DEVIOT - CÙNG NHAU HỌC LẬP TRÌNH IOT

THỰC CHIẾN Lập trình C

```c
#include <stdio.h>

int main(){
    printf("Hello World!");
    return 0;
}
```

www.deviot.vn

www.facebook.com/groups/deviot.vn

THỰC CHIẾN LẬP TRÌNH C CÙNG DEVIOT.VN

Lời nói đầu

Deviot là trung tâm đào tạo về lập trình nhúng và IoT. Với đội ngũ là các anh chị cựu sinh viên trường Đại học Bách Khoa Hà Nội, có nhiều năm kinh nghiệm trong lĩnh vực và giữ nhiều vị trí quan trọng tại các công ty, tập đoàn hàng đầu cả nước. Với sự kỷ luật, kiên trì và tận tâm của mình, đội ngũ đã dành hết tâm huyết để cho ra mắt những sản phẩm chất lượng nhất đến với cộng đồng các bạn sinh viên kỹ thuật. Hi vọng cuốn sách sẽ trở thành hành trang không thể thiếu với các bạn sinh viên.

Hà Nội, ngày …… tháng…… năm……

Xin chào tất cả các bạn độc giả, mình là Ngô Vũ Trường Giang.

Bản thân mình là một cựu sinh viên trường Đại học Bách Khoa Hà Nội, khóa K58 chuyên nghành Kỹ thuật đo và Tin học công nghiệp, sau nhiều năm đi làm, nhận thấy có những vùng kiến thức được sử dụng rất nhiều trong công việc mà hầu hết các bạn sinh viên còn trong trường chưa biết hoặc chưa được tiếp cận dẫn đến có thể đi sai hướng. Mình quyết định cùng đội ngũ **Deviot – Cùng nhau học lập trình nhúng và IoT** ra mắt một sản phẩm có tên "*Thực chiến lập trình C cùng Deviot*", sản phẩm này chắt lọc những vùng kiến thức về lập trình ngôn ngữ C mà bọn mình hay gặp và sử dụng cho các dự án công ty nhất. Hi vọng sản phẩm sẽ đem đến một cái nhìn rõ ràng hơn cho các bạn sinh viên kỹ thuật.

Thông tin liên hệ:

Fanpage: https://www.facebook.com/deviot.vn

Group Facebook: https://www.facebook.com/groups/deviot.vn

Địa chỉ: **Số 101C, ngõ Xã Đàn 2, Hà Nội.**

Xin gửi lời cảm ơn tới những người bạn và các thầy cô giáo giảng viên đã dành thời gian duyệt qua nội dung bản thảo và đưa ra các góp ý để cuốn sách thêm phần cải thiện về nội dung.

Tiến sĩ: Bùi Đình Bá
Hiện đang là giảng viên bộ môn Cơ điện tử - Đại học Bách Khoa Hà Nội.
"*Cuốn sách có nội dung chi tiết, được trình bày dễ hiểu và đào sâu vào các kĩ thuật hay sử dụng trong lập trình C. Phù hợp với các bạn muốn tìm hiểu từ đầu cũng như chuyên sâu.*"

Bạn: Nguyễn Minh Huy
Founder đội ngũ BKStar. Từng dẫn dắt cả đội tham gia các cuộc thi về Robotcon toàn quốc. Từ khi còn học tập trong trường Huy đã tham gia nhiều dự án kỹ thuật lớn và giành luôn tấm bằng xuất sắc của Đại học Bách Khoa Hà Nội.
"*Nội dung cuốn sách hay và được trình bày rất khoa học. Đi từ nội dung cơ bản đến nâng cao nhưng lại rất dễ hiểu chứ không hề hàn lâm. Mình sẽ khuyên các bạn sinh viên sử dụng cuốn sách này.*"

Địa điểm mua sách Offline

1. Linh kiện điện tử Tuhu

Địa chỉ: Số 2, ngõ 106 Lê Thanh Nghị, phường Bách Khoa, Hai Bà Trưng, Hà Nội.

Website: https://mualinhkien.vn/

SDT: 0941344233

2. Quán photo sau thư viên Tạ Quang Bửu

Mục Lục

Bài 1: Giới thiệu về ngôn ngữ lập trình C

1. Ngôn ngữ C là gì ?

C là một ngôn ngữ lập trình cấp trung được phát triển bởi **Dennis M. Ritchie** để phát triển hệ điều hành UNIX tại Bell Labs. C được thực thi lần đầu tiên trên máy tính DEC PDP-11 vào năm 1972.

Năm 1978, **Brian Kernighan** và **Dennis Ritchie** đưa ra mô tả C đầu tiên công khai về C, nay được gọi là tiêu chuẩn K & R.

Ngôn ngữ C được phát triển để tạo ra các ứng dụng hệ thống trực tiếp tương tác với các thiết bị phần cứng như trình điều khiển, kernals vv.

Ngôn ngữ lập trình Java, Hệ điều hành UNIX, trình biên dịch C và tất cả các chương trình ứng dụng UNIX đều đã được viết bằng C.

Lập trình C được coi là cơ sở cho các ngôn ngữ lập trình khác, đó là lý do tại sao nó được biết đến như là ngôn ngữ mẹ. Hầu hết các trình biên dịch, JVMs, Kernals vv được viết bằng ngôn ngữ C và hầu hết các ngôn ngữ theo cú pháp C, như C ++, Java vv.

Nó cung cấp các khái niệm cốt lõi như mảng, chức năng, xử lý tập tin vv được sử dụng trong nhiều ngôn ngữ như C ++, java, C # v.v.

2. Bạn có cần học lập trình C không ?

Câu trả lời là rất rất cần thiết. Vì sao vậy ?

Như giới thiệu ở trên, C được coi là ngôn ngữ mẹ của tất cả ngôn ngữ hiện đại. Vì vậy việc học và biết cách lập trình C gần như là bắt buộc, nó giống như việc các bạn hành trang cho mình một kiến thức nền thật chắc chắn thì mới có thể học lên các ngôn ngữ lập trình bậc cao hơn dễ dàng được.

Vậy lập trình ngôn ngữ C trong nghành IT và lập trình ngôn ngữ C trong ngành nhúng và IoT có gì khác nhau ?

Các bạn biết rằng hầu hết các lập trình viên nghành IT đều viết các chương trình để chạy trên các thiết bị có cấu hình rất mạng ví dụ như máy tính, laptop, cloud… Trong khi đặc thù của nghành nhúng là lập trình cho những con chip, máy tính nhúng có bộ nhớ RAM, ROM và tài nguyên vô cùng hạn chế. Vì vậy chúng ta không thể tùy tiện khai báo sử dụng tài nguyên RAM một cách hoang phí được. Thay vào đó các bạn sẽ phải tính toán và tiết

kiệm từng phần tài nguyên nhỏ. Bất cứ một thao tác sử dụng quá nhiều tài nguyên cũng có thể gây ra các lỗi Hard Fault, Stack overflow…

3. IDE là gì, Text Editor là gì, Compiler là gì ?

IDE (Integrated Development Environment) là môi trường tích hợp dùng để viết code để phát triển ứng dụng. Ngoài ra IDE tích hợp sẵn các tool hỗ trợ khác như trình biên dịch (Compiler), trình thông dịch (Interpreter), kiểm tra lỗi (Debugger), định dạng hoặc highlight code, tổ chức thư mục code, tìm kiếm code…

Text Editor là một trình soạn thảo, không tích hợp sẵn trình biên dịch hoặc trình thông dịch bên trong nó, nghĩa là muốn chạy được ứng dụng, bạn phải dùng riêng compiler bên ngoài. Những Text Editor này thường dùng cho phát triển ứng dụng web, tiêu biểu như Sublime text, Atom, Bracket, Notepad++, VScode…v.v.

