LẬP TRÌNH C CƠ BẢN

(from https://viettuts.vn/)

PART 1: C BASIC

1. Giới thiệu chương trình C cơ bản

```c
#include <stdio.h>

int main() {
    printf("Hello C Language");
    return 0;
}
```

Giải thích:

- `#include<stdio.h>` : bao gồm các hàm đầu vào tiêu chuẩn đầu vào. Hàm printf() được định nghĩa trong `stdio.h` .
- `int main()` : là hàm được gọi đầu tiên của chương trình
- `printf()` : được sử dụng để in dữ liệu trên console.
- `return 0` : câu lệnh return 0, trả về trạng thái thực thi cho hệ điều hành. Giá trị 0 được sử dụng để thực hiện thành công và 1 cho việc thực hiện không thành công.

2. Flow thực thi

1. Chương trình C (mã nguồn) được gửi đến Preprocessor đầu tiên. Preprocessor có trách nhiệm chuyển đổi các chỉ thị tiền xử lý thành các giá trị tương ứng. Bộ tiền xử lý tạo ra một mã nguồn mở rộng.
2. Mã nguồn được mở rộng được gửi tới trình Compiler biên dịch mã và chuyển đổi nó thành mã assembly.
3. Mã assembly được gửi đến Assembler để lắp ráp mã và chuyển đổi nó thành mã đối tượng. Bây giờ tệp hello.obj được tạo ra.
4. Mã đối tượng được gửi đến Linker liên kết nó tới thư viện như các tệp header. Sau đó, nó được chuyển đổi thành mã thực thi. Một tập tin hello.exe được tạo ra.
5. Mã thực thi được gửi đến Loader nạp nó vào bộ nhớ và sau đó nó được thực thi. Sau khi thực hiện, output của chương trình được gửi đến console.

3. Biến trong C

- Một biến trong C là tên của vị trí bộ nhớ.
- Có thể thay đổi giá trị và được sử dụng nhiều lần trong chương trình.
- Biến là một cách để thể hiện **vị trí bộ nhớ** thông qua một cái tên để nó có thể được xác định một cách dễ dàng.

Cú pháp khai báo biến

```
type variable_list;
```

Ví dụ:

```
int a = 10, b = 20; // Khai báo 2 biến kiểu số nguyên
float = 20,8;
char c = 'A';
```

Một số quy tắc:

- Một biến có thể có các chữ cái, chữ số và dấu gạch dưới.
- Tên biến chỉ có thể bắt đầu bằng bảng chữ cái và dấu gạch dưới. Nó không thể bắt đầu bằng chữ số.
- Không có khoảng trắng trong tên biến.
- Tên biến không là bất kỳ từ hoặc từ khóa dành riêng như `int` , `float` ,...
- Tên biến có phân biệt hoa thường,

Tên biến hợp lệ:

```
int a; int _ab; int a30
```

Tên biến không hợp lệ:

```
int 2; int a b; int long;
```

Các kiểu biến trong C

1. Biến local (cục bộ).

Một biến được khai báo bên trong hàm hoặc khối lệnh được gọi là biến cục bộ

```
void function1() {
    int x = 10; // bien local
}
```

2. Biến global (toàn cục).

Một biến được khai báo bên ngoài hàm hoặc khối lệnh được gọi là biến toàn cục. Bất kỳ hàm nào cũng có thể thay đổi giá trị của biến toàn cục. Nó có sẵn cho tất cả các chức năng.
Trong ví dụ dưới đây, biến a là biến global.

```
int a = 20; // bien global

void function1() {
    int x = 10; // bien local
}
```

3. Biến static (biến tĩnh).

Nó giữ lại giá trị của nó sau nhiều lần gọi hàm. (Chỉ khởi tạo 1 lần)

```
#include <stdio.h>

int main() {
    function1(); // 11, 11
    function1(); // 11, 12
    function1(); // 11, 13
    return 0;
}

int function1() {
    int x = 10; // bien local
    static int y = 10; // bien static
    x = x + 1;
    y = y + 1;
    printf("\n %d, %d", x, y);
}
```

4. Biến automatic. (tự động)

Tất cả các biến trong C được khai báo trong khối lệnh, là các biến tự động theo mặc định. Bởi chúng ta có thể khai báo một cách rõ ràng biến tự động bằng cách sử dụng từ khóa auto.

```
void main(){
    int x = 10; // bien local (cung la bien automatic)
    auto int y = 20; // bien automatic
}
```

5. Biến external.

Chúng ta có thể chia sẻ một biến trong nhiều tập tin mã nguồn C bằng cách sử dụng biến external.
Để khai báo biến bên ngoài, bạn cần sử dụng từ khóa extern

File: myfile.h

```
extern int x = 10; // bien external variable (cung la bien global)
```

