

Cấu trúc	Giá trị
<a href="#">setup()</a>	Hằng số
<a href="#">loop()</a>	<a href="#">HIGH</a>   <a href="#">LOW</a>
Cấu trúc điều khiển	<a href="#">INPUT</a>   <a href="#">INPUT_PULLUP</a>   <a href="#">OUTPUT</a>
<a href="#">if</a>	<a href="#">LED_BUILTIN</a>
<a href="#">if...else</a>	<a href="#">true</a>   <a href="#">false</a>
<a href="#">switch / case</a>	Hằng số nguyên (integer constant)
<a href="#">for</a>	Hằng số thực (floating point constant)
<a href="#">while</a>	Kiểu dữ liệu
<a href="#">break</a>	<a href="#">void</a>
<a href="#">continue</a>	<a href="#">boolean</a>
<a href="#">return</a>	<a href="#">char</a>
<a href="#">goto</a>	<a href="#">unsigned char</a>
Cú pháp mở rộng	<a href="#">byte</a>
<a href="#">;</a> (dấu chấm phẩy)	<a href="#">int</a>
<a href="#">{ }</a> (dấu ngoặc nhọn)	<a href="#">unsigned int</a>
<a href="#">//</a> (single line comment)	<a href="#">word</a>
<a href="#">/* */</a> (multi-line comment)	<a href="#">long</a>
<a href="#">#define</a>	<a href="#">unsigned long</a>
<a href="#">#include</a>	<a href="#">short</a>
Toán tử số học	<a href="#">float</a>
<a href="#">=</a> (phép gán)	<a href="#">double</a>
<a href="#">+</a> (phép cộng)	<a href="#">array</a>
<a href="#">-</a> (phép trừ)	<a href="#">string</a> (chuỗi kí tự biểu diễn bằng mảng)
<a href="#">*</a> (phép nhân)	<a href="#">String</a> (object)
<a href="#">/</a> (phép chia)	Chuyển đổi kiểu dữ liệu
<a href="#">%</a> (phép chia lấy dư)	<a href="#">char()</a>
Toán tử so sánh	<a href="#">byte()</a>

== (so sánh bằng)

!= (khác bằng)

> (lớn hơn)

< (bé hơn)

>= (lớn hơn hoặc bằng)

<= (bé hơn hoặc bằng)

### **Toán tử logic**

&& (và)

|| (hoặc)

! (phủ định)

^ (loại trừ)

### **Phép toán hợp nhất**

++ (cộng thêm 1 đơn vị)

-- (trừ đi 1 đơn vị)

+= (phép rút gọn của phép cộng)

-= (phép rút gọn của phép trừ)

\*= (phép rút gọn của phép nhân)

/= (phép rút gọn của phép chia)

int()

word()

long()

float()

### **Phạm vi của biến và phân loại**

Phạm vi hiệu lực của biến

static - biến tĩnh

const - biến hằng

volatile

### **Hàm hỗ trợ**

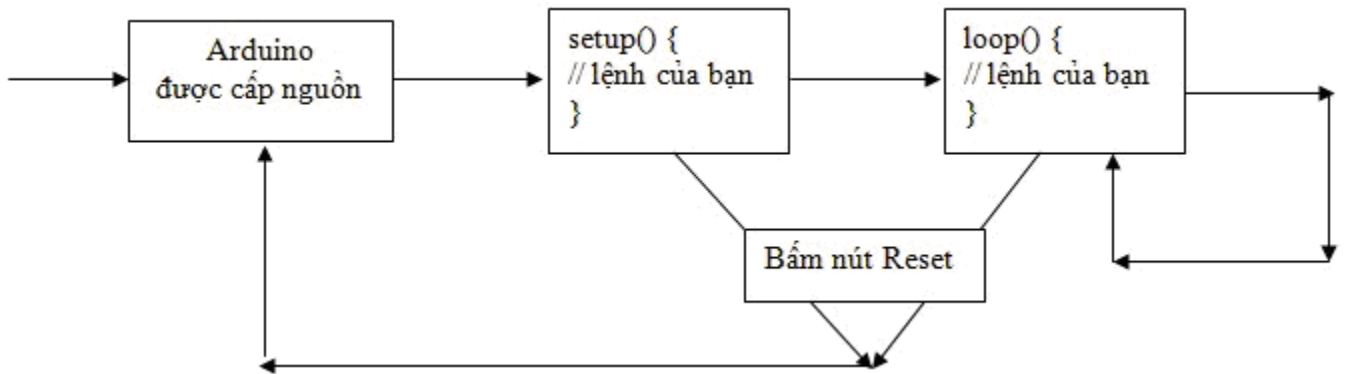
sizeof()

1. Setup và loop: Những lệnh trong setup() sẽ được chạy khi chương trình của bạn khởi động. Bạn có thể sử dụng nó để khai báo giá trị của biến, khai báo thư viện, thiết lập các thông số,...