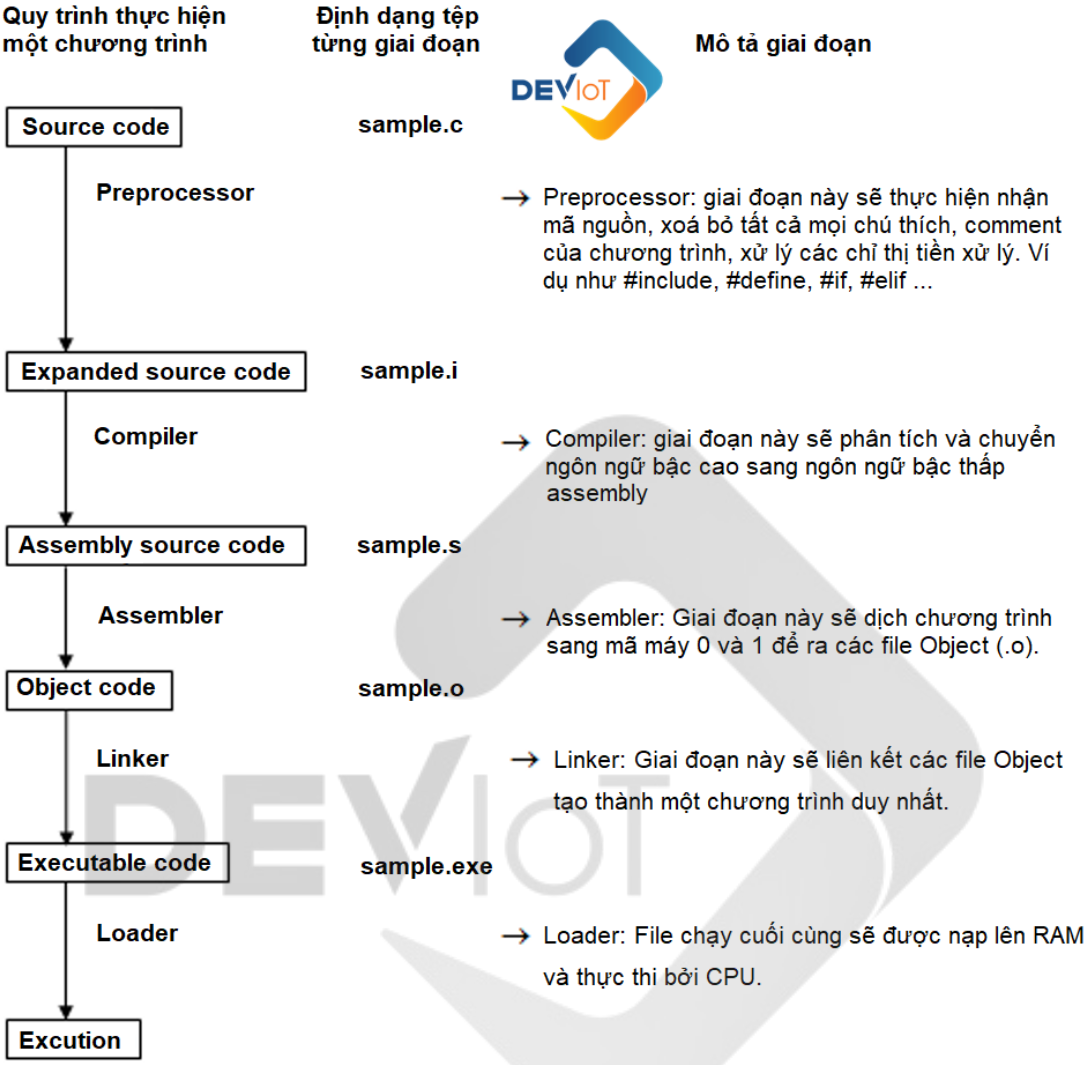
Compiler hay còn gọi là Trình biên dịch, là một chương trình có nhiệm vụ dịch các các code của một ngôn ngữ lập trình tương ứng thành một chương trình tương đương của ngôn ngữ cấp thấp hơn (thường là ngôn ngữ máy).

4. Quá trình biên dịch một chương trình C/C++

Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao sang ngôn ngữ máy. Hiểu một cách đơn giản, các ngôn ngữ như C, C++, Java, Python… đều là các ngôn ngữ lập trình viên. Máy tính sẽ không thể hiểu được các ngôn ngữ này. Đối với ngôn ngữ của máy tính nó chỉ có các mức logic 0 và 1 được đặc trưng bởi các mức điện áp. Chính vì vậy chúng ta cần một quá trình có thể biên dịch ngôn ngữ người dùng sang ngôn ngữ máy. Quá trình đó bao gồm các bước chính sau:

- **Giai đoạn tiền xử lý (Pre-processor):** Giai đoạn này sẽ thực hiện nhận mã nguồn, xóa bỏ tất cả mọi chú thích, comment của chương trình, xử lý các chỉ thị tiền xử lý. Ví dụ như #include, #define, #if, #elif…
- **Giai đoạn dịch Ngôn ngữ bậc cao sang Asembly (Compiler):** Giai đoạn này sẽ phân tích và chuyển ngôn ngữ bậc cao sang ngôn ngữ bậc thấp assembly.
- **Giai đoạn dịch asembly sang ngôn ngữ máy (Asembler):** Giai đoạn này sẽ dịch chương trình sang mã máy 0 và 1 để ra các file Object (.o).
- **Giai đoạn liên kết (Linker):** Giai đoạn này sẽ liên kết các file Object tạo thành một chương trình duy nhất.

- **Giai đoạn thực thi (Loader):** File chạy cuối cùng sẽ được nạp lên RAM và thực thi bởi CPU.



5. Một số trang web học C bằng tiếng việt hiệu quả

- deviot.vn
- vietjact.com
- nguyenvanhieu.vn
- stdio.vn
- daynhauhoc.com
- codelearn.io

6. Cách học C hiệu quả

Một trong những cách học lập trình hiệu quả chính là thực hành code thật nhiều và thật nhiều. Kiến thức sẽ được cải thiện rõ rệt qua từng dự án thay vì chúng ta chỉ đọc sách và xem lời giải.

Trong lập trình nhúng, ngôn ngữ C đặc biệt quan trọng, do tài nguyên bị giới hạn, nhiều dòng chip chỉ có thể được lập trình bằng ngôn ngữ C thay vì các ngôn ngữ bậc cao khác. Việc nắm chắc kiến thức C là tiên quyết nếu không bạn sẽ rất dễ gặp phải những lỗi hóc búa liên quan tới tràn bộ nhớ Heap, Stack …

7. Công cụ lập trình

Đối với việc thực hành lập trình C trên máy tính, có rất nhiều công cụ hỗ trợ lập trình có thể kể đến như:

- Dev-C++
  https://sourceforge.net/projects/orwelldevcpp/
- Code Block
  http://www.codeblocks.org/
- Visual Studio Code
  https://code.visualstudio.com/Download

  Các bạn chỉ việc tải phần mềm về cài đặt vào máy tính cá nhân và bắt đầu lập trình được rồi.

8. Thực hành chương trình đầu tiên “Hello World”

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

Sau đây là các phân tích theo từng dòng mã của ví dụ trên.

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

Dòng đầu tiên này là một chỉ thị tiền xử lý #include. Điều này sẽ làm cho bộ tiền xử lý (bộ tiền xử lý này là một công cụ để kiểm tra mã nguồn trước khi nó được dịch) tiến hành thay dòng lệnh đó bởi toàn bộ các dòng mã hay thực thể trong tập tin mà nó đề cập đến

(tức là tập tin stdio.h). Dấu ngoặc nhọn bao quanh stdio.h cho biết rằng tập tin này có thể tìm thấy trong các nơi đã định trước cho bộ tiền xử lý biết thông qua các đường tìm kiếm đến các tập tin header. Tập hợp các tập tin được khai báo sử dụng qua các chỉ thị tiền xử lý còn được gọi là các tập tin bao gồm.

```
int main()
```

Dòng trên biểu thị một hàm chuẩn tên main. Hàm này có mục đích đặc biệt trong C. Khi chương trình thi hành thì hàm main() được gọi trước tiên. Phần mã int chỉ ra rằng giá trị trả về của hàm main (tức là giá trị mà main() sẽ được trả về sau khi thực thi) sẽ có kiểu là một số nguyên. Còn phần mã (void) cho biết rằng hàm main sẽ không cần đến tham số để gọi nó.

```
{
```

Dấu '{' cho biết sự bắt đầu của định nghĩa của hàm main.

```
printf("Hello World!");
```

Dòng trên gọi đến một hàm chuẩn khác tên là printf. Hàm này đã được khai báo trước đó trong tập tin stdio.h. Dòng này sẽ cho phép tìm và thực thi mã (đã được hỗ trợ sẵn) với ý nghĩa là hiển thị lên đầu ra chuẩn dòng chữ “Hello World”.

```
return 0;
```

Dòng này sẽ kết thúc việc thực thi mã của hàm main và buộc nó trả về giá trị 0.

```
}
```

Dấu '}' cho biết việc kết thúc mã cho hàm main.

Bài 2: Toán tử trong C và các thuật toán sắp xếp

1. Toán tử trong C

1.1 Toán tử số học

Toán tử	Ý nghĩa
+	phép toán cộng
–	phép toán trừ
*	phép toán nhân
/	phép toán chia lấy phần nguyên
%	phép toán lấy số dư (chỉ áp dụng cho số nguyên)

Ví dụ:

```
1+2=3
3-2=1
1*3=3
10/3 = 3
10%3 = 1
```

1.2 Toán tử tăng giảm

Toán tử **++**: Tăng giá trị lên 1 đơn vị.

- **x++:** thực hiện lệnh trước rồi mới tăng x lên 1 đơn vị.
- **++x:** tăng x lên 1 đơn vị rồi mới thực hiện lệnh.

Toán tử **--**: Giảm giá trị đi 1 đơn vị.

- **x--:** thực hiện lệnh trước rồi mới giảm x đi 1 đơn vị.
- **--x:** giảm x đi 1 đơn vị rồi mới thực hiện lệnh.