File: test.c

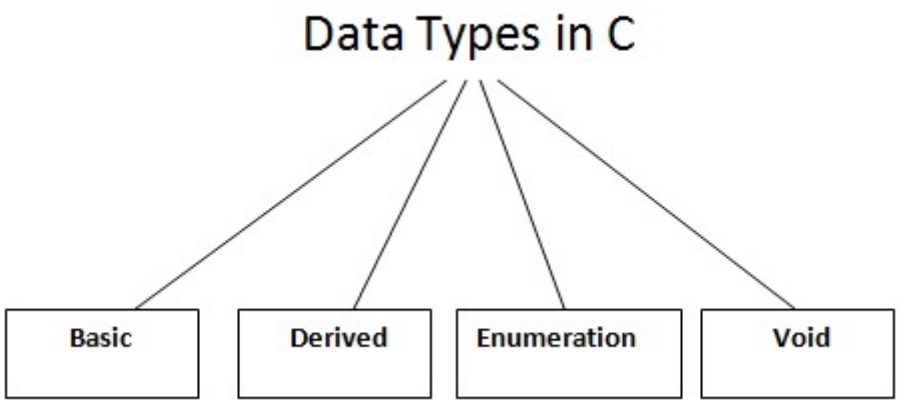
```
#include "myfile.h"
#include <stdio.h>

int main() {
    printValue();
}

int printValue() {
    printf("x: %d", x);
}
```

4. Các kiểu dữ liệu trong C

> **Kiểu dữ liệu trong C** xác định loại dữ liệu mà một biến có thể lưu trữ như số nguyên, số thực, ký tự vv.



Có 4 loại dữ liệu trong ngôn ngữ C:

#	Kiểu và mô tả
1	**Kiểu dữ liệu cơ bản** Chúng là các loại số học và được phân loại thành: các kiểu số nguyên và các kiểu số thực.
2	**Kiểu Enumeration** Chúng lại là các loại số học và chúng được sử dụng để xác định các biến chỉ có thể chỉ định một số giá trị số nguyên rời rạc trong suốt chương trình.

#	Kiểu và mô tả
3	**Kiểu dữ liệu Derived** Chúng bao gồm các kiểu con trỏ (Pointer), mảng (Array), cấu trúc (Structure), Union và Function.
4	**Kiểu Void** Kiểu Void (vô nghĩa) chỉ định rằng không có giá trị nào có sẵn.

Bảng tóm tắt các loại dữ liệu cơ bản

Kiểu	Kích thước bộ nhớ	Vùng giá trị
char	1 byte	–128 to 127
signed char	1 byte	–128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	–32,768 to 32,767
signed short	2 byte	–32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	–32,768 to 32,767
signed int	2 byte	–32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	–32,768 to 32,767
signed short int	2 byte	–32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	1.2E-38 to 3.4E+38
double	8 byte	2.3E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

5. Ép kiểu dữ liệu trong C

> **Ép kiểu trong C** là việc gán giá trị của một biến có kiểu dữ liệu này tới biến khác có kiểu dữ liệu khác.

Cú pháp

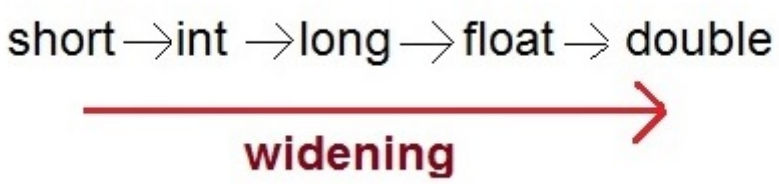
```
(type) value;
```

Ví dụ:

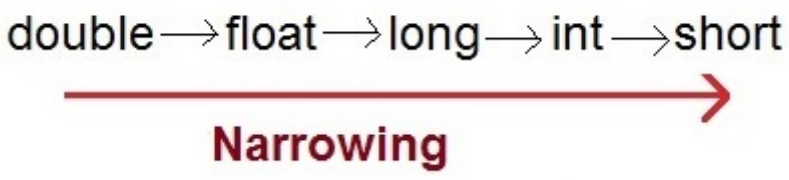
```
float c = 35.8f;
int b = (int)c + 1;
```

Phân loại ép kiểu trong C
Trong c, có hai loại ép kiểu:

1. **Nới rộng (widening):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin.



2. **Thu hẹp (narrowwing):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin



6. Từ khoá trong C

> **Từ khóa** là một từ dành riêng. Bạn **không** thể sử dụng nó như một tên biến, tên hằng số... Chỉ có 32 từ dành riêng (từ khoá) trong ngôn ngữ lập trình C.

Danh sách 32 từ khoá trong C được đưa ra dưới đây:

```
auto, break, case, char, const, continue, default, do
double, else, enum, extern, float, for, goto, if
int, long, register, return, short, signed, sizeof, static
struct, switch, typedef, union, unsigned, void, volatile, while
```

7. Toán tử trong C

> Toán tử trong C là một ký hiệu được sử dụng để thực hiện một phép tính/chức năng nào đó.

Ngôn ngữ lập trình C cung cấp các dạng toán tử sau:

- Toán tử số học.
- Toán tử quan hệ.
- Toán tử logic.
- Toán tử so sánh bit.
- Toán tử gán.
- Toán tử hỗn hợp.
- Thứ tự ưu tiên.

