương trình với pause, và nó sẽ hiện thị:



Cuối cùng, chương trình hiện thị "Xin chào" đã được hoàn thành.

❓ Nhưng thật sự nó không hoàn toàn hiển thị “xin chào”, có một dòng khác cùng hiển thị sau nó.

Thưa bạn, không có việc gì nghiêm trọng ở đây cả, chúng ta sẽ học cách sửa chữa nó ngay đây.

Bạn muốn kết quả sẽ đưa ra màn hình một dòng khác nằm dưới dòng “Xin chào” của chúng ta, tương tự như việc gõ phím "enter" để xuống dòng khi chat vậy.

Tất nhiên khi chat hay viết code source bạn sẽ xuống dòng bằng cách nhấn enter, nhưng chúng ta đang nói đến việc xuống dòng cho đoạn văn được in ra màn hình console.

Để làm điều đó chúng ta phải sử dụng những kí tự đặc biệt.

Những kí tự đặc biệt:

Những kí tự đặc biệt là những kí tự cho máy tính hiểu rằng ta muốn xuống dòng hay nhấn tab để cách khoảng ...

Những kí tự này tương đối dễ dàng nhận biết. Trước chúng lúc nào cũng có một dấu anti-slash “\”, kế tiếp là một chữ cái hay một số, \n và \t là 2 kí tự đặc biệt được sử dụng khá thường xuyên mà bạn chắc chắn cần dùng. Bên cạnh đó tôi sẽ cung cấp cho bạn 1 danh sách các ký tự đặc biệt khác để tham khảo trong trường hợp bạn cần đến chúng.

Escape sequence	Hex value in ASCII	Character represented
\a	07	Alarm (Beep, Bell)
\b	08	Backspace
\f	0C	Formfeed
\n	0A	Newline (Line Feed); see notes below
\r	0D	Carriage Return
\t	09	Horizontal Tab
\v	0B	Vertical Tab
\\	5C	Backslash
\'	27	Single quotation mark
\"	22	Double quotation mark
\?	3F	Question mark
\nnn	any	The character whose numerical value is given by nnn interpreted as an octal number
\xhh	any	The character whose numerical value is given by hh interpreted as a hexadecimal number

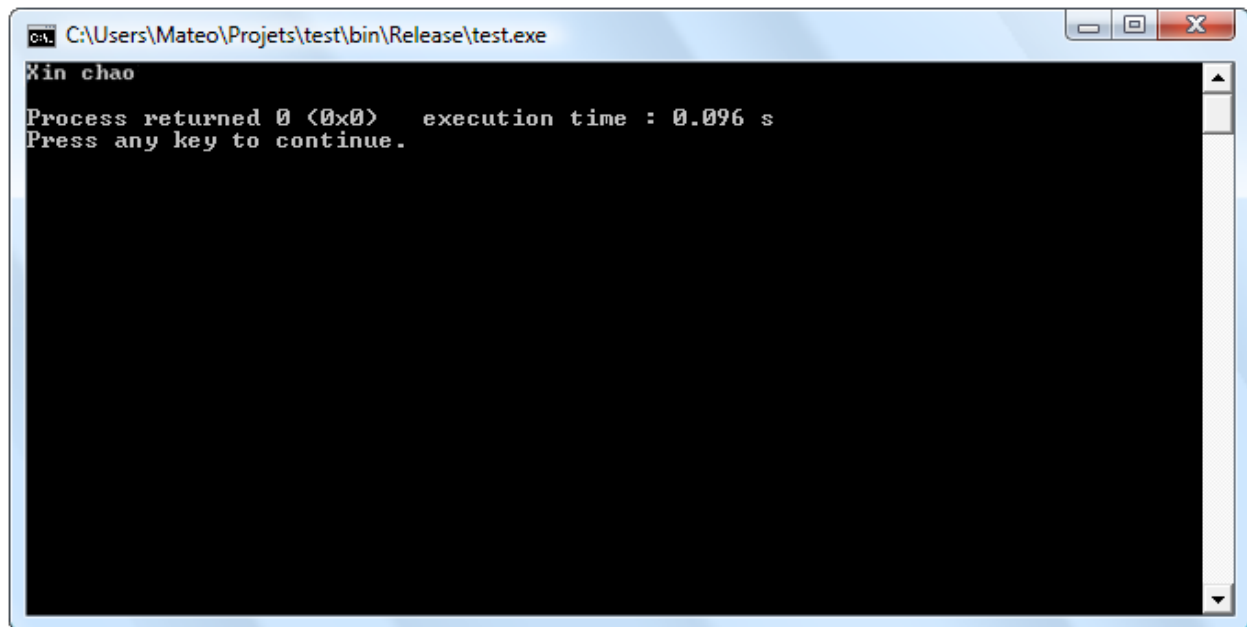
Danh sách các ký tự đặc biệt bạn có thể sử dụng khi lập trình

Trong trường hợp này, chúng ta chỉ cần thêm vào `\n` để xuống dòng.

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
    printf ("Xin chao\n");
    system ("PAUSE");

    return 0;
}
```

Và bây giờ chương trình của bạn đã rõ ràng hơn rồi.



Một chương trình hiển thị rõ ràng

⚠ Bạn có thể viết trong printf duy nhất một ký tự `\n`, điều đó có nghĩa là bạn muốn xuống dòng ở câu kế tiếp. Bạn hãy tập viết những câu thể này:
`printf ("Xin chao\nTam biet\n");`
Nó sẽ hiển thị “Xin chao” ở câu đầu tiên và “Tam biet” ở câu kế tiếp.

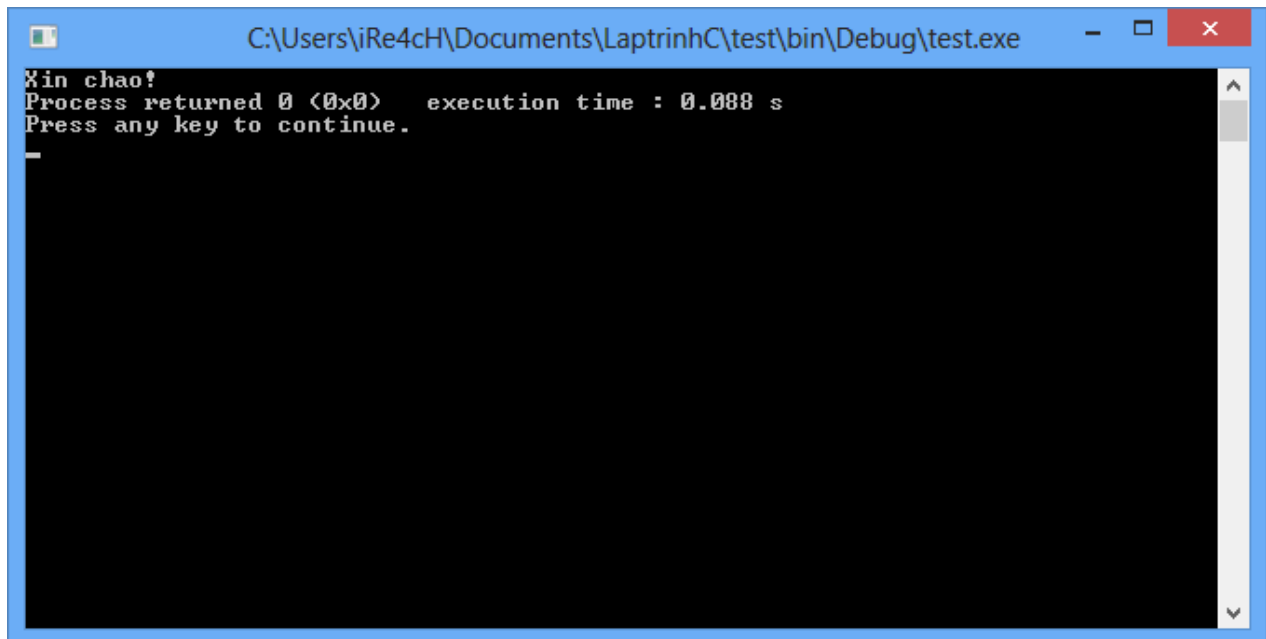
Ví dụ khi sử dụng Code::Blocks phiên bản mới:

Cũng là một chương trình in ra màn hình câu “Xin chào!” nhưng khi viết bằng Code::Blocks phiên bản hiện tại thì chỉ đơn giản như sau:

C Code:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    printf ("Xin chào!");
    return 0;
}
```

Chương trình hiển thị:



Code::Blocks phiên bản mới chương trình tự động dừng lại và tự động xuống dòng ở cuối.

Bạn có thể nhận ra rằng khi dùng Code::Blocks, bạn không cần phải thêm `\n` để xuống dòng cũng như câu lệnh `system ("PAUSE")` để dừng chương trình như những dòng code tôi đã hướng dẫn bạn trước đó.

Code::Blocks đã thay chúng ta làm việc đó (IDE này khá thông minh đúng không) 😊

Hội chứng Gérard

❓ Xin chào, tôi tên là Gérard và tôi muốn sửa đổi chương trình với tên là “Hello Gérard”. Chỉ vậy thôi, nhưng thật bất ngờ khi máy tính không hiển thị chính xác những gì tôi muốn. Tôi phải làm gì bây giờ?

Đầu tiên xin chào bạn, Gérard 😊

Đây là một câu hỏi khá hay dành cho tôi, và tôi rất vui khi thấy rằng bạn đã bắt đầu có những ý tưởng cải tiến chương trình.

Và đây là câu trả lời, tôi có một tin hơi buồn dành cho bạn: console trên Windows không hiển thị được những dấu trọng âm, nhưng ngược lại trên Linux ta có thể làm điều đó. 😊

Trong trường hợp này bạn có 2 lựa chọn:

- **Chuyển sang Linux:** lựa chọn này khá là phức tạp vì lúc đó tôi phải giải thích cho riêng bạn cách sử dụng Linux. Nếu bạn chưa đủ trình độ để sử dụng vào lúc này, hãy quên lựa chọn này đi.
- **Không sử dụng những dấu trọng âm.** Cách này hơi miễn cưỡng nhưng lúc này bạn phải lựa chọn nó. Console của Windows có những hạn chế, nó chỉ hiển thị những tin nhắn không có dấu.

Và bạn sẽ ghi là:

C Code:

```
printf("Hello Gerard\n");
```

Tôi xin cảm ơn bạn Gérard đã giúp tôi nhớ lại vấn đề này

ps: Nếu tên các bạn cũng có dấu như bạn Gérard, thì cũng làm tương tự vậy nhé.

Những lời chú thích, vô cùng tiện dụng!

Trước khi kết thúc phần này, tôi nhất thiết phải chỉ cho bạn một cái khá hay, mà ta gọi chúng là các comment. Trên các ngôn ngữ lập trình ta luôn có thể thêm vào những ghi chú vào trong mã nguồn của bạn. Và đối với ngôn ngữ C bạn cũng có thể làm như vậy.

Có nghĩa là bạn thêm vào một đoạn văn vào code source để giải thích là phải làm gì ở đó, dòng này có nhiệm vụ gì, kí hiệu này cho mục đích gì ..v.v..

Đó thật sự là một điều không thể thiếu vì kể cả những thiên tài về lập trình cũng cần phải thêm vào các chú thích ở đây hay ở kia. Những ghi chú này sẽ giúp bạn có thể:

- Dễ dàng đi vào trọng tâm của những gì bạn đã viết. Vì ta có thể dễ dàng quên mất nguyên tắc hoạt động chương trình mà bạn đã viết. 😓 Bạn có thể mất nhiều ngày để suy nghĩ lại điều đó, bạn sẽ cần những chú thích của bản thân bạn để có thể tự hiểu lại ý nghĩa của việc mình làm.
- Nếu bạn đưa mã nguồn của bạn cho một ai khác và nếu người đó không hiểu nhiều lắm về nguyên tắc hoạt động chương trình của bạn, thì những ghi chú đó sẽ giúp họ làm quen nhanh hơn.
- Cuối cùng, cái đó cho phép tôi có thể thêm những chú thích vào những đoạn mã trong bài học khi hướng dẫn cho bạn. Điều đó giúp tôi giải thích cho bạn tốt hơn về tác dụng của những dòng code.

Có nhiều cách để thêm vào một lời chú thích. Tất cả phụ thuộc vào chiều dài của lời chú thích mà bạn muốn viết:

- Nếu **ngắn**: chỉ gồm 1 dòng, hoặc vài từ. Trong trường hợp đó bạn đánh vào double slash (//) sau đó là chú thích của bạn.

Ví dụ:

C Code:

```
// Đây là một chú thích ngắn.
```

hoặc

```
printf("Xin chào"); // instruction này hiện thị lên màn hình "Xin chào"
```

- Nếu lời chú thích của bạn **dài**: bạn có nhiều cái để thuật lại, bạn cần viết rất nhiều câu và trên rất nhiều dòng. Trong trường hợp này :
 - i. Để mở đầu lời chú thích: hãy đánh một slash sau đó đánh dấu sao (/*)
 - ii. Để kết thúc: Đánh dấu sao rồi sau đó là slash (*/)

Ví dụ:

C Code:

```
/* Đây là một chú thích gồm nhiều dòng */
```

Trở lại với chương trình hiển thị “Xin chào”, và thêm vào những lời chú thích để luyện tập:

C Code:

```
/*
Sau đây là những preprocessor directives.
Nhưng dòng này cho phép thêm một số file vào project của bạn, nhưng file này thường được
chung ta gọi tên là “thu viện”
Nho vào các file thu viện, chúng ta luôn có những hàm sẵn sàng làm việc.
ví dụ như hàm printf: hiển thị một đoạn văn lên màn hình
*/
#include <stdio.h>
#include <stdlib.h>
/*
Sau đây là function chính của chương trình tên là “main”. Nho function này mà chương trình của
bạn có thể bắt đầu
Chương trình này sẽ hiển thị “Xin chào” lên màn hình, đưa chương trình vào trạng thái pause,
kết thúc
*/
int main(int argc, char *argv[])
{
    printf("Xin chào"); // instruction này hiển thị “Xin chào” lên màn hình
    return 0; // Chương trình trả về giá trị 0 và kết thúc
}
```

Trên đây là một chương trình với những dòng chú thích 😊

Khi ta biên dịch chương trình, tất cả những chú thích sẽ được bỏ qua, máy tính sẽ không đọc các dòng này. Những chú thích sẽ không xuất hiện khi ta chạy chương trình, chúng chỉ dành cho những người lập trình.

Bình thường thì ta không ghi chú ở mỗi dòng code của chương trình. Tôi đã nói rằng viết chú thích trong code source là một điều quan trọng nhưng chúng ta cần biết khi nào cần dùng đến, vì chú thích từng dòng như vậy sẽ tốn thời gian vô ích.

VD như khi mọi người đã biết rằng printf là hàm hiển thị một tin nhắn lên màn hình, bạn không cần phải chú thích thêm nữa về tác dụng của nó mỗi lần lập trình. 😊

Tốt hơn là bạn hãy chú thích nhiều cái trong một lần, chẳng hạn như để giải thích ý nghĩa của một dãy instruction nào đó, nó sẽ được sử dụng vào việc gì.

Và người lập trình chỉ cần ngó qua những lời chú thích, họ sẽ tự hiểu lấy toàn bộ.

Nắm vững: Những lời chú thích hướng dẫn người lập trình trong code source, nó cho phép chúng ta nhận ra nó, vì vậy hãy tập chú thích từng nhóm cùng lúc hơn là bạn chú thích cho từng dòng.

Và để kết thúc bài học này, tôi xin trích dẫn **một luật của IBM**:

Nếu đọc những chú thích mà bạn không hiểu chương trình hoạt động thế nào, hãy xóa bỏ tất cả.

Như bạn nhận thấy, chúng ta vẫn chưa hoàn toàn kết thúc hết toàn bộ bài học.

Và đây cũng là lần đầu tiên bạn thấy thế nào là mã lập trình thật sự, các từ ngữ, các kí hiệu, có thể khiến đầu óc hơi choáng váng một tí.

Thật ra điều đó cũng bình thường thôi, tất cả ai cũng đều như vậy trong lần đầu tiên. 😊

Trước khi bạn bước sang một giai đoạn mới, bạn hãy test lại những gì bạn đã biết.

Tôi cố tránh việc dạy bạn nhiều thứ trong một lúc, đơn giản là bạn sẽ không lãnh ngộ được gì cả nếu bạn học một cách quá nhanh và nhồi nhét.

Và tôi xin báo trước cho bạn biết, trong các phần tiếp theo sẽ có rất nhiều điều mới lạ mà bạn chưa biết.

TRẮC NGHIỆM KIẾN THỨC.

<p>❓ Một dòng preprocessor directives được bắt đầu bởi</p> <p>A. #</p> <p>B. {</p> <p>C. //</p>
<p>❓ Tên của function chính trong chương trình là ?</p> <p>A. printf</p> <p>B. master</p> <p>C. main</p>
<p>❓ Thư viện là gì?</p> <p>A. Những file source đã được viết trước đó gồm các function luôn sẵn sàng chờ bạn gọi ra.</p> <p>B. Một file cho phép bạn viết một đoạn văn lên màn hình</p> <p>C. Một nơi để ta có thể mượn những quyển sách về khoa học viễn tưởng</p>
<p>❓ Một instruction luôn được kết thúc bởi kí tự nào ?</p> <p>A. /*</p> <p>B. ;</p> <p>C. }</p>
<p>❓ Tên của hàm cho phép hiển thị một đoạn văn lên màn hình ?</p> <p>A. printf</p> <p>B. print</p> <p>C. afficher</p>
<p>❓ Kí tự nào cho phép ta xuống dòng khi hiển thị tin nhắn lên màn hình console?</p> <p>A. \t</p> <p>B. \n</p> <p>C. Chỉ đơn giản là nhấn phím enter để xuống dòng.</p>
<p>❓ Chú thích chỉ dành cho một dòng bắt đầu bởi :</p> <p>A. /*</p> <p>B. */</p> <p>C. //</p>

Đáp án:

1 – A

2 – C

3 – A

4 – B

5 – A

6 – B

7 – C

Bài 4: Thế giới của các biến số (variable)

Đây là một chương quan trọng, và bạn cần phải tập trung nhiều (Nói cách khác, đây không phải thời điểm để bạn phân tích đường bay của một con ruồi đang quanh quẩn bên cạnh). 😊

Tóm tắt lại những gì đã học:

Ở những bài trước trước, bạn đã được học cách để tạo một project mới trên IDE Code::Blocks. Tôi đã đặc biệt giải thích với bạn rằng việc tạo ra một chương trình trên cửa sổ khá phức tạp (và tôi cũng không nói với bạn về việc tạo ra một game 3D chơi trên mạng 😊).

Chúng ta bắt đầu học lập trình với việc học cách làm việc trên console. Và chúng ta đã học những điều khá hay ho như việc hiển thị một tin nhắn lên màn hình.

Tôi biết rằng bạn sắp sửa bảo với tôi rằng cái đó chưa giúp bạn điều gì cả. 😊

Và tại thời điểm này, bạn vẫn chưa biết cách làm thế nào gọi ra một **biến số**, thứ mà tất cả những ngôn ngữ lập trình như C đều bắt buộc phải sử dụng.

Nào chúng ta hãy nói về nó !

❓ **Vậy thì chính xác biến số là gì ?**

Tôi sẽ giải thích tất cả về nó trong phần này, bạn sẽ không phải chờ đợi lâu đâu, nhìn một cách tổng quát chúng ta sẽ học cách đưa những con số vào trong bộ nhớ của máy tính.

Tôi sẽ bắt đầu với những lời giải thích về bộ nhớ của máy tính, nguyên tắc hoạt động, máy tính có thể nhớ bao nhiêu thứ khác nhau?

Vấn đề này có thể đơn giản đối với một số người, nhưng bài giảng của tôi chỉ dành cho những người vẫn chưa biết bộ nhớ máy tính là gì.

Phần này ta sẽ học :

Công việc của bộ nhớ:

- Khai báo một biến số
- Hiển thị giá trị một biến số
- Lưu lại giá trị được chọn
- TRẮC NGHIỆM KIẾN THỨC.

Công việc của bộ nhớ

Bài giảng này có mối liên hệ trực tiếp với bộ nhớ của máy tính.

Con người cũng như máy tính đều cần lưu giữ lại một số cái gì đó, con người chỉ có duy nhất bộ não nhưng trên máy tính thì có nhiều dạng bộ nhớ khác nhau.

❓ Tại sao máy tính cần nhiều loại bộ nhớ khác nhau? Một bộ nhớ duy nhất không đủ cho máy tính, có phải vậy không?

Không, thực tế người ta chỉ cần một bộ nhớ có **tốc độ lưu nhanh** và **khả năng chứa lớn** (để có thể lưu lại nhiều thứ quan trọng).

Nhưng cho đến thời điểm hiện tại, chúng ta vẫn chưa tạo được những bộ nhớ giống như vậy. Vì các bộ nhớ nhanh thì đắt tiền nên các bộ nhớ được tổ chức thành nhiều cấp, cấp có dung lượng ít thì nhanh nhưng đắt tiền hơn cấp có dung lượng cao hơn. Những bộ nhớ có tốc độ lưu càng nhanh sẽ có dung lượng càng nhỏ.

Vậy máy tính của chúng ta được lắp đặt gồm:

- Những bộ nhớ có tốc độ lưu nhanh nhưng khả năng chứa nhỏ.
- Những bộ nhớ có tốc độ lưu chậm nhưng khả năng chứa lớn hơn rất nhiều.

Bạn vẫn theo kịp tôi chứ 😊

Những khác biệt về bộ nhớ:

Để cho bạn dễ hiểu, đây là những loại bộ nhớ khác nhau có trong một máy tính được sắp xếp từ nhanh đến chậm:

1. *Registers*: Bộ nhớ cực nhanh được đặt trực tiếp trong bộ xử lý của máy tính (processor).
2. *Memory cache*: Làm cầu nối giữa registers và RAM.
3. *Main memory (RAM)*: Là một bộ nhớ mà chúng ta sử dụng thường xuyên nhất.
4. *Ổ cứng (Hard Disk Drive)*: Cái này các bạn biết đến nhiều nhất, người ta thường lưu trữ dữ liệu ở đây.

Những registers chỉ có thể chứa được một vài số, trái ngược hẳn với ổ cứng có thể chứa một số lượng lớn các tập tin.

⚠ Khi tôi nói một bộ nhớ “chậm” là đang dựa theo thang đo máy tính, 8 phần nghìn giây để vào đến ổ cứng thật sự là quá lâu!

❓ Có cần phải nắm tất cả những điều này?

Từ bây giờ, các bạn sẽ học về lập trình, và các bạn thường chỉ làm việc trên RAM nên các bạn cần biết đôi chút về nó. Chúng ta sẽ tìm hiểu cách đọc và lưu các tập tin lên ổ cứng (nhưng có lẽ là trong các bài học sau). Còn về Memory cache và registers thì không cần phải chạm đến vì máy tính của bạn sẽ tự làm việc đó.