Ví dụ 1:

```
#include <stdio.h>

int main()
{
    int x = 10;
    int y = 10;
```

```
    printf("Truoc x = %d\n",++x);    // cộng trước rồi thực hiện in
    printf("Truoc x = %d\n",x);

    printf("Truoc y = %d\n",y++);    // in trước rồi thực hiện cộng
    printf("Truoc y = %d\n",y);
    return 0;
}
```

Kết quả:

```
Truoc x = 11
Truoc x = 11
Truoc y = 10
Truoc y = 11
```

Ví dụ 2:

```
#include <stdio.h>

int main()
{
    int x = 10;
    int y = 10;

    printf("Truoc x = %d\n",--x);    // trừ trước rồi thực hiện in
    printf("Truoc x = %d\n",x);

    printf("Truoc y = %d\n",y--);    // in trước rồi thực hiện trừ
    printf("Truoc y = %d\n",y);
    return 0;
}
```

Kết quả:

```
Truoc x = 9
Truoc x = 9
Truoc y = 10
Truoc y = 9
```

1.3 Toán tử gán

Toán tử	Viết gọn	Viết đầy đủ
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Ví dụ:

```
x += 2;  tương đương x = x + 2;
x -= 2;  tương đương x = x - 2;
x *= 2;  tương đương x = x * 2;
x /= 2;  tương đương x = x / 2;
x %= 2;  tương đương x = x % 2;
```

1.4 Toán tử quan hệ

Toán tử	Ý nghĩa	Ví dụ
==	so sánh bằng	6 == 4 cho kết quả là 0
>	so sánh lớn hơn	6 > 4 cho kết quả là 1
<	so sánh nhỏ hơn	6 < 4 cho kết quả là 0
!=	so sánh khác	6 != 4 cho kết quả là 1
>=	lớn hơn hoặc bằng	6 >= 4 cho kết quả là 1
<=	nhỏ hơn hoặc bằng	6 <= 4 cho kết quả là 0

Ví dụ:

```
#include <stdio.h>

int main()
{
    int x = 100;
    int y = 100;

    if(x == y)
        printf("x bang y\n");

    if(x > y)
        printf("x lon hon y\n");

    if(x < y)
        printf("x nho hon y\n");

    if(x != y)
        printf("x khac y\n");

    if(x >= y)
        printf("x lon hon hoac bang y\n");

    if(x <= y)
        printf("x nho hon hoac bang y\n");
    return 0;
}
```

Kết quả :

```
x bang y
x lon hon hoac bang y
x nho hon hoac bang y
```

1.5 Toán tử logic

Toán tử &&: là toán tử AND, trả về true khi và chỉ khi tất cả các toán hạng đều đúng.

Toán tử ||: là toán tử OR, trả về true khi có ít nhất 1 toán hạng đúng.

Toán tử !: là toán tử NOT, phủ định giá trị của toán hạng.

Ví dụ:

```
#include <stdio.h>

int main()
{
    int x = 10;
    int y = 10;

    if((!x) || (!y) || ((x == 10) && (y == 10)))
        printf("x,y thoa man dieu kien\n");
    else
```

```
        printf("x,y khong thoa man dieu kien\n");
    return 0;
}
```

Kết quả:

```
x,y thoa man dieu kien
```

1.6 Toán tử thao tác trên bit

Phép thao tác trên bit	Kí hiệu
Phép AND	&
Phép OR	|
Phép phủ định NOT	~
Phép XOR	^
Phép dịch trái - Shift left	<<
Phép dịch phải - Shift right	>>

Phép AND bit

Ta có bảng logic của phép AND như sau

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Ví dụ:

0011
1001

0001

Phép OR bit

Ta có bảng logic của phép OR như sau

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Ví dụ:

0011
1001

1011

Phép NOT bit

Ta có bảng logic của phép NOT như sau

A	NOT A
0	1
1	0

Ví dụ:

0011

1100

Phép XOR bit

Ta có bảng logic của phép XOR như sau

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Ví dụ:

0011
1001

1010

Phép dịch trái bit

Khi toán tử dịch trái được thực hiện trên một toán hạng, những bit của toán hạng được dịch về bên trái. Các bit bị chuyển sang trái bị mất và 0 thay vào phía bên phải của toán hạng.

Ta cũng có: A << n = A * 2n

Ví dụ:

(00000001b) << 1 = (00000010b)

(00001111b) << 2 = (00111100b)

(00111100b) << 4 = (11000000b)

Phép dịch phải bit

Khi toán tử dịch phải được thực hiện trên một toán hạng, những bit của toán hạng được dịch về bên phải. Các bit bị chuyển sang phải bị mất và 0 thay vào phía bên trái của toán hạng.

Ta cũng có: A << n = A / 2n

Ví dụ:

(11000000b) >> 1 = (01100000b)

(11000000b) >>2 = (00110000b)

(00111100b) >> 4 = (00000011b)

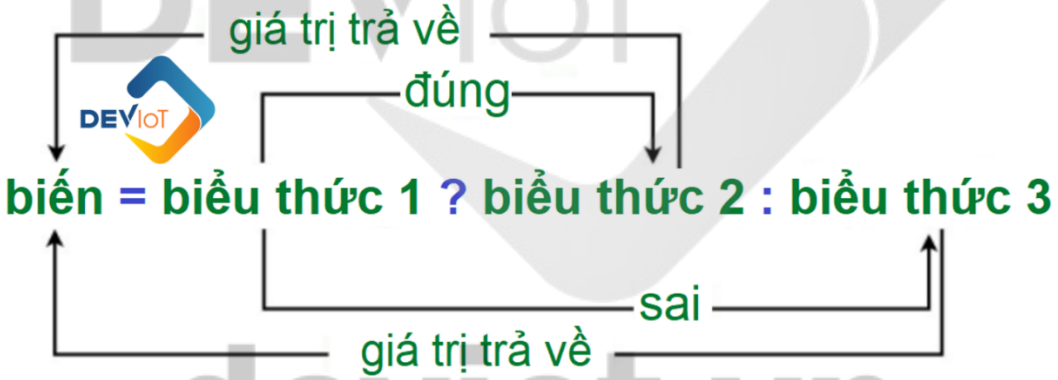
Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

1.7 Toán tử 3 ngôi

Cú pháp

```
bieu_thuc_1 ? bieu_thuc_2 : bieu_thuc_3;
```

Nếu như điều kiện ở `bieu_thuc_1` đúng thì `bieu_thuc_2` trở thành giá trị của toàn bộ biểu thức. Ngược lại nếu như điều kiện ở `bieu_thuc_1` sai thì `bieu_thuc_3` trở thành giá trị của toàn bộ biểu thức.



Ta xét ví dụ sau :

```
giave = (tuoi < 10) ? 10000 : 50000;
```

biểu thức này tương đương với

```
if(tuoi < 10)
{
    giave = 10000;
```

```
}
else
{
    giave = 50000;
}
```

Như vậy ta thấy việc sử dụng toán tử điều kiện đã giúp thu gọn được đáng kể số lượng dòng code phải không nào.

Ví dụ:

```
#include <stdio.h>

int main()
{
    // định nghĩa các biến
    int n1 = 16, n2 = 8, max;

    // Số lớn nhất giữa n1 và n2
    max = (n1 > n2) ? n1 : n2;

    // In ra số lớn nhất
    printf("So lon nhat giua %d va %d la %d.", n1, n2, max);

    return 0;
}
```

Kết quả:

```
So lon nhat giua 16 va 8 la 16.
```

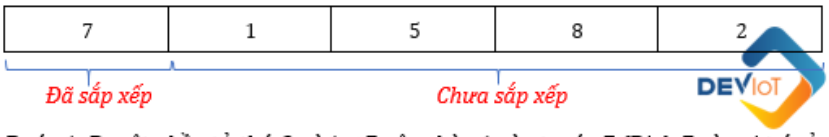
Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

2. Ba thuật toán sắp xếp được sử dụng nhiều trong C

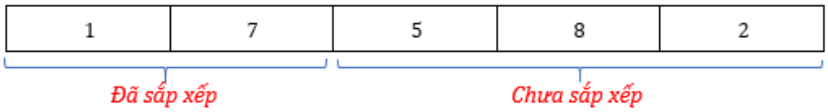
2.1 Thuật toán chèn(insertion sort)

Sắp xếp chèn (insertion sort) là thuật toán sắp xếp cho một dãy đã có thứ tự. Ta chia dãy thành dãy đã sắp xếp và dãy chưa sắp xếp. Chèn thêm một phần tử c từ dãy chưa sắp xếp vào vị trí thích hợp của dãy số đã sắp xếp sao cho dãy số vẫn là dãy sắp xếp có thứ tự. Sau đó ta lặp lại các thao tác cho đến khi dãy chưa sắp xếp không còn phần tử nào nữa.

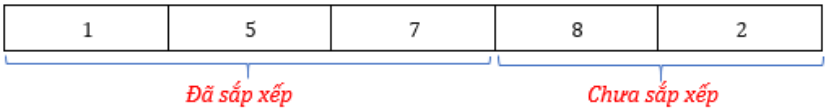
Ta xét ví dụ sau để hiểu nguyên lý sắp xếp



Bước 1: Duyệt phần tử thứ 2, vì 1 < 7 nên chèn 1 vào trước 7 (Dịch 7 vào vị trí của 1, rồi chèn 1 vào vị trí trống)

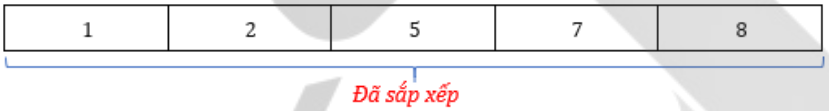


Bước 2: Duyệt phần tử thứ 3, vì 1 < 5 < 7 nên chèn 5 vào giữa 1 và 7 (Dịch 7 vào vị trí của 5, rồi chèn 5 vào vị trí trống)



Bước 3: Duyệt phần tử thứ 4, vì 8 > 7 nên giữ nguyên 8 tại vị trí

Bước 4: Duyệt phần tử thứ 5, vì 1 < 2 < 5, nên dịch 5, 7, 8 sang phải một vị trí (Đặt 2 vào vị trí trống



Ta có đoạn code thực hành thuật toán như sau:

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

void insertSort(int arr[], int N);
void printArr(int arr[], int N);

void main()
{
    int Arr[] = {0, -5, -10, 10, 5, 15, 20, 35, 25, 40, 30};
    int sizeArr = sizeof(Arr)/sizeof(int);
    insertSort(Arr, sizeArr);
    printArr(Arr, sizeArr);
    getch();
}

// Insertion sort
// Sắp xếp từ nhỏ đến lớn
void insertSort(int arr[], int N)
{
    int i, j, temp;
    for (i = 1; i < N; i++)
    {
        j = i - 1;
        temp = arr[i];
        while(arr[j] > temp && j >= 0)
        {
```