7.1. Toán tử số học trong C

Toán tử	Miêu tả	Ví dụ
+	Thêm hai toán hạng	A + B sẽ cho kết quả là 30
-	Trừ giá trị toán hạng hai từ toán hạng đầu	A - B sẽ cho kết quả là -10
*	Nhân hai toán hạng	A * B sẽ cho kết quả là 200
/	Chia lấy phần nguyên hai toán hạng	B / A sẽ cho kết quả là 2
%	Chia lấy phần dư	B % A sẽ cho kết quả là 0
++	Lượng gia giá trị toán hạng thêm 1 đơn vị	A++ sẽ cho kết quả là 11
--	Lượng giảm giá trị toán hạng một đơn vị	A-- sẽ cho kết quả là 9

7.2. Toán tử quan hệ trong C

> Được sử dụng kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị **Boolean** (logic “true” hoặc “false”). Các toán tử quan hệ được sử dụng trong các **cấu trúc điều khiển**.

Giải sử A = 10; B = 20;

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	(A == B) là false.
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	(A != B) là true.
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	(A > B) là false.
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	(A < B) là true.
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	(A >= B) là false.
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	(A <= B) là true.

7.3. Toán tử logic trong C

Giả sử biến A = true (hoặc 1) và biến B = false (hoặc 0):

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	(A && B) là false.
II	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	(A II B) là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	!(A && B) là true.

7.4. Toán tử bit trong C

Giả sử A = 60 (= 0011 1100) và B = 13 (= 0000 1101)

Toán tử	Miêu tả	Ví dụ
&	Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng.	(A & B) sẽ cho kết quả là 12, tức là 0000 1100
I	Toán tử OR (hoặc) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng.	(A I B) sẽ cho kết quả là 61, tức là 0011 1101
^	Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai.	(A ^ B) sẽ cho kết quả là 49, tức là 0011 0001
~	Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại).	(~A) sẽ cho kết quả là -61, tức là 1100 0011.
<<	Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải.	A << 2 sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit)
>>	Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải.	A >> 2 sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit)

7.5. Toán tử gán trong C

> **Toán tử gán** dùng để gán một giá trị vào một biến và có thể gán nhiều giá trị cho nhiều biến cùng một lúc.

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	C = A + B sẽ gán giá trị của A + B vào trong C
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	C += A tương đương với C = C + A
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	C -= A tương đương với C = C - A
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	C *= A tương đương với C = C * A

Toán tử	Miêu tả	Ví dụ
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	C /= A tương đương với C = C / A
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	C %= A tương đương với C = C % A
<<=	Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải.	C <<= 2 tương đương với C = C << 2
>>=	Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải.	C >>= 2 tương đương với C = C >> 2
&=	Phép AND bit	C &= 2 tương đương với C = C & 2
^=	Phép OR loại trừ bit	C ^= 2 tương đương với C = C ^ 2
I=	Phép OR bit.	C I= 2 tương đương với C = C I 2

7.6. Toán tử hỗn hợp trong C

Có một số toán tử hỗn hợp quan trọng là sizeof và ? : được hỗ trợ bởi ngôn ngữ C.

Toán tử	Miêu tả	Ví dụ
sizeof()	Trả lại kích cỡ của một biến	sizeof(a), với a là integer, thì sẽ trả lại kết quả là 4.
&	Trả lại địa chỉ của một biến.	&a; sẽ cho địa chỉ thực sự của biến a.
*	Trỏ tới một biến.	*a; sẽ trỏ tới biến a.
? :	Biểu thức điều kiện	Nếu điều kiện là true ? thì giá trị X : Nếu không thì giá trị Y

7.7. Thứ tự ưu tiên

> **Thứ tự ưu tiên quyết** định **trật tự thực hiện** các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong C.

Loại	Toán tử	Thứ tự ưu tiên
Postfix	() -> . ++ - -	Trái sang phải
Unary	+ - ! ~ ++ - - (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit	|	Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic	||	Trái sang phải
Điều kiện	?:	Phải sang trái
Gán	= += -= *= /= %=>>= <<= &= ^= |=	Phải sang trái
Dấu phảy	,	Trái sang phải

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn ():

Phần được giới hạn trong ngoặc đơn được thực hiện trước.
Nếu dùng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài.
Trong phạm vi một cặp ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.
Ví dụ, x = 10 + 5 * 2; ở đây, x được gán giá trị 20, chứ không phải 30 bởi vì toán tử * có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân 5 * 2 và sau đó thêm với 10.

Để thay đổi thứ tự ưu tiên ta dùng dấu (), ví dụ, x = (10 + 5) * 2; ở đây x = 30.

8. Comment trong C

> **Comment** trong C được sử dụng để cung cấp thông tin (giải thích) các dòng code.

1. Comment một dòng

Comment một dòng trong C được đại diện bởi dấu gạch chéo kép //.

```
#include<stdio.h>

int main() {
    // in thong tin
    printf("Hello C");
    return 0;
}
```

2. Comment nhiều dòng

Comment nhiều dòng trong C được đại diện bởi dấu gạch chéo /*...*/. Nó có thể chiếm nhiều dòng nhưng nó không thể được lồng nhau.

```
#include<stdio.h>

int main() {
    /* in thong tin
      comment nhieu dong */
    printf("Hello C");
    return 0;
}
```

9. Hằng số trong C

> **Hằng số** là một giá trị hoặc biến không thể thay đổi trong chương trình

Sử dụng từ khóa const để khai báo hằng trong C

```
const float PI = 3.14;
```

10. printf() và scanf() trong C

> Các hàm **printf()** và **scanf()** trong C được sử dụng cho đầu ra và đầu vào. Cả hai chức năng là các chức năng của thư viện có sẵn, được định nghĩa trong tập tin tiêu đề **stdio.h**.