⚠ Trong ngôn ngữ lập trình bậc thấp, như assembler (viết tắt của "ASM"), một ngôn ngữ tôi đã từng sử dụng, chúng ta phải làm việc trực tiếp với registers, việc làm một phép toán nhân đơn giản thật sự là cả một quá trình chiến đấu gian nan! May mắn là việc đó trên C (và trên nhiều ngôn ngữ lập trình khác) thực hiện đơn giản hơn rất nhiều.

Cần phải nói thêm một điều quan trọng cuối cùng: chỉ có ổ cứng giữ lại tất cả những gì mà nó chứa. Tất cả các bộ nhớ khác (registers, Memory cache, RAM) đều là những bộ nhớ nhất thời: khi mà bạn tắt máy tính đi thì tất cả dữ liệu trong đó sẽ mất đi.

May mắn là dữ liệu trong ổ cứng của bạn vẫn không đổi để nhắc nhở máy tính của bạn ở tình trạng nào khi bật lên. 😊

Hình ảnh của RAM:

Chúng ta sắp sửa làm việc với RAM, tôi nghĩ rằng tôi nên giới thiệu nó với bạn 😊

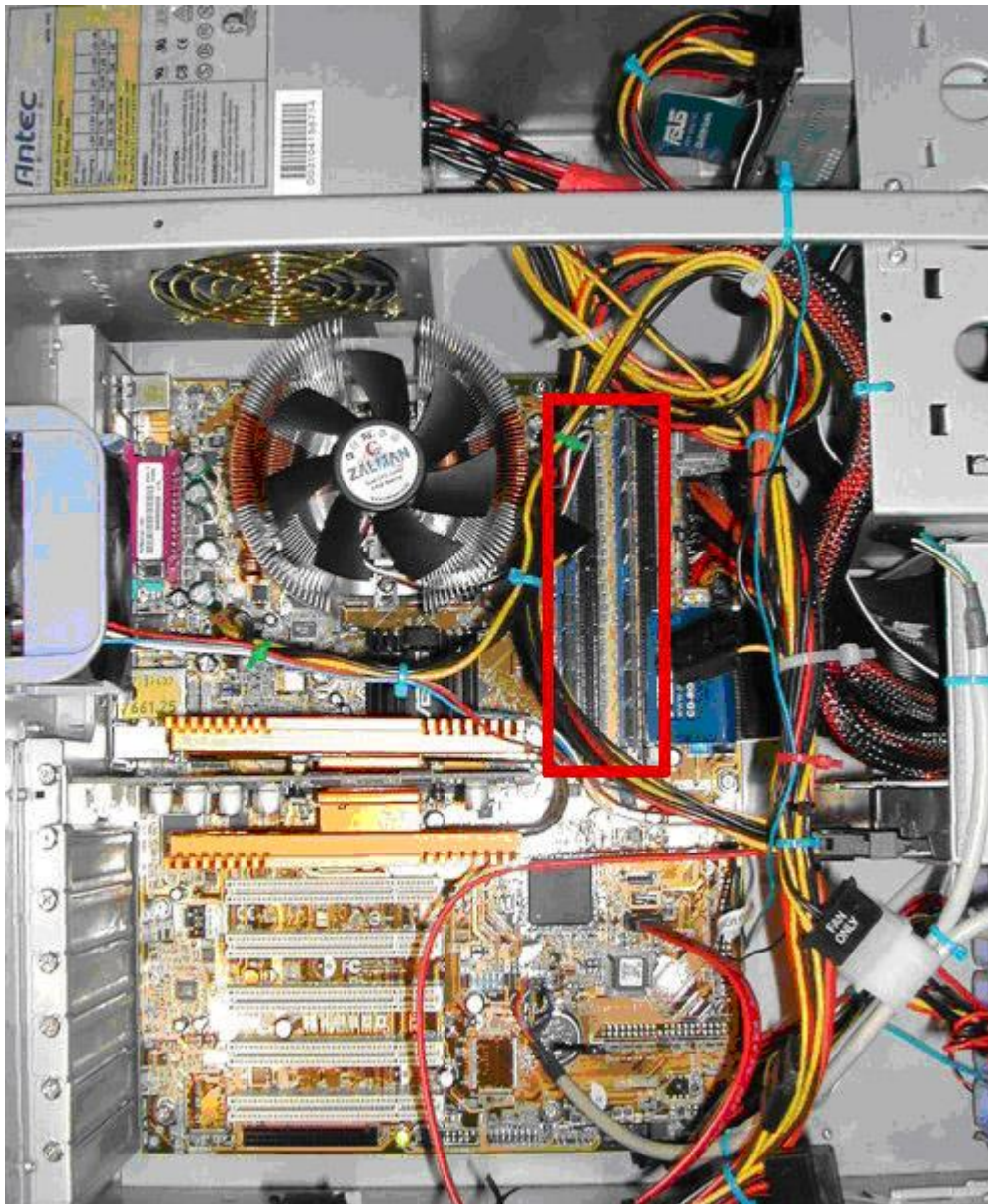
Đây là máy tính của bạn:



Các bạn đã biết thế nào là bàn phím, chuột, màn hình và thùng máy. Bây giờ chúng ta chỉ quan tâm đến thùng máy của bạn, trung tâm của máy tính, nó chứa tất cả các loại bộ nhớ:



Cái mà ta đang hứng thú tìm hiểu nằm bên trong thùng máy, khi mở ra:



Bạn có cảm thấy thích nó không? 😊

Các bạn hãy yên tâm, tôi sẽ không yêu cầu các bạn phải biết chúng hoạt động như thế nào, tôi chỉ muốn bạn biết chỗ để tìm thấy RAM trong thùng máy, nó nằm trong ô chữ nhật màu đỏ.

Tôi sẽ không chỉ ra những bộ nhớ khác (registers và cache) nằm ở đâu vì nó khá nhỏ để có thể thấy được bằng mắt của chúng ta. 😊

Và đây là hình dáng thật sự của RAM:



Biểu đồ của RAM:

Nếu ta nhìn một cách bình thường trên RAM thì chẳng thấy được gì cả. Nhưng, điều quan trọng là cần biết bên trong nó hoạt động như thế nào. Đây chính là điều tôi muốn hướng dẫn các bạn.

Tôi sẽ vẽ cho các bạn một biểu đồ về cách hoạt động của RAM, nó cực kì đơn giản. 😊 Nếu bạn nắm được biểu đồ này thì điều đó vô cùng tốt đối với bạn. 😊

Địa chỉ	Giá trị
0	145
1	3.8028322
2	0.827551
3	3901930
...	...
3 448 765 900 126	940.5118

Biểu đồ hoạt động của RAM

Như bạn thấy, nó được chia làm 2 cột:

- Một cột **địa chỉ (address)**: địa chỉ là một số cho phép máy tính có thể xác định vị trí trong RAM. Nó bắt đầu từ địa chỉ 0 và kết thúc ở địa chỉ 3 448 765 900 126... Hic, tôi không hề biết rõ số lượng địa chỉ chứa trong RAM, tôi chỉ biết rằng nó có rất nhiều. Bởi vì nó phụ thuộc vào dung lượng bộ nhớ mà bạn có. Chỉ có thể nói là, bạn có RAM, bạn có thể để vào đó nhiều thứ. 😊
- Và mỗi địa chỉ chứa một **giá trị** (một số, value): Máy tính của bạn đưa vào RAM những số này để có thể nhớ ngay lập tức. Và người ta chỉ có thể đưa vào một số cho một địa chỉ trong RAM!

Và RAM không thể chứa gì khác ngoài những con số.

❓ **Vậy làm cách nào để chúng ta có thể lưu lại những chữ cái ?**

Đó là một câu hỏi thú vị, trên thực tế, đối với máy tính thì những chữ cái cũng là những con số!

Một câu văn chính là một dãy những con số !

Có một bảng viết về sự tương ứng giữ chữ cái và số (bảng mã ASCII), ví dụ số 67 tương ứng với chữ Y, tôi không nói nhiều về vấn đề này, nếu có cơ hội chúng ta sẽ tìm hiểu về nó sau.

Trở lại với biểu đồ của chúng ta. Hãy xem xét một vấn đề đơn giản: nếu máy tính muốn lưu lại giá trị 5 (có thể là số mạng sống của nhân vật mà bạn chơi trong game nào đó), nó sẽ đặt số 5 vào một vị trí nào đó trong bộ nhớ. (Ví dụ tại địa chỉ 3 062 199 902).

Sau đó, khi muốn tìm lại giá trị này, máy tính sẽ đến “ô” bộ nhớ n° 3 062 199 902, tại đó nó tìm thấy 5 !

Và đó là nguyên tắc hoạt động của bộ nhớ, có thể bạn vẫn còn một chút mập mờ (Đâu là lợi ích của việc đặt một số vào một địa chỉ của bộ nhớ?), bạn sẽ hiểu rõ hơn vấn đề này ở những phần sau của bài hướng dẫn. 😊

Cách khai báo một biến số

Bạn hãy tin tôi rằng một ít giới thiệu về bộ nhớ sẽ rất tiện lợi và tốt hơn cho bạn, giúp bạn có thể tưởng tượng dễ dàng hơn.

Nhưng bây giờ chúng ta phải biết làm cách nào để sử dụng nó. 😊

Vậy thế nào là một biến số ?

Chỉ đơn giản là một thông tin nhỏ được lưu trữ trong RAM.

Chúng ta gọi nó là « biến số » vì nó có thể thay đổi trong quá trình thực hiện chương trình. Ví dụ, số 5 vừa rồi của chúng ta có khả năng bị giảm đi (khi mà nhân vật bạn chết thì mạng sống sẽ giảm xuống). Khi mà giá trị này tiến đến 0 thì trò chơi sẽ kết thúc, game over.

Các bạn sẽ thấy chương trình của chúng ta sẽ chứa đầy những biến số.

Trên ngôn ngữ C, một biến số có 2 thành phần:

- Một **giá trị**: đó là số mà nó chứa, ví dụ như 5.
- Một **tên gọi**: tên gọi này sẽ giúp ta nhận ra nó. Trên ngôn ngữ C, chúng ta không cần phải nhớ địa chỉ của biến số, chúng ta chỉ cần chỉ ra tên của biến số. Và bộ dịch (Compiler) sẽ thực hiện việc chuyển đổi giữa chữ và số.

Gọi tên một biến số:

Trong ngôn ngữ của chúng ta, biến số chỉ số mạng sống của nhân vật trong một trò chơi điện tử nào đó thường được gọi là “mạng sống nhân vật”, hoặc một tên nào khác cùng loại.

Trong ngôn ngữ C, mỗi biến số có một tên gọi, nhưng không phải muốn đặt tên thế nào tùy theo ý thích của bạn cũng được đâu. Dưới đây là một số nguyên tắc khi đặt tên cho biến số:

- Chúng ta chỉ có thể đặt tên nó bằng những chữ cái viết thường hay viết hoa và những con số (abcABC012...).
- Tên của biến số phải **bắt đầu bằng một chữ cái**. Chúng ta **không được sử dụng khoảng trống** « » , thay vào đó chúng ta có thể sử dụng **kí tự** « _ » (underscore). Đó là kí tự duy nhất không thuộc dạng chữ cái hay số được phép sử dụng.
- Bạn cũng không được phép sử dụng chữ cái mang dấu trọng âm. (ví dụ éèèà...).

Và một điều hết sức quan trọng mà bạn cần phải nắm đó là trong ngôn ngữ C (C++ cũng như thế) **có sự khác nhau giữa chữ thường và chữ hoa**: chieu_rong, CHIEU_RONG và CHieu_RoNg là tên của 3 biến số khác nhau trong ngôn ngữ C. Đối với chúng ta thì chúng có vẻ hoàn toàn giống nhau! Và đây là các biến số được đặt tên chính xác: mangsongNhanvat, mangsong_nhanvat, ho, ten, so_dien_thoi, sodidong.

Mỗi người có cách thức gọi tên biến số khác nhau. Trong phần này, tôi giới thiệu cho bạn cách thức gọi tên biến số của riêng tôi:

- Tên của biến số, tôi luôn bắt đầu bằng chữ cái thường.
- Nếu tên của biến số gồm nhiều chữ, thì mỗi chữ tôi sẽ viết hoa ở kí tự đầu tiên

Tôi thích bạn thực hiện giống như tôi, vì điều đó giúp chúng ta có thể làm việc dễ dàng với nhau.



Bạn hãy đặt cho biến số những tên gọi rõ ràng. Chúng ta có thể rút ngắn tên của mangsong_NhanVat bằng ms_NV. Điều đó có thể giúp tên gọi ngắn hơn, nhưng không hề rõ ràng khi bạn viết chương trình. Bạn đừng ngại việc đặt tên dài cho biến số vì điều đó sẽ giúp chương trình của bạn dễ đọc, dễ hiểu hơn.

Những dạng của biến số:

Các bạn có thể xem máy tính không khác gì một cỗ máy lớn dành cho công việc tính toán, nó không biết gì khác hơn ngoài những con số.

Và tôi có một tin đặc biệt là **có nhiều dạng biến số !**

Ví dụ, có những số tự nhiên dương:

- 45
- 398
- 7650

Cũng có những số thực:

- 75,909
- 1,7741
- 9810,7

Hơn nữa cũng có những số nguyên âm:

- -87
- -916

Và những số thực âm:

- -76,9
- -100,11

Và chiếc máy tính đáng thương của bạn cần sự hỗ trợ! Khi bạn yêu cầu nó lưu lại một số, bạn phải nói con số đó thuộc dạng nào. Máy tính của bạn không thể nào có khả năng tự nhận biết chúng, điều đó giúp nó rất nhiều trong việc tự tổ chức, và hạn chế việc sử dụng bộ nhớ một cách vô ích.


Khi bạn tạo một biến số, **phải ghi nó thuộc dạng nào.**


Đây là những dạng biến số cơ bản thường dùng trong ngôn ngữ C (sẽ còn một số loại biến số khác trong C++):

Type	Dung lượng (octets)	Giá trị chấp nhận
signed char	1	-128 đến 127
int	2 (on processor 16 bits) 4 (on processor 32 bits)	-32 768 đến 32 767 -2 147 483 648 đến 2 147 483 647
long	4	-2 147 483 648 đến 2 147 483 647
float	4	-3.4*10⁻³⁸ đến 3.4*10³⁸
double	8	-1.7*10⁻³⁰⁸ đến 1.7*10³⁰⁸

3 dạng đầu cho phép chúng ta khai báo những số nguyên (1, 2, 3, 4...)

2 dạng cuối khai báo những số thực (13.8, 16.911...)

 float và double cho phép khai báo những số thực rất lớn.
Nếu nhưng bạn không hiểu rõ lắm về lũy thừa của 10, tôi sẽ nói rõ hơn cho bạn rằng
những số dạng double có thể lưu lại số được viết bởi số 1 và 308 số 0 tiếp theo !
100...
(xin lỗi nhưng tôi không rảnh để ghi hết 308 số 0 cho bạn nhìn thấy đâu) 😊

 Bạn cần lưu ý rằng int và long có vẻ như giống nhau nhưng thực sự thì int sẽ nhỏ hơn long, nhưng ngày nay bộ nhớ đã phát triển rất nhiều và chúng ta luôn đủ chỗ để chứa những số vô cùng lớn, chúng ta không cần chú ý lắm đến sự khác biệt của chúng. Thực tế tôi chỉ thường dùng những dạng **char**, **long** và **double**.

Và bạn sẽ thấy phần lớn chúng ta chỉ sử dụng những số tự nhiên vì nó dễ dàng sử dụng. 😊



Hãy chú ý với những số thực! Máy tính của bạn không hiểu dấu phẩy là gì đâu, chúng ta chỉ sử dụng dấu chấm. Bạn không thể viết 54.9, thay vào đó là 54.9!

Và không chỉ như vậy! Đối với những biến số dạng số tự nhiên (char, int, long), còn có thêm các loại đặc biệt khác mang tên « unsigned » (không có dấu), tại đó chúng ta chỉ có thể đưa vào những số tự nhiên. Để sử dụng, chỉ cần đặt « unsigned » ở phía trước :

unsigned char	0 đến 255
unsigned int	0 đến 4 294 967 295
unsigned long	0 đến 4 294 967 295

Như bạn đã thấy, những biến dạng unsigned không thể chứa những số âm, nhưng nó có lợi thế là mở rộng giới hạn chứa những số dương lên gấp đôi (ví dụ: signed char có giới hạn 128, trong khi đó unsigned char có giới hạn 255).



Bạn cần lưu ý rằng dạng biến số char nên được khai báo hoặc có signed, hoặc unsigned, không nên đứng một mình. Lý do đơn giản là dạng biến số này sẽ có dấu hay không dấu tùy vào các loại máy tính khác nhau. Trước khi khai báo một biến số, hãy suy nghĩ dạng biến số nào bạn sẽ cần dùng đến.



Tại sao phải tạo ra 3 dạng biến số cho những số tự nhiên như vậy? Chúng ta chỉ cần 1 dạng là đủ rồi mà, không phải vậy sao?

Người ta tạo nhiều dạng biến số khác nhau như thế để tiết kiệm bộ nhớ. Khi mà chúng ta bảo máy tính rằng chúng ta cần một biến số dạng “char”, thì máy tính sẽ sử dụng bộ nhớ ít hơn khi chúng ta bảo rằng cần bộ nhớ dạng “long”.

Việc này sẽ có ý nghĩa trong giai đoạn bộ nhớ máy tính còn nhiều giới hạn. Ngày nay, RAM máy tính đã tiên tiến hơn rất nhiều nên việc này không còn là vấn đề thật sự nữa. Chúng ta không cần nghĩ nhiều đến việc chọn dạng biến số nào để sử dụng. Nếu biến số của bạn có nhu cầu nhận một giá trị tương đối lớn thì hãy nghĩ đến việc sử dụng long.

Tôi nói nghiêm túc rằng bạn không cần phải suy nghĩ nhiều lắm về cách chọn dạng biến số trong thời điểm hiện tại. 😊

Chúng ta chỉ cần phân biệt sự khác biệt giữa dạng số nguyên và số thực:

- Đối với số tự nhiên, người ta thường dùng int.
- Đối với số thực, người ta thường dùng double.

Khai báo một biến số

Cuối cùng chúng ta cũng đến được đây, và bây giờ bạn hãy tạo một project mới lấy tên là “variables”.(biến số)

Bạn sẽ thấy làm cách nào để chúng ta khai báo một biến số, hay nói cách khác là bạn sẽ **yêu cầu quyền sử dụng một ít bộ nhớ của máy tính**.

Bạn chỉ cần làm theo trình tự sau:

1. Chỉ ra dạng của biến số cần tạo.
2. Nhấn phím spacebar để cách khoảng.
3. Chỉ ra tên của biến số cần tạo.
4. Cuối cùng là chấm phẩy ; đừng quên điều đó.

Ví dụ nếu như tôi muốn khai một biến số mangsongNhanVat, tôi sẽ làm như sau:

C Code:

```
int mangsongNhanVat;
```

Chỉ đơn giản vậy thôi! 😊

Và một vài ví dụ khá ngu khác :

C Code:

```
int diemToan;  
double tongChiPhiNhanDuoc;  
unsigned soluongNguoiChuanBiXemTenCuaMotBienSoKhaLaDai;
```

Tôi nghĩ bạn cũng đã hiểu được nguyên tắc của nó rồi. 😊

Việc chúng ta vừa làm gọi là **variable declaration (khai báo biến số)**, hãy nắm vững thuật ngữ này 😊

Bạn phải thực hiện việc khai báo biến số ở vị trí bắt đầu của các function. Và trong thời điểm này chúng ta chỉ có được duy nhất một function (function *main*), bạn hãy khai báo biến số như sau:

C Code:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) // Tương tự với int main()
{
    int mangsongNhanVat;
    return 0;
}
```

Nếu bạn thực hiện việc dịch và chạy chương trình vào lúc này thì chắc hẳn bạn sẽ ngạc nhiên rằng chẳng có gì xảy ra cả 😊

Giải thích

Trước khi bạn cho rằng tôi đùa với bạn thì hãy nghe tôi giải thích vài lời. 😊

Thực tế đã nó một vài thứ diễn ra nhưng bạn không thể nào thấy được. Khi mà chương trình chạy đến đoạn mã khai báo biến số của bạn, nó chỉ yêu cầu máy tính một cách lịch sự rằng nó sẽ sử dụng một ít khoảng trống trong RAM của máy tính.

Nếu không có vấn đề gì, máy tính sẽ trả lời « dùng đi, tự nhiên như ở nhà mày vậy ».

⚠ Vấn đề chỉ xảy ra khi bộ nhớ của bạn không còn khoảng trống nữa. May mắn là điều này vô cùng khó xảy ra nếu sử dụng những biến số dạng int để làm đầy bộ nhớ của máy tính. 😊

Và biến số của bạn đã được tạo ra một cách hoàn hảo.

⚠ Có một điều bạn cần biết: nếu bạn có nhiều biến số cần khai báo và các biến số này cùng một dạng, bạn không cần thiết phải khai báo mỗi biến số cho mỗi dòng. Bạn chỉ cần phân biệt các biến số bởi những dấu phẩy trên cùng một dòng :

C Code:

```
int mangsongNhanVat, capdoTroChoi, capdoNhanVat;
```

Đoạn code này đã khai báo 3 biến số dạng int cho các biến số mangsongNhanVat, capdoTroChoi, capdoNhanVat.

Và bây giờ ?

Sau khi đã khai báo xong biến số, chúng ta có thể đưa cho chúng những giá trị. 😊

Đưa giá trị vào biến số

Không có gì là khó khăn, nếu bạn muốn cho biến số mangsongNhanVat một giá trị, bạn chỉ cần làm như sau :

C Code:

```
mangsongNhanVat = 5;
```

Vậy là xong rồi, bạn không cần làm thêm điều gì khác. Bạn chỉ cần đặt tên của biến số, cho một dấu bằng, kế tiếp là giá trị bạn muốn đặt vào nó. Ở đây chúng ta cho mangsongNhanVat giá trị 5.

Dưới đây là chương trình hoàn thiện:

C Code:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int mangsongNhanVat;
    mangsongNhanVat = 5;

    return 0;
}
```

Và khi chạy chương trình thì màn hình vẫn chưa có gì thay đổi, nó chỉ diễn ra trong bộ nhớ.

Tại một ô bộ nhớ nào đó trong máy tính, giá trị 5 đã được đặt vào. Tuyệt vời đúng không? 😊

Và hay hơn nữa là:

C Code:

```
int mangsongNhanVat;
mangsongNhanVat = 5;
mangsongNhanVat = 4;
mangsongNhanVat = 3;
```

Trong ví dụ này, khi chạy chương trình, biến nhận giá trị 5 đầu tiên, sau đó là 4 và cuối cùng là 3. Việc này diễn ra vô cùng nhanh trên máy tính, chương trình kết thúc khi bạn chưa kịp chớp mắt xong 😊

Giá trị của một biến số mới

Đây là một câu hỏi khá quan trọng mà tôi muốn nhấn mạnh :

❓ Khi mà ta khai báo một biến, thì nó sẽ nhận giá trị nào đầu tiên ?