```
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
}

void printArr(int arr[], int N)
{
    int i;
    for (i = 0; i < N; i++)
    {
        printf("%d\t", arr[i]);
    }
}
```

Kết quả:

```
-10 -5 0 5 10 15 20 25 30 35 40
```

2.2 Thuật toán sắp xếp lựa chọn (selection sort)

Sắp xếp chọn (selection sort) là phương pháp sắp xếp bằng cách chọn phần tử bé nhất xếp vào vị trí thứ nhất, sau đó ta xét tiếp tập các phần tử còn lại, ta lại chọn phần tử bé nhất xếp vào vị trí thứ hai, tiếp đó ta lặp lại các bước y hệt cho đến phần tử cuối cùng.

Ta xét ví dụ sau để hiểu nguyên lý sắp xếp

6	3	-1	10	0

Bước 1: chọn phần tử nhỏ nhất -1, đặt -1 vào vị trí thứ nhất (đổi chỗ -1 và 6)

-1	3	6	10	0

Bước 2: chọn phần tử nhỏ thứ nhì là 0, đặt 0 vào vị trí thứ hai

-1	0	6	10	3

Bước 3: chọn phần tử nhỏ thứ ba là 3, đặt 3 vào vị trí thứ ba

-1	0	3	10	6

Bước 4: chọn phần tử nhỏ thứ tư là 6, đặt 6 vào vị trí thứ tư, phần tử lớn nhất là 10 được đẩy về sau cùng

-1	0	3	6	10

Ta có đoạn code thực hành thuật toán như sau:

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

void selSort(int arr[], int N);
void printArr(int arr[], int N);

void main()
{
    int Arr[] = {0, -5, -10, 10, 5, 15, 20, 35, 25, 40, 30};
    int sizeArr = sizeof(Arr)/sizeof(int);
    selSort(Arr, sizeArr);
    printArr(Arr, sizeArr);
    getch();
}

// Sắp xếp từ lớn đến nhỏ
void selSort(int arr[], int N)
{
    int i, j, idx, temp;
    for (i = 0; i < N-1; i++)
    {
        idx = i;
        for (j = i+1; j < N; j++)
        {
            if (arr[i] < arr[j])
            {
                idx = j;
            }
        }
        swap(&arr[i], &arr[idx]);
    }
}

void swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

void printArr(int arr[], int N)
{
    int i;
    for (i = 0; i < N; i++)
    {
        printf("%d\t", arr[i]);
    }
}
```

Kết quả

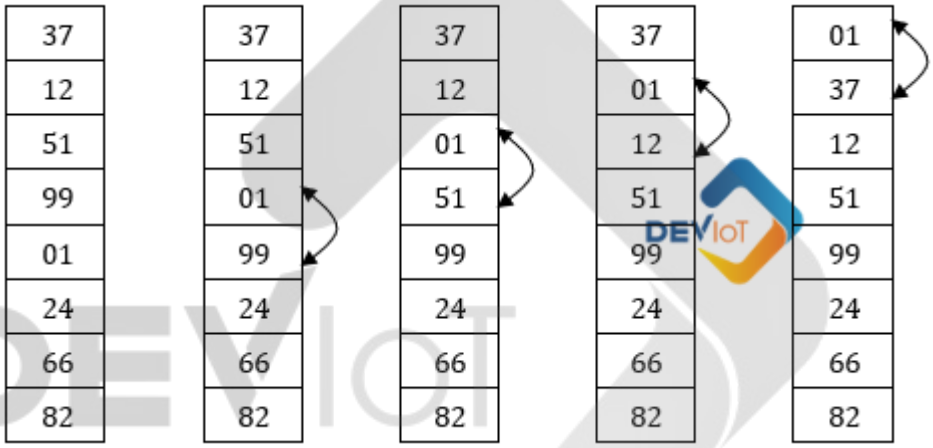
```
-10 -5 0 5 10 15 20 25 30 35 40
```

2.3 Thuật toán sắp xếp nổi bọt (Bubble Sort)

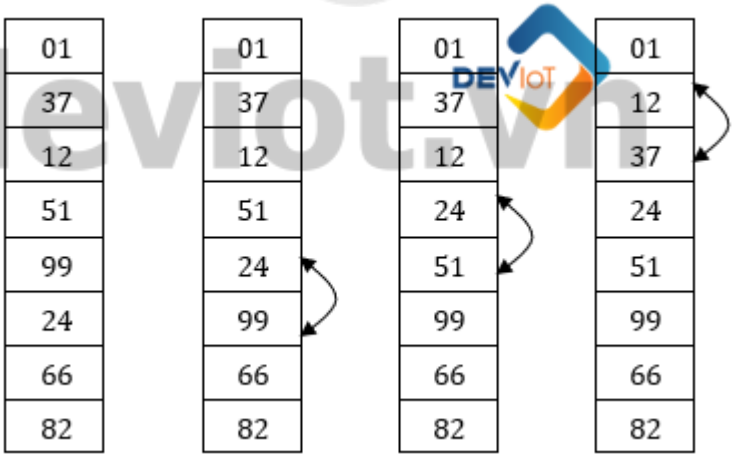
Sắp xếp nổi bọt (bubble sort) là phương pháp sắp xếp đơn giản, dễ hiểu thường được dạy trong khoa học máy tính. Nó bắt đầu so sánh từ 2 phần từ cuối của dãy, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau (trong trường hợp sắp xếp tăng dần). Sau khi so sánh đến phần từ đầu của dãy, ta đã sắp xếp được phần tử có giá trị nhỏ nhất lên đầu dãy rồi. Ta sẽ tiếp tục quay lại lượt so sánh tiếp theo làm đúng theo quy tắc trên cho đến khi không còn cần phải đổi chỗ phần tử nào nữa.

Ta xét ví dụ sau để hiểu nguyên lý sắp xếp

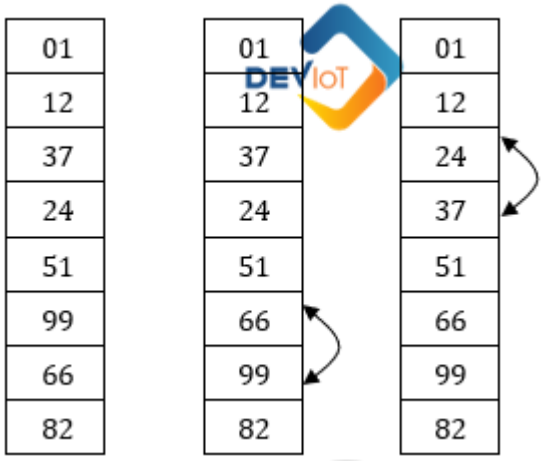
B1: Ta so sánh từ cuối dãy lên đầu dãy, các cặp cần đổi chỗ là (99,01), (51,01), (12,01), (37,01). Phần từ 01 đã nổi lên đầu dãy và là phần tử có giá trị nhỏ nhất.



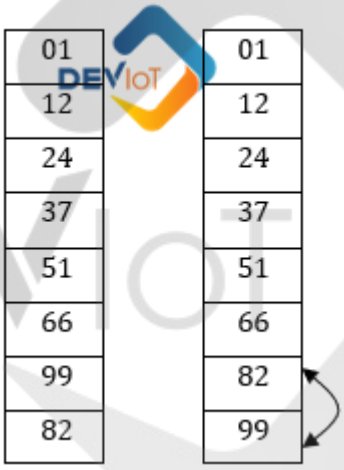
B2: Khi so sánh từ cuối dãy lên, các cặp cần phải đổi chỗ là (99, 24), (51, 24), (12, 37). Phần tử 12 nổi lên vị trí thứ 2.



B3: Khi so sánh từ cuối dãy lên, các cặp cần phải đổi chỗ (99, 66), (37, 24). Phần tử 24 nổi lên vị trí thứ 3.



B4: Khi so sánh từ cuối dãy lên, cặp cần phải đổi chỗ (99, 82).



Ta có đoạn code thực hành thuật toán như sau:

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

void BubbleSort(int arr[], int N);
void printArr(int arr[], int N);

void main()
{
    int Arr[] = {0, -5, -10, 10, 5, 15, 20, 35, 25, 40, 30};
    int sizeArr = sizeof(Arr)/sizeof(int);
    BubbleSort(Arr, sizeArr);
    printArr(Arr, sizeArr);
    getch();
}

// Bubble sort
void BubbleSort(int arr[], int N)
{
```

```
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = N-1; j > i; j--)
        {
            if(arr[j] < arr[j-1])
                swap(&arr[j], &arr[j-1]);
        }
    }
}

void swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

void printArr(int arr[], int N)
{
    int i;
    for (i = 0; i < N; i++)
    {
        printf("%d\t", arr[i]);
    }
}
```

Kết quả:

```
-10 -5 0 5 10 15 20 25 30 35 40
```

Vậy là mình đã giới thiệu tới các bạn 3 thuật toán sắp xếp hay bắt gặp nhất trong lập trình C rồi.

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

Buổi 3: Kiểu dữ liệu và biến

3. Kiểu dữ liệu

3.1 Kiểu số nguyên

Type	Kích thước (byte)	Phạm vi giá trị
Char	1	-128 to 127 hoặc 0 to 255
Unsigned char	1	0 tới 255
Signed char	1	-128 tới 127
Int	2 or 4	-32,768 tới 32,767 hoặc -2,147,483,648 tới 2,147,483,647
Unsigned int	2 or 4	0 tới 65,535 hoặc 0 tới 4,294,967,295
Short	2	-32,768 tới 32,767
Unsigned short	2	0 tới 65,535
Long	8	-9223372036854775808 tới 9223372036854775807
Unsigned long	8	0 tới 18446744073709551615
Uint8_t	1	0 tới 255
Uint16_t	2	0 tới 65535
Uint32_t	4	0 tới 4,294,967,296

3.2 Kiểu số thực

Type	Kích thước (byte)	Phạm vi giá trị
Float	4	1.2E-38 to 3.4E+38
Double	8	2.3E-308 to 1.7E+308
Long double	10	3.4E-4932 to 1.1E+4932

3.3 Kiểu ký tự

Type	Kích thước (byte)	Phạm vi giá trị
Char	1	-128 to 127
Unsigned char	1	0 to 255

3.4 Kiểu void

Kiểu void có nghĩa là "không có giá trị", nó không được dùng trong khai báo biến thông thường mà được sử dụng để chỉ định kiểu của các hàm không có giá trị trả về.

Ví dụ:

```
void tinh_tong (int a, int b);
int gia_tri_trung_binh(void);
void hien_thi_gia_tri_trung_binh(void);
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

4. Định dạng trong C

Đối với hàm printf để có thể in ra được đúng kết quả chúng ta mong muốn cần có kiến thức về các kiểu định dạng trong C.