10.1. Hàm printf()

Hàm printf() được sử dụng cho đầu ra. Nó in ra lệnh đã cho vào bảng điều khiển (console).

```
printf("format string", argument_list);
```

Format string có thể là %d (số nguyên), %c (ký tự) ,%s (chuỗi), %f (float), vv.

10.2. Hàm scanf()

Hàm scanf() được sử dụng cho đầu vào. Nó đọc dữ liệu đầu vào từ bàn điều khiển.

```
scanf("format string", argument_list);
```

Ví dụ nhập số nguyên n từ console và hiển thị bình phương của nó ra console.

```
#include <stdio.h>

int main() {
    int number;
    printf("Nhap so nguyen: ");
    scanf("%d",&number);
    printf("Binh phuong cua %d la: %d ", number, number*number);
    return 0;
}
```

Kết quả:

```
Nhap so nguyen: 9
Binh phuong cua 9 la: 81
```

11. Ký tự đặc biệt trong C

Ký tự đặc biệt trong C là một dãy các ký tự không đại diện cho chính nó khi được sử dụng bên trong chuỗi ký tự hoặc ký tự.

Nó bao gồm hai hoặc nhiều ký tự bắt đầu bằng dấu gạch chéo ngược . Ví dụ: \n đại diện cho dòng mới.

Kí tự	Ý nghĩa
\a	Báo thức hoặc tiếng bíp
\b	Dấu Backspace
\f	Form Feed
\n	Xuống dòng mới (LF)

Kí tự	Ý nghĩa
\r	Xuống dòng (CR)
\t	Dấu Tab ngang
\v	Dấu Tab dọc
\	Dấu ngạch chéo ngược
'	Dấu nháy đơn
"	Dấu nháy kép
?	Dấu chấm hỏi
\nnn	Số octal
\xhh	Số hexadecimal
\0	Null

```c
#include <stdio.h>

int main() {
    printf("Ban Dang\nHoc Lap Trinh\n\'C\'\n\"Chuc Ban Hoc Vui Ve!\"");
    return 0;
}
```

```
Ban Dang
Hoc Lap Trinh
'C'
"Chuc Ban Hoc Vui Ve!"
```

12. Tệp Header trong C

Cú pháp include:
Cả tệp người dùng và tiêu đề hệ thống đều được bao gồm bằng cách sử dụng chỉ thị tiền xử lý #include.

```c
#include <file>
```

13. Toán tử sizeof trong C

> **Toán tử sizeof trong C** là một toán tử chứ không phải là một hàm. Toán tử sizeof nhận một tham số là bất kỳ kiểu dữ liệu nào và trả về kích cỡ của kiểu dữ liệu đó.

Toán tử sizeof có cú pháp:

```
sizeof (kieu_du_lieu)
```

Ví dụ:

```
#include<stdio.h>

int main() {

   printf("\nKich co cua kieu du lieu char la: %d", sizeof(char));
   printf("\nKich co cua kieu du lieu int la: %d", sizeof(int));
   printf("\nKich co cua kieu du lieu float la: %d", sizeof(float));
   printf("\nKich co cua kieu du lieu double la: %d", sizeof(double));
   return (0);
}
```

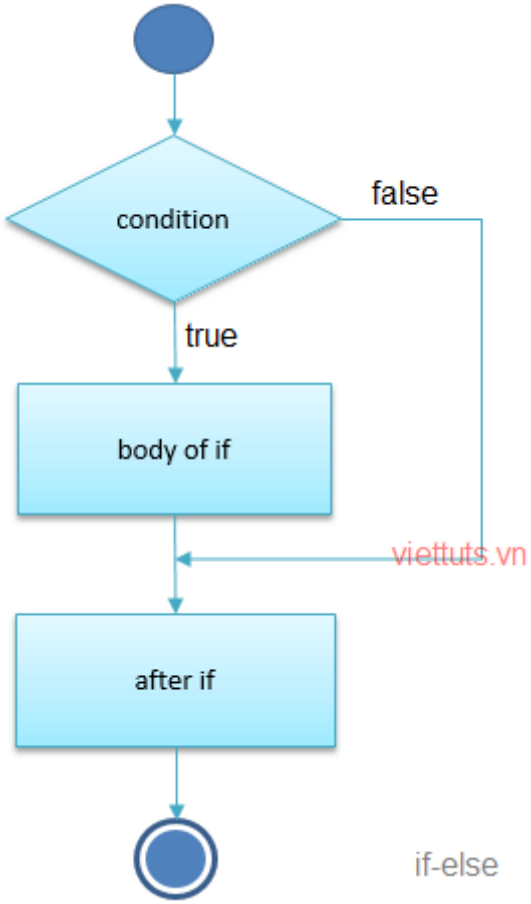
PART 2. CÂU LỆNH ĐIỀU KHIỂN

1. Mệnh đề if-else trong C

> **Mệnh đề if-else trong C** được sử dụng để kiểm tra một biểu thức điều kiện nào đó có đúng hay không, nếu đúng thì thực thi những câu lệnh bên trong khối lệnh if và ngược lại nếu sai thì nó sẽ bỏ qua những câu lệnh đó.