Thực tế, khi mà máy tính bạn đọc dòng này :

C Code:

```
int mangsongNhanVat;
```

Đồng ý là biến sẽ chiếm 1 vị trí trong bộ nhớ của RAM. Nhưng giá trị của biến số lúc này là bao nhiêu ? Là 0 lúc khởi đầu đúng không ?

Câu trả lời là không. Không, không và không. Không có giá trị nào lúc khởi đầu cả. Bộ nhớ sẽ giành chỗ cho biến số nhưng tại vị trí đó, giá trị sẽ không đổi. Máy tính sẽ không hề xóa những gì đã được đặt vào trước đó (có thể vị trí đó đã được dùng cho một chương trình cũ từng chạy trên máy tính trước đây)

Nếu vị trí này vẫn chưa sử dụng qua lần nào thì có thể nó sẽ mang giá trị là 0. Nhưng nếu một chương trình nào khác đã sử dụng qua rồi thì nó có thể mang giá trị là 368, 18 hay một số nào khác bất kỳ.

Chúng ta cần phải chú ý kĩ vấn đề này để tránh các sai sót về sau. Tốt nhất bạn hãy gán cho nó giá trị ngay sau khi vừa khai báo xong. Trình biên dịch có thể hiểu được nếu như ta khai báo và gán giá trị một biến số trong cùng một lúc:

C Code:

```
int mangsongNhanVat = 5;
```

Như trên, biến số mangsongNhanVat đã được khai báo và nhận tức khắc giá trị 5.

Lợi ích của việc này là bạn luôn chắc rằng biến số đó luôn nhận giá trị chính xác như bạn muốn.

Những constants (hằng số)

Đôi khi chúng ta cần sử dụng những giá trị không đổi trong suốt quá trình sử dụng chương trình. Có nghĩa là sau khi khai báo, biến số sẽ nhận một giá trị mà không cách nào có thể thay đổi được.

Những biến số này chúng ta gọi là các **constants** (hằng số), lý do là giá trị mà nó mang sẽ luôn được giữ nguyên như thế.

Để khai báo một constant, ta làm như sau: chúng ta thêm từ « const » trước dạng biến số mà bạn khai báo.

Mặt khác, chúng ta bắt buộc phải gán cho nó một giá trị ngay trong thời điểm bạn khai báo nó. Giống như cách mà ta đã thấy vừa rồi. Sau đó, bạn không thể nào thay đổi giá trị đó nữa, vì mọi thứ đã được qui định xong hết rồi.

Ví dụ về cách khai báo một constants:

C Code:

```
const MANGSONG_NHANVAT_KHOIDAU = 5;
```

⚠Việc tôi chỉ sử dụng những chữ cái in hoa để đặt tên cho constants là không bắt buộc. Làm như thế để giúp tôi có thể dễ dàng phân biệt những biến số với những constants. Ghi thêm rằng tôi vẫn sử dụng dấu underscore _ vào vị trí của khoảng trống « ».

Sau đó, bạn có thể sử dụng constants như một biến số bình thường. Khác biệt duy nhất là nếu bạn thử thay đổi giá trị của nó sau đó và thực hiện dịch chương trình thì compiler sẽ báo lỗi. 😊

Tôi gọi nó là « death zone » (hay là vùng chết). Trong trường hợp đó, compiler sẽ hiển thị lên màn hình: [Warning] assignment of read-only variable 'MANGSONG_NHANVAT_KHOIDAU' (Dịch ra: “bạn thật là ngu ngốc, tại sao bạn lại cố gắng thay đổi giá trị của một constant chứ?”)

Hiển thị giá trị của biến số

Chúng ta đã biết cách hiển thị một đoạn văn với function *printf*.

Bây giờ, chúng ta sẽ xem làm sao để hiển thị một giá trị của biến số cũng với function này. Chúng ta cũng sẽ sử dụng *printf* với phương pháp cũ, nhưng thêm vào một kí tự đặc biệt tại vị trí mà chúng ta muốn giá trị của biến số đó.

Ví dụ :

C Code:


```
printf("Ban con %d hoisinh");
```

Kí tự đặc biệt mà tôi đã nói với bạn đó là một « **%** » sau đó là những chữ cái « **d** ». Những kí tự này cho phép chúng ta hiển thị dạng của biến số.

« d » có nghĩa là tôi muốn hiển thị một số dạng int.

Còn rất nhiều kí tự đặc biệt khác có thể sử dụng. Nhưng để dễ dàng, lúc này bạn chỉ cần nắm những loại sau:

Format	Type
"%d"	int
"%ld"	long
"%f"	float
"%lf"	double

 Cần lưu ý rằng format dùng để hiển thị một float và một double là giống nhau.

Tôi sẽ nói cho bạn biết những kí tự đặc biệt khác về sau.

Chúng ta sắp xong rồi, chúng ta đã chỉ ra vị trí cần hiển thị một số, nhưng chúng ta vẫn chưa nói là hiển thị số nào. Vì thế chúng ta cần chỉ cho function *printf* biết phải hiển thị biến số nào.

Bằng cách đánh tên của biến số đó sau khi thêm vào một dấu phẩy sau khi kết thúc dấu '' giống như sau:

C Code:

```
printf("Ban con %d hoisinh", mangsongNhanVat);
```

%d sẽ thay thế bởi biến số mà ta đã chỉ ra sau dấu phẩy, trường hợp này là mangsongNhanVat.

Chúng ta thử chạy chương trình nhé ?

<-----

C Code:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int mangsongNhanVat = 5; // Khoi dau, ban co 5 lan hoi sinh
    printf("Ban co %d lan hoi sinh\n", mangsongNhanVat);
    printf("***** B U M *****\n"); // Ban bi trung mot phat sung vao dau
    mangsongNhanVat = 4; // Ban bi mat mot lan hoi sinh !
    printf("Xin chia buon, ban chi con %d lan hoi sinh !\n\n", mangsongNhanVat);

    return 0;
}
```

Cái này gần giống như một game điện tử rồi (bạn hãy tưởng tượng nhiều một tí)
Chương trình hiển thị cái này ra màn hình:

Console:

```
Ban co 5 lan hoi sinh
***** B U M *****
Xin chia buon, ban chi con 4 lan hoi sinh !

Press any key to continue.
```

Bạn phải hiểu được điều gì diễn ra trong chương trình của bạn :

1. Đầu tiên, nhân vật có 5 mạng sống, chúng ta hiển thị bằng printf
2. Sau đó, « bùm » nhân vật trúng phải một phát súng vào đầu.
3. Cuối cùng, nhân vật còn 4 mạng sống, chúng ta hiển thị bằng printf.


Đơn giản là như vậy.

Hiển thị nhiều biến số trong một function printf

Chúng ta luôn có thể hiển thị giá trị của nhiều biến số chỉ trong một function *printf* duy nhất. Chỗ này sẽ hiển thị %ld và chỗ kia hiển thị %lf, tùy theo bạn muốn, sau đó chỉ ra theo thứ tự lần lượt những biến số tương ứng, cách nhau bởi những dấu phẩy.

Ví dụ :

```
printf("Ban co %d lan hoi sinh va dang o man choi thu %d", mangsongNhanVat, capdo);
```

 Hãy chỉ ra những giá trị của bạn theo đúng thứ tự. %d đầu tiên sẽ thay thế bằng biến số đầu tiên (mangsongNhanVat), và %d thứ hai sẽ thay thế bởi biến số thứ hai (capdo). Nếu như bạn nhầm lẫn vị trí, những gì bạn muốn hiển thị sẽ không còn đúng nữa.

Và chúng ta hãy thử test lại một tí, ghi thêm rằng trong đoạn code bên dưới tôi không ghi những dòng ở trên cùng (những preprocessor directives bắt đầu bởi những #), và tôi sẽ **giả định rằng bạn sẽ luôn thêm nó vào ở đầu chương trình**.

C Code:

```
int main(int argc, char *argv[])
{
    int mangsongNhanVat = 5, capdo = 1;

    printf("Ban co %d lan hoi sinh va ban dang o man choi thu %d\n", mangsongNhanVat, capdo);

    return 0;
}
```

Và nó sẽ hiển thị:

Console:

```
Ban co 5 lan hoi sinh va ban dang o man choi thu 1
```

Cách gán giá trị cho biến số

Những biến số bắt đầu làm bài học này thú vị hơn rồi nhỉ. Chúng ta sẽ học cách yêu cầu người dùng nhập một số vào console. Số này sẽ được một biến số lưu lại. Một khi bạn thực hiện được điều này, bạn có thể làm thêm rất nhiều việc sau đó. 😊

Để yêu cầu người dùng đưa vào một cái gì đó vào trong console, chúng ta sẽ sử dụng một function khác, và function này đã có sẵn trong thư viện của bạn: function đó là *scanf*

Cách sử dụng *scanf* khá giống với *printf*. Bạn phải đặt %d hay %lf trong cặp dấu "..." để giải thích với máy tính rằng bạn muốn người dùng đưa vào một số nguyên hay một số thực. Sau đó bạn phải chỉ ra tên của biến số sẽ nhận lấy giá trị đó.

Bạn sẽ thấy điều đó trong ví dụ này :

C Code:

```
scanf ("%d", &tuoi);
```

Chúng ta phải đặt %d (hoặc %lf) trong cặp dấu "..."
Một khác chúng ta thêm vào & trước tên của biến số.

❓ Vậy tại sao phải thêm & trước tên của biến số ?

Tôi sẽ không giải thích cho bạn tất cả ở đây. Nhưng hãy tin tôi, tôi sẽ giải thích cho bạn vấn đề này trong một bài khác sau này, tôi hứa đấy ! 😊

Trở lại, khi mà chương trình của bạn chạy đến *scanf*, nó sẽ dừng lại và đợi người sử dụng đưa vào một số. Số này sẽ được đưa vào biến số « *tuoi* ».



Hãy chú ý, có một vài sự khác nhau giữa *printf* và *scanf* ! Để gán một giá trị dạng float, ta dùng format "%f", nhưng để gán một giá trị dạng double ta dùng format "%lf"

Đây là một chương trình nhỏ yêu cầu biết tuổi của người sử dụng và nó sẽ hiển thị ra sau đó :

Code C:

```
int main(int argc, char *argv[])
{
    int tuoi = 0; // Khởi tạo biến số giá trị là 0

    printf("Ban bao nhieu tuoi?\n");
    scanf("%d", &tuoi); // Máy tính yêu cầu nhập tuổi với scanf
    printf("Oh! tuoi cua ban la %d !\n\n", tuoi);

    return 0;
}
```

Console:

```
Ban bao nhieu tuoi?
20
Oh! tuoi cua ban la 20 !
```

Chương trình sẽ dừng lại và hiển thị « Ban bao nhieu tuoi? ». Dấu nhảy sẽ xuất hiện trên màn hình. Các bạn phải đánh vào một số tự nhiên (tuổi của bạn). Sau đó nhấn Enter để xác nhận, và chương trình sẽ tiếp tục hoạt động.

Sau đó, chương trình sẽ hiển thị giá trị của biến số « tuoi » lên màn hình (“Oh! tuoi cua ban la 20 !”).

Nguyên tắc hoạt động là như vậy. 😊

Nhờ vào function *scanf* chúng ta có thể yêu cầu người sử dụng đưa ra một số thông tin cá nhân.

Viết thêm rằng bạn chỉ có thể đưa vào đó một số tự nhiên :

- Nếu bạn nhập vào đó một số thực, ví dụ như 2.9, nó sẽ tự động làm tròn, nghĩa là nó chỉ giữ lại phần nguyên. Trong trường hợp này số 2 sẽ được biến số lưu lại.
- Nếu bạn đánh vào bất kì một chữ cái nào đó («èèydf »), biến số sẽ không thay đổi giá trị. Điều này cũng tốt vì trước chúng ta đã gán cho biến số giá trị 0. Sau khi nhập những chữ cái vào thì ngay lập tức, chương trình hiển thị « 0 tuoi », chứng tỏ scanf không được thực hiện. Nếu sau khi khai báo biến số chúng ta không gán cho nó giá trị nào, chương trình bạn có thể hiển thị bất cứ cái gì !

Chúng ta sắp kết thúc bài học về các biến số 😊

Tôi xin nhắc lại là biến số sẽ được sử dụng thường xuyên khi lập trình. Nếu như bạn hiểu rằng biến số là một thông tin được đưa vào bộ nhớ tạm thời thì bạn đã hiểu bài giảng này. Không có điều gì khác ngoài việc bạn cần biết những dạng biến số (char, int, long, double...).

Hãy tự luyện tập cách hiển thị những biến số lên màn hình và cách nhập vào giá trị một biến số bằng bàn phím với scanf.

Trong chương tiếp theo, chúng ta sẽ học cách làm sao thực hiện các tính toán trên ngôn ngữ C. Yêu cầu bạn phải sử dụng tốt *printf* và *scanf*. 😊

TRẮC NGHIỆM KIẾN THỨC.

❓ Khi ta khai báo một biến số, bộ nhớ nào sẽ được sử dụng ?

- A. Registers
- B. Memory cache
- C. RAM (main memory)
- D. Hard Disk Drive

❓ Khi tắt máy tính, bộ nhớ nào sẽ không bị mất dữ liệu ?

- A. Registers
- B. Memory cache
- C. RAM (main memory)
- D. Hard Disk Drive

❓ Biến số nào không được đặt tên chính xác ?

- A. vitriMenu
- B. chieurongCửaSổ
- C. tuoi_Capital

❓ Dạng biến số nào có thể lưu trữ số 76.8 ?

- A. **char**
- B. **long**
- C. **double**
- D. **int**

❓ Dạng biến số nào có thể lưu trữ số -1000 ?

- A. **int**
- B. **unsigned int**
- C. **unsigned double**

❓ Nếu như biến số "taikhoanNganHang" thuộc dạng int có giá trị là 6 500 000 , màn hình sẽ hiển thị đoạn mã này thế nào ?

Code:

```
printf("Ban co %d dong trong tai khoan", taikhoanNganHang);
```

- A. **Ban co %d dong trong tai khoan**
- B. **Ban co 6 500 000 dong trong tai khoan**
- C. **Ban co d dong trong tai khoan, taikhoanNganHang**

Đáp án:

- 1- C
- 2- D
- 3- B
- 4- C
- 5- A
- 6- B

Bài 5: Các công cụ để tính toán

Tôi đã nói vấn đề này ở phần trước: máy tính của các bạn cũng giống như một cỗ máy tính toán khổng lồ.

Khi mà bạn muốn nghe nhạc, xem film hay chơi game điện tử, máy tính của bạn không làm điều gì khác hơn việc tính toán

Ở phần này tôi sẽ hướng dẫn cho các bạn thực hiện phần lớn những phép tính mà máy tính có thể thực hiện. Các bạn đã biết thế nào là biến số, và ý tưởng là chúng ta sẽ thực hiện những tính toán trên các biến số đó: hãy cho một biến số giá trị nào đó, sau đó hãy nhân nó lên, và giá trị nhận được sẽ đưa vào một biến số khác .v.v...

Kể cả khi bạn không phải là một fan của toán học, các bạn cũng nên biết nội dung của phần hướng dẫn này.

Sự thật là nếu bạn không biết cách thực hiện phép cộng, bạn không thể nào có thể thực hiện việc lập trình. 😊

Nội dung bài học này sẽ gồm:

Những phép toán cơ bản

- Những cách viết rút gọn
- Thư viện toán học
- TRẮC NGHIỆM KIẾN THỨC.

Những phép toán cơ bản

Những gì chúng ta sẽ tìm hiểu không có gì khác ngoài những phép toán bình thường, máy tính của bạn là một cỗ máy tính toán đơn giản vì nó chỉ có thể làm những phép toán:

- Phép cộng
- Phép trừ
- Phép nhân
- Phép chia
- Phép module (Tôi sẽ giải thích nếu như bạn không biết nó là gì)

Nếu như bạn muốn sử dụng những phép toán phức tạp hơn (bình phương, lũy thừa, logarit, và một số những phép toán khác mà bạn thích) thì bạn phải lập trình ra nó, có nghĩa là bạn sẽ hướng dẫn máy tính làm cách nào để thực hiện những phép toán đó.

May mắn là bạn có thể mượn những quyển sách này trong thư viện của ngôn ngữ C: có rất nhiều function toán học được viết sẵn. Bạn không cần phải viết lại chúng nữa 😊

Chúng ta bắt đầu với phép cộng.

Để thực hiện một phép cộng, chúng ta sử dụng kí tự + (đúng không nhỉ? 🤔)

Bạn cần phải đưa kết quả nhận được vào trong một biến số. Chúng ta sẽ sử dụng biến số «ketqua» dạng int để thực hiện phép tính:

C Code:

```
int ketqua = 0;  
ketqua = 5 + 3;
```

Bạn không cần phải có một đầu óc pro về tính toán để có thể hiểu rằng « ketqua » sẽ mang giá trị 8 sau khi ta chạy chương trình. 😊

Chắc chắn là, màn hình sẽ không hiển thị bất cứ điều gì nếu như ta chỉ sử dụng đoạn mã trên, hãy thêm vào một function printf.

Trên màn hình sẽ cho ta :

Console:

```
Ket qua = 8
```

Và đó là phép cộng.

Và với những phép toán khác, cũng tương tự, chỉ cần thay đổi kí tự tính toán :

- Phép cộng: +
- Phép trừ: -
- Phép nhân: *
- Phép chia: /
- Module: %

Nếu như bạn đã từng tính toán trên máy tính của bạn thì chắc hẳn là bạn biết những kí tự này .
"Dấu trừ" tương ứng với dấu gạch ngang "-", "dấu nhân" tương ứng dấu sao "*", "dấu chia" tương ứng dấu slash "/" và "module" sẽ tương ứng với dấu phần trăm "%".
Không có gì đặc biệt khó khăn để sử dụng được chúng. Ở hai phép tính cuối cùng (phép chia và module) có một số khác biệt nhỏ, chúng ta sẽ nói rõ hơn.

Phép chia

Phép chia hoạt động bình thường trên máy tính nếu như không có số dư. Ví dụ, $6 / 3$ bằng 2, máy tính của bạn sẽ cho một kết quả đúng, không hề sai sót.

Bây giờ chúng ta thử thực hiện một phép chia có dư như $5 / 2$.
 $5 / 2$ theo như ta tính sẽ cho kết quả là 2.5.

Tuy nhiên! hãy xem kĩ kết quả của đoạn mã này :

C Code:

```
int ketqua=0;  
ketqua = 5 / 2;  
printf ("5 / 2 = %d",ketqua);
```

Console:

Ket qua = 2

Có một vấn đề lớn ở đây, chúng ta yêu cầu máy tính thực hiện $5 / 2$, chúng ta chờ đợi kết quả là 2.5, nhưng máy tính cho kết quả là 2 !

Có một cái gì đó kì lạ ở đây. Không lẽ máy tính của chúng ta bị ngu ở phép tính này ?
Thực sự, khi máy tính nhận được những số 5 và 2, máy tính của bạn thực hiện phép tính với dạng số tự nhiên, điều đó có nghĩa là máy tính đã làm tròn kết quả, nó chỉ giữ lại phần nguyên (số 2).

❓ Tôi biết rồi! tại vì biến số « ketqua » mà chúng ta khai báo có dạng int! nếu nó ở dạng double thì nó sẽ chứa một số thực!

Cũng không phải. 😊

Hãy thử lại đoạn mã trên nhưng chúng ta đổi biến số « ketqua » thành double, nó cũng chỉ hiển thị kết quả là 2.

Nếu như ta muốn máy tính hiển thị một kết quả chính xác, chúng ta phải biến đổi những số 5 và 2 đó về dạng số thực, nghĩa là 5.0 và 2.0 (Đối với chúng ta, chúng giống nhau nhưng đối với máy tính, những số thực khác với số tự nhiên và nó sẽ thực hiện phép toán với dạng số thực) :

C Code:

```
double ketqua = 0;  
ketqua = 5.0 / 2.0;  
printf("5 / 2 = %lf", ketqua);
```

Console:

```
Ket qua = 2.500000
```

Mặc dù nó hiển thị một dãy những số 0 ở phía sau nhưng kết quả này hoàn toàn chính xác.

Đặc điểm này của phép chia rất quan trọng, bạn cần chú ý:

$$\begin{aligned}5 / 2 &= 2 \\10 / 3 &= 3 \\4 / 5 &= 0\end{aligned}$$

Những số trong phép tính phải thuộc dạng số thực :

$$\begin{aligned}5.0 / 2.0 &= 2.5 \\10.0 / 3.0 &= 3.33333 \\4.0 / 5.0 &= 0.8\end{aligned}$$

Thực tế, nếu ta thực hiện phép tính « 5 / 2 », ở dạng số tự nhiên. Máy tính sẽ trả lời câu hỏi: «Trong 5, có bao nhiêu lần 2 ?». Câu trả lời là 2 lần. Giống như vậy, « trong 10, có bao nhiêu lần 3 ? đáp án là 3 lần ».

Nhưng làm sao để giữ lại số dư của phép chia ?

Và đây chính là công việc của phép module. 😊

Phép module

Module là một phép toán cho ta **số dư của một phép chia**. Module ít được biết đến hơn các phép toán cơ bản còn lại, nhưng nó giúp máy tính có thể thực hiện đầy đủ tất cả những phép toán với những số tự nhiên. Module được biểu thị bởi kí tự %.

Một số ví dụ :

- $5 \% 2 = 1$
- $14 \% 3 = 2$
- $4 \% 2 = 0$

Module $5 \% 2$ là số dư của $5 / 2$, bằng 1. Máy tính tính toán như sau $5 = 2 * 2 + 1$ (module cho kết quả 1).

Tương tự, $14 \% 3$, tính như sau $14 = 3 * 4 + 2$ (module cho kết quả 2).

Cuối cùng, $4 \% 2$, phép chia không có dư nên module sẽ cho kết quả là 0.

Và không nói gì nhiều hơn về module, tôi chỉ giải thích với những bạn nào chưa biết. 😊

Và tôi có thêm một tin tốt nữa, chúng ta đã biết tất cả những phép toán cơ bản. 😊

Những tính toán sử dụng biến số

Vấn đề này khá lý thú, bạn đã biết cách sử dụng các phép toán cơ bản, chúng ta hãy luyện tập bằng cách tính toán với nhiều biến số, bạn có thể thực hiện :

C Code:

```
ketqua = so1 + so2;
```

Đoạn mã này tính tổng của các biến số so1 và so2, sau đó kết quả sẽ đưa vào biến số ketqua. Ah! tôi có một ý tưởng cho bạn đây, bây giờ bạn có khả năng thực hiện một công cụ tính toán nhỏ. Chắc mà, tôi bảo đảm các bạn có thể làm được ! 😊

Hãy tưởng tượng một chương trình đòi hỏi người sử dụng nhập vào 2 số hạng. Đó là giá trị của 2 biến số, sau đó bạn hãy thực hiện tổng của 2 biến số này, kết quả là giá trị của biến số « ketqua ». Sau đó bạn hiển thị nó lên màn hình. Chương trình này khá đơn giản, hãy luyện tập với nó. 😊

Đây là kết quả:

C Code:

```
int main(int argc, char *argv[])
{
    int ketqua = 0, so1 = 0, so2 = 0;

    // Chúng ta yêu cầu người sử dụng nhập vào giá trị của so1 và so2 :

    printf("Gia tri so thu 1 : ");
    scanf("%d", &so1);
    printf("Gia tri so thu 2 : ");
    scanf("%d", &so2);

    // Thực hiện phép tính :

    ketqua = so1 + so2;

    // Và ta hiển thị lên màn hình :

    printf("%d + %d = %d\n", so1, so2, ketqua);

    return 0;
}
```

Console:

```
Gia tri so thu 1 : 30
Gia tri so thu 2 : 25
30 + 25 = 55
```

Các bạn có thể thử chương trình này với bất kì số hạng nào (nếu số hạng đó không nằm ngoài giới hạn của biến số dạng int), máy tính của bạn sẽ hoàn thành phép tính với vận tốc ánh sáng 😊. Tôi khuyên bạn hãy tạo thêm các chương trình sử dụng những phép toán khác (phép trừ, phép nhân...)

Bạn hoàn toàn có thể thực hiện với nhiều biến số hơn nữa, không vấn đề gì nếu như ta dùng 3 biến số cùng lúc:

C Code:

```
ketqua = so1 + so2 + so3;
```

Phương pháp viết rút gọn

Như tôi đã nói với các bạn, chúng ta không còn phép toán nào mới để học nữa, đó là tất cả. 😊 Chúng ta không cần thêm các phép toán nào khác vì chúng ta có thể tạo ra chúng.

Thật khó tin nếu như tôi nói rằng một game 3D chỉ sử dụng không gì khác ngoài phép cộng và phép trừ. Nhưng đó hoàn toàn là sự thật. 😊

Và cũng giống như dưới đây, trong C có những phương pháp giúp ta viết ngắn gọn những phép toán.

Tại sao phải dùng phương pháp viết rút gọn ?

Tại vì chúng ta thường xuyên phải lặp lại một phép toán. Bạn sẽ hiểu rõ hơn những gì tôi nói. Với cái gọi là increment.

Incrementing (Phương pháp tăng giá trị)

Bạn sẽ thấy rằng bạn sẽ phải thường xuyên tăng giá trị một biến số lên 1. Ví dụ như biến số của bạn là « sohang ». Và ta làm như sau:

C Code:

```
sohang = sohang + 1;
```

Và chuyện gì diễn ra ở đây? Chúng ta lấy sohang + 1, và sau đó chúng ta đưa giá trị nhận được vào chính « sohang ». Nếu số hạng này có giá trị ban đầu là 4 thì nó sẽ thành 5, nếu là 8 thì sẽ thành 9...

Thao tác này sẽ được sử dụng lại. Những nhà lập trình đều là những người đặc biệt lười biếng, hầu như họ đều không hứng thú với việc viết lại một cái nào đó đã có (việc này khá mệt nhọc !).



Và họ đã tạo ra một cách viết rút gọn gọi là **increment**. Đoạn mã sau cũng biểu thị điều tương tự với đoạn mã ta vừa thấy ở trên :

C Code:

```
sohang++;
```

Đoạn mã này khá ngắn so với những gì ta thấy trước đó, nó có nghĩa là « thêm 1 vào biến số sohang ». Chỉ cần viết tên biến số, sau đó thêm vào hai dấu +, và đừng quên dấu chấm phẩy đặt ở cuối cùng.

Chỉ là như vậy, chúng ta sẽ gặp lại nó thường xuyên vì có rất nhiều trường hợp cần sử dụng phương pháp này.

⚠ Nếu bạn chịu để ý một tí, bạn sẽ thấy dấu hiệu này được tìm thấy trong « C++ ». Và với con mắt của một nhà lập trình, bạn có thể hiểu được ý nghĩa của nó! C++ có nghĩa là ngôn ngữ C « được tăng thêm một cấp » 🤖, mặc dù vậy so với C nó không hề hơn.

Decrementing (Phương pháp giảm giá trị)

Đơn giản có thể hiểu là phương pháp này trái ngược hoàn toàn với increment. Chúng ta sẽ giảm giá trị của biến số đi 1.

Và chúng ta cũng sẽ sử dụng nó thường xuyên như increment.

Nếu như ta viết nó đầy đủ :

C Code:

```
sohang = sohang - 1;
```

Thì đây là dạng rút gọn :

C Code:

```
sohang--;
```

Hẳn là bạn có thể tự đoán ra được, ở vị trí ta đặt ++, thì thay thế bằng --. Nếu biến số có giá trị là 6 ban đầu, thì nó sẽ thành 5 sau khi thực hiện decrement.

Những dạng viết rút gọn khác

Trong C còn nhiều cách viết rút gọn khác cũng hoạt động tương tự. Tất cả các phép toán cơ bản : + - * / đều có phương pháp viết rút gọn.

Nó giúp ta phải viết lại tên của biến số cùng một dòng.

Và nếu bạn muốn tăng lên 2 lần giá trị của một biến số :

C Code:

```
sohang = sohang * 2;
```

Bạn có thể viết dưới dạng rút gọn :

C Code:

```
sohang *= 2;
```

Nếu số đó là 5 lúc ban đầu thì sau đó nó sẽ trở thành 10.

Với những phép toán cơ bản khác cũng hoạt động y như vậy, đây là một chương trình làm ví dụ :

C Code:

```
int sohang = 5;  
sohang += 4; // sohang tro thanh 9...  
sohang -= 3; // ... sohang tro thanh 2  
sohang *= 5; // ... sohang tro thanh 25  
sohang /= 3; // ... sohang tro thanh 1  
sohang %= 3; // ... sohang tro thanh 2 (vi  $5 = 1 * 3 + 2$ )
```

Đó là những cách viết rút gọn mà bạn sẽ sử dụng trong một ngày nào đó, 😊 hãy nắm vững increment vì đây là phương pháp viết rút gọn được sử dụng nhiều nhất. 😊

Thư viện toán học

Trong ngôn ngữ C, tồn tại một số cái gọi là những thư viện « standard », đó là những thư viện «cơ bản» luôn sẵn sàng để sử dụng, và được sử dụng thường xuyên.

Tôi xin nhắc lại, thư viện là tập hợp những function đã được viết sẵn bởi những nhà lập trình khác trước đó để tránh việc phải viết lại.

Chúng ta đã từng sử dụng function *printf* và *scanf* trong thư viện «stdio.h» .

Chúng ta phải biết rằng còn nhiều thư viện khác nữa, trong đó có «math.h», nó chứa một số lớn những function toán học đã được viết trước.

⚠ Ngoài phép toán cơ bản mà bạn đã biết, thì thư viện toán học chứa những phép toán phức tạp khác mà tôi chắc là bạn sẽ cần đến, ví dụ như là các hàm lũy thừa (nếu như bạn không biết đây là gì thì có thể bạn còn quá trẻ hay là bạn học toán vẫn chưa đủ).

Trong trường hợp, chúng ta muốn thực hiện những phép tính lũy thừa trong C! Làm sao tính một số mũ 2 ? Bạn có thể viết 5^2 trong đoạn mã của bạn, nhưng máy tính sẽ không hiểu cái đó là gì cả... Ít nhất bạn phải giải thích cho nó bằng cách sử dụng những thư viện toán học !

Để có thể sử dụng những function trong thư viện toán học, chúng ta bắt buộc phải thêm preprocessor directives ở đầu chương trình:

C Code:

```
#include <math.h>
```

Một khi mà bạn đã làm điều đó, bạn có thể sử dụng tất cả các function trong thư viện này.

Tôi nghĩ là tôi nên giới thiệu chúng với bạn. Tôi sẽ không làm một list đầy đủ ở đây, vì nó có rất nhiều, các ngón tay đáng thương của tôi sẽ phồng lên trước khi kết thúc phần hướng dẫn này. 😊
Tôi chỉ hướng dẫn các bạn một số function chính, những function có vẻ quan trọng nhất.

⚠️ Có thể trình độ toán học của bạn không đủ để hiểu làm cách nào có thể viết được những function này. Nếu đúng là như vậy, bạn không cần phải lo lắng gì cả. Chỉ cần đọc, việc này không có hại cho bạn đâu.
Và tôi sẽ cho bạn một lời khuyên miễn phí: hãy chú tâm vào những tiết toán, họ không nói như tôi ở đây đâu. 🤔

fabs

Function này sẽ trả về giá trị tuyệt đối của một số, trong toán học viết là $|x|$.

- Nếu bạn đưa function này giá trị là -53, nó sẽ trả về giá trị 53.
- Nếu bạn đưa function này giá trị là 53, nó sẽ trả về giá trị 53.

C Code:

```
double giatri_tuyetdoi = 0, sohang = -27;  
giatri_tuyetdoi = fabs(sohang); // giá trị tuyệt đối của sohang sẽ là 27
```

Function này sẽ **trả về một số dạng double** vì vậy biến số của bạn đưa vào cũng phải thuộc dạng double.

⚠️ Trong thư viện « `stdlib.h` » cũng có một function tương tự gọi là « `abs` », nó cũng hoạt động như vậy, chỉ trừ việc nó sử dụng những số nguyên « `int` » và nó trả về giá trị dạng số nguyên `int`.

ceil

Function này sẽ **trả về giá trị dạng số nguyên** nếu như ta đưa cho nó một số thực.

Đó là một dạng làm tròn. Nó sẽ luôn cho một **số nguyên có giá trị lớn hơn**.

Ví dụ, nếu như ta cho nó giá trị là 26.512, function sẽ trả lại 27.

Nó **sử dụng và trả lại giá trị dạng double**:

C Code:

```
double lamtronLen = 0, sohang = 52.71;  
lamtronLen = ceil(sohang); // lamtronLen sẽ bằng 53
```

floor

Trái ngược với function ceil, function này cho ta **số nguyên có giá trị nhỏ hơn**.
Nếu như ta cho nó 37.91, function *floor* sẽ trả lại 37.

C Code:

```
double lamtronXuong = 0, sohang = 37.91;  
lamtronXuong = floor(sohang); // giá trị của lamtronXuong sẽ bằng 37
```

pow

Function này cho phép tính lũy thừa một số. Chúng ta phải chỉ ra cho nó 2 giá trị: số hạng và cấp lũy thừa của số đó. Đây là cấu trúc của function này:

C Code:

```
pow(sohang, capLuyThua);
```

Ví dụ, « 2 lũy thừa 3 » (chúng ta thường ghi là 2^3 trên máy tính), là phép toán $2 * 2 * 2$, cho kết quả là 8 :

C Code:

```
double ketqua = 0, sohang = 2;  
ketqua = pow(sohang, 3); // ketqua sẽ được  $2^3 = 8$ 
```

Bạn có thể sử dụng function này để tính bình phương của một số.

sqrt

Function này **tính căn bậc 2** của một số, nó **cho ta giá trị dạng double**.

C Code:

```
double ketqua = 0, sohang = 100;  
ketqua = sqrt(sohang); // ketqua trở thành 10
```

sin, cos, tan

Đây là 3 function được sử dụng trong lượng giác.
Cách hoạt động của chúng như nhau, **trả về giá trị dạng double**.

Chúng ta phải cho nó giá trị **radian**.
(một radian bằng 180 độ)

asin, acos, atan

Đây là những function arc sinus (arcsin), arc cosinus (arccos) và arc tangente (arctan), các function lượng giác khác.

Cũng hoạt động tương tự như trên, **trả về giá trị dạng double**.

exp

Function tính exponential, hay còn gọi là **lũy thừa cơ số e**. **Trả về giá trị dạng double**.

VD: $\exp(4) = e^4$

log

Function tính logarit tự nhiên, là logarit cơ số **e**. (chúng ta thường ghi là « ln ») ngày trước còn đi học ông thầy bảo đọc cái này là « lóc nê be »

log10

Function này tính le logarit cơ số 10 của một số. ông thầy bảo đọc là « lóc mười »

Lời kết:

Tóm lại, tôi đã không nói về các function khác. 😊 (Thật sự là tôi không biết là chúng được dùng để làm gì 😊)

Với những function này bạn có thể sử dụng cho phần lớn các trường hợp liên quan đến toán học.

Xin nói thêm lần nữa, nếu bạn không hiểu những điều tôi nói ở trên thì cũng không có gì nghiêm trọng cả vì những phép toán này chúng ta không nhất thiết cần đến. Trừ khi bạn phải làm một chương trình tính toán một vấn đề khoa học nào đó. 😊

Dù gì bạn cũng nên nắm vững function **floor**, **ceil**, và **pow**, nó rất cần thiết cho chúng ta trong tính toán.

Bài hướng dẫn chi tiết về các công thức toán tôi xin nhường cho những thầy giáo dạy toán. Nếu bạn vẫn còn đi học, tôi cho bạn một lời khuyên chân thành: Hãy học tốt môn toán, điều đó sẽ giúp ích rất nhiều trong lập trình. Nhưng chúng ta rất ít khi phải tính lũy thừa và tiếp tuyến khi viết chương trình, nó phụ thuộc vào chương trình mà chúng ta viết, tôi nói lại.

Ví dụ, nếu có ai trong các bạn hứng thú với những việc liên quan đến 3D (tôi sẽ hướng dẫn về 3D sau), bạn cần phải có một số hiểu biết về hình học không gian (đồ thị, vec-tơ...)

TRẮC NGHIỆM KIẾN THỨC.

❓ "Dấu nhân" trên máy tính là kí hiệu nào ?

- A. *
- B. +
- C. /
- D. -
- E. %

❓ Một câu hỏi đơn giản về module
Kết quả sẽ là bao nhiêu: $17 \% 5$?

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4
- F. 5
- G. 15

❓ "Ketqua" sẽ mang giá trị bao nhiêu ?

C Code:

```
int ketqua = 0;
```

```
ketqua = (8 / 3) - 2;
```

- A. -2
- B. 0
- C. 1
- D. 2

❓ Phương pháp này là gì ?

C Code:

```
sohang++;
```

- A. Increment
- B. Increasing
- C. Supplementation

❓ Biến số "sohang" sẽ mang giá trị bao nhiêu ?

C Code:

```
int sohang = 4;  
sohang--;  
sohang *= 4;  
sohang %= 12;  
sohang += 1;
```

- A. 1
- B. 4
- C. 12
- D. 14

❓ Function nào sau đây sẽ làm tròn 5.47 thành 5 ?

- A. pow
- B. ceil
- C. floor
- D. sqrt

Đáp án:

- 1- A
- 2- C
- 3- B
- 4- A
- 5- A
- 6- C

Bài 6: Condition (Điều kiện)

Trong bài 1, ta đã thấy rằng có rất nhiều ngôn ngữ lập trình, trong đó một số ngôn ngữ hoạt động tương tự nhau, ví dụ như ngôn ngữ PHP khá giống với C, tuy nhiên người ta sử dụng PHP để thiết kế web nhiều hơn. 😊

Các ngôn ngữ thường có một số điểm giống nhau do chúng cùng sử dụng lại các nguyên tắc cơ bản của ngôn ngữ đời trước. Ngôn ngữ C được tạo ra cách đây khá lâu, và mô hình của phần lớn các ngôn ngữ mới hiện giờ đều được tạo ra dựa trên C.

Nguyên tắc cơ bản thì có rất nhiều như cách khai báo biến số, các cách thực hiện phép tính (hầu như các ngôn ngữ lập trình đều giống nhau ở mặt này!) và cách sử dụng condition.

Phần này gồm :

- Condition "if... else"
- Boolean, trung tâm của những conditions
- Condition "switch"
- Ternaries: những condition rút gọn
- TRẮC NGHIỆM KIẾN THỨC.

Condition « if...else »

Chúng ta thường muốn kiểm tra giá trị của một biến số. Ví dụ « Nếu biến số maymoc có giá trị là 50, hãy thực hiện công việc ... » . Hoặc nếu biến số nhỏ hơn 50, nhỏ hơn hoặc bằng 50, lớn hơn, lớn hơn hoặc bằng...

Điều đó có thể được thực hiện trong C thông qua việc sử dụng condition if...else. Condition dùng để kiểm tra giá trị của biến số. Và để biết cách sử dụng nó, chúng ta sẽ đi theo sơ đồ sau:

1. Một số kí tự cần biết trước khi bắt đầu.
2. Test if
3. Test else
4. Test “else if”
5. Cách thiết lập nhiều conditions cùng lúc
6. Những sai phạm thường gặp mà người mới học cần tránh

Một vài kí hiệu cơ bản cần biết trước khi học cách sử dụng condition “if...else” trên C

Chúng ta cần phải **thuộc lòng** bảng kí hiệu này: 😊

Kí tự	Ý nghĩa
=	Bằng
>	Lớn hơn
<	Bé hơn
>=	Lớn hơn hoặc bằng
<=	Bé hơn hoặc bằng
!=	Khác



Hãy chú ý, để kiểm tra bằng nhau, ta cần nhập 2 kí tự « == ». Những người bắt đầu học lập trình thường mắc lỗi chỉ nhập một kí tự =, chúng không có cùng ý nghĩa trong ngôn ngữ C, tôi sẽ nói về vấn đề này ở dưới.

if

Chúng ta sẽ thực hiện một chương trình đơn giản, nó sẽ nói với máy tính:

NẾU biến số thỏa điều kiện ...
THÌ thực hiện ...

Trong tiếng anh, từ « nếu » sẽ dịch thành « if », từ này cũng được sử dụng trong C để khai báo một condition.

Để viết một condition if, đầu tiên hãy viết từ **if**, kế đó mở ngoặc đơn. Trong ngoặc đơn, hãy viết điều kiện.

Sau đó, mở một dấu gộp { và hãy đóng lại ở phía sau }. Trong đó sẽ chứa tất cả những instruction sẽ được thực hiện nếu điều kiện thỏa mãn.

Chúng ta sẽ viết như sau:

C Code:

```
if (/* Điều kiện của bạn */)
{
    // Các Instructions sẽ được thực hiện nếu như điều kiện thỏa mãn
}
```

Tại vị trí chú thích « điều kiện của bạn », chúng ta sẽ viết một điều kiện để kiểm tra biến số.

Ví dụ chúng ta có thể kiểm tra một biến số « tuổi » sẽ chứa giá trị là tuổi của bạn. Chúng ta sẽ xét xem bạn có phải là người trưởng thành hay không, có nghĩa là tuổi của bạn có lớn hơn hoặc bằng 18 hay không:

C Code:

```
if (tuoi >= 18)
{
    printf ("Ban la nguoi truong thanh !");
}
```

Kí tự **>=** có nghĩa là « lớn hơn hoặc bằng », chúng ta đã thấy trong bảng liệt kê các kí tự đặc biệt ở trên.

⚠️ Các dấu gộp {...} không bắt buộc nếu bên trong nó chỉ chứa duy nhất một instruction. Bạn có thể viết như bên dưới nhưng tôi khuyên bạn hãy luôn đặt những dấu gộp để chương trình bạn được rõ ràng.

C Code:

```
if (tuoi >= 18)
    printf ("Ban la nguoi truong thanh !");
```

Test thử đoạn code trên

Nếu như bạn muốn test những đoạn mã trước để xem if hoạt động như thế nào, chúng ta phải đặt condition if bên trong một function main và đừng quên khai báo biến số tuổi, và cho nó một giá trị nào đó theo ý thích của bạn. 😊

Tôi nghĩ rằng bạn có thể tự mình viết ra đoạn mã này và sau đó chạy thử nó.

Đây là đoạn mã hoàn chỉnh:

C Code:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    long tuoi = 20;
    if (tuoi >= 18)
    {
        printf("Ban la nguoi truong thanh !\n");
    }
    return 0;
}
```

Ở đây, biến số tuổi bằng 20, vậy « Ban la nguoi truong thanh ! » sẽ được hiển thị
Hãy thử thay đổi giá trị của biến số. Cho giá trị mới là 15: điều kiện không thỏa mãn và « Ban la nguoi truong thanh ! » không hiển thị lần này. 😊
Hãy giữ lại đoạn mã này để sử dụng cho ví dụ tiếp theo. 😊

Vấn đề cần giải thích

Cách bạn đặt những dấu gộp { } không quan trọng, chương trình của bạn sẽ cũng chạy tốt nếu như bạn viết tất cả trên cùng một hàng. Ví dụ:

C Code:

```
if (tuoi >= 18) { printf("Ban la nguoi truong thanh !"); }
```

Mặc dù bạn có thể viết như vậy nhưng cách viết này không hề được khuyến khích (điều này thực sự rất quan trọng nhé). 😊

Thực sự là viết tất cả trên cùng một hàng sẽ khiến cho **việc đọc đoạn mã của bạn vô cùng khó khăn**. Bạn cần tập cách trình bày mã nguồn của mình ngay từ bây giờ, nếu không sau này khi bạn viết các chương trình lớn hơn, bạn sẽ không tìm thấy được cái bạn cần tìm trong đó!

Hãy thử trình bày lại mã nguồn của bạn theo cách thức của tôi: một dấu gộp mở { duy nhất trên 1 hàng, các dòng sau đó là các instruction (nhấn tab để có thể « cách về bên phải »), sau đó một dấu gộp đóng } duy nhất trên một hàng.

⚠️ Có rất nhiều cách thức hay để trình bày mã nguồn và nó không làm thay đổi hoạt động của chương trình.

Trong suốt quá trình từ giờ về sau bạn sẽ bắt gặp các đoạn mã được trình bày bởi các style hơi khác. Nhưng về cơ bản hầu hết các đoạn mã đó đều có cách trình bày thoáng và dễ nhìn.

« else » để nói « nếu không »

Chúng ta đã biết cách viết condition đơn giản, hãy đi xa hơn một chút: điều kiện không thỏa mãn (sai), chúng ta sẽ yêu cầu máy tính thi hành một instruction khác.

Trong ngôn ngữ của chúng ta, điều đó sẽ được ghi với dạng tương tự như sau :

NẾU biến số thỏa điều kiện...
THÌ thực hiện ...
NẾU KHÔNG hãy thực hiện ...

Chỉ cần thêm vào từ **else** sau dấu gộp kết thúc của **if**.

Một ví dụ nhỏ :

C Code:

```
if (tuoi >= 18) // Neu tuoi lon hon hoac bang 18
{
    printf ("Ban la nguoi truong thanh !");
}
else // Neu khong...
{
    printf ("Hehe, ban con la con nit !");
}
```

Ta có thể hiểu đơn giản: nếu như biến số tuổi có giá trị lớn hơn hoặc bằng 18, sẽ hiển thị « Ban la nguoi truong thanh ! »

Nếu không hiển thị « Hehe, ban con la con nit ! »

« else if » để nói « nếu không nếu »

Chúng ta đã thấy bằng cách nào để tạo một « nếu » và một « nếu không ». Chúng ta cũng có thể tạo một “nếu không nếu”. Nếu như điều kiện đầu không thỏa mãn, chúng ta sẽ kiểm tra biến số với một điều kiện khác. “Nếu không nếu” đặt bên trong if và else.