- **%c** : Ký tự đơn
- **%s** : Chuỗi
- **%d** : Số nguyên hệ 10 có dấu
- **%f** : Số chấm động (VD 6.33 khi in sẽ ra 6.330000)
- **%e** : Số chấm động (ký hiệu có số mũ)
- **%g** : Số chấm động (VD 6.33 khi in sẽ in ra 6.33)
- **%x** : Số nguyên hex không dấu (hệ 16)
- **%o** : Số nguyên bát phân không dấu (hệ 8)
- **%p** : Địa chỉ con trỏ
- **l** : Tiền tố dùng kèm với %d, %x, %o để chỉ số nguyên dài (ví dụ%ld)

Ví dụ 1:

```
#include <stdio.h>

int main()
{
    int a = 18;
    float b = 22.5;
    char c = 'B';
    long d = 3333;
    char* s = "deviot.vn"; // khai bao kieu chuoi

    printf("VI DU VE PRINTF\n");
    printf("Tong cua %d va %f la %f \n", a, b, a+b);
    printf("Tich cua %d va %ld la %ld \n", a, d, a*d);
    printf("Ky tu c la: %c \n", c);
    printf("Chuoi s la: %s \n", s);
    printf("So he 16 va he 8 cua %d la %x va %o \n", a, a, a);
    printf("Ma ASCII cua %c la %d", c, c);

    return 0;
}
```

Kết quả :

```
VI DU VE PRINTF
Tong cua 18 va 22.500000 la 40.500000
Tich cua 18 va 3333 la 59994
Ky tu c la: B
Chuoi s la: deviot.vn
So he 16 va he 8 cua 18 la 12 va 22
Ma ASCII cua B la 66
```

Ví dụ 2:

```
#include <stdio.h>

int main()
{
    int a = 10;
    float b = 23.6;

    printf("%6d \n",a);
    printf("%06d \n",a);
    printf("%6.3f \n",b);
    printf("%.3f \n",b);
    return 0;
}
```

Kết quả:

```
    10
000010
23.600
23.600
```

Chúng ta có thể rút ra 1 số kết luật như sau:

- **%6d**: xuất kí tự có bề rộng là 6.
- **%06d**: xuất kí tự có bề rộng là 6 nhưng hiển thị cả chữ số 0.
- **%6.3f**:xuất ra chữ số thực có bề rộng là 6 (tính cả dấu ".") trong đó có 3 chữ số sau dấu phẩy.
- **%.3f**: xuất ra chữ số thực có 3 chữ số sau dấu phẩy.

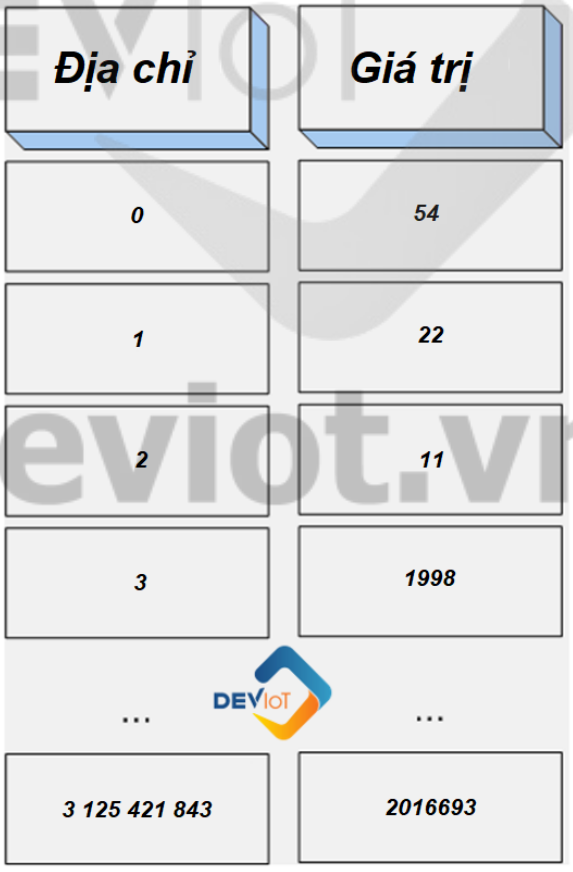
Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

5. Biến số là gì ?

5.1 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.

Ta hình dung 1 chút về cách tổ chức dữ liệu trong RAM, nó hiểu đơn giản như sau:



Bộ nhớ RAM được chia thành 2 cột:

- **Cột địa chỉ:** cho phép máy tính có thể xác định vị trí của biến trong bộ nhớ. Giống như địa chỉ của 1 ngôi nhà vậy.
- **Cột giá trị :** Những con số (giá trị của biến) sẽ được lưu trữ ở đây. Và chúng ta sẽ chỉ đưa vào 1 con số cho 1 địa chỉ.

5.2 Cách khai báo một biến số ?

Cú pháp `kieu_du_lieu danh_sach_bien;`

- `kieu_du_lieu` là kiểu dữ liệu của ngôn ngữ C.

Ví dụ:

```
char, int, float, double ...
```

- `danh_sach_bien` có thể bao gồm một hoặc nhiều tên định danh ngăn cách bởi dấu phẩy.

Ví dụ:

```
int    i, j;
char   ho, ten, c, ch;
float  f, luongthuong, diemthi;
double d;
```

Biến có thể được khởi tạo (được gán các giá trị ban đầu) trong khai báo của nó như sau:

```
char c = 'B';
int x = 200;
float y = 3.14;
```

Ví dụ:

```
#include <stdio.h>

int main ()
{
  /* phan dinh nghia bien: */
  int a, b;
  int c;
  float f;

  /* phan khoi tao gia tri thuc su */
  a = 15;
  b = 35;

  c = a + b;
  printf("Gia tri cua c la : %d \n", c);
```

```
  f = 50.0/3.0;
  printf("Gia tri cua f la : %f \n", f);

  return 0;
}
```

Kết quả:

```
Gia tri cua c la : 50
Gia tri cua f la : 16.666666
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

6. Biến toàn cục

Biến toàn cục được khai báo ở bên ngoài tất cả các hàm.

Biến toàn cục có thể được truy xuất và sử dụng ở mọi hàm trong chương trình.

Biến toàn cục được tồn tại cho tới khi chương trình kết thúc.

Ví dụ:

```
#include<stdio.h>

char x = 'B';   // bien toan cuc
int y = 100;     // bien toan cuc

void hienthi(void) {
    printf("%c\r\n",x);
    printf("%d\r\n",y);
}

int main() {
    hienthi();
    return 0;
}
```

Kết quả:

```
B
100
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

7. Biến cục bộ

Là các biến được khai báo trong một hàm. Biến đó sẽ chỉ có thể tồn tại và sử dụng bên trong hàm. Nó sẽ được giải phóng khi hàm kết thúc thực thi.

Ví dụ:

```
#include<stdio.h>

int ham_cong(int x, int y)
{
    int a;  // bien cuc bo
    int b;  // bien cuc bo
    a = x;
    b = y;
    return a+b;
}

int main() {
    int ket_qua;
    ket_qua = ham_cong(1,2);
    printf("Ket qua 1 + 2 = %d\r\n", ket_qua);
    return 0;
}
```

Kết quả :

```
Ket qua 1 + 2 = 3
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

8. Biến static (biến tĩnh)

8.1 Biến static trong khai báo biến cục bộ

Khi 1 biến cục bộ được khai báo với từ khóa static. Biến sẽ chỉ được khởi tạo 1 lần duy nhất và tồn tại suốt thời gian chạy chương trình. Giá trị của nó không bị mất đi ngay cả khi kết thúc hàm. Mỗi lần hàm được gọi, giá trị của biến chính bằng giá trị tại lần gần nhất hàm được gọi.

Ví dụ:

```
#include<stdio.h>

int in_so_thu_tu(void)
{
    static int x = 0;
    x = x + 1;
```

```
    printf("%d\r\n",x);
}

int main() {
    in_so_thu_tu ();
    in_so_thu_tu ();
    in_so_thu_tu ();
    in_so_thu_tu ();
    in_so_thu_tu ();
    return 0;
}
```

Kết quả:

```
1

2

3

4

5
```

8.2 Biến static trong khai báo biến toàn cục và khai báo hàm

Mỗi project thường sẽ được viết trên nhiều File vì mục đích phân chia module cũng như là để dễ bảo trì. Do có nhiều File nên rất có thể ở các File sẽ có sự trùng lặp trong cách đặt tên biến. Để tránh sự cố sai sót này người ta đưa ra khái niệm biến toàn cục tĩnh và hàm tĩnh.

- Biến toàn cục tĩnh sẽ chỉ có thể được truy cập và sử dụng trong File khai báo nó, các File khác không có cách nào truy cập được.
- Hàm tĩnh sẽ chỉ có thể gọi trong File khai báo nó, các File khác không có cách nào gọi hàm này được.

Ví dụ:

```
//-----------------
//A.c

// biến a này chỉ được sử dụng trong file A.c
static int a;

// hàm hienthi() này chỉ được sử dụng trong file A.c
static void hien_thi() {};

int c;


//------------------
//B.c
```

```
// biến a này chỉ được sử dụng trong file B.c
static int a;

// hàm hienthi() này chỉ được sử dụng trong file B.c
static void hien_thi() {};

int d;
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

9. Từ khóa const

Từ khóa const được sử dụng để chỉ ra rằng một giá trị nào đó sẽ không thể thay đổi được trong suốt quá trình chạy của chương trình.