Có ba dạng của câu lệnh if trong C.

- Mệnh đề if.
- Mệnh đề if-else.
- Mệnh đề if-elseif-else.



Mệnh đề if.

```
if (condition) {
  // khối lệnh này được thực thi nếu condition = true
}
```

Ví dụ:

```
#include <stdio.h>

int main () {
   int num = 10;
        if (num % 2 == 0) {
            printf("num la so chan.");
        }
   return 0;
}
```

Mệnh đề if-else.

```
if (condition) {
  // khối lệnh này được thực thi nếu condition = true
} else {
  // khối lệnh này được thực thi nếu condition = false
}
```

Ví dụ:

```
#include <stdio.h>

int main() {
    int num = 11;
    if (num % 2 == 0) {
        printf("num la so chan.");
    } else {
        printf("num la so le.");
    }
    return 0;
}
```

Mệnh đề if-elseif-else.

```
if (condition1) {
  // khối lệnh này được thực thi nếu condition1 = true
} else if (condition2) {
  // khối lệnh này được thực thi nếu condition1 = false và condition2 = true
...
} else {
  // khối lệnh này được thực thi nếu nếu tất cả những điều kiện trên = false
}
```

Ví dụ:

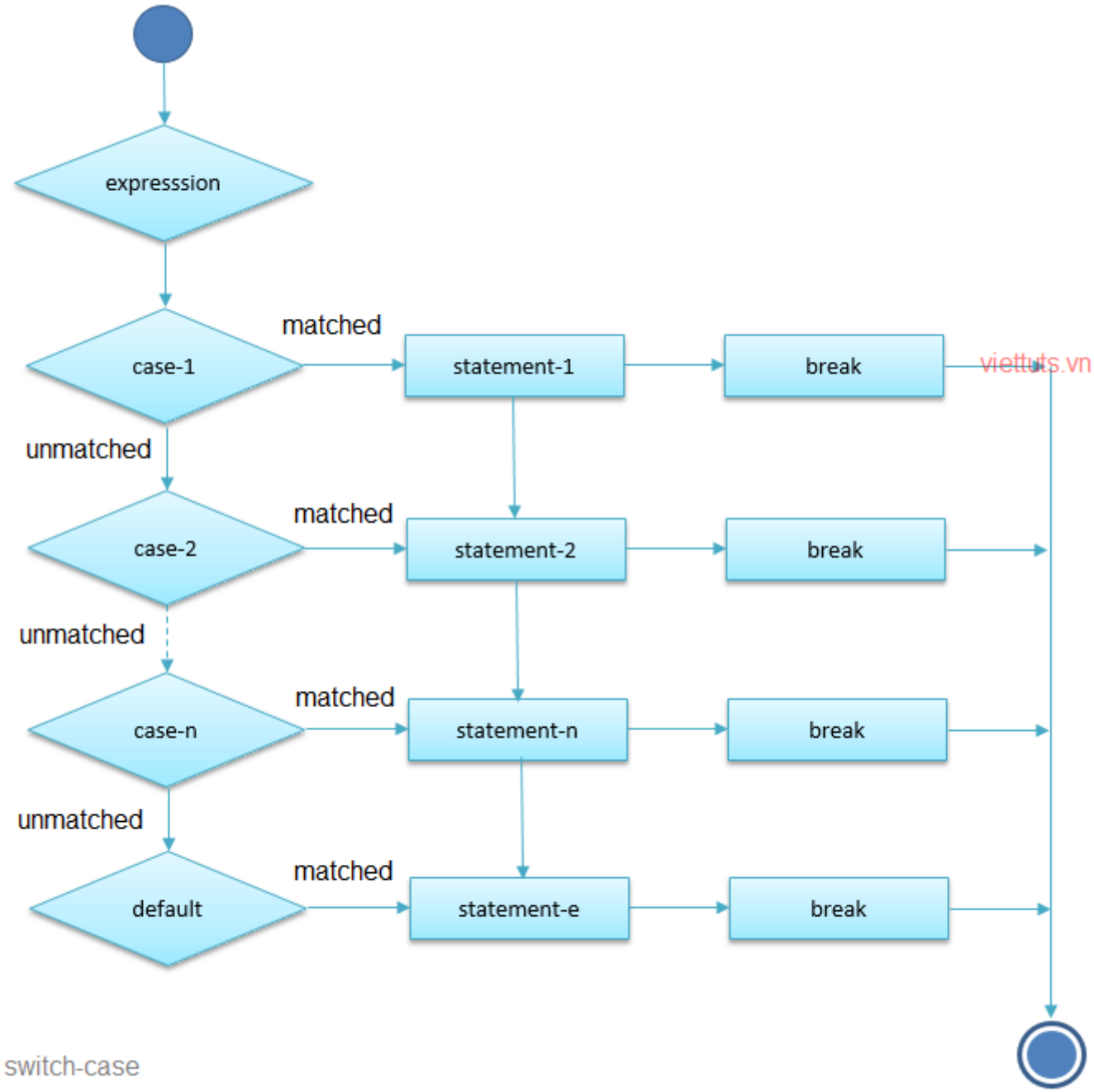
```c
#include <stdio.h>

int main() {
    int num;
    printf("Nhap 1 so de kiem tra thang diem:");
    scanf("%d", &num);
    if (num < 0 || num >= 100) {
        printf("Ban nen nhap so tu 1 den 100");
    } else if (num > 0 && num < 50) {
        printf("Fail");
    } else if (num >= 50 && num < 60) {
        printf("D Grade");
    } else if (num >= 60 && num < 70) {
        printf("C Grade");
    } else if (num >= 70 && num < 80) {
        printf("B Grade");
    } else if (num >= 80 && num < 90) {
        printf("A Grade");
    } else if (num >= 90 && num <= 100) {
        printf("A+ Grade");
    }
}
```