Sau khi `setup()` chạy xong, những lệnh trong `loop()` được chạy. Chúng sẽ lặp đi lặp lại liên tục cho tới khi nào bạn ngắt nguồn của board Arduino mới thôi.

Bất cứ khi nào bạn nhấn nút Reset, chương trình của bạn sẽ trở về lại trạng thái như khi Arduino mới được cấp nguồn.

Quá trình này có thể được miêu tả như sơ đồ dưới đây



### Ví dụ

```
int led = 13;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

### Giải thích

Khi bạn cấp nguồn cho Arduino, lệnh “*pinMode(led, OUTPUT);*” sẽ được chạy 1 lần để khai báo.

Sau khi chạy xong lệnh ở `setup()`, lệnh ở `loop()` sẽ được chạy và được lặp đi lặp lại liên tục, tạo thành một chuỗi:

```
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
```

.....

*Trước khi đọc về lệnh if, các bạn cần xem qua các mục sau:*

### **Các toán tử logic (Boolean operators)**

Toán tử	Ý nghĩa	Ví dụ
and (&&)	Và	(a && b) trả về TRUE nếu a và b đều mang giá trị TRUE. Nếu một trong a hoặc b là FALSE thì (a && b) trả về FALSE
or (    )	Hoặc	(a    b) trả về TRUE nếu có ít nhất 1 trong 2 giá trị a và b là TRUE, trả về FALSE nếu a và b đều FALSE
not ( ! )	Phủ định	nếu a mang giá trị TRUE thì (!a) là FALSE và ngược lại
xor (^)	Loại trừ	(a ^ b) trả về TRUE nếu a và b mang hai giá trị TRUE/FALSE khác nhau, các trường hợp còn lại trả về FALSE

## Các toán tử so sánh (Comparison operators)

Các toán tử so sánh thường dùng để so sánh 2 số có cùng một kiểu dữ liệu.

	Ví dụ
	(a == b) trả về TRUE nếu a bằng b và ngược lại
bằng	(a != b) trả về TRUE nếu a khác b và ngược lại
	(a > b) trả về TRUE nếu a lớn hơn b và FALSE nếu a bé hơn hoặc bằng b
	(a < b) trả về TRUE nếu a bé hơn b và FALSE nếu ngược lại
ng	(a <= b) tương đương với ((a < b) or (a = b))
ằng	(a >= b) tương đương với ((a > b) or (a = b))

## Câu lệnh if

### Cú pháp:

```
if ([biểu thức 1] [toán tử so sánh] [biểu thức 2]) { //biểu thức điều kiện
    [câu lệnh 1]
} else {
    [câu lệnh 2]
}
```

Nếu biểu thức điều kiện trả về giá trị TRUE, [câu lệnh 1] sẽ được thực hiện, ngược lại, [câu lệnh 2] sẽ được thực hiện.

### Ví dụ:

```
int a = 0;
if (a == 0) {
    a = 10;
```

```

} else {
    a = 1;
}
// a = 10

```

Lệnh if không bắt buộc phải có nhóm lệnh nằm sau từ khóa else

```

int a = 0;
if (a == 0) {
    a = 10;
} // a = 10

```

Bạn có thể kết hợp nhiều biểu thức điều kiện khi sử dụng lệnh if. Chú ý rằng mỗi biểu thức con phải được bao bằng một ngoặc tròn và phải luôn có một cặp ngoặc tròn bao toàn bộ biểu thức con.

Cách viết đúng	Cách viết sai
<pre> int a = 0; if ((a == 0) &amp;&amp; (a &lt; 1)) {     a = 10; } </pre>	<pre> int a = 0; if (a == 0) &amp;&amp; (a &lt; 1) {     a = 10; } </pre>
	<pre> int a = 0; if (a == 0 &amp;&amp; a &lt; 1) {     a = 10; } </pre>

### Chú ý:

- `a = 10;` là một câu lệnh gán, giá trị logic của nó luôn là TRUE (vì lệnh gán này luôn thực hiện được)
- `(a == 10)` là một biểu thức logic có giá trị TRUE hay FALSE tùy thuộc vào giá trị của biến x.

Nếu bạn viết ...

```
int a = 0;
if (a = 1) {
    a = 10;
}
```

... thì giá trị của a sẽ bằng 10, vì (a = 1) là một câu lệnh gán, trong trường hợp này nó được xem như một biểu thức logic và luôn trả về giá trị TRUE.

Giống như [if](#), switch / case cũng là một dạng lệnh nếu thì, nhưng nó được thiết kế chuyên biệt để bạn xử lý giá trị trên một biến chuyên biệt.

Ví dụ, bạn có một biến là action sẽ nhận trị từ những module khác qua serial. Nhưng action sẽ nằm trong một các giá trị nào đó thì lúc này bạn hãy sử dụng switch / case.

### **Ví dụ**