Bạn có thể đặt từ khóa "**const**" ở những nơi nào trong chương trình của mình?

Ở đây, trong bài viết này, mình sẽ liệt kê ra một số nơi bạn có thể đặt từ const.

9.1 Khi khai báo một biến

```
const int x = 8;
int const y = 10;

x = 15; // ERROR
y = 5;  // ERROR
```

Hai cách khai báo trên là tương đương. Giá trị của hai biến x và y sẽ không thể thay đổi được.

9.2 Khi khai báo một con trỏ

Ví dụ 1:

```
int x = 10, y = 20;
const int *px = &x;
```

Khi này, giá trị của vùng nhớ mà px trỏ đến là không thể thay đổi thông qua (*px).
Do đó, ta có các lệnh như sau:

```
*px = 15;   // ERROR
px = &y;    // OK
x = 15;     // OK
```

Ví dụ 2:

```
int x = 10, y = 20;
int* const px = &x;
```

Khi này giá trị của vùng nhớ mà px trỏ tới có thể thay đổi được thông qua việc thay đổi (*px). Tuy nhiên, ta không thể thay đổi vùng nhớ mà px đang trỏ tới. Ta có các lệnh như sau:

```
*px = 15;    // OK
px = &y;     // ERROR
```

Ví dụ 3:

```
int x = 10; y = 20;
const int* const px = &x;
```

Khi này, bạn không thể thay đổi vùng nhớ con trỏ px đang trỏ tới và cũng không thể thay đổi giá trị vùng nhớ đó thông qua (*px).

```
*px = 15;    // ERROR
px = &y;     // ERROR
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

10. Từ khóa extern

Để có thể truy cập giá trị một biến toàn cục ở một File khác, chúng ta sử dụng từ khóa extern.

Ví dụ:

```
//-----------------
//A.c

int a;

//-----------------
//B.c

extern int a;     // khai báo để sử dụng biến int a từ File A.c

void hienthi() {
    printf("A = %d",a);
};
```

Từ khóa extern được sử dụng cực kì nhiều trong lập trình nhúng, khi mà chúng ta cần chia sẻ quyền truy cập của các biến toàn cục giữa các file trong project. Như ở ví dụ trên

biến int a được khai báo file A.c vì vậy ở file B.c nếu muốn có thể truy cập vào biến int a, chúng ta cần extern biến int a.

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

11. Từ khóa volatile

Trong lập trình nhúng (Embedded System), ta rất thường hay gặp khai báo biến với từ khóa volatile. Việc khai báo biến **volatile** là rất cần thiết để tránh những lỗi sai khó phát hiện do tính năng optimization của compiler.

Cú pháp:

```
volatile int var;
int volatile var;
```

Một biến cần được khai báo dưới dạng biến volatile khi nào?

Khi mà giá trị của nó có thể thay đổi một cách không báo trước. Trong thực tế, có 3 loại biến mà giá trị có thể bị thay đổi như vậy:

- Memory-mapped peripheral registers (thanh ghi ngoại vi có ánh xạ đến ô nhớ)
- Biến toàn cục được truy xuất từ các tiến trình con xử lý ngắt (interrupt service routine)
- Biến toàn cục được truy xuất từ nhiều tác vụ trong một ứng dụng đa luồng.

Ví dụ:

Trong lập trình nhúng, chúng ta hay gặp đoạn code khi ta khai báo 1 biến đếm count, mỗi khi bấm nút xảy ra ngắt ngoài, chúng ta tăng biến đếm count. Tuy nhiên, khi chúng ta bật tính năng tối ưu code của compiler, nó sẽ hiểu rằng các biến như vậy dường như không thay đổi giá trị bởi phần mềm nên compiler có xu hướng loại bỏ biến count để có thể tối ưu kích cỡ file code chạy được sinh ra.

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

Buổi 4: Vòng lặp

1. Điều kiện If, else if, else

1.1 Câu lệnh If

Đây là 1 câu lệnh điều kiện để kiểm tra 1 điều kiện nào có được thỏa mãn không. Nếu điều kiện được thỏa mãn thì sẽ thực thi đoạn code bên trong nó.

Cấu trúc câu lệnh If

```
if (điều kiện){
    // Khối lệnh sẽ được thực hiện nếu <điều kiện> đúng.
}
```

Ví dụ:

```
#include <stdio.h>

int main(){
    int a;
    printf("Nhap a = "); scanf("%d", &a);

    if (a % 2 == 0) // a chia hết cho 2
    {
        printf("%d la so chan", a);
    }
    printf("\nXong!");
}
```

Kết quả:

```
Nhap a = 2
2 la so chan
Xong!
```

1.2 Câu lệnh If else

Cấu trúc câu lệnh if else

```
if (condition){
    // statement1
    // khối lệnh sẽ thực hiện nếu điều kiện đúng
}else{
    // statement2
    // khối lệnh sẽ thực hiện nếu điều kiện sai
}
```

Ví dụ:

```
#include <stdio.h>

int main(){
    int a;
    printf("Nhap a = "); scanf("%d", &a);
```

```
    if (a % 2 == 0) // a chia hết cho 2
    {
        printf("%d la so chan", a);
    }else{
        printf("%d la so le", a);
    }
}
```

Kết quả:

```
Nhap a = 1
1 la so le
```

1.3 Câu lệnh if ... else if ... else

Cấu trúc if ... else if ... else

```
if (test expression1)
{
   // statement(1)
}
else if(test expression2)
{
   // statement(2)
}
else if (test  expression3)
{
   // statement(3)
}
…
else
{
   // statement(n)
}
```

Ví dụ:

```
#include <stdio.h>

int main(){
    int a, b;
    printf("Nhap a = "); scanf("%d", &a);
    printf("Nhap b = "); scanf("%d", &b);


    // a, b
    if(a > b){
        //
        printf("%d lon hon %d", a, b);
    }else if(a == b){
        printf("%d bang %d", a, b);
    }else{
        printf("%d nho hon %d", a, b);
    }
}
```

Kết quả:

```
Nhap a = 3
Nhap b = 6
3 nho hon 6
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

2. Lệnh switch case

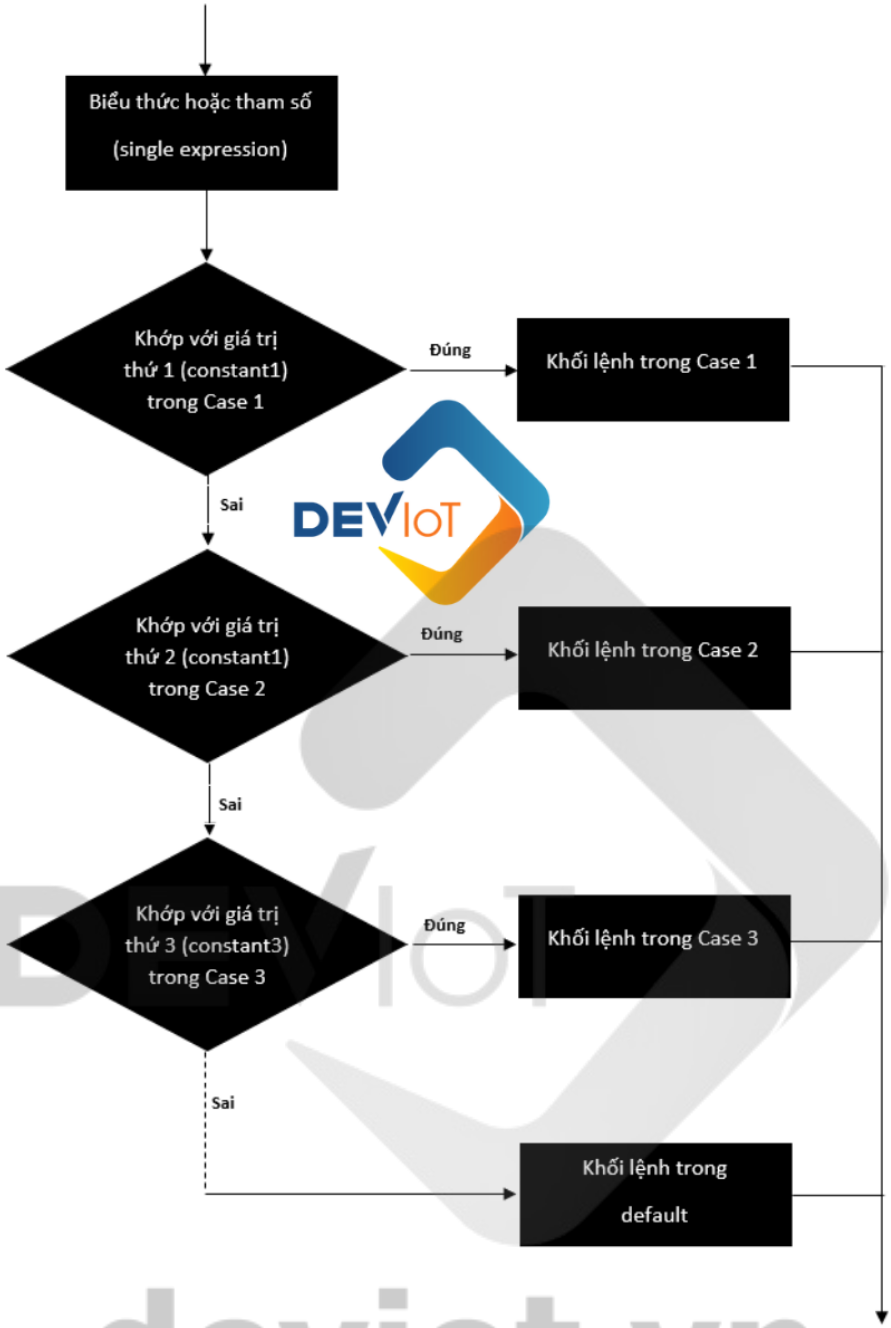
Lệnh switch case là một cấu trúc điều khiển & rẽ nhánh hoàn toàn có thể được thay thế bằng cấu trúc if else. Tuy nhiên, việc sử dụng switch case sẽ giúp code của chúng ta dễ viết và dễ đọc hơn. Một điều nữa là sử dụng switch case có vẻ như cho hiệu năng tốt hơn so với sử dụng if else trong trường hợp có nhiều điều kiện có thể xảy ra.