Chúng ta yêu cầu máy tính:

NẾU biến số thỏa điều kiện 1
THÌ thực hiện việc 1
NẾU KHÔNG NẾU biến số thỏa điều kiện 2
THÌ thực hiện việc 2
NẾU KHÔNG thực hiện việc 3

Dịch sang ngôn ngữ C:

C Code:

```
if ( tuoi >= 18 ) // Neu tuoi lon hon hoac bang 18
{
    printf ("Ban la nguoi truong thanh !");
}
else if ( tuoi > 4 ) // Neu khong, Neu tuoi nho hon 18 va lon hon 4
{
    printf ("Hehe, ban con la con nit !");
}
else // Neukhong...
{
    printf ("Oe oe"); // Ngon ngu tre so sinh, ban khong the hieu duoc dau
}
```

Máy tính sẽ thực hiện những kiểm tra theo thứ tự:

- Đầu tiên nó sẽ kiểm tra **if** đầu tiên: nếu như điều kiện được thỏa mãn, nó sẽ thi hành các instructions có trong các dấu gộp {...} đầu tiên.
- Nếu không, nó sẽ đi đến « nếu không nếu » để kiểm tra các điều kiện mới: nếu đúng, thì nó sẽ thi hành các instruction trong các dấu gộp {...} thứ 2 của **else if**.
- Cuối cùng, nếu không có điều kiện nào được thỏa mãn, nó sẽ chạy những instruction của **else** « nếu không »



« else » và « else if » không bắt buộc phải có. Cần phải có ít nhất một “if” để tạo một condition

Chúng ta có thể đặt bao nhiêu “else if” nếu như ta muốn. Chúng ta có thể viết như sau:

```
NẾU biến số thỏa điều kiện 1
THÌ thực hiện việc 1
NẾU KHÔNG NẾU biến số thỏa điều kiện 2
THÌ thực hiện việc 2
NẾU KHÔNG NẾU biến số thỏa điều kiện 3
THÌ thực hiện việc 3
NẾU KHÔNG NẾU biến số thỏa điều kiện 4
THÌ thực hiện việc 4
NẾU KHÔNG thực hiện việc 5
```


Thiết lập nhiều điều kiện cùng lúc

Ta có thể kiểm tra nhiều điều kiện cùng lúc trong if. Ví dụ, các bạn muốn xem thử nếu như tuổi lớn hơn 18 VÀ nhỏ hơn 25.

Chúng ta cần phải sử dụng các kí hiệu sau đây:

Kí hiệu	Ý nghĩa
&&	VÀ
	HOẶC
!	KHÔNG

Test VÀ

Đoạn mã sử dụng kí hiệu vừa nêu

C Code:

```
if (age > 18 && age < 25);
```

Những kí hiệu « && » có nghĩa là VÀ. Diễn đạt bằng ngôn ngữ của chúng ta : « Nếu tuổi lớn hơn 18 VÀ bé hơn 25 »

Test HOẶC

Để tạo một HOẶC, chúng ta sử dụng 2 kí tự ||.

Chúng ta hãy tưởng tượng một chương trình kiểm tra và quyết định quyền mở tài khoản ngân hàng. Để mở một tài khoản ngân hàng, tuổi khách hàng không được nhỏ quá (chúng ta sẽ tùy tiện đặt điều kiện tuổi phải lớn hơn 30) hoặc khách hàng có nhiều tiền (tất nhiên ngân hàng sẽ đang tay đón chào kể cả khi tuổi bạn bé hơn 10) 😊

Đây là đoạn mã nhận biết quyền mở một tài khoản ngân hàng :

C Code:

```
if (tuoi > 30 || tien > 100000)
{
    printf("Chao mung ban den voi chung toi !");
}
else
{
    printf("Cut !");
}
```

Điều kiện sẽ được thỏa mãn nếu người này có tuổi trên 30 hoặc có số tiền nhiều hơn 100.000 euros.

Test KHÔNG

Kí tự cuối cùng còn lại để test là dấu chấm than. Trong tin học, dấu chấm than có nghĩa là «Không»

Các bạn phải đặt kí tự này trước điều kiện để nói rằng «Nếu điều này không đúng »:

C Code:

```
if (!(tuoi < 18))
```

Đoạn mã trên có thể dịch là « *Nếu người này không phải là trẻ con* »

Nếu dấu « ! » ở phía trước bị lấy đi, đoạn mã sẽ có nghĩa trái ngược : « *Nếu người này là trẻ con* »

Một số lỗi thường gặp của người mới học

Đừng quên có đến 2 dấu ==

Nếu chúng ta muốn kiểm tra xem người này có phải 18 tuổi hay không, chúng ta phải ghi:

C Code:

```
if (tuoi == 18)
{
    printf ("Ban vua moi truong thanh !");
}
```

Đừng quên việc đặt **2 kí tự « bằng »** trong một if, như thế này : ==

Nếu bạn chỉ đặt mỗi một kí tự =, thì biến số của bạn sẽ nhận giá trị 18 (giống như ta đã học trong phần biến số). Tất cả những điều ta muốn ở đây, là kiểm tra giá trị của biến số chứ không phải thay đổi nó! Hãy chú ý điểm này, có rất nhiều người trong các bạn chỉ đặt một dấu = và khi chương trình bạn được bắt đầu thì tất nhiên là nó không chạy giống như ý họ muốn. 😊

Dấu chấm phẩy dư thừa



Một lỗi khác rất thường xuyên của những người mới bắt đầu học lập trình: có khi các bạn đặt một dấu chấm phẩy sau dòng của if.

if là một condition, chúng ta sẽ không đặt dấu chấm phẩy ở cuối condition mà phải là cuối một instruction.

Đoạn code sau đây sẽ không chạy vì có một dấu chấm phẩy ở cuối condition if :

C Code:

```
if (tuoi == 18); // dấu chấm phẩy nay KHÔNG được phép ở đây
{
    printf("Bạn chỉ vừa mới trưởng thành");
}
```

Có một vấn đề khó khăn nữa là "condition if...else..." không biết phải dịch như thế nào là đúng, vì nếu dịch là "điều kiện if...else..." thì có thể bạn sẽ nhầm lẫn bởi điều kiện bên trong dấu ngoặc (...), nếu dịch là "hàm điều kiện..." thì có thể hiểu nhầm là cần đặt một dấu chấm phẩy ở cuối cùng vì nó cũng là một hàm.

Nên mình quyết định, gọi là "condition" nếu đề cập đến "if...else...", gọi "điều kiện" nếu đề cập đến điều kiện bên trong (...) của if.

Booleans, trung tâm của những conditions

Chúng ta sẽ tìm hiểu chi tiết hơn về cách thức hoạt động của condition if...else.

Thực tế, condition được tác động bởi một thứ mà trong tin học người ta gọi là những Boolean.

Đây là một khái niệm khá quan trọng, cần phải nghe rõ (đúng hơn là phải nhìn rõ 😊)

Một số ví dụ để có thể hiểu rõ hơn

Trong những bài học về Vật Lý–Hóa học, thầy giáo của tôi thường có thói quen bắt đầu từ một số thí nghiệm nhỏ trước khi giảng về một khái niệm mới.

Bây giờ tôi sẽ bắt chước ông ấy 😊

Bạn hãy chạy thử đoạn mã đơn giản sau :

C Code:

```
if (1)
{
    printf("Đúng");
}
else
{
    printf("Sai");
}
```

Kết quả :

Console:

Đúng

? Nhưng ??? Chúng ta không hề đưa điều kiện vào trong if, chúng ta chỉ cho nó một số. Vậy đó nghĩa là gì ?

Chúng ta thử một ví dụ khác, bây giờ bạn hãy thay thế 1 bởi 0 :

C Code:

```
if(0)
{
    printf("Đúng");
}
else
{
    printf("Sai");
}
```

Kết quả:

Console:

Sai

Thử một số những ví dụ khác, thay thế 0 bởi bất kì một số nguyên nào, như là 4, 15, 226, -10, -36 .v.v...

Mỗi lần như vậy, nó đưa ra kết quả là gì ? Nó trả lời là : « **Điều đó đúng** ».

Tóm tắt lại các thử nghiệm trên: nếu đặt 0, điều kiện không thỏa mãn, và nếu chúng ta để vào giá trị 1 hay bất kì một số hạng nào, điều kiện thỏa mãn.

Giải thích

Thực tế, mỗi khi ta kiểm tra một điều kiện trong một if, nó sẽ trả về giá trị là 1 nếu điều đó đúng và 0 nếu điều đó sai.

Ví dụ :

C Code:

```
if(tuoi >= 18)
```

Ở đây, chúng ta cần kiểm tra điều kiện “tuoi >= 18”.

Giả định rằng tuổi có giá trị là 23, sẽ cho kết quả đúng, và máy tính « thay thế » « tuoi >= 18 » bởi 1. Sau đó, máy tính sẽ ghi nhớ một « if (1) ». Khi mà số này là 1, như chúng ta đã thấy, máy tính sẽ hiểu là điều kiện này là đúng, và nó sẽ thi thành các instruction khi điều kiện đúng.

Tương tự nếu như điều kiện này là sai, máy tính sẽ thay thế “tuoi >= 18” bởi số 0, và ngay tức khắc máy tính hiểu điều kiện là sai. Máy tính sẽ thi hành instruction của « else ».

Kiểm tra với một biến số

Bây giờ chúng ta làm cái khác: gửi kết quả của điều kiện vào trong một biến số, như chúng ta làm một phép tính (bởi vì đối với máy tính điều kiện cũng là một phép tính !).

C Code:

```
int tuoi = 20;  
int truongthanh = 0;  
truongthanh = tuoi >= 18;  
printf("truongthanh co gia tri: %ld\n", truongthanh);
```

Console:

```
Truongthanh co gia tri: 1
```

Như các bạn vừa thấy, điều kiện `tuoi >= 18` đã trả về giá trị 1 vì `20 >= 18` (đúng).

Biến số `truongthanh` nhận giá trị 1, chúng ta kiểm chứng bằng cách làm một `printf` để thấy rõ `truongthanh` đã thay đổi giá trị từ 0 thành 1.

Thực hiện lại ví dụ trên, cho `tuoi = 10`. Trong trường hợp này, `truongthanh` có giá trị 0.

Biến số `truongthanh` là một Boolean

Hãy nắm vững :


Biến số nhận giá trị 0 và 1 được gọi là một **Boolean**.

Và :

0 = Sai

1 = Đúng

Chính xác hơn, 0 = sai và tất cả các số còn lại là đúng (đã được kiểm chứng). Để đơn giản, chúng ta không sử dụng những số khác, sử dụng 0 và 1 để nói « điều đó là đúng hay sai »

 Trong ngôn ngữ C, không có dạng biến số « Boolean ». Một dạng biến số mới là “bool” được thêm vào trong C++ , sử dụng để tạo ra những biến số dạng Boolean.

Trong thời điểm này, chúng ta chỉ làm việc trên C, chúng ta không có các dạng biến số đặc biệt khác.

Booleans trong các conditions

Chúng ta sẽ làm một kiểm tra condition "if" bằng cách sử dụng một boolean :

C Code:

```
int truongthanh = 1;
if (truongthanh)
{
    printf ("Ban la nguoi truong thanh !");
}
else
{
    printf ("Ban la con nit");
}
```

Vì biến số truongthanh mang giá trị 1, điều kiện đúng, "Ban la nguoi truong thanh !" sẽ hiển thị. Điều này rất tiện lợi, người khác đọc điều kiện của bạn sẽ được dễ dàng hơn. Khi họ thấy « if (truongthanh) » thì họ có thể hiểu là « Nếu bạn là người trưởng thành »

Những điều kiện dùng boolean thường rõ ràng và dễ hiểu nhưng ít nhất bạn cũng phải đặt tên rõ ràng cho biến số, như tôi đã nói kể từ khi bắt đầu. 😊

Đây là một ví dụ khác giúp bạn hiểu rõ hơn :

C Code:

```
if (truongthanh && nam)
```

Có nghĩa là "Nếu bạn là người trưởng thành và bạn là nam".
nam ở đây là một biến số khác dạng boolean có giá trị là 1 nếu bạn là nam, và 0 nếu bạn là nữ.

❓ **Câu hỏi :** nếu chúng ta viết « if (truongthanh == 1) » thì nó vẫn hoạt động đúng không?

Vẫn hoạt động như bình thường nhưng nếu ghi là « if (truongthanh) » thì dễ hiểu hơn 😊

Nắm vững vấn đề : nếu biến số của bạn mang giá trị là một số, hãy viết điều kiện dưới dạng « if (variable == number) ».

Ngược lại, nếu biến số của bạn mang giá trị là một boolean (có nghĩa là sẽ mang giá trị 1 hoặc 0 để nói rằng điều đó là đúng hay sai), thì hãy viết điều kiện dưới dạng «if (boolean)».

Condition "switch"

Chúng ta thấy rằng condition "if... else" là dạng condition được dùng thường xuyên nhất. Đôi khi "if... else" bị lặp lại khá thường xuyên. Xem ví dụ này:

C Code:

```
if (tuoi == 2)
{
    printf ("Chao baby !");
}
else if (tuoi == 6)
{
    printf ("Chao nhoc !");
}
else if (tuoi == 12)
{
    printf ("Chao cau be !");
}
else if (tuoi == 16)
{
    printf ("Chao chang trai !");
}
else if (tuoi == 18)
{
    printf ("Chao anh !");
}
else if (tuoi == 68)
{
    printf ("Chao ong !");
}
else
{
    printf ("Toi khong co cau chao danh cho ban ");
}
```

Thiết lập một switch

Những người lập trình rất ghét việc làm một việc gì đó nhiều lần, chúng ta đã có cơ hội chứng minh điều này trước đó. Và để tránh lặp đi lặp lại nhiều lần việc kiểm tra giá trị của mỗi một biến số, người ta đã tạo ra một cấu trúc khác ngoài "if... else"

Và người ta gọi nó là "switch". Đây là một switch được thực hiện trên ví dụ mà ta vừa thấy :

C Code:

```
switch (tuoi)
{
    case 2:
        printf("Chao baby !");
        break;
    case 6:
        printf("Chao nhoc !");
        break;
    case 12:
        printf("Chao cau be !");
        break;
    case 16:
        printf("Chao chang trai !");
        break;
    case 18:
        printf("Chao anh!");
        break;
    case 68:
        printf("Chao ong !");
        break;
    default:
        printf("Toi khong co cau chao danh cho ban ");
        break;
}
```

Hãy lấy ví dụ của tôi làm cơ sở cho việc tạo một switch khác của riêng bạn. Tuy rằng chúng ta ít sử dụng nó, nhưng việc này khá tiện lợi vì nó giúp ta viết ít hơn. 😊

Ý nghĩa viết "**switch (bienso)**" là "**Tôi sẽ kiểm tra giá trị của biến số bienso**".

Tiếp đó bạn hãy mở một dấu gộp và đóng nó lại ở phía sau { ... }

Bên trong, bạn sẽ tạo nên tất cả các trường hợp: **case 1, case 2, case 4, case 5, case 45 ...**



Các bạn bắt buộc phải ghi instruction break ở cuối mỗi trường hợp. Nếu bạn không làm, nó sẽ đọc những instruction dành cho các trường hợp khác ở phía dưới.

Instruction break ; là lệnh yêu cầu máy tính ra khỏi những dấu gộp.

Cuối cùng, trường hợp « default » tương ứng với « else » mà chúng ta đã biết trước đó. Nếu biến số không mang giá trị nào được nêu ra ở trước đó, thì máy tính sẽ đọc instruction nằm trong default.

Tạo một menu với switch

switch thường xuyên được sử dụng để tạo những menu trên console.
Trên console, để tạo một menu, chúng ta sử dụng *printf* để hiển thị những lựa chọn khác nhau.
Mỗi lựa chọn sẽ được đánh số, người sử dụng phải nhập vào số của lựa chọn mà họ muốn

Đây là một ví dụ mà console sẽ phải hiển thị:

Console:

```
=== Menu ===  
1. Pho  
2. Bun bo Hue  
3. Mi Quang  
4. Thit cay  
Lua chon cua ban ?
```

(Bạn cần phải biết là khi tôi đánh ra các dòng này tôi đang cảm thấy rất đói bụng)

Và đây là nhiệm vụ của các bạn: Hãy tạo lại một menu như thế nhờ vào hàm *printf* (quá dễ), sử dụng *scanf* để lưu lại lựa chọn của người sử dụng vào biến số *luachonMenu* (quá dễ 😊), và cuối cùng hãy sử dụng một *switch* để nói với người sử dụng biết rằng "Bạn đã lựa chọn".

Nào, tiến hành thôi.



Đáp án

Đây là kết quả mà tôi mong muốn rằng tự bạn có thể tìm ra :

C Code:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int luachonMenu;

    printf("=== Menu ===\n\n");
    printf("1. Pho\n");
    printf("2. Bun bo Hue\n");
    printf("3. Mi Quang\n");
    printf("4. Thit cay\n");
    printf("\nLua chon cua ban ? ");
    scanf("%d", &luachonMenu);

    printf("\n");

    switch (luachonMenu)
    {
        case 1:
            printf("Ban da chon Pho. Lua chon tuyet voi !");
            break;
        case 2:
            printf("Ban da chon Bun bo Hue. Lua chon chinh xac !");
            break;
        case 3:
            printf("Ban da chon Mi Quang. Qua tuyet !");
            break;
        case 4:
            printf("Ban da chon Thit cay. Hay den quan nhau !");
            break;
        default:
            printf("Ban da khong nhap dung so can thiet, ban khong duoc an gi het !");
            break;
    }
    printf("\n\n");
    return 0;
}
```

Tôi hi vọng rằng bạn đã không quên việc đặt "default" ở cuối của *switch* !

Trên thực tế, nếu bạn lập trình thì bạn phải nghĩ đến tất cả các trường hợp. Bạn đã nói rõ ràng rằng hãy nhập vào một số từ 1 đến 4, nhưng sẽ có một thằng đàn nào đó sẽ nhập vào "10" hay có thể là "gkfhgs". Tất nhiên đó không phải là điều chúng ta mong muốn. 😊

Tóm lại bạn hãy luôn cẩn trọng: đừng bao giờ tin tưởng vào người dùng, đôi khi họ có thể nhập vào bất cứ cái gì.

Hãy tiên đoán trước một trường hợp « default » hoặc một « else » nếu bạn thực hiện ví dụ trên bởi *if*.

⚠ Tôi khuyên bạn hãy làm quen với cách hoạt động của những menu trên console. Vì chúng ta sẽ thường xuyên lập trình những chương trình chạy trên console và tôi chắc là bạn sẽ cần đến.

Ternary: những conditions rút gọn

Có một cách khác để viết những condition, nhưng rất hiếm.
Chúng ta gọi đó là **ternary expression**.

Cụ thể, nó cũng tương tự như "if... else", chỉ trừ việc tất cả chỉ nằm trên 1 dòng !
Thay vì phải giải thích dài dòng, tôi sẽ cho bạn 2 condition giống nhau: cái thứ nhất sử dụng "if... else", và cái thứ 2 cũng như thế nhưng sẽ sử dụng dạng ternary.

Một condition if... else được biết đến khá nhiều

Hãy giả định rằng chúng ta có một biến số dạng boolean "truongthanh" mang giá trị đúng (1), và sai (0) nếu người đó là con nít.

Chúng ta muốn thay đổi giá trị của biến số "tuoi" dựa vào hoạt động của boolean, sẽ cho giá trị « 18 » nếu người đó trưởng thành và « 17 » nếu người đó là con nít. Tôi đồng ý rằng đây là một ví dụ khá ngu ngốc, nhưng để giới thiệu cho bạn thấy làm cách nào chúng ta có thể sử dụng những **ternary**.

Chúng ta sẽ thực hiện điều đó với *if... else* :

C Code:

```
if (truongthanh)
    tuoi = 18;
else
    tuoi = 17;
```

⚠ Chú thích rằng tôi đã xóa đi những dấu gộp { .. } vì ở đây chỉ có một instruction duy nhất, tôi đã giải thích cho bạn điều này trước đó.

Condition ternary

Đây là một đoạn mã hoàn toàn tương tự đoạn mã vừa rồi, nhưng lần này chúng ta sẽ viết dưới dạng **ternary**:

C Code:

```
tuoi = (truongthanh) ? 18 : 17;
```

Các **ternary** cho phép, chỉ trên 1 dòng, thay đổi giá trị của biến số dựa vào hoạt động của một điều kiện. Ở đây, điều kiện của chúng ta chỉ đơn giản là « **truongthanh** », nhưng nó còn có thể hoạt động trên bất kì điều kiện khác không kể là dài hay ngắn. 😊

Dấu chấm hỏi "?" ở đây có nghĩa là « **có phải bạn là người trưởng thành ?** ». Nếu đúng, nó sẽ đưa giá trị 18 vào biến số **tuoi**, nếu không (dấu ":" có nghĩa là **else** ở đây), nó sẽ đưa giá trị 17.

Những **ternary** thật sự không cần thiết, về cá nhân tôi nghĩ là không nên sử dụng nó nhiều quá vì nó có thể khiến cho việc đọc một đoạn mã khó khăn hơn.

Tuy nhiên, bạn cũng phải hiểu rõ nó vì sẽ có một ngày, bạn rơi vào một đoạn mã với đầy những **ternary** với mọi cách 😊. Bạn sẽ hiểu được nó hoạt động như thế nào.

Và kể từ giây phút này, bạn sẽ thực hiện các **condition** khắp mọi nơi trong chương trình của bạn, vì vậy tốt hơn bạn hãy luyện tập với nó 😊

Đây là một ý tưởng để luyện tập (lần này sẽ không có đáp án 😊): hãy tạo một công cụ tính toán trên console. Hiện thị đầu tiên menu yêu cầu người sử dụng chọn lựa những phép tính: (cộng, trừ, nhân, chia... có thể thêm vào căn bậc 2, bằng cách sử dụng thư viện toán học)

Khi mà người sử dụng lựa chọn xong, hãy yêu cầu họ nhập vào các giá trị cần thiết và hiện thị đáp án! 😊

Bạn sẽ sử dụng những gì bạn học được từ trong phần này, tôi muốn nhấn mạnh ở một điểm khác: những boolean.

Thật sự cực kì quan trọng việc nắm vững rằng boolean là những biến số có nghĩa là đúng hay sai tùy theo giá trị của nó (0 là sai, 1 là đúng).

Chương tiếp theo sẽ sử dụng lại những **boolean** và các **condition**, vì vậy bạn hãy chuẩn bị tốt trước khi sẵn sàng 😊

Cố gắng lên nào !

Bài 7: Loop (Vòng Lặp)

Chúng ta đã biết làm cách nào để thiết lập những conditions, bây giờ chúng ta sẽ học cách thực hiện các vòng lặp. 😊 Vậy loop (vòng lặp) là gì ?

Là một phương pháp giúp ta có thể lặp lại nhiều lần một nhóm các instructions. Rất tiện lợi, đặc biệt là trong bài thực hành đầu tiên đang đợi bạn sau khi kết thúc phần này 😊
Hãy thư giãn, không có gì phức tạp cả. Ở phần trước, chúng ta đã thấy thế nào là những Boolean, tương đối khó nuốt.

Nhưng bây giờ thì trôi chảy hơn rồi, và bài thực hành sắp tới cũng không khiến bạn gặp quá nhiều rắc rối đâu.

Nói chung là hãy cố hiểu rõ nó, vì chúng ta chuẩn bị bước vào chương thứ II, và sẽ có nhiều điều hứng thú hơn rất nhiều.

Thế nào là một vòng lặp?

Tương tự như các conditions, có nhiều cách để thực hiện một vòng lặp. Nhưng dù thực hiện bằng cách nào, thì chúng đều thực hiện một chức năng: **lặp lại nhiều lần các instruction**.

Chúng ta sẽ tìm hiểu 3 dạng vòng lặp thường sử dụng trong C:

1. while
2. do... while
3. for



Biểu đồ thể hiện cách hoạt động của các vòng lặp

Máy tính thực hiện những instruction từ cao xuống thấp (giống như mọi khi)

- i. Khi đã đến cuối của vòng lặp, nó quay trở lại instruction đầu tiên.
- ii. Nó lại đọc những instruction từ cao xuống thấp...
- iii. ... Và lại bắt đầu từ instruction đầu tiên.

Vấn đề xảy ra là nếu ta không dừng nó lại thì máy tính sẽ không ngừng lặp lại những instruction này. Điều này không khiến ta bận tâm vì nó lặp lại theo yêu cầu của chúng ta. 🇪🇺

Và tại đây lại xuất hiện ... **những điều kiện!**

Khi chúng ta tạo một vòng lặp, chúng ta luôn phải tạo một điều kiện. Điều kiện này có nghĩa là “Lặp lại vòng lặp nếu điều kiện này vẫn đúng.”

Có rất nhiều cách để thực hiện nó giống như tôi đã nói. Và sau đây là cách thực hiện một vòng lặp while trên C 😊

While loop (vòng lặp While)

Cách tạo ra một vòng lặp while.

Code C:

```
while (* Dieukien *)
{
    // Cac instructions duoc lap lai
}
```

Chỉ đơn giản như vậy 😊

While có nghĩa là "hễ còn". Chúng ta sẽ nói với máy tính “*Hễ điều kiện vẫn đúng thì lặp lại các instruction được viết trong dấu gộp.*”

Tôi thực hiện một ví dụ đơn giản: Yêu cầu người sử dụng nhập vào số 47. Hễ người sử dụng không thực hiện đúng, thì máy tính sẽ tiếp tục yêu cầu nhập vào số 47. Chương trình sẽ không dừng lại nếu như người sử dụng vẫn không nhập vào số 47.

Code C:

```
int giatriCanNhap = 0;
while (giatriCanNhap != 47)
{
    printf("Hay nhap vao so 47 ! ");
    scanf("%d", &giatriCanNhap);
}
```

Và đây là cách mà ví dụ trên thực hiện. Ghi thêm rằng tôi đã ép mình đánh sai 2-3 lần trước khi đánh vào số chính xác. 🤪

Console:

```
Hay nhập vào số 47 ! 10
Hay nhập vào số 47 ! 27
Hay nhập vào số 47 ! 40
Hay nhập vào số 47 ! 47
```

Chương trình tự dừng lại cho đến khi số 47 được nhập vào.
Vòng lặp sẽ lặp lại các instruction nếu như người sử dụng vẫn không nhập vào đúng số 47. Chỉ đơn giản như vậy.

Bây giờ, chúng ta sẽ thử làm một vài điều thú vị hơn: chúng ta muốn vòng lặp sẽ lặp lại nhiều lần một instruction.

Chúng ta sẽ tạo một biến số “counter” có giá trị 0 lúc bắt đầu, chúng ta sẽ tăng dần giá trị đó lên. Bạn còn nhớ **increment** ko? Chúng ta sẽ cộng thêm 1 vào biến số bằng cách viết “bienso++”.

Hãy đọc kĩ đoạn mã này và hãy thử hiểu cách hoạt động:

Code C:

```
long counter = 0;
while (counter < 10)
{
    printf("Xin chao cac ban !\n");
    counter++;
}
```

Kết quả :

Console:

```
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
Xin chao cac ban !
```

Đoạn mã này sẽ lặp lại 10 lần câu “Xin chao cac ban !”.



Chính xác nó hoạt động như thế nào?

Trình tự hoạt động của vòng lặp như sau:

1. Lúc bắt đầu, chúng ta có một biến số *counter* có giá trị là 0.
2. Vòng lặp **while** vẫn lặp lại hễ giá trị counter vẫn bé hơn 10, vì counter có giá trị là 0 lúc bắt đầu, máy tính sẽ đi vào vòng lặp
3. Hàm *printf* sẽ hiển thị ra màn hình câu “Xin chào các bạn !”
4. Máy tính sẽ **tăng giá trị của biến số counter lên 1**, nhờ vào instruction “*counter++*;”. Bây giờ counter có giá trị là 1
5. Đã đến cuối của vòng lặp (dấu **}**), bây giờ chúng ta quay lại từ khi bắt đầu, từ **while**. Chúng ta sẽ kiểm tra lại điều kiện: “*có phải giá trị của counter vẫn bé hơn 10 ?*”. Vâng, giá trị của nó hiện giờ là 1, vậy hãy lặp lại các instruction của vòng lặp. 😊

Và cứ thế tiếp tục... Counter tăng dần các giá trị 0, 1, 2, 3, ..., 8, 9, 10. Đến khi counter có giá trị là 10, điều kiện *counter < 10* không còn chính xác, nên chúng ta sẽ ra khỏi vòng lặp

Mặt khác, chúng ta có thể thấy giá trị của biến *counter* tăng dần theo kích cỡ của vòng lặp. Nếu bạn đã hiểu vấn đề này, thì xem như bạn đã hiểu tất cả về vòng lặp **while**. 😊

Bạn có thể tăng giới hạn của vòng lặp (“<100” thay vì “<10”). Việc này rất hữu ích nếu bỗng dưng bạn bị chép phạt 100 lần 😊

Chú ý những vòng lặp không giới hạn

Khi bạn tạo một vòng lặp, hãy chắc chắn rằng nó có thể dừng lại tại một thời điểm nào đó ! Nếu điều kiện luôn luôn đúng, chương trình của bạn sẽ không bao giờ dừng lại ! Đây là một ví dụ về một vòng lặp không giới hạn:

Code C:

```
while (1)
{
    printf("Vong lap khong gioi han\n");
}
```

Bạn có nhớ những Boolean: **1 = đúng**, **0 = sai**. Tại đây, điều kiện luôn luôn đúng, và máy tính sẽ hiển thị “vong lap khong gioi han” liên tục và không ngừng !



Để dừng lại chương trình, trên Windows bạn không có lựa chọn nào khác ngoài việc nhấn vào dấu X ở góc phải bên trên của console. Trên Linux, bạn có thể nhấn Ctrl + C để dừng chương trình.

Tóm lại hãy chú ý: bằng mọi cách phải tránh rơi vào vòng lặp không giới hạn.

Đôi khi vòng lặp không giới hạn sẽ có lợi, nhất là trong các game điện tử ta sẽ thấy sau này.

Do... while loop (vòng lặp do... while)

Dạng vòng lặp này tương tự như while, chỉ khác một điều là nó ít được sử dụng hơn.

Điều khác biệt so với while là vị trí của điều kiện. Đối với *while* điều kiện nằm ở vị trí bắt đầu vòng lặp, còn ở *do... while*, **điều kiện nằm ở cuối cùng**:

Code C:

```
long counter = 0;
do
{
    printf("Xin chao cac ban !\n");
    counter++;
} while (counter < 10);
```

Có điều gì khác biệt ở đây ?

Rất đơn giản: vòng lặp while luôn chắc chắn rằng nó sẽ không bao giờ hoạt động nếu như điều kiện sai từ khi bắt đầu. Ví dụ, nếu như ta gán cho counter giá trị là 50, nhưng điều kiện sai kể từ khi bắt đầu thì chương trình sẽ không tiến vào vòng lặp.

Đối với vòng lặp do... while thì khác: vòng lặp này luôn luôn thực hiện ít nhất một lần. Thực tế, điều kiện sẽ được kiểm tra ở vị trí kết thúc giống như ta đã thấy. Nếu như biến *counter* có giá trị là 50, mặc dù điều kiện bị sai nhưng vòng lặp vẫn sẽ thực hiện ít nhất một lần.

Đôi khi vòng lặp này khá tiện dụng vì luôn chắc rằng chương trình luôn chạy nó một lần. Nhưng dù gì thì nó vẫn khá hiếm. 😊



Có một điều đặc biệt trong vòng lặp do... while là có một dấu chấm phẩy phía sau while, bạn đừng quên điều đó. Nếu không có nó, chương trình của bạn sẽ không thể biên dịch được

For loop (vòng lặp for)

Về nguyên tắc, vòng lặp while cho phép thực hiện tất cả những vòng lặp mà ta muốn. Nhưng trong nhiều trường hợp, người ta cần một loại vòng lặp khác gọn gàng hơn.

Vòng lặp for được sử dụng khá nhiều trong lập trình. Tôi không thể tính chính xác nhưng tôi chắc rằng nó được sử dụng gấp nhiều lần while, vì vậy bạn cần phải biết rõ 2 loại vòng lặp này. Như tôi đã nói với bạn, vòng lặp for cũng chính là một dạng khác của while. Đây là một ví dụ về while mà ta đã thấy trước đó

Code C:

```
long counter = 0;
while (counter < 10)
{
    printf("Xin chao cac ban !\n");
    counter++;
}
```

Cũng trong trường hợp tương tự nhưng ta nếu ta dùng vòng lặp for:

Code C:

```
long counter;
for (counter = 0 ; counter < 10 ; counter++)
{
    printf("Xin chao cac ban !\n");
}
```

Có bao nhiêu điểm khác biệt?

- Bạn có thể thấy rằng chúng ta đã không khai báo giá trị của biến số ngay sau khi tạo ra nó (nhưng chúng ta có quyền làm điều đó).
- Có rất nhiều thứ trong ngoặc sau for (chúng ta sẽ xem xét sau).
- Cũng không có counter++ trong vòng lặp giống như khi dùng vòng lặp while.

Nó được tìm thấy trong ngoặc () và cũng chính điểm này khiến vòng lặp *for* trở nên thú vị.

Có 3 instructions viết ngắn gọn trong ngoặc và chúng **cách nhau bởi những dấu chấm phẩy**:

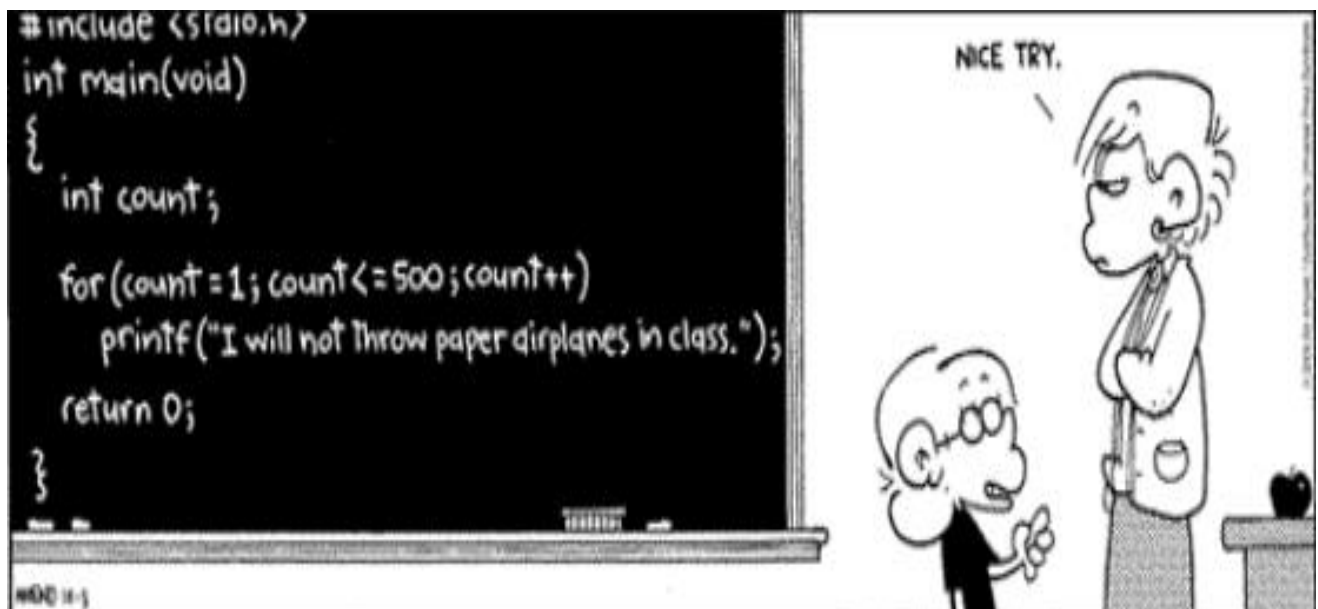
- instruction đầu tiên **dùng để khai báo**: khai báo biến số *counter*. Trong trường hợp của chúng ta, biến số có giá trị là 0.
- instruction thứ hai là **điều kiện**: giống như vòng lặp *while*, đây là điều kiện để vòng lặp được thực hiện. Khi điều kiện vẫn còn đúng, thì vòng lặp lại sẽ được tiếp tục.
- Cuối cùng có một **increment** ở đây: instruction cuối cùng sẽ được thực hiện ở cuối mỗi vòng lặp để cập nhật giá trị của biến số *counter*. Tương tự, chúng ta cũng có thể thực hiện decrement (*counter--*;) hoặc bất kì dạng phép tính nào (*counter += 2*); để tăng hoặc giảm giá trị cho những biến số.

Tóm lại, giống như ta đã thấy vòng lặp *for* không có gì khác biệt ngoài một số thứ được viết ngắn gọn hơn so với vòng lặp *while* 😊

Hãy nắm vững nó, chúng ta sẽ cần sử dụng nó rất nhiều lần! Trong chương tiếp theo, có lẽ chúng ta sẽ mệt mỏi với một ít bài thực hành.

Như bạn đã biết, trong những bài thực hành hầu như sẽ không có thêm kiến thức mới, đây là cơ hội để bạn có thể ứng dụng những gì đã được học trong những bài học trước.

Để kết thúc phần này với một hình vui mà tôi chắc là bạn đã có thể hiểu được ý nghĩa của nó 😄



Bài 8: Test Program: Lớn hơn hay nhỏ hơn

Test Program: Lớn hơn hay nhỏ hơn, chương trình đầu tiên của bạn

Bây giờ chúng ta đến phần Test Program đầu tiên. Phần này sẽ kiểm chứng xem bạn có thể thực hiện lại những gì bạn đã học được không. Bởi vì, lý thuyết thì chỉ có vậy nhưng nếu chúng ta không biết cách thực hành như thế nào thì thật uổng phí thời gian bạn bỏ ra để học chúng. 😊

Bạn có tin không? Nếu bạn đã đến được đây có nghĩa là bạn đã có đủ khả năng tạo ra một chương trình khá thú vị rồi đấy.

Một trò chơi trên console. Khá đơn giản, vì thế tôi chọn nó làm đề bài Test Program đầu tiên.

Chuẩn bị và một vài gợi ý

Nguyên tắc của chương trình

Trước tiên, tôi phải giải thích các bạn chương trình của chúng ta hoạt động dựa vào đâu.

Đây là một trò chơi gọi là «Lớn hơn hay nhỏ hơn»

Nguyên tắc là như sau:

1. Máy tính sẽ chọn ngẫu nhiên một số từ 1 đến 100.
2. Máy tính sẽ yêu cầu bạn đoán số đó là bao nhiêu.
3. Máy tính sẽ so sánh số bạn chọn và số « bí mật » đó. Máy tính sẽ gợi ý cho bạn biết số bí mật này lớn hay nhỏ hơn số mà bạn đã chọn. Nếu đoán đúng thì chương trình dừng lại.
4. Nếu bạn đoán sai ở bước 3, thì máy tính lại kêu bạn đoán tiếp.
5. ... Và máy tính lại gợi ý số bí mật lớn hơn hay nhỏ hơn.
6. Và nó vẫn tiếp tục cho đến khi bạn tìm thấy số bí mật đó.

Mục đích của trò chơi đương nhiên là tìm số bí ẩn với số lần đoán nhỏ nhất 😊

Và đây là màn hình của một phần chơi, và đây là những gì bạn phải làm :

Console:

```
So can tim la bao nhieu ? 50
Lon hon !
So can tim la bao nhieu ? 75
Lon hon !
So can tim la bao nhieu ? 85
Nho hon !
So can tim la bao nhieu ? 80
Nho hon !
So can tim la bao nhieu ? 78
Lon hon !
So can tim la bao nhieu ? 79
Chuc mung ! Ban da tim duoc so bi mat !!!
```

Cách chọn ngẫu nhiên một số



Vậy làm sao để máy tính chọn ngẫu nhiên một số? Tôi vẫn chưa biết làm như thế nào !

Và. 😊

Đến lúc này, bạn vẫn chưa biết cách yêu cầu máy tính chọn ra một số ngẫu nhiên. Tuy máy tính tính rất giỏi việc tính toán nhưng nó không biết cách nào để thực hiện công việc trên nên điều này không hề đơn giản. 😊

Trên thực tế, để « thử » có được một số ngẫu nhiên, người ta phải yêu cầu máy tính thực hiện những phép toán khá phức tạp...

Ta có 2 cách sau:

- Ta yêu cầu người sử dụng nhập vào một số ngẫu nhiên với hàm scanf trước. Điều này dẫn đến phải có hai người cùng chơi trò chơi này: một người nhập vào số bí mật ngẫu nhiên và người còn lại đoán số bí mật đó.
- Hoặc là ta cố gắng khiến máy tính tự tạo ra một số ngẫu nhiên. Lợi thế là chúng ta có thể chơi ngay trò này một mình. Bất lợi... là tôi phải hướng dẫn các bạn làm cách nào để thực hiện 😊

Chúng ta sẽ chọn cách thứ hai, và tôi không hề ngăn cản bạn code cách thứ nhất nếu như bạn muốn. 😊

Để tạo ra một số ngẫu nhiên, người ta sử dụng function *rand()*. Function này sẽ tạo ra một số ngẫu nhiên bất kì. Nhưng chúng ta lại cần một số ngẫu nhiên từ 1 đến 100. (Vì nếu chúng ta không biết được giới hạn của số bí mật, thì trò chơi sẽ vô cùng khó). 😊

Để làm được những việc trên, chúng ta sẽ sử dụng công thức sau (tất nhiên tôi sẽ giải thích cho bạn ý nghĩa của nó là gì ! 🤖):

Code C:

```
srand(time(NULL));  
soBiMat = (rand() % (MAX - MIN + 1)) + MIN;
```

Dòng thứ nhất (với hàm *srand*) nó có tác dụng hỗ trợ cho các hàm random bắt đầu công việc khởi tạo một số ngẫu nhiên. Vâng, hơi phức tạp một tí, tôi đã báo trước cho bạn rồi, *soBiMat* là một biến số chứa số được chọn ra ngẫu nhiên.



Instruction srand phải được chạy duy nhất một lần (ở đầu chương trình). Bắt buộc chỉ một lần, duy nhất một lần. Tác dụng của dòng `srand (time(NULL))`; là để giúp chương trình tránh gặp phải trường hợp chọn trùng 2 số ngẫu nhiên (trường hợp này rất hi hữu nhưng nó vẫn xảy ra, và ta phải tránh nó)

Sau đó bạn có thể sử dụng `rand()` bao nhiêu lần tùy thích nhưng đừng quên rằng không để máy tính thực hiện `srand` nhiều hơn hai lần trong một chương trình.

MAX và MIN là những biến số constants, số đầu tiên là giới hạn lớn nhất (100) và số thứ hai là giới hạn nhỏ nhất (1). Tôi khuyên bạn nên xác định những constants này ở đầu chương trình. Như sau:

Code C:

```
const int MAX = 100, MIN = 1;
```

Những thư viện kèm theo

Để chương trình bạn không bị lỗi, chúng ta cần phải thêm vào ba thư viện: `stdlib`, `stdio` và `time` (cái cuối cùng dùng cho những số ngẫu nhiên).

Và chương trình phải được bắt đầu bởi:

Code C:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Tôi chỉ nói nhiều đó thôi !

Tôi sẽ không giải thích thêm cho bạn vì nếu tiếp tục tôi sẽ lộ ra hết toàn bộ code của chương trình ! 🤔



Để bạn có thể tạo được những số ngẫu nhiên, tôi bắt buộc chỉ cho bạn những dòng codes « ăn liền » ở trên và không giải thích rõ nó hoạt động như thế nào. Tôi thực sự không thích điều này lắm, nhưng vì tôi không còn lựa chọn nào khác trong thời điểm hiện tại. Sau này tôi sẽ chỉ bạn những kiến thức mới giúp bạn hiểu được những vấn đề trên. 😊

Tóm lại, bạn đã có đủ kiến thức, tôi cũng đã giải thích cho bạn nguyên tắc của trò chơi, tôi đã cho bạn thấy một phần màn hình trò chơi của chương trình.

Với nhiều đó thông tin, tôi nghĩ bạn đã có thể tự mình viết chương trình này.

Nào hãy bắt đầu thôi !
Chúc bạn may mắn ! 🍀

Đáp án Test Program "Lớn hơn hay Nhỏ hơn"

Đây là đáp án tôi đưa ra cho bạn:

C Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char** argv)
{
    int soBiMat = 0, soHienTai = 0;
    const int MAX = 100, MIN = 1;
    // Tao mot so ngau nhien
    srand(time(NULL));
    soBiMat = (rand() % (MAX - MIN + 1) + MIN);
    /*Vong lap cua chuong trinh. No se tu lap lai cho den khi nguoi
    choi tim duoc so bi mat*/
    do
    {
        //Yeu cau doan so bi mat
        printf("So can tim la bao nhieu ? ");
        scanf("%d", &soHienTai);
        //So sanh so hien tai voi so bi mat
        if (soBiMat > soHienTai)
            printf("Lon hon !\n");
        else if (soBiMat < soHienTai)
            printf("Nho hon !\n");
        else
            printf("Chuc mung ! Ban da tim duoc so bi mat !!!\n");
    } while (soHienTai != soBiMat);

    return 0;
}
```

Giải thích

Bây giờ tôi sẽ giải thích cho bạn đoạn code của tôi, chúng ta bắt đầu từ dòng đầu tiên.

Preprocessor directives
(những chỉ thị tiền xử lý)

Đó là những dòng bắt đầu bằng # ở đầu đoạn code. Nó tích hợp các thư viện ta cần vào chương trình. Tôi vừa đưa luôn 3 dòng này cho bạn ở phần Test Program, nếu bạn có lỗi ở chỗ này, khả năng của bạn thật sự bá đạo. 😊

Những biến số

Chúng ta không cần nhiều lắm. Chỉ một biến số để chứa số mà người dùng đoán (*soHienTai*) và một biến số khác chứa số bí mật được chọn ngẫu nhiên bởi máy tính (*soBiMat*).