```
switch (action) {
    case "callMyMom":
        //gọi điện cho mẹ của tôi
        break;
    case "callMyDad":
        //gọi điện cho ba của tôi
        break;
    default:
        // mặc định là không làm gì cả
        // bạn có thể có default: hoặc không
}
```

### **Cú pháp**

```
switch (var) {
    case label:
        //đoạn lệnh
        break;
    case label:
```

```

    // Đoạn lệnh
    break;
/*
case ... more and more
*/
default:
    // statements
}

```

## **Tham số**

var: biến mà bạn muốn so sánh

label: sẽ đem giá trị của biến SO SÁNH BẰNG với nhãn này

Hàm for có chức năng làm một vòng lặp. Vậy vòng lặp là gì? Hãy hiểu một cách đơn giản, nó làm đi làm lại một công việc có một tính chất chung nào đó. Chẳng hạn, bạn bật tắt một con LED thì dùng digitalWrite xuất HIGH delay rồi lại LOW rồi lại delay. Nhưng nếu bạn muốn làm nhiều hơn 1 con LED thì mọi đoạn code của bạn sẽ dài ra (không đẹp và khi chỉnh sửa thì chẳng lẽ ngồi sửa lại từng dòng?

Với 1 con led, bạn lập trình như thế này

```

digitalWrite(led1,HIGH);
delay(1000);
digitalWrite(led1,LOW);
delay(1000);

```

Với 10 con led, nếu bạn không dùng for, đoạn code nó sẽ dài như thế này

```

digitalWrite(led1,HIGH);
delay(1000);
digitalWrite(led1,LOW);
delay(1000);
digitalWrite(led2,HIGH);
delay(1000);

```



```
digitalWrite(led2,LOW);  
delay(1000);
```

...

```
digitalWrite(led10,HIGH);  
delay(1000);  
digitalWrite(led10,LOW);  
delay(1000);
```

Nếu như vậy thì bạn có còn muốn lập trình và suy nghĩ về một led ma trận có còn nữa không ? Chắc chắn là không rồi, vì vậy hàm for ra đời để giúp bạn nhìn cuộc sống một cách tươi đẹp hơn ! 😊

Bây giờ hãy lấy một ví dụ đơn giản như sau:

Tôi muốn xuất 10 chữ số (từ 1 - 10) ra Serial. Hãy giúp tôi lập trình trên Arduino để làm được việc ấy!

Nếu bạn chưa đọc bài này và cũng chưa biết kiến thức về for, bạn sẽ lập trình như sau:

```
void setup() {  
    Serial.begin(9600);  
    Serial.println(1);  
    Serial.println(2);  
    Serial.println(3);  
    Serial.println(4);  
    Serial.println(5);  
    Serial.println(6);  
    Serial.println(7);  
    Serial.println(8);  
    Serial.println(9);  
    Serial.println(10);  
}  
void loop() {
```

```
// không làm gì cả;  
}
```

Đoạn code khá dài và lặp đi lặp lại câu lệnh `Serial.println`

Nhưng sau khi biết về hàm `for` bạn chỉ cần một đoạn code cực kì ngắn như sau:

```
void setup(){  
    Serial.begin(9600);  
    int i;  
    for (i = 1;i<=10;i=i+1) {  
        Serial.println(i);  
    }  
}  
void loop(){  
}
```

### Cấu trúc

Theo quan điểm của tôi, nếu bạn chưa biết về vòng lặp hoặc hàm `for`, để hiểu được hàm `for`, bạn cần nắm được 4 phần:

1. Hàm `for` là một vòng lặp có giới hạn - nghĩa là chắc chắn nó sẽ kết thúc (không sớm thì muộn).
2. Nó sẽ bắt đầu từ một vị trí xác định và đi đến một vị trí kết thúc.
3. Cứ mỗi bước xong, nó lại thực hiện một đoạn lệnh
4. Sau đó, nó lại bước đi tiếp, nó có thể bước 1 bước hoặc nhiều bước, nhưng không được thay đổi theo thời gian.

Theo ví dụ trên, ta có đoạn code sử dụng hàm `for` như sau:

```
for (i = 1;i<=10;i = i + 1) {  
    Serial.println(i);  
}
```

Hàm `for` trong ví dụ này sẽ :

1. Chắc chắn nó sẽ xử lý đoạn code `Serial.println(i);` 10 lần

2. Chạy từ vị trí xuất phát là 1, đến vị trí kết thúc là 10 //  $i = 1 ; i \leq 10$
3. Cứ mỗi lần bước xong (tính luôn cả vị trí xuất phát tại thời điểm  $i = 1$ ) thì nó lại chạy lệnh `Serial.println(i);`. Trong đó biến  $i$ , dân khoa học gọi là biến con chạy, còn tôi gọi là vị trí của thằng  $i$  😊
4. Mỗi lần chạy xong, thằng  $i$  lại bước thêm 1 bước nữa. Chừng nào mà thằng  $i$  còn  $\leq 10$  thì nó còn quay về bước 3

Bạn có thấy nó dễ hiểu không? Bây giờ tôi sẽ nói nó theo một cách khoa học qua cú pháp của hàm `for` (tôi sẽ chia làm 2 loại để các bạn dễ dàng ứng dụng vào code của mình).

- For tiến (xuất phát từ một vị trí nhỏ chạy đến vị trí lớn hơn) <vị trí kết thúc> bé hơn <vị trí kết thúc>  
`for (<kiểu dữ liệu nguyên> <tên thằng chạy> = <vị trí xuất phát>; <tên thằng chạy> <= <vị trí kết thúc>; <tên thằng chạy> += <mỗi lần bước mấy bước>) {`  
    <đoạn câu lệnh>;  
}
- For lùi (xuất phát từ một vị trí lớn chạy về vị trí nhỏ hơn) <vị trí xuất phát> lớn hơn <vị trí kết thúc>  
`for (<kiểu dữ liệu nguyên> <tên thằng chạy> = <vị trí xuất phát>; <tên thằng chạy> <= <vị trí kết thúc>; <tên thằng chạy> -= <mỗi lần lùi mấy bước>) {`  
    <đoạn câu lệnh>;  
}

Và khi đã hiểu được một cách sâu sắc thì đây là cú pháp chính của hàm `For`:

```
for (<biến chạy> = <start>; <điều kiện>; <bước>) {  
//lệnh
```

} Vòng lặp `while` là một dạng vòng lặp theo điều kiện, mình không thể biết trước số lần lặp của nó, nhưng mình quản lý lúc nào thì nó ngừng lặp!

**Cách hiểu dành cho Newbie**

Giống như [for](#), cũng có vài khái niệm mà bạn cần nắm, tôi đã "vui" hóa cho nó nên hãy thoải mái khi đọc 😊!

1. While là một vòng lặp không biết trước số lần lặp, nó dựa vào điều kiện, điều kiện còn đúng thì còn chạy. Điều này cũng giống như, nếu chúng ta còn đang "xanh" trong LOL thì không ngại gì mà đi lẻ. Tất nhiên, nếu không "xanh" thì không đi lẻ nữa 😈
2. Chạy một đoạn lệnh (trong đó có những hàm ảnh hưởng đến điều kiện). Nếu cứ chạy mãi như void loop() thì biết khi nào vòng lặp While mới dừng!

### Cú pháp

```
while (<điều kiện>) {  
    //các đoạn lệnh;  
}
```

### Ví dụ

```
int day = 1;  
int nam = 2014; // Năm 2014  
while (day < 365) { //Chừng nào day < 365 thì còn chạy (<=364). Khi  
    day == 365 thì hết 1 năm...  
    day += 1; //  
    delay(60*60*24); // Một ngày có 24 giờ, mỗi giờ có 60 phút, mỗi phút  
    có 60 giây  
}  
nam += 1; //... bây giờ đã là một năm mới ! Chúc mừng năm mới :)  
Đó chỉ là một ví dụ vui để bạn hiểu cách hoạt động của vòng lặp while.  
Chúc vui vẻ!
```

break là một lệnh có chức năng dừng ngay lập tức một vòng lặp (do, for, while) chứa nó trong đó. Khi dừng vòng lặp, tất cả những lệnh phía sau break và ở trong vòng lặp chịu ảnh hưởng của nó sẽ bị bỏ qua.

### Ví dụ

```
int a = 0;
```

```

while (true) {
    if (a == 5) break;
    a = a + 1;
}
//a = 5
while (true) {
    while (true) {
        a++;
        if (a > 5) break;
    }
    a++;
    if (a > 100) break;
}
//a = 101

```

continue là một lệnh có chức năng bỏ qua một chu kì lặp trong một vòng lặp (for, do, while) chứa nó trong đó. Khi gọi lệnh continue, những lệnh sau nó và ở trong cùng vòng lặp với nó sẽ bị bỏ qua để thực hiện những chu kì lặp kế tiếp.

### **Ví dụ**

```

int a = 0;
int i = 0;
while (i < 10) {
    i = i + 1;
    continue;
    a = 1;
}
//a vẫn bằng 0

```

return có nhiệm vụ trả về một giá trị (cùng kiểu dữ liệu với hàm) mà nó được gọi!

### **Cú pháp**

return;

return value; // cả 2 đều đúng

### **Thông số**

value: bất kỳ giá trị hoặc một đối tượng.

### **Ví dụ**

//Hàm kiểm tra giá trị của cảm biến có hơn một ngưỡng nào đó hay không