Cú pháp câu lệnh switch case:

```
switch (expression)
{
    case constant1:
      // statements
      break;
    case constant2:
      // statements
      break;
    .
    .
    .
    default:
      // default statements
}
```

- ***expression*** phải bắt buộc là giá trị hằng, có thể là biểu thức nhưng kết quả cần là hằng số. Trong đó, ***expression*** sẽ được so sánh với các giá trị của các ***case***.
- Nếu có 1 case nào đó khớp giá trị, các khối lệnh tương ứng sau case đó sẽ được thực hiện cho tới khi gặp lệnh ***break***. Do đó, nếu chúng ta không sử dụng ***break*** thì tất cả các case kể từ case khớp giá trị đều được thực hiện.
- Case ***default*** sẽ được thực hiện nếu không có case nào khớp giá trị với ***expression***.

Sơ đồ khối mô tả hoạt động:



Ví dụ 1:

```
#include <stdio.h>

int main()
{
    int a, b;
    char opera;
    printf("\nNhap phep toan: ");
```

```
    scanf("%c", &opera);

    printf("\nNhap vao 2 so a, b: ");
    scanf("%d%d", &a, &b);

    switch (opera)
    {
    case '+':
        printf("%d + %d = %d", a, b, a + b);
        break;
    case '-':
        printf("%d - %d = %d", a, b, a - b);
        break;
    case '*':
        printf("%d * %d = %d", a, b, a * b);
        break;
    case '/':
        if(b == 0){
            printf("Khong the chia cho 0!");
        }else{
            printf("%d / %d = %.2f", a, b, (float)a / b);
        }
        break;

    default:
    printf("Khong co phep toan %c!", opera);
        break;
    }
}
```

Kết quả:

```
Nhap phep toan: +

Nhap vao 2 so a, b: 10  20
10 + 20 = 30
```

Ví dụ 2:

```
#include <stdio.h>
int main()
{
     int i=2;
     switch (i)
     {
        case 1:
           printf("Case1 ");
        case 2:
           printf("Case2 ");
        case 3:
           printf("Case3 ");
        case 4:
           printf("Case4 ");
        default:
           printf("Default ");
     }
    return 0;
```

```
}
```

Kết quả:

```
Case2 Case3 Case4 Default
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

3. Vòng lặp for

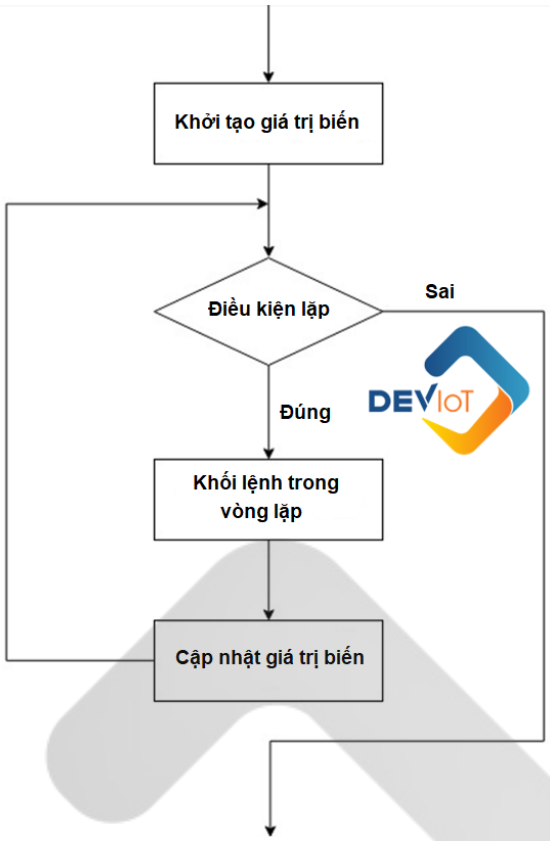
Mục đích của vòng lặp là để giải quyết các công việc có sự lặp đi lặp lại nhiều lần bằng những dòng code ngắn gọn nhất.

Được sử dụng để lặp đi lặp lại 1 khối code có số lần lặp xác định.

Cú pháp

```
for (khởi tạo giá trị biến lặp; điều kiện lặp; cập nhật biến lặp)
{
    // các lệnh cần lặp
}
```

Flowchart của vòng lặp



- **B1**: Khởi tạo giá trị biến lặp lần duy nhất
- **B2**: Kiểm tra điều kiện lặp, nếu sai chuyển sang B5
- **B3**: Thực hiện nội dung trong thân vòng lặp
- **B4**: Cập nhật giá trị lặp và quay lại B2
- **B5**: Kết thúc vòng lặp.

Ví dụ 1:

```
int main()
{
    int i;
    for (i = 1; i <= 10; i++)
    {
        printf("Deviot.vn\r\n");
    }
    return 0;
}
```

Kết quả:

```
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
Deviot.vn
```

Ví dụ 2:

```
#include <stdio.h>

int main()
{
    int i;
    for (i = 1; i <= 20; i++)
    {
        if( i % 2 == 0)
        {
            printf("So chan %d\r\n", i);
        }
    }
    return 0;
}
```

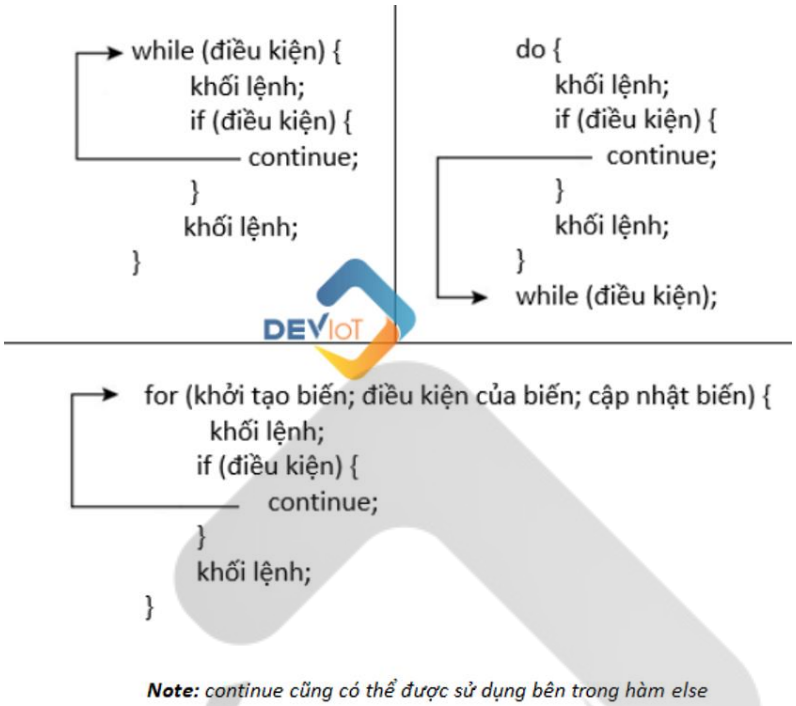
Kết quả:

```
So chan 2
So chan 4
So chan 6
So chan 8
So chan 10
So chan 12
So chan 14
So chan 16
So chan 18
So chan 20
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

4. Từ khóa continue

Nếu một vòng lặp đang chạy mà gặp lệnh continue, tất cả các lệnh trong thân vòng lặp nằm phía dưới lệnh continue sẽ bị bỏ qua ở lần lặp hiện tại. Vòng lặp sẽ chuyển sang kiểm tra điều kiện và tiếp tục lặp ở lần tiếp theo(nếu điều kiện lặp còn thỏa mãn).