Tôi cũng khai báo những *constants* mà tôi giới thiệu với bạn ở đầu chương hướng dẫn này. Lợi ích của việc khai báo *constant* ở đầu chương trình, là bạn có thể thay đổi độ khó của trò chơi dễ dàng (ví dụ *MAX* = 1000). Chúng ta đơn giản chỉ cần thay đổi giá trị của dòng này và dịch lại chương trình.

Vòng lặp

Tôi chọn vòng lặp *do...while*. Về mặt lý thuyết, vòng lặp *while* cũng hoạt động, nhưng tôi thấy rằng ý tưởng *do...while* sẽ có tính logic hơn.

Tại sao? Bởi vì, bạn hãy nhớ lại xem, *do...while* là vòng lặp chạy ít nhất một lần. Và chúng ta cũng yêu cầu người chơi đoán số bí mật ít nhất một lần (người chơi không thể nào có được kết quả nếu chưa đoán lần nào, làm được thì là thánh rồi!). 😊

Mỗi lần vòng lặp hoạt động, ta sẽ yêu cầu người chơi đoán một số. Ta sẽ đặt số đó vào biến số *soHienTai*. Sau đó so sánh với biến số *soBiMat*. Có 3 khả năng xảy ra:

1. *soBiMat* lớn hơn *soHienTai*, sẽ thông báo "Lon hon !".
2. *soBiMat* nhỏ hơn *soHienTai*, sẽ thông báo "Nho hon !"
3. Vậy nếu *soBiMat* không lớn hơn cũng không nhỏ hơn *soHienTai* ? Bằng nhau là điều chắc chắn ! ở vị trí **else**. Tại trường hợp này, thông báo "Chuc mung ! Ban da tim duoc so bi mat !!!"

Cần phải có một điều kiện cho vòng lặp. Khá đơn giản để nhận ra: vòng lặp vẫn còn lặp lại hễ *soHienTai* khác với *soBiMat*. Nếu 2 số bằng nhau, đồng nghĩa với người chơi đã tìm ra đáp án, vòng lặp ngừng. Chương trình kết thúc. 😊

Ý tưởng cải tiến

Bạn tưởng phần Test Program kết thúc ở đây à? Tôi muốn bạn tiếp tục cải tiến chương trình, đồng thời tập luyện. Đừng bao giờ quên rằng chỉ có luyện tập thường xuyên mới có thể khiến bạn ngày càng tiến bộ! Với những ai đọc bài hướng dẫn của tôi mà không test lại một lần nào thì đó là một sai lầm rất lớn, tôi đã từng nói điều này và tôi sẽ tiếp tục nói lại đây!

Báo cho bạn biết đầu tôi lúc nào cũng đầy ý tưởng, kể cả với một chương trình nhỏ như thế này, tôi không hề thiếu ý tưởng để cải tiến nó đâu ! 🤖

Chú ý: lần này tôi sẽ không đưa cho bạn đáp án nữa, bạn cần phải tự xoay sở lấy! Nếu bạn thật sự gặp vấn đề nan giải, hãy lên các diễn đàn có liên quan đến C. Tìm kiếm trước để xem thắc mắc của bạn đã có người giải thích chưa, nếu không thì tạo một topic khác để đặt câu hỏi. 🗣️

- Tạo một bộ đếm "số lần đoán". Bộ đếm này sẽ là một biến số tăng dần mỗi khi vòng lặp được lặp lại. VD: Khi người chơi tìm ra đáp án, sẽ thông báo "Chúc mừng ! Ban da tim duoc so bi mat trong 8 lan doan !!!"
- Khi người chơi tìm ra đáp án, chương trình dừng lại. Tại sao ta không hỏi người chơi có muốn chơi một ván khác không? Nếu bạn thực hiện điều này, chúng ta cần một vòng lặp bao lấy gần như toàn bộ chương trình, để người chơi không muốn dừng thì trò chơi vẫn được thực hiện lại. Tôi khuyên bạn hãy tạo một biến số dạng **Boolean** *tieptucGame* khởi tạo với giá trị 1. Nếu người chơi muốn dừng lại, hãy cho biến số giá trị 0 để chương trình dừng lại.
- Thêm vào mode chơi cho 2 người ! Cần chú ý là người chơi cần chọn giữa mode dành cho một người hay mode dành cho 2 người ! Bạn phải thêm vào một Menu ở đầu chương trình yêu cầu người chơi lựa chọn mode mong muốn. Sự khác nhau giữa 2 mode chơi là cách tạo *soBiMat*, trường hợp sử dụng *rand()* như ta vừa thấy, trường hợp sử dụng *scanf*.
- Tạo ra nhiều cấp độ khó khác nhau cho người chơi lựa chọn. Bắt đầu, tạo một Menu yêu cầu người chơi lựa chọn. Ví dụ:
 - 1 = từ 1 đến 100.
 - 2 = từ 1 đến 1000.
 - 3 = từ 1 đến 10000.

Nếu bạn thực hiện, bạn cần thay đổi *constant MAX*... Vâng! một *constant* thì không thể nào thay đổi giá trị suốt quá trình chạy chương trình! Đổi tên biến thành *soToiDa* (đồng thời bạn cần xóa đi từ khóa *const* nếu không giá trị đó vẫn là một *constant* đấy !). Giá trị của biến số này phụ thuộc vào cấp độ người chơi lựa chọn

Vậy đó, việc này sẽ khiến bạn phải mất thêm chút ít thời gian. 🧐

Hãy tận hưởng nó và đừng ngại thêm vào những ý tưởng khác để cải tiến thêm trò chơi của bạn. Và nếu như bạn có câu hỏi, đừng ngại lên các 4rum để thảo luận nhé 🗣️..

Bài 9: Function (Hàm)

Bài hướng dẫn về function trong ngôn ngữ C sẽ khép lại chương thứ nhất. Tôi sắp giới thiệu bạn trong phần hướng dẫn này một yếu tố mà mọi chương trình viết bằng ngôn ngữ C đều phải dùng đến.

Chúng ta sẽ học cách xây dựng một chương trình từng chút một... giống như cách bạn chơi Lego.



Tất cả chương trình viết bằng C là tập hợp bởi những mảnh code, và các mảnh code này được gọi là các **functions** !

Cách tạo và gọi một function

Trong các phần trước, chúng ta đã biết một chương trình viết bằng C được bắt đầu bởi một function gọi là *main*. Tôi có một biểu đồ để nhắc lại một số từ ngữ đã học. Chờ tí, để tôi tìm lại...



Ah thấy rồi, nó đây... 🤔

Quen không? Thấy quen chứ?

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Preprocessor Directives

Instructions

Function

Ở trên, chúng ta thấy có những *Preprocessor directives* (Những chỉ thị tiền xử lý). Những *directives* này khá dễ dàng nhận ra: nó bắt đầu bởi dấu # và luôn đặt ở phần đầu *file source*.

Kế tiếp, có một đoạn code mà ta gọi đó là một "function". Và trong hình vẽ bạn nhận ra đó là function *main*. 🤔

Tôi từng nói với bạn là một chương trình viết bằng ngôn ngữ C bắt đầu bởi function *main*, tôi bảo đảm với bạn rằng, điều đó luôn đúng ! 🤖

Nhưng có một điều cần nói, những bài học trước chúng ta chỉ hoạt động trong function *main*, chúng ta chưa bao giờ ra khỏi nó. Hãy nhìn lại các dòng code trước mà chúng ta đã viết, chúng đều nằm trong hai dấu ngoặc { ... } của function *main*.



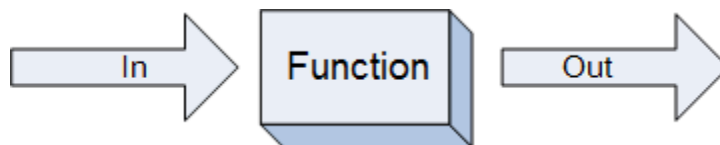
Vậy, việc này có gì không tốt à ?

Không phải, việc này vẫn tốt nhưng đây không phải là một chương trình viết bằng C trong thực tế. Hầu như chẳng có chương trình C nào được viết toàn bộ trong những dấu ngoặc này. Và cho đến thời điểm này, chương trình của chúng ta vẫn còn khá ngắn, nên việc này cũng không xảy ra bất cứ vấn đề nào, nhưng hãy tưởng tượng đến các chương trình viết bởi hàng triệu dòng code! Nếu tất cả đều nằm trong function *main*, loạn ngay ! 😊

Bây giờ chúng ta bắt đầu học cách sắp xếp chúng. Chúng ta sẽ cắt chương trình ra thành nhiều mảnh ghép (cố gắng liên tưởng đến những mảnh lego mà tôi nói với bạn lúc nãy). Mỗi mảnh ghép này chúng ta sẽ gọi là một function.

Một function sẽ thực thi hành động và gửi một kết quả. Đây là một đoạn code dùng để làm những việc có chủ đích nào đó. Một function có đầu vào và ra.

Xem hình:



1. *In (đầu vào)*: ta sẽ đưa vào các thông tin sẽ sử dụng trong function
2. *Tính toán*: Nhờ vào những thông tin đó mà function hoạt động
3. *Out (đầu ra)*: khi được tính toán xong, function sẽ gửi kết quả. Người ta gọi là *Out* hay *return*.

Cụ thể hơn, chúng ta sẽ xem ví dụ về function *triple* có tác dụng nhân 3 một số đưa vào.



Function "triple" sẽ nhân 3 lần
giá trị nhận được

Mục đích của các functions là để đơn giản hóa *code source*, để không phải đánh đi đánh lại cùng một đoạn *code* nhiều lần.

Tưởng tượng một trường hợp sau: về sau, chúng ta sẽ tạo ra một function *moCuaSo* có tác dụng mở một cửa sổ trên màn hình. Một khi function được viết xong (viết là giai đoạn khó khăn nhất), chúng ta chỉ cần bảo "Ê ! function *moCuaSo*, mở giúp tao một cửa sổ !". 😊

Ngoài ra, chúng ta còn dùng nó để viết một function *dichuyenNhanVat*, mục đích là di chuyển một nhân vật trong một trò chơi ra màn hình, v...v... 😊

Biểu đồ của một function

Bạn đã nhận ra cách tạo một function với function *main*. Bây giờ để bạn có thể hiểu rõ hơn, tôi sẽ chỉ ra cho bạn làm cách nào để tạo một function.

Đoạn code sau đây sẽ là biểu đồ của một function. Bạn cần biết và hiểu rõ nó:

Code C:

```
type tenFunction (parameters)
{
    // Các instructions sẽ được thêm vào đây
}
```

Về cấu tạo của function là như vậy, và đây là những điều bạn cần biết thêm:

- *Type* (tương ứng với **out**): là dạng của function. Giống như các *biến số*, các function cũng có dạng tương ứng. *Type* function phụ thuộc vào kết quả cho ra của function. Ví dụ, nếu function cho kết quả *một số thực*, bạn sẽ đặt vào đây *double*. Nếu là *một số tự nhiên*, bạn sẽ đặt vào đây *int* hoặc *long*. Nhưng có những function *không cho ra kết quả gì cả* ! Điều này dẫn đến có 2 loại function khác nhau:
 - Những function *cho ra giá trị* : ta đặt vào đây *type* mà ta biết (*char*, *int*, *double*,...).
 - Những function *không cho giá trị*: ta đặt vào đây một *type* đặc biệt *void* (có nghĩa là trống rỗng).
- *tenFunction*: tên function của bạn. Bạn có thể đặt bất cứ tên gì, nhưng bạn phải tuân thủ cách đặt tên giống như đặt tên *biến số*. (Không dấu, không khoảng trống, ...)
- *parameters* hay có thể gọi là **tham số** (tương ứng với **in**): Trong dấu ngoặc (...), bạn có thể đưa vào đây những *parameters* cho function. Đó là những giá trị sẽ được dùng đến. Lấy ví dụ, function *triple* ở trên, bạn sẽ gửi một số vào vị trí *parameter*. Function *triple* sẽ lấy giá trị của số đó và nhân 3 lên. Và sau đó nó sẽ gửi cho ta kết quả



Bạn có thể gửi bao nhiêu *parameter* vào đây cũng được. Bạn cũng có thể không gửi *parameter* nào vào nhưng việc này ít xảy ra hơn.

- Kế tiếp bạn có dấu mở và đóng ngoặc {...} tương ứng với đầu và cuối của function. Trong đó, bạn sẽ đưa vào những *instructions* mà bạn muốn. Trong ví dụ function *triple*, bạn phải viết *instruction* nhân 3 giá trị nhận được.

Function giống như một cỗ máy, nó nhận những giá trị ở đầu vào và cho kết quả ở đầu ra.

Cách tạo một function

Chúng ta sẽ xem ngay ví dụ về function *triple* mà tôi cứ nhắc đi nhắc lại này giờ. Ta sẽ cho function này nhận *một số tự nhiên* dạng *int* và gửi kết quả *một số tự nhiên* cũng dạng *int*. Function này sẽ thực hiện phép tính nhân 3 cho giá trị ta cho nó :

Code C:

```
int triple(int soHang)
{
    int ketqua = 0;
    ketqua = 3 * soHang; // Ta nhân giá trị nhân được lên 3 lần
    return ketqua;      // Ta trả về biến so ketqua có giá trị gấp 3 lần soHang
}
```

Và đó là function thứ nhất của chúng ta ! 😊

Có một điều khá quan trọng ở đây: bạn nhận thấy rằng function này *type int*. Nó sẽ trả về giá trị cũng *type int*.

Trong dấu ngoặc (...) đó là những biến số mà function sẽ **nhận được**. Tại đây, function *triple* nhận một biến số dạng *int* gọi là *soHang*.

Dòng *return* có tác dụng "trả về một giá trị" sau khi tính toán và thường nằm ở cuối của function

Code C:

```
return ketqua;
```

Dòng code này bảo với function: "*dừng lại ở đây và trả về giá trị chứa trong ketqua*". Type của biến số *ketqua* bắt buộc phải là *int* bởi vì function trả về một *int* như tôi nói ở trên. 😊

Biến số *ketqua* được khai báo trong function *triple*. Điều đó có nghĩa là nó chỉ được dùng trong function này, chứ không dùng cho bất kì một function nào khác (ví dụ như function *main*). Tóm lại đây là một biến số riêng của function *triple*.

Nhưng đó có phải là cách viết ngắn gọn nhất của function *triple* không?

Không, vì người ta còn có thể viết chỉ trong một dòng như sau: 😊

Code C:

```
int triple (int soHang)
{
    return 3 * soHang;
}
```

function này thực hiện chính xác như function ở trên và nó được viết lại ngắn gọn

Có chứa parameter hoặc không có parameter

Chứa parameter

Function *triple* của chúng ta chỉ chứa duy nhất một *parameter*, ta còn có thể tạo ra những function khác có chứa nhiều *parameter*. Lấy ví dụ function *congHaiSo* thực hiện việc tính tổng hai số *a* và *b*:

Code C:

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

Chúng ta chỉ cần phân biệt hai *parameter* bằng dấu phẩy như các bạn thấy. 😊

Không chứa parameters

Có nhiều function khác xuất hiện ít hơn, không chứa bất kì *parameter* nào. Những function này sẽ thực hiện những công việc giống nhau. Nếu như function không nhận bất kì giá trị nào để làm việc thì nó chỉ dùng để thực hiện các công việc, ví dụ: hiển thị một đoạn text lên màn hình.

Ta xem ví dụ function *xinchao* có tác dụng hiển thị "Xin Chao" lên màn hình:

Code C:

```
void xinchao()
{
    printf("Xin Chao");
}
```

Trong ngoặc (...) tôi không đặt vào đó thứ gì tại vì function này không cần *parameter*. Thêm nữa, tôi sử dụng *type void* mà tôi đã giới thiệu với bạn cho function *xinchao* vì một function không trả về bất cứ giá trị nào sẽ mang *type void*.

Cách gọi một function

Chúng ta sẽ cùng test một đoạn code để xem lại những gì ta vừa học. Chúng ta sẽ sử dụng function *triple* để nhân 3 giá trị của một số.

Trong thời điểm hiện tại, tôi yêu cầu bạn viết function *triple* TRƯỚC function *main*. Nếu bạn đặt ở phía sau, chương trình sẽ không hoạt động. Tôi sẽ giải thích lý do ở các phần sau. 🤔

Và đây là đoạn code bạn cần hiểu và test lại:

Code C:

```
#include <stdio.h>
#include <stdlib.h>
int triple(int soHang)
{
    return 3 * soHang;
}

int main(int argc, char *argv[])
{
    int soNhapVao = 0, soTriple = 0;

    printf("Nhap vao mot gia tri... ");
    scanf("%d", &soNhapVao);

    soTriple = triple(soNhapVao);
    printf("Triple (x3) của gia tri vua roi la %d\n", soTriple);

    return 0;
}
```

Chương trình bắt đầu bằng function *main* như chúng ta đã biết. Ta yêu cầu người dùng cho một giá trị số. Chúng ta sẽ đưa giá trị này vào function *triple*, và sau đó ta thu được kết quả là giá trị của *soTriple*. Hãy chú ý dòng code này, đó là cách ta gọi một function:

Code C:

```
soTriple = triple(soNhapVao);
```

Trong dấu ngoặc, ta đưa vào biến số *soNhapVao*, giá trị của *soNhapVao* sẽ được sử dụng trong function *triple*. Sau đó function này sẽ cho ra một giá trị, giá trị này sẽ được chứa trong biến số *soTriple*. Cũng giống như ta ra lệnh bảo với máy tính : "Yêu cầu function *triple* tính cho tao 3 lần giá trị của *soNhapVao*, và chứa kết quả nhận được trong biến số *soTriple*".

Giải thích lại theo dạng biểu đồ

Tôi nghĩ bạn vẫn còn khó hiểu về việc nó hoạt động chính xác như thế nào đúng không?
Không sao, tôi chắc bạn sẽ hiểu rõ hơn qua biểu đồ sau đây. 😊

Đoạn code sau đã được chú thích thứ tự hoạt động của chương trình. Chương trình sẽ hoạt động theo thứ tự bắt đầu từ 1 đến 9 (tôi nghĩ bạn đã bắt đầu hiểu được vấn đề rồi đó 😊) :

Code C:

```
#include <stdio.h>
#include <stdlib.h>
int triple(int soHang) //6
{
    return 3 * soHang; //7
}

int main(int argc, char *argv[]) //1
{
    int soNhapVao = 0, soTriple = 0; //2

    printf("Nhap vao mot gia tri... "); //3
    scanf("%d", &soNhapVao); //4

    soTriple = triple(soNhapVao); //5
    printf("Triple (x3) cua gia tri vua roi la %d\n", soTriple); //8

    return 0; //9
}
```

Và đây là những gì diễn ra.

1. Chương trình bắt đầu bằng function *main*.
2. Nó chạy những *instructions* theo thứ tự dòng trước đến dòng sau.
3. Nó chạy *instruction* kế tiếp và thực hiện lệnh *instruction* đó yêu cầu (*printf*).
4. Vẫn thế, nó chạy *instruction* kế tiếp và thực hiện lệnh *instruction* đó yêu cầu (*scanf*).
5. Nó chạy *instruction* kế tiếp À! ở đây yêu cầu gọi function *triple*, nên nó phải nhảy lên đoạn code của function *triple* phía trên.
6. Nó chạy function *triple* và nhận giá trị cho *parameter* (*soHang*).
7. Nó tính toán giá trị và kết thúc function. *return* có nghĩa là kết thúc function và cho ra giá trị kết quả.
8. Nó quay về *main* và chạy *instruction* kế tiếp.
9. Một *return* nữa xuất hiện! Vậy là function *main* kết thúc và chương trình kết thúc

Nếu bạn hiểu thứ tự máy tính đọc những *instruction*, coi như bạn đã hiểu vấn đề. 😊
Bây giờ, bạn cần hiểu cách một function nhận giá trị vào tại vị trí *parameters* và trả lại một giá trị khác ở *return*.

2) Function *triple* trả về (*return*) một giá trị. Giá trị này bằng với 3 lần giá trị ban đầu ta đưa vào.

Giá trị trả về sẽ được chứa trong một biến số *soTriple* của function *main*

Dấu "=" có thể hiểu là "gửi giá trị trả về của function đó vào trong biến số này"

```
#include <stdio.h>
#include <stdlib.h>
int triple(int soHang)
{
    return 3 * soHang;
}

int main(int argc, char *argv[])
{
    int soNhapVao = 0, soTriple = 0;

    printf("Nhap vao mot gia tri... ");
    scanf("%d", &soNhapVao);

    soTriple = triple(soNhapVao);
    printf("Triple (x3) cua gia tri vua xoi la %d\n", soTriple);

    return 0;
}
```

1) Giá trị của biến số *soNhapVao* sẽ được đưa vào vị trí *parameter* của function *triple*. Tại đây, giá trị này sẽ được chứa vào một biến số khác được đặt tên là "*soHang*".

Ghi chú: Chúng ta cũng có thể đặt tên giống nhau các biến số của 2 functions khác nhau. Việc này không dẫn đến xung đột vì biến số phụ thuộc vào function chứa nó

Chú thích: Đây không phải trường hợp cho hầu hết các function. Vì nhiều lúc, function không có *parameter* nào, hoặc có rất nhiều *parameter*. Và đôi khi function cho ra một giá trị, hoặc nó không trả về một giá trị nào cả (trường hợp không có *return*).

Test chương trình

Màn hình test chương trình: 😊

Console:

```
Nhap vao mot gia tri ... 10
Triple (x3) cua gia tri vua roi la 30
```



Bạn không bắt buộc phải đưa giá trị của kết quả nhận được vào một biến số! Bạn có thể gửi trực tiếp giá trị kết quả nhận được của function *triple* vào một function khác, giống như việc xem *triple(soNhapVao)* là một biến số.

Tham khảo đoạn code dưới đây, cũng tương tự với đoạn code trên nhưng ở dòng *printf* cuối cùng có một chút thay đổi và ta không sử dụng biến số *soTriple*.

Code C:

```
#include <stdio.h>
#include <stdlib.h>
int triple(int soHang)
{
    return 3 * soHang;
}

int main(int argc, char *argv[])
{
    int soNhapVao = 0;

    printf("Nhap vao mot gia tri... ");
    scanf("%d", &soNhapVao);

    // ket qua cua function triple duoc gui truc tiep den printf ma khong can dua vao bat ki bien so nao

    printf("Triple (x3) cua gia tri vua roi la %d\n", triple(soNhapVao));

    return 0;
}
```

Nhận thấy, *triple(soNhapVao)* được đưa trực tiếp vào *printf*. Máy tính hoạt động như thế nào nếu nó đọc đến dòng này?