```
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    }
    else{
        return 0;
    }
}
```

### **Goto**

Nó có nhiệm vụ tạm dừng chương trình rồi chuyển đến một nhãn đã được định trước, sau đó lại chạy tiếp chương trình!

### **Cú pháp**

label: //Khai báo một nhãn có tên là label

goto label; //Chạy đến nhãn label rồi sau đó thực hiện tiếp những đoạn chương trình sau nhãn đó

### **Thủ thuật**

Không nên dùng lệnh goto trong chương trình Program hay bất cứ chương trình nào sử dụng ngôn ngữ C. Nhưng nếu sử dụng một cách khôn ngoan bạn sẽ tối ưu hóa được nhiều điều trong một chương trình!

Vậy nó hữu ích khi nào, đó là lúc bạn đang dùng nhiều vòng lặp quá và muốn thoát khỏi nó một cách nhanh chóng!

### **Ví dụ**

```

for(byte r = 0; r < 255; r++){
    for(byte g = 255; g > -1; g--){
        for(byte b = 0; b < 255; b++){
            if (analogRead(0) > 250){ goto bailout;}
            //thêm nhiều câu lệnh nữa
        }
    }
}

```

bailout:

*#define* là một đối tượng của ngôn ngữ C/C++ cho phép bạn đặt tên cho một hằng số nguyên hay hằng số thực. Trước khi biên dịch, trình biên dịch sẽ thay thế những tên hằng bạn đang sử dụng bằng chính giá trị của chúng. Quá trình thay thế này được gọi là quá trình tiền biên dịch (pre-compile).

## Cú pháp

*#define* [tên hằng] [giá trị của hằng]

## Ví dụ

```
#define pi 3.14
```

Nếu bạn viết code thế này ...

```
#define pi 3.14
```

```
float a = pi * 2.0; // pi = 6.28
```

thì sau khi pre-compile trước khi biên dịch, chương trình của bạn sẽ như thế này:

```
#define pi 3.14
```

```
float a = 3.14 * 2.0; // a = 6.28
```

## Chú ý

Nếu một biến có tên trùng với tên hằng số được khai báo bằng *#define* thì khi pre-compile, cái biến ấy sẽ bị thay thế bằng giá trị của hằng số kia. Hệ quả tất yếu là khi biên dịch, chương trình của bạn sẽ

bị lỗi cú pháp (bạn sẽ không dễ dàng nhận ra điều này). Đôi khi nó cũng dẫn đến lỗi logic - một lỗi rất khó sửa !

Nếu bạn viết thế này...

```
#define pi 3.14
```

```
float pi = 3.141592654;
```

thì sau khi pre-compile trước khi biên dịch, bạn sẽ được thế này:

```
#define pi 3.14
```

```
float 3.14 = 3.141592654; // lỗi cú pháp
```

Vì vậy, bạn nên hạn chế tối đa việc sử dụng *#define* để khai báo hằng số khi không cần thiết. Bạn có thể sử dụng cách khai báo sau để thay thế:

```
const float pi = 3.14;
```

*#include* cho phép chương trình của bạn tải một thư viện đã được viết sẵn. Tức là bạn có thể truy xuất được những tài nguyên trong thư viện này từ chương trình của mình. Nếu bạn có một đoạn code và cần sử dụng nó trong nhiều chương trình, bạn có thể dùng *#include* để nạp đoạn code ấy vào chương trình của mình, thay vì phải chép đi chép lại đoạn code ấy.

## Cú pháp

```
#include <[đường dẫn đến file chứa thư viện]>
```

## Ví dụ

Giả sử bạn có thư mục cài đặt Arduino IDE tên là ArduinoIDE, thư viện của bạn có tên là EEPROM (được lưu ở \ArduinoIDE\libraries\EEPROM\)

Một đoạn code lưu ở file code.h nằm trong thư mục function của thư viện EEPROM thì được khai báo như sau:

```
#include <function/code.h> //đường dẫn đầy đủ:  
\ArduinoIDE\libraries\EEPROM\function\code.h
```

Trong lập trình trên Arduino, **HIGH** là một hằng số có giá trị nguyên là 1. Trong điện tử, **HIGH** là một mức điện áp lớn hơn 0V. *Giá trị của HIGH được định nghĩa khác nhau trong các mạch điện khác nhau, nhưng thường được quy ước ở các mức như 1.8V, 2.7V, 3.3V 5V, 12V, ...*



## **HIGH là một hằng số có giá trị nguyên là 1**

Xét đoạn code ví dụ sau:

```
int led = 13;

void setup() {
    pinMode(led, OUTPUT);
    digitalWrite(led, HIGH);
}
```

```
void loop() {
}
```

Đoạn code này có chức năng bật sáng đèn led nối với chân số 13 trên mạch Arduino (Arduino Nano, Arduino Uno R3, Arduino Mega 2560, ...). Bạn có thể tải đoạn chương trình này lên mạch Arduino của mình để kiểm chứng. Sau đó, hãy thử tải đoạn chương trình này lên:

```
int led = 13;

void setup() {
    pinMode(led, OUTPUT);
    digitalWrite(led, 1);
}
```

Trong lập trình trên Arduino, **LOW** là một hằng số có giá trị nguyên là 0. Trong điện tử, **LOW** là mức điện áp 0V hoặc gần bằng 0V, giá trị này được định nghĩa khác nhau trong các mạch điện khác nhau, nhưng thường là 0V hoặc hơn một chút xíu.

## **LOW là một hằng số có giá trị nguyên là 0**

Xét đoạn code ví dụ sau:

```
int led = 13;

void setup() {
    pinMode(led, OUTPUT);
}

void loop() {
```

```
digitalWrite(led, HIGH);  
delay(1000);  
digitalWrite(led, LOW);  
delay(1000);  
}
```

Đoạn code này có chức năng bật sáng đèn led nối với chân số 13 trên mạch Arduino (Arduino Nano, Arduino Uno R3, Arduino Mega 2560, ...). Bạn có thể tải đoạn chương trình này lên mạch Arduino của mình để kiểm chứng. Sau đó, hãy thử tải đoạn chương trình này lên:

```
int led = 13;  
void setup() {  
  pinMode(led, OUTPUT);  
}
```

```
void loop() {  
  digitalWrite(led, HIGH);  
  delay(1000);  
  digitalWrite(led, 0);  
  delay(1000);  
}
```

Sẽ xuất hiện 2 vấn đề:

- Trong đoạn code thứ 2, "**LOW**" đã được sửa thành "**0**".
- Đèn led trên mạch Arduino vẫn sáng bình thường với 2 chương trình khác nhau.

Điều này khẳng định "**LOW** là một **hằng số** có giá trị nguyên là 0" đã nêu ở trên.

### **LOW là một điện áp lớn hơn 0V**

Điện áp (điện thế) tại một điểm là trị số hiệu điện thế giữa điểm đó và cực âm của nguồn điện (0V). Giả sử ta có một viên pin vuông 9V thì ta

có thể nói điện áp ở cực dương của cực pin là 9V, ở cực âm là 0V, hoặc hiệu điện thế giữa 2 cực của cực pin là 9V.

Điện áp ở mức **LOW** không có giá trị cụ thể như 3.3V, 5V, 9V, ... mà trong mỗi loại mạch điện, nó có một trị số khác nhau nhưng **thường là 0V hoặc gần bằng 0V**. Trong các mạch Arduino, **LOW** được quy ước là mức 0V mặc dù 0.5V vẫn có thể được xem là **LOW**. Ví dụ như trong mạch Arduino Uno R3, theo nhà sản xuất, điện áp được xem là ở mức **LOW** nằm trong khoảng từ 0V đến 1.5V ở các chân I