Note: *continue cũng có thể được sử dụng bên trong hàm else*

Ví dụ 1:

```
#include <stdio.h>

/*
    In ra các chữ số lẻ từ 1 đến 10.
 */

int main(){
    for(int i = 0; i <= 10; i++){
        if(i % 2 == 0){
            continue;
        }
        printf("%d ", i);
    }
}
```

Kết quả:

```
1 3 5 7 9
```

Với mỗi giá trị của i chia hết cho 2 sẽ không chạy tiếp các lệnh bên dưới mà chuyển sang lần lặp kế tiếp luôn.

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

5. Vòng lặp while

Được sử dụng để lặp đi lặp lại 1 khối lệnh mà không biết trước số lần lặp.

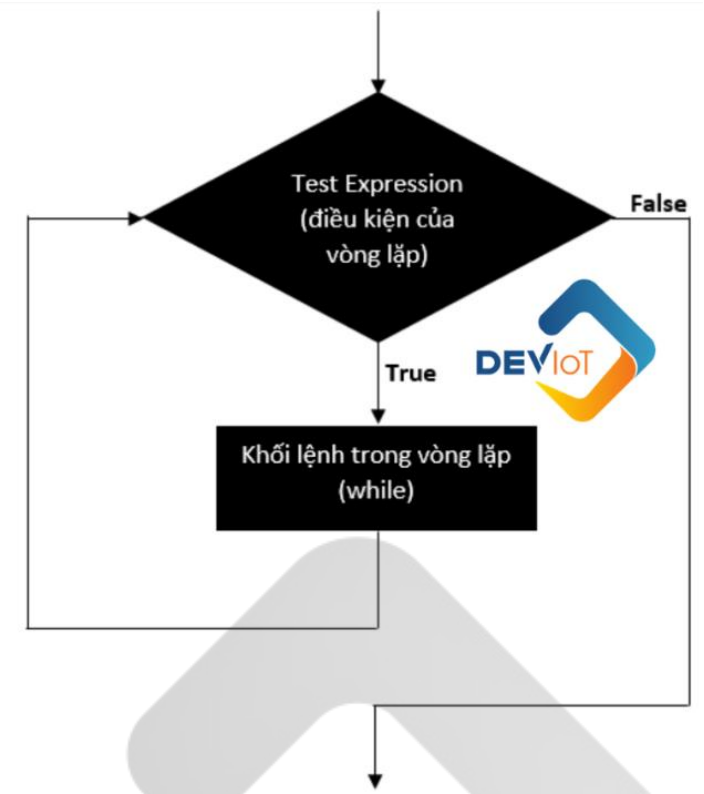
Cú pháp

```
while (testExpression)
{
    // code
}
```

1. Nếu điều kiện `testExpression` đúng => còn lặp

2. Nếu điều kiện `testExpression` sai => thoát vòng lặp

Flowchart của vòng lặp while



Ví dụ 1:

```
#include <stdio.h>

int main()
{
    int n = 10;

    while (n--)
    {
        printf("%d ", n);
    }

    printf("\n");

    return 0;
}
```

Kết quả

```
9 8 7 6 5 4 3 2 1 0
```

Ví dụ 2:

```
#include <stdio.h>

int main () {
    int a = 10;
```

```
    while( a < 15 ) {
        printf("Gia tri cua a: %d\n", a);
        a++;
    }

    return 0;
}
```

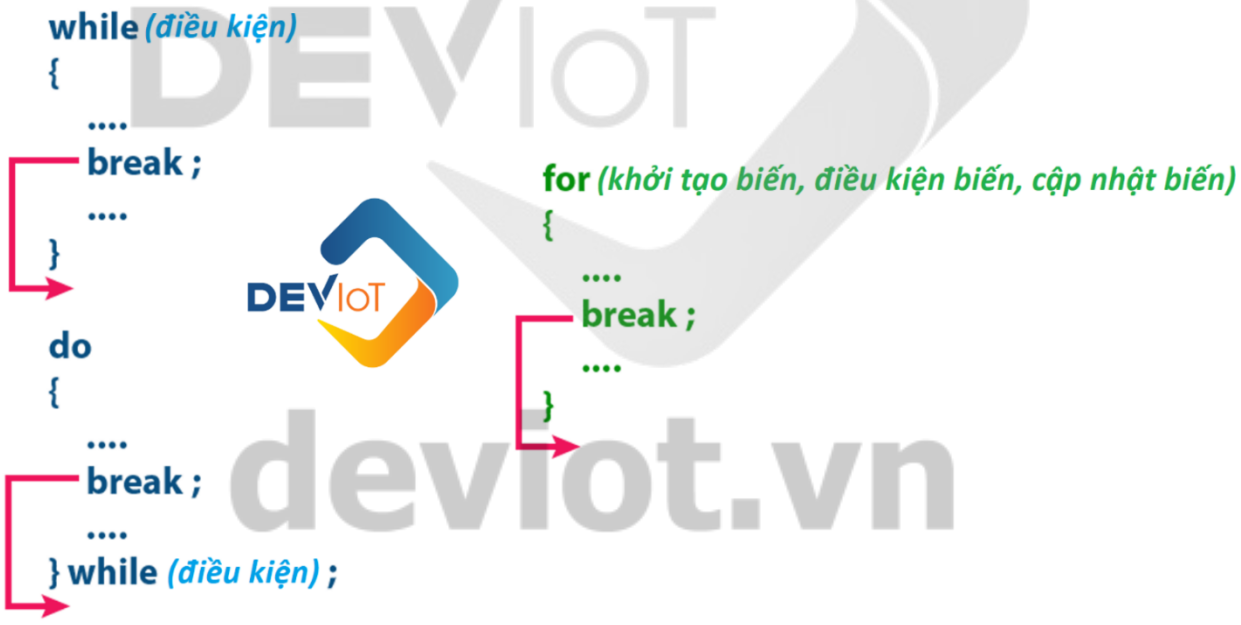
Kết quả:

```
Gia tri cua a: 10
Gia tri cua a: 11
Gia tri cua a: 12
Gia tri cua a: 13
Gia tri cua a: 14
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.

6. Từ khóa break

Một vòng lặp nếu đang thực hiện và gặp lệnh break sẽ ngay lập tức thoát ra khỏi vòng lặp. Vòng lặp ở đây có thể là vòng lặp for, vòng lặp while...



Ví dụ 1:

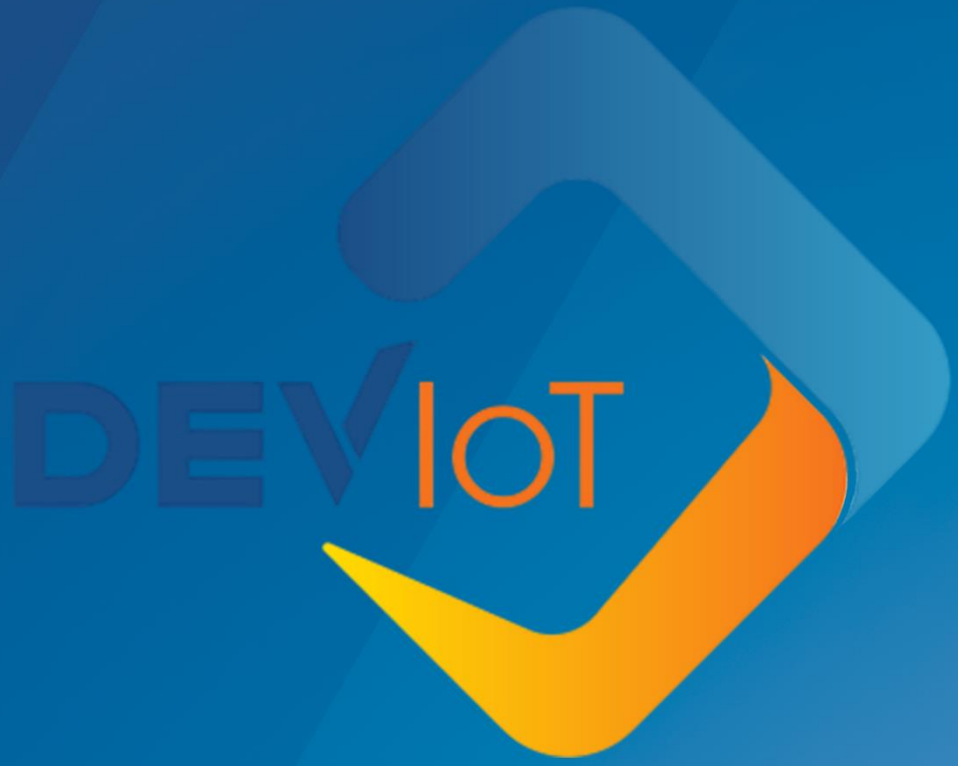
```
#include <stdio.h>
 /*
    In từ 10 đến 15
 */
int main () {
    int a = 10;
```

```
    while( a < 20 ) {
        printf("Gia tri cua a: %d\n", a);
        a++;

        if( a > 15) {
            /* ket thuc vong lap khi a lon hon 15 */
            break;
        }
    }

   return 0;
}
```

Kết quả

```
Gia tri cua a: 10
Gia tri cua a: 11
Gia tri cua a: 12
Gia tri cua a: 13
Gia tri cua a: 14
Gia tri cua a: 15
```

Các bạn truy cập nhóm https://www.facebook.com/groups/deviot.vn để tìm hiểu thêm các ví dụ phần này nhé.



- ĐIỆN TỬ
- CƠ KHÍ
- LẬP TRÌNH NHÚNG
- XỬ LÝ ẢNH
- IOT

www.deviot.vn

www.facebook.com/groups/deviot.vn