2. Mệnh đề switch trong C

> **Mệnh đề switch trong C** cho phép một biến được kiểm tra xem giá trị của nó có bằng một giá trị trong một danh sách hay không. Mỗi giá trị được gọi là một trường hợp (case).

```c
switch (bieu_thuc) {
case gia_tri_1:
    // Khối lệnh 1
    break;  //tùy chọn
case gia_tri_2:
    // Khối lệnh 2
    break;  //tùy chọn
......
case gia_tri_n:
    // Khối lệnh n
    break;  //tùy chọn
default:
    // Khối lệnh này được thực thi
    // nếu tất cả các điều kiện trên không thỏa mãn
}
```



switch-case

Các quy tắc sau đây áp dụng cho mệnh đề switch trong C:

- Biểu thức được sử dụng trong mệnh đề switch phải là một số nguyên, kiểu char, hoặc kiểu enum, hoặc kiểu lớp có một hàm chuyển đổi duy nhất thành một số nguyên hoặc kiểu enum.
- Bạn có thể khai báo bất kỳ số lượng lệnh case bên trong một switch. Đằng sau từ khóa case là một giá trị được sử dụng để so sánh và một dấu hai chấm (😃 .
- Biểu thức hằng số cho một case phải giống kiểu dữ liệu như biến trong switch và nó phải là một hẳng số hoặc một chuỗi.
- Khi gặp case có giá trị phù hợp các câu lệnh sau đó được thực hiện cho tới khi gặp lệnh break.
- Khi gặp lệnh break, mệnh đề switch kết thúc và luồng điều khiển chuyển sang dòng tiếp theo sau câu lệnh switch.
- Lệnh break là tùy chọn.

- Một câu lệnh switch có thể có một trường hợp mặc định (default) tùy chọn, nó được khai báo ở cuối switch. Case mặc định có thể được sử dụng để thực hiện tác vụ khi không có case nào đúng. Không cần khai báo break trong case mặc định.

Ví dụ:

```
#include <stdio.h>

int main() {
    char xeploai = 'B';

    switch (xeploai) {
    case 'A':
        printf("Gioi!\n");
        break;
    case 'B':
    case 'C':
        printf("Kha\n");
        break;
    case 'D':
        printf("Trung Binh\n");
        break;
    case 'F':
        printf("Kem\n");
        break;
    default:
        printf("Dup\n");
    }

    printf("Ban xep loai  %c\n", xeploai);

    return 0;
}
```

Kết quả:

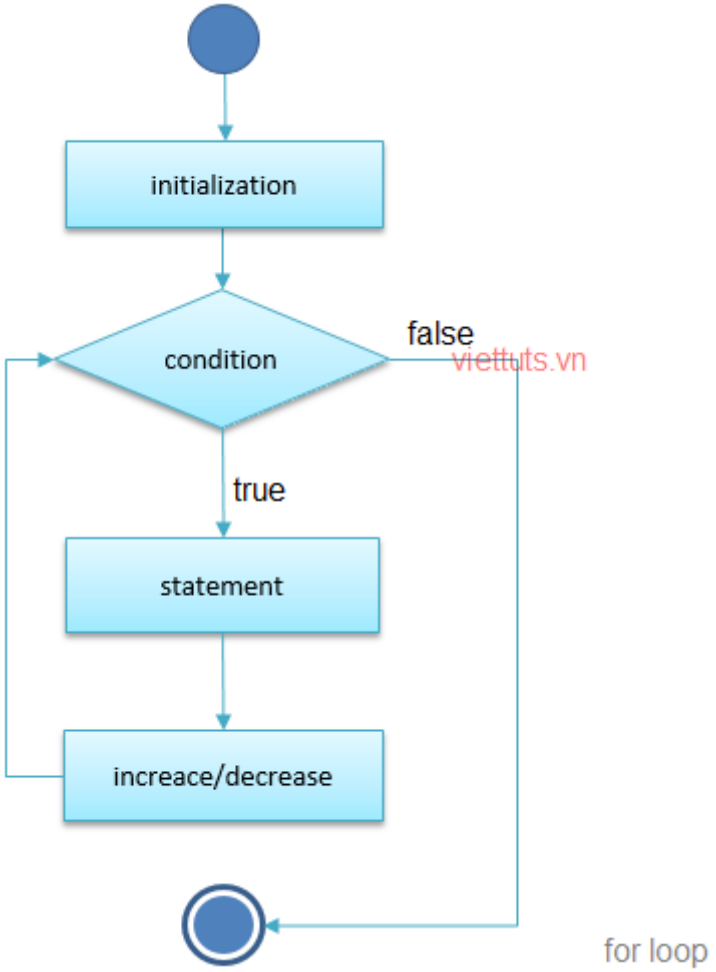
```
Kha
Ban xep loai  B
```

3. Mệnh đề for trong C

> **Vòng lặp For trong C** là một cấu trúc điều khiển lặp được sử dụng để thực thi số lần lặp cụ thể.

Cú pháp:

```
for (khoi_tao_bien ; check_dieu_kien ; tang/giam_bien) {
    // Khối lệnh được thực thi
}
```



Luồng điều khiển trong vòng lặp for:

- Bước **khoi_tao_bien** được thực thi đầu tiên, và chỉ một lần. Bước này cho phép bạn khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp nào. Bạn cũng có thể không cần phải đặt một khai báo ở đây, miễn là khai báo dấu chấm phẩy.
- Tiếp theo, **check_dieu_kien** được đánh giá. Nếu nó là true, phần thân của vòng lặp được thực thi. Nếu nó là false, phần thân của vòng lặp không thực thi và luồng điều khiển nhảy đến câu lệnh kế tiếp ngay sau vòng lặp for.
- Sau khi phần thân của vòng lặp for được thực thi, luồng điều khiển nhảy ngược lại lên câu lệnh **tang/giam_bien**. Câu lệnh này cho phép bạn cập nhật bất kỳ biến điều khiển vòng lặp nào. Câu lệnh này có thể để trống, miễn là khai báo dấu chấm phẩy.
- **check_dieu_kien** được đánh giá lại. Nếu nó là true, vòng lặp thực hiện và quá trình lặp lại chính nó (phần thân của vòng lặp, sau đó là bước **tang/giam_bien**, và sau đó lại **check_dieu_kien**). Sau khi điều kiện là false, vòng lặp for kết thúc.

```c
#include <stdio.h>

int main () {
    int a;

    for (a = 10; a < 20; a++) {
        printf("Gia tri cua a: %d\n", a);
    }
    return 0;
}
```

Kết quả:

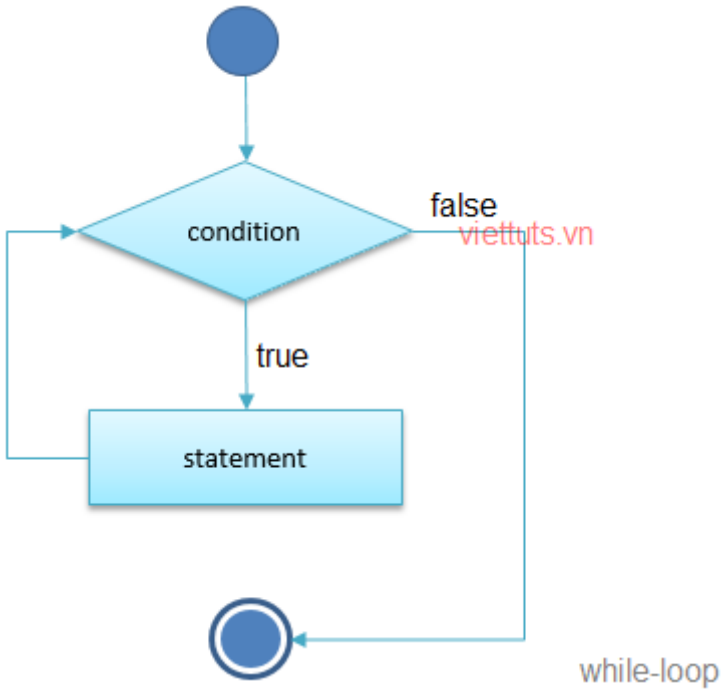
```
Gia tri cua a: 10
Gia tri cua a: 11
Gia tri cua a: 12
Gia tri cua a: 13
Gia tri cua a: 14
Gia tri cua a: 15
Gia tri cua a: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```

4. Mệnh đề while trong C

> **Vòng lặp while trong C** được sử dụng để lặp một phần của chương trình một vài lần. Nếu số lần lặp không được xác định trước thì vòng lặp lặp while được khuyến khích sử dụng trong trường hợp này.
> Cú pháp:

```c
while(condition) {
    // Khối lệnh được lặp lại cho đến khi condition = False
}
```

condition có thể là bất kỳ biểu thức nào. Khối lệnh trong vòng lặp while được thực thi trong khi condition là true. Khi condition là false thì điều khiển chương trình sẽ chuyển đến dòng ngay sau vòng lặp.



while-loop

Ví dụ:

```
#include <stdio.h>

int main () {
    int a = 10;

    while( a < 20 ) {
        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
Gia tri cua a: 15
Gia tri cua a: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```