```
void loop() {  
}
```

Sẽ xuất hiện 2 vấn đề:

- Trong đoạn code thứ 2, "**HIGH**" đã được sửa thành "**1**".
- Đèn led trên mạch Arduino vẫn sáng bình thường với 2 chương trình khác nhau.

Điều này khẳng định "**HIGH** là một **hằng số** có giá trị nguyên là 1" đã nêu ở trên.

### **HIGH là một điện áp lớn hơn 0V**

Điện áp (điện thế) tại một điểm là trị số hiệu điện thế giữa điểm đó và cực âm của nguồn điện (0V). Giả sử ta có một viên pin vuông 9V thì ta có thể nói điện áp ở cực dương của cực pin là 9V, hoặc hiệu điện thế giữa 2 cực của cực pin là 9V.

Điện áp ở mức **HIGH** không có giá trị cụ thể như 3.3V, 5V, 9V, ... mà trong mỗi loại mạch điện, nó có trị số khác nhau và đã được quy ước trước. Trong các mạch Arduino, **HIGH** được quy ước là mức 5V mặc dù 4V vẫn có thể được xem là **HIGH**. Ví dụ như trong mạch Arduino Uno R3, theo nhà sản xuất, điện áp được xem là ở mức HIGH nằm trong khoảng từ 3V đến 5V.

Dù HIGH không có một trị số nào rõ ràng nhưng nhất quyết rằng giá trị của nó luôn lớn hơn 0V.

Thiết đặt Digital Pins như là INPUT, INPUT\_PULLUP, và OUTPUT

[ksp](#) gửi vào Thứ tư, 21 Tháng 5, 2014 - 20:27

- [\*\*9400 LƯỢT XEM\*\*](#)

Chân kỹ thuật số có thể được sử dụng như là INPUT, INPUT\_PULLUP, hoặc OUTPUT. Để thay đổi cách sử dụng một pin, chúng ta sử dụng hàm pinMode().

### **Cấu hình một pin là INPUT**

Các pin của Arduino ( Atmega ) được cấu hình là một INPUT với pinMode ( ) có nghĩa là làm cho pin ấy có trở kháng cao (không cho dòng điện đi ra). Pin được cấu hình là INPUT làm việc tiêu thụ năng lượng điện của mạch rất nhỏ, nó tương đương với một loạt các điện trở 100 Mega-ôm ở phía trước của pin. Điều này làm cho chúng cực kỳ hữu ích cho việc đọc một cảm biến, nhưng không cung cấp năng lượng một đèn LED.

Nói một cách nôm na, dân dã, thì khi một pin được cấu hình là INPUT thì bạn sẽ dễ dàng đọc được các tín hiệu điện và đọc được từ bất cứ thứ gì (Có điện  $\leq 5V$ )!

Nếu bạn đã cấu hình pin là INPUT, bạn sẽ muốn pin có một tham chiếu đến mặt đất (GND, cực âm), thường được thực hiện với một điện trở kéo xuống ( một điện trở đi xuống mặt đất ) như mô tả trong kỹ thuật số đọc nối tiếp.

### **Cấu hình một pin là INPUT\_PULLUP**

Chip Atmega trên Arduino có nội kéo lên điện trở (điện trở kết nối với hệ thống điện nội bộ) mà bạn có thể truy cập. Nếu bạn không thích mắc thêm một điện trở ở mạch ngoài, bạn có thể dùng tham số INPUT\_PULLUP trong pinMode(). Mặc định khi không được kết nối với một mạch ngoài hoặc được kết nối với cực dương thì pin sẽ nhận giá trị là HIGH, khi pin được thông tới cực âm xuống đất thì nhận giá trị là LOW.

### **Cấu hình một pin là đầu ra (OUTPUT)**

Để thiết đặt pin là một OUTPUT, chúng ta dùng pinMode ( ), điều này có nghĩa là làm cho pin ấy có một trở kháng thấp (cho dòng điện đi ra). Điều này có nghĩa, pin sẽ cung cấp một lượng điện đáng kể cho các mạch khác. Pin của vi điều khiển Atmega có thể cung cấp một nguồn điện liên tục 5V hoặc thả chìm ( cho điện thế bên ngoài chạy vào ) lên đến 40 mA ( milliamps ). Điều này làm cho chúng hữu ích để tạo năng

lượng đèn LED nhưng vô dụng đối với các cảm biến đọc!

Lúc bấy giờ, nếu bạn làm làm ngắn mạch (digitalWrite 1 pin là HIGH rồi nối trực tiếp đến cực âm hoặc digitalWrite 1 pin là LOW rồi mắc trực tiếp đến cực dương, hoặc những việc làm tương tự) thì mạch sẽ bị hỏng! Ngoài ra, với dòng điện chỉ 40mA thì trong một số trường hợp chúng ta không thể làm cho mô tơ hoặc relay hoạt động được. Để làm chúng hoạt động thì chúng ta cần chuẩn bị cho mình một số mạch sử dụng các IC khuếch đại chuyên dụng (gọi là mạch giao tiếp)

Thiết đặt Digital Pins như là INPUT, INPUT\_PULLUP, và OUTPUT

[ksp](#) gửi vào Thứ tư, 21 Tháng 5, 2014 - 20:27

- **9401 LƯỢT XEM**

Chân kỹ thuật số có thể được sử dụng như là INPUT, INPUT\_PULLUP, hoặc OUTPUT. Để thay đổi cách sử dụng một pin, chúng ta sử dụng hàm pinMode().

### **Cấu hình một pin là INPUT**

Các pin của Arduino ( Atmega ) được cấu hình là một INPUT với pinMode ( ) có nghĩa là làm cho pin ấy có trở kháng cao (không cho dòng điện đi ra) . Pin được cấu hình là INPUT làm việc tiêu thụ năng lượng điện của mạch rất nhỏ, nó tương đương với một loạt các điện trở 100 Mega-ôm ở phía trước của pin . Điều này làm cho chúng cực kỳ hữu ích cho việc đọc một cảm biến, nhưng không cung cấp năng lượng một đèn LED.