Khá đơn giản. Nó đọc đến dòng bắt đầu bằng *printf*, nó sẽ gọi function *printf*. Nó tiếp tục đưa cho *printf* nhưng *parameter* cần thiết. *Parameter* đầu tiên là đoạn văn cần hiển thị và thứ hai là một số. Máy tính thấy rằng để đưa số này vào *printf* thì nó cần dùng function *triple*. Và nó gọi function *triple*, nó thực hiện những phép toán có trong *triple* và khi nó có được kết quả, nó sẽ gửi trực tiếp vào function *printf* !

Thấy có vẻ hơi lằng nhằng giữa các functions phải không? 😊

Để nói đơn giản hơn, một function được phép gọi một function khác khi đến lượt nó, hoặc tự gọi lại chính nó,... Đây là nguyên tắc cơ bản của lập trình trên ngôn ngữ C ! Và tất cả sẽ được tập hợp lại, giống như việc lắp ghép Lego vậy. 😊

Cuối cùng, việc khó khăn nhất chính là cách bạn viết ra các functions. Khi đã viết xong, bạn chỉ cần gọi lại chúng để thực hiện những công việc mà nó có thể làm bằng những phép toán được chứa bên trong. Việc này sẽ đơn giản hóa cách viết chương trình của chúng ta. Và hãy tin tôi, chúng ta sẽ dùng rất thường xuyên đấy! 🙌

Xem thêm vài ví dụ để hiểu rõ hơn

Bạn có thể coi tôi như một kẻ bị ám ảnh bởi những ví dụ. Thật sự lý thuyết rất tốt, nhưng nếu chúng ta chỉ học mỗi lý thuyết thì có nguy cơ trong đầu sẽ không giữ lại nhiều điều đã học và trên hết đôi khi ta không biết sử dụng như thế nào, bạn có thấy tiếc thời gian bỏ ra không? 😞

Bây giờ, tôi sẽ chỉ cho bạn một vài ví dụ về cách sử dụng function và để cho bạn cảm thấy thích nó hơn, tôi bắt buộc phải cho bạn những ví dụ khác nhau trong các trường hợp khác nhau, mục đích cho bạn thấy các sự khác nhau của các *type* function có thể tồn tại.

Tôi cũng không chỉ bạn thêm điều mới mẻ nào ở đây. Và nếu bạn đã hiểu những gì tôi hướng dẫn trước đó thì những ví dụ sau đây bạn có thể hiểu được dễ dàng 😊

Chuyển từ euros sang vietnamese dong

Chúng ta bắt đầu bởi một function khá giống với *triple*, không phải vì thế mà bạn mất hứng thú nhé: một function cho phép ta chuyển từ euros sang vietnamdong. Cho những bạn không thường xuyên theo dõi về tiền tệ thì 1 euro = 22864.0426 vietnamese dong (cập nhật ngày 10/03/2015)

Chúng ta sẽ tạo một function gọi là *conversion*. Function này sẽ nhận một biến số *type double* và trả lại một giá trị *type double* vì chúng ta cần sử dụng đến những số thực. Hãy nghiên cứu đoạn code sau:

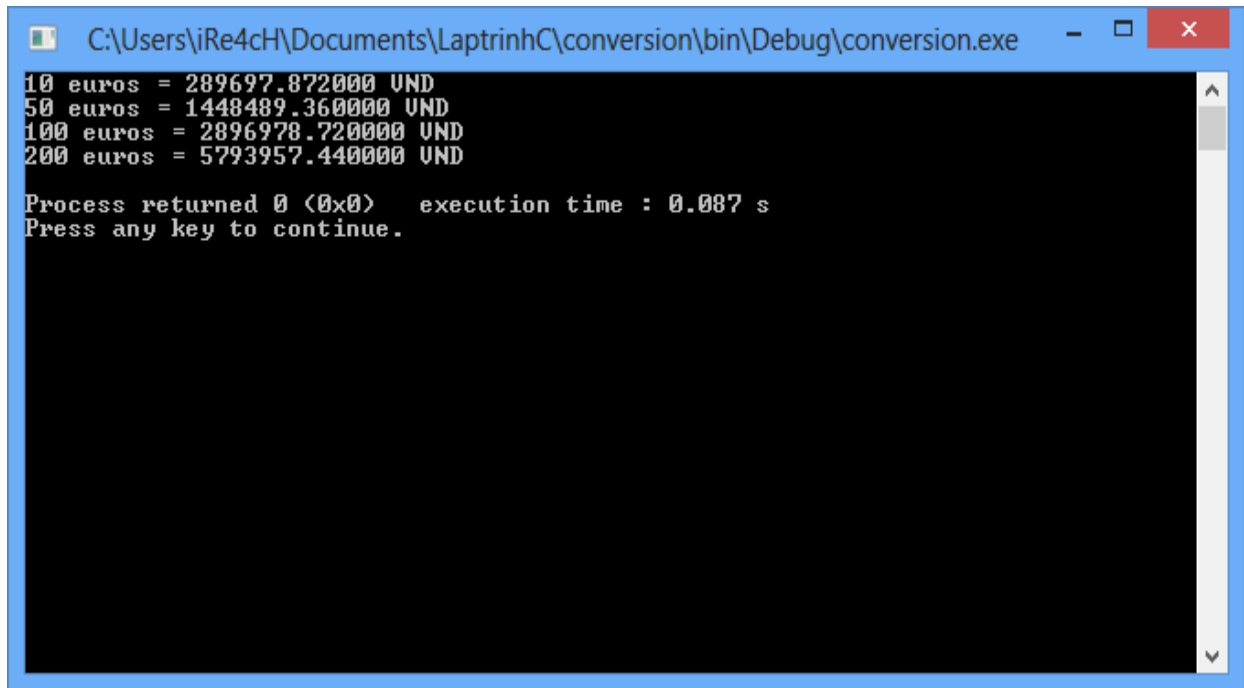
Code C:

```
double conversion(double euros)
{
    double vietnamdongs= 0;

    vietnamdongs = 22864.0426* euros;
    return vietnamdongs;
}
int main(int argc, char *argv[])
{
    printf("10 euros = %f VND\n", conversion(10));
    printf("50 euros = %f VND\n", conversion(50));
    printf("100 euros = %f VND\n", conversion(100));
    printf("200 euros = %f VND\n", conversion(200));

    return 0;
}
```

Console:



```
C:\Users\iRe4cH\Documents\LaptrinhC\conversion\bin\Debug\conversion.exe
10 euros = 289697.872000 VND
50 euros = 1448489.360000 VND
100 euros = 2896978.720000 VND
200 euros = 5793957.440000 VND

Process returned 0 (0x0)   execution time : 0.087 s
Press any key to continue.
```

Như đã báo trước, không có thay đổi lớn nào so với function *triple*. Mặt khác, function *conversion* này có vẻ dài hơn một tí và có thể rút ngắn lại trong một dòng, tôi để bạn làm điều này vì tôi hình như đã hướng dẫn ở phía trên rồi. 😊

Trong function *main*, tôi cố ý để thật nhiều *printf* để bạn thấy lợi ích của việc sử dụng function.

Để có được giá trị của 50 euros, tôi viết *conversion(50)*. Và để có được giá trị của 100 euros tôi chỉ cần thay đổi *parameter* mà tôi đưa vào (50 thành 100).

Và bây giờ đến phiên bạn ! Hãy viết một function thứ 2 (luôn đặt trước function *main*) có tác dụng chuyển đổi ngược lại: từ vietnamese dong thành euros.

Việc này không có gì là khó khăn cả, các bạn chỉ việc thay đổi một phép tính. 😊

Chép phạt

Chúng ta sẽ tham khảo một function không trả về một giá trị nào. Là một function thực hiện việc hiển thị lên màn hình những câu giống nhau với số lượng tùy theo ta yêu cầu. Function này có một *parameter* ở *in*: là số lượng câu mà bạn phải chép phạt.

Code C:

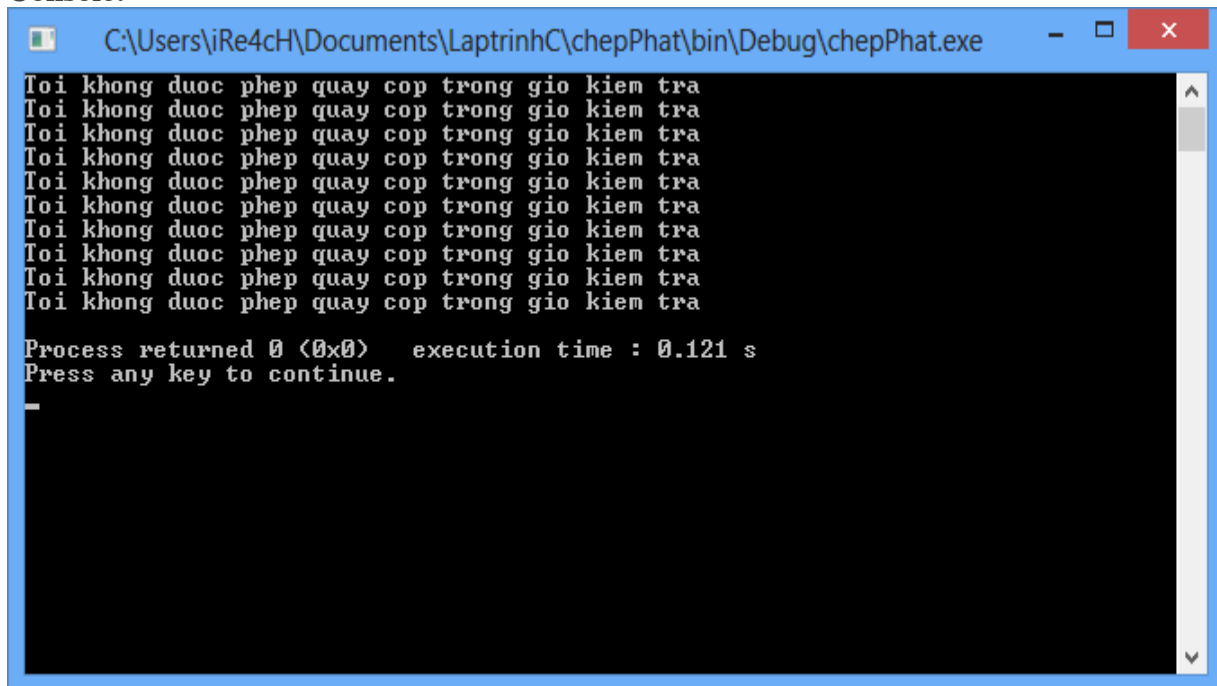
```
void chepPhat(int soDong)
{
    int i;

    for (i = 0 ; i < soDong; i++)
    {
        printf("Toi khong duoc phep quay cop trong gio kiem tra\n");
    }
}

int main(int argc, char *argv[])
{
    chepPhat(10);

    return 0;
}
```

Console:



Đến đây, tôi đã thực hiện một function không trả về giá trị nào. Function dạng này được dùng để thi hành một hành động cụ thể nào đó (ở đây, nó hiển thị lên màn hình những tin nhắn).

Function không trả về giá trị nào có *type* là *void*, cũng chính vì thế trước tên function tôi viết *void*.

Nâng cấp thú vị hơn của bài tập này là ta tạo một function *chepPhat* khác để có thể sử dụng cho bất kì một trường hợp chép phạt nào.

Chúng ta sẽ cho nó 2 *parameter* (tham số): đoạn văn cần chép phạt và số lần cần chép phạt. Vấn đề là, bạn vẫn chưa biết cách sử dụng chuỗi kí tự trong C.

Bạn sẽ được học cách sử dụng những biến số để chứa những chuỗi kí tự ở những phần sau.

Diện tích của một hình chữ nhật

Khá đơn giản để tính diện tích của một hình chữ nhật: chiều dài x chiều rộng. Và function với tên gọi là *dientichHinhChuNhat* sẽ nhận 2 *parameters* (tham số): *chieuDai* và *chieuRong*, nó sẽ trả về giá trị diện tích.

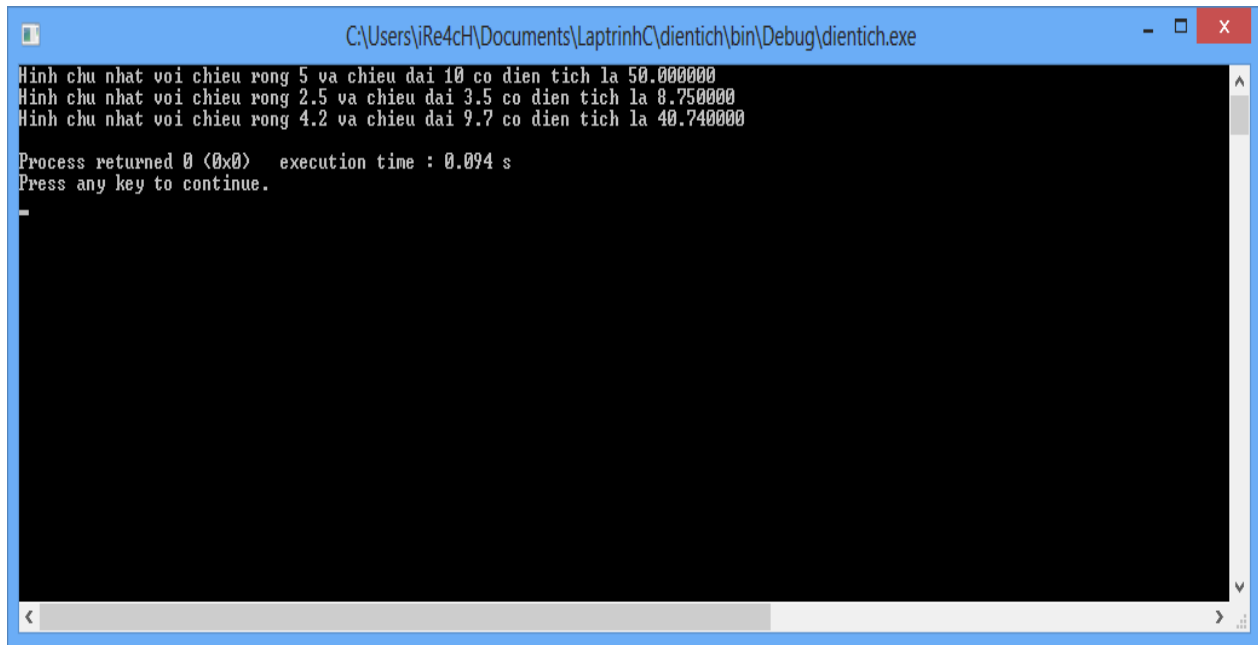
Code C:

```
double dientichHinhChuNhat(double chieuRong, double chieuDai)
{
    return chieuRong * chieuDai;
}

int main(int argc, char *argv[])
{
    printf("Hình chu nhat voi chieu rong 5 va chieu dai 10 co dien tich la %f\n",
    dientichHinhChuNhat(5, 10));
    printf("Hình chu nhat voi chieu rong 2.5 va chieu dai 3.5 co dien tich la %f\n",
    dientichHinhChuNhat(2.5, 3.5));
    printf("Hình chu nhat voi chieu rong 4.2 va chieu dai 9.7 co dien tich la %f\n",
    dientichHinhChuNhat(4.2, 9.7));

    return 0;
}
```

Console:



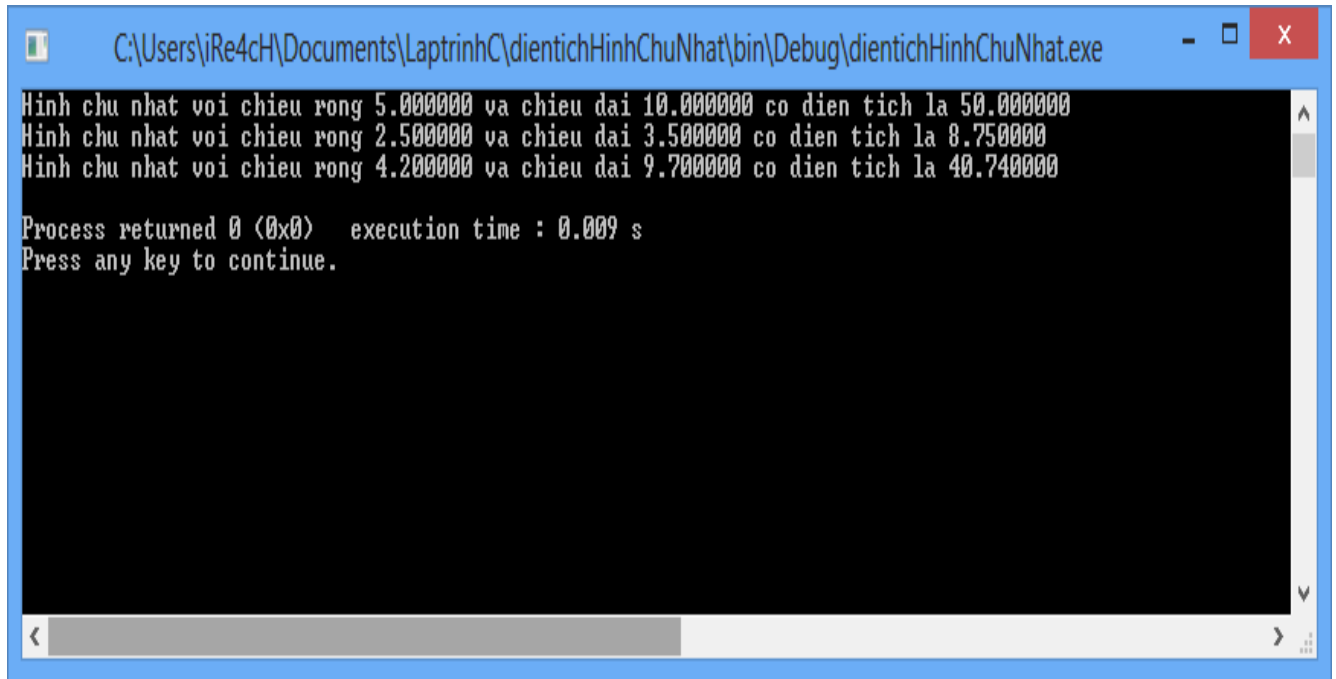
Có cách viết nào khác không?

Có chứ! Trong trường hợp này, function sẽ không trả về kết quả nào, nó sẽ tính toán diện tích và hiển thị luôn kết quả.

Code C:

```
void dientichHinhChuNhat(double chieuRong, double chieuDai)
{
    double dientich = 0;
    dientich = chieuRong * chieuDai;
    printf("Hinh chu nhat voi chieu rong %f va chieu dai %f co dien tich la %f\n", chieuRong,
    chieuDai, dientich);
}
int main(int argc, char *argv[])
{
    dientichHinhChuNhat(5, 10);
    dientichHinhChuNhat(2.5, 3.5);
    dientichHinhChuNhat(4.2, 9.7);
    return 0;
}
```

Console:



```
C:\Users\jRe4ch\Documents\LaptrinhC\dientichHinhChuNhat\bin\Debug\dientichHinhChuNhat.exe
Hinh chu nhat voi chieu rong 5.000000 va chieu dai 10.000000 co dien tich la 50.000000
Hinh chu nhat voi chieu rong 2.500000 va chieu dai 3.500000 co dien tich la 8.750000
Hinh chu nhat voi chieu rong 4.200000 va chieu dai 9.700000 co dien tich la 40.740000

Process returned 0 (0x0) execution time : 0.009 s
Press any key to continue.
```

Giống như bạn thấy, *printf* nằm bên trong function *dientichHinhChuNhat* và hiển thị tương tự đoạn code trước.

Đây là ví dụ về hai cách viết khác nhau của cùng một chương trình. 😊

Một Menu

Đoạn code sau khá thú vị và cụ thể, người ta tạo một *menu()* không nhận bất kì *parameter* (tham số) nào.

Function này sẽ hiển thị một menu và yêu cầu người sử dụng lựa chọn. Function sẽ trả về lựa chọn đó.

Code C:

```
int menu()
{
    int luachon = 0;
    while (luachon < 1 || luachon > 4)
    {
        printf("=== Menu ===\n");
        printf("1. Pho\n");
        printf("2. Bun bo Hue\n");
        printf("3. Mi Quang\n");
        printf("4. Thit cay\n");
        printf("Lua chon cua ban ? ");
        scanf("%d", &luachon);
    }
    return luachon;
}

int main(int argc, char *argv[])
{
    switch (menu())
    {
        case 1:
            printf("Ban da chon Pho. Mot lua chon tuyet voi !\n");
            break;
        case 2:
            printf("Ban da chon Bun bo Hue. Mot lua chon chinh xac !\n");
            break;
        case 3:
            printf("Ban da chon Mi Quang. Qua tuyet !\n");
            break;
        case 4:
            printf("Ban da chon Thit cay. Nao ta cung den quan nhau !\n");
            break;
    }
    return 0;
}
```

Tranh thủ, tôi đã cải tiến luôn menu trên mà bạn đã thấy trước đó. Lần này menu sẽ hiển thị lại mỗi khi người sử dụng không nhập đúng giá trị từ 1 đến 4. Như vậy, không có lỗi xảy ra nếu như function trả lại một con số không liên quan gì đến các lựa chọn trong menu !

Trong main, bạn thấy ta sử dụng một switch (*menu()*). Một khi function *menu()* kết thúc, nó sẽ trả về lựa chọn của người sử dụng trực tiếp vào function *switch*. Một phương pháp khá nhanh và hiệu quả. 🍌

Nào đến phiên bạn! Đoạn code trên vẫn có thể cải tiến tiếp được: Thay đổi chương trình để hiển thị một thông báo lỗi nếu như người sử dụng lựa chọn một số khác ngoài menu thay vì hiển thị lại toàn bộ menu như trên. 😊

Một bài tập nhỏ trước khi kết thúc

Bạn còn nhớ trò chơi "Lớn hơn hay nhỏ hơn" không? Tôi hi vọng bạn vẫn không quên nó. 😊
Bạn sẽ thay đổi bằng cách sử dụng những function. Bạn sẽ tạo ra 2 function: *taoSoNgauNhiem* (có tác dụng tạo ngẫu nhiên một số nằm giữa *MIN* và *MAX*) và *sosanh* (có tác dụng so sánh 2 số và hiển thị lớn hơn, nhỏ hơn hay kết quả đúng) 😊

Bạn không cần gấp gáp để tiến sang chương II của bài hướng dẫn vì ở đó tôi sẽ hướng dẫn các bạn về con trỏ. Khá khó nuốt đấy 😊

Và chương hiện tại là con đường bắt buộc bạn phải bước qua để có thể học những kiến thức mới hơn, nên hãy cố gắng nắm thật vững. 😊

Nhớ là phải luôn cố gắng, đừng bỏ dở giữa chừng, vì phần khó khăn nhất bạn sẽ học nằm ở chương 2, nếu qua được, bạn sẽ thu được kết quả lớn hơn rất nhiều. 😊

Sau đó, ở chương 3, bạn sẽ học cách viết những trò chơi, mở những cửa sổ, sử dụng bàn phím, chuột, joystick, âm thanh ... 😊

Bạn thấy hứng thú hơn chưa? 😊

Thế là xong những kiến thức nền tảng cho một người mới tập tành về C.

Những điều mới mẻ nghiêm túc hơn sẽ bắt đầu ở chương thứ 2 !!! 😊