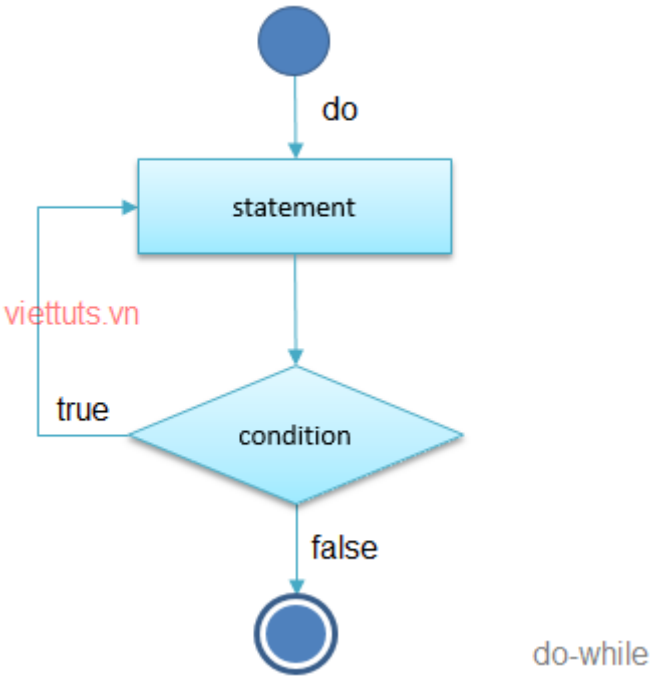
5. Mệnh đề do-while trong C

Không giống như vòng lặp for và while, trong đó kiểm tra điều kiện lặp ở đầu vòng lặp, vòng lặp do-while trong C kiểm tra điều kiện lặp của nó ở dưới cùng của vòng lặp.

Vòng lặp do-while tương tự như một vòng lặp while, ngoại trừ nó được đảm bảo để **thực hiện ít nhất một lần**.

Cú pháp:

```
do {
    // Khối lệnh được thực thi
} while(condition);
```



do-while

Ví dụ:

```
#include <stdio.h>

int main () {
    int a = 10;

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

    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
Gia tri cua a: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```

6.Lệnh break trong C

Câu lệnh break trong C có hai cách sử dụng như sau:

- Khi gặp câu lệnh break trong một vòng lặp, vòng lặp bị kết thúc ngay lập tức và câu lệnh kế tiếp sau vòng lặp được thực thi.
- Lệnh break có thể được sử dụng để kết thúc một case trong câu lệnh switch.
  Nếu bạn sử dụng vòng lặp lồng nhau, câu lệnh break sẽ dừng việc thực hiện vòng lặp trong cùng và bắt đầu thực hiện cấu lệnh kế tiếp sau vòng lặp trong cùng.

Cú pháp:

```
break;
```

Ví dụ:

```c
#include <stdio.h>

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
```

7. Lệnh continue trong C

Câu lệnh continue trong C hoạt động giống như câu lệnh break. Thay vì buộc kết thúc vòng lặp, nó buộc trở về kiểm tra điều kiện để thực hiện vòng lặp tiếp theo và bỏ qua các lệnh bên trong vòn lặp hiện tại sau lệnh continue.

Đối với vòng lặp for, câu lệnh continue làm cho điều khiển chương trình tăng hoặc giảm biến đếm của vòng lặp. Đối với vòng lặp while và do-while, câu lệnh continue làm cho điều khiển chương trình quay về đầu vòng lặp và kiểm tra điều kiện của vòng lặp.

Cú pháp:

```c
continue;
```

Ví dụ:

```c
#include <stdio.h>

int main () {
    int a = 10;

    do {
        if( a == 15) {
         // quay ve do khi a = 15 (bo qua lenh print)
            a = a + 1;
            continue;
        }

        printf("Gia tri cua a: %d\n", a);
        a++;

    } while( a < 20 );

   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: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```

8. Lệnh goto trong C

Câu lệnh goto trong C cung cấp một bước nhảy vô điều kiện từ 'goto' đến một câu lệnh có nhãn trong cùng một hàm.

Chú ý: Việc sử dụng câu lệnh goto không được khuyến khích sử dụng trong bất kỳ ngôn ngữ lập trình nào vì nó rất khó để theo dõi luồng điều khiển của chương trình, làm cho chương trình khó hiểu và khó bảo trì. Bất kỳ chương trình nào sử dụng goto đều có thể được viết lại theo cách bình thường.

Cú pháp:

```
goto label;
..
.
label: statement;
```

Ở đây nhãn (label) có thể là bất kỳ văn bản thuần tuý trừ từ khóa C và nó có thể được đặt ở bất kỳ vị trí nào trong chương trình C, bên trên hoặc bên dưới câu lệnh goto.

Ví dụ:

```
#include <stdio.h>

int main () {
    int a = 10;

    TEST:do {
        if( a == 15) {
            // quay ve do khi a = 15 (bo qua lenh print)
            a = a + 1;
            goto TEST;
        }
        printf("Gia tri cua a: %d\n", a);
        a++;
    } while( a < 20 );

   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: 16
Gia tri cua a: 17
Gia tri cua a: 18
Gia tri cua a: 19
```