Nói một cách nôm na, dân dã, thì khi một pin được cấu hình là INPUT thì bạn sẽ dễ dàng đọc được các tín hiệu điện và đọc được từ bất cứ thứ gì (Có điện  $\leq 5V$ )!

Nếu bạn đã cấu hình pin là INPUT, bạn sẽ muốn pin có một tham chiếu đến mặt đất (GND, cực âm), thường được thực hiện với một điện trở kéo xuống ( một điện trở đi xuống mặt đất ) như mô tả trong kỹ thuật số đọc nối tiếp.

### **Cấu hình một pin là INPUT\_PULLUP**

Chip Atmega trên Arduino có nội kéo lên điện trở (điện trở kết nối với hệ thống điện nội bộ) mà bạn có thể truy cập. Nếu bạn không thích mắc thêm một điện trở ở mạch ngoài, bạn có thể dùng tham số

INPUT\_PULLUP trong pinMode(). Mặc định khi không được kết nối với một mạch ngoài hoặc được kết nối với cực dương thì pin sẽ nhận giá trị là HIGH, khi pin được thông tới cực âm xuống đất thì nhận giá trị là LOW.

### **Cấu hình một pin là đầu ra (OUTPUT)**

Để thiết đặt pin là một OUTPUT, chúng ta dùng pinMode ( ), điều này có nghĩa là làm cho pin ấy có một trở kháng thấp (cho dòng điện đi ra). Điều này có nghĩa, pin sẽ cung cấp một lượng điện đáng kể cho các mạch khác. Pin của vi điều khiển Atmega có thể cung cấp một nguồn điện liên tục 5V hoặc thả chìm ( cho điện thế bên ngoài chạy vào ) lên đến 40 mA ( milliamps ). Điều này làm cho chúng hữu ích để tạo năng lượng đèn LED nhưng vô dụng đối với các cảm biến đọc!

Lúc bấy giờ, nếu bạn làm làm ngắn mạch (digitalWrite 1 pin là HIGH rồi nối trực tiếp đến cực âm hoặc digitalWrite 1 pin là LOW rồi mắc trực tiếp đến cực dương, hoặc những việc làm tương tự) thì mạch sẽ bị hỏng! Ngoài ra, với dòng điện chỉ 40mA thì trong một số trường hợp chúng ta không thể làm cho mô tơ hoặc relay hoạt động được. Để làm chúng hoạt động thì chúng ta cần chuẩn bị cho mình một số mạch sử dụng các IC khuếch đại chuyên dụng (gọi là mạch giao tiếp)

Thiết đặt Digital Pins như là INPUT, INPUT\_PULLUP, và OUTPUT

[ksp](#) gửi vào Thứ tư, 21 Tháng 5, 2014 - 20:27

### **• 9402 LƯỢT XEM**

Chân kỹ thuật số có thể được sử dụng như là INPUT, INPUT\_PULLUP, hoặc OUTPUT. Để thay đổi cách sử dụng một pin, chúng ta sử dụng hàm pinMode().

### **Cấu hình một pin là INPUT**

Các pin của Arduino ( Atmega ) được cấu hình là một INPUT với pinMode ( ) có nghĩa là làm cho pin ấy có trở kháng cao (không cho dòng điện đi ra). Pin được cấu hình là INPUT làm việc tiêu thụ năng lượng điện của mạch rất nhỏ, nó tương đương với một loạt các điện trở 100 Mega-ôm ở phía trước của pin. Điều này làm cho chúng cực kỳ hữu ích cho việc đọc một cảm biến, nhưng không cung cấp năng lượng một đèn LED.

Nói một cách nôm na, dân dã, thì khi một pin được cấu hình là INPUT thì bạn sẽ dễ dàng đọc được các tín hiệu điện và đọc được từ bất cứ thứ gì (Có điện  $\leq 5V$ )!

Nếu bạn đã cấu hình pin là INPUT, bạn sẽ muốn pin có một tham chiếu đến mặt đất (GND, cực âm), thường được thực hiện với một điện trở kéo xuống ( một điện trở đi xuống mặt đất ) như mô tả trong kỹ thuật số đọc nối tiếp.

### **Cấu hình một pin là INPUT\_PULLUP**

Chip Atmega trên Arduino có nội kéo lên điện trở (điện trở kết nối với hệ thống điện nội bộ) mà bạn có thể truy cập. Nếu bạn không thích mắc thêm một điện trở ở mạch ngoài, bạn có thể dùng tham số INPUT\_PULLUP trong pinMode(). Mặc định khi không được kết nối với một mạch ngoài hoặc được kết nối với cực dương thì pin sẽ nhận giá trị là HIGH, khi pin được thông tới cực âm xuống đất thì nhận giá trị là LOW.

### **Cấu hình một pin là đầu ra (OUTPUT)**

Để thiết đặt pin là một OUTPUT, chúng ta dùng pinMode ( ), điều này có nghĩa là làm cho pin ấy có một trở kháng thấp (cho dòng điện đi ra). Điều này có nghĩa, pin sẽ cung cấp một lượng điện đáng kể cho các mạch khác . Pin của vi điều khiển Atmega có thể cung cấp một nguồn điện liên tục 5V hoặc thả chìm ( cho điện thế bên ngoài chạy vào ) lên đến 40 mA ( milliamps ). Điều này làm cho chúng hữu ích để tạo năng lượng đèn LED nhưng vô dụng đối với các cảm biến đọc!

Lúc bấy giờ, nếu bạn làm làm ngắn mạch (digitalWrite 1 pin là HIGH rồi nối trực tiếp đến cực âm hoặc digitalWrite 1 pin là LOW rồi mắc trực tiếp đến cực dương, hoặc những việc làm tương tự) thì mạch sẽ bị hỏng! Ngoài ra, với dòng điện chỉ 40mA thì trong một số trường hợp chúng ta không thể làm cho mô tơ hoặc relay hoạt động được. Để làm chúng hoạt động thì chúng ta cần chuẩn bị cho mình một số mạch sử dụng các IC khuếch đại chuyên dụng (gọi là mạch giao tiếp)

Hầu hết các mạch Arduino đều có một pin kết nối với một on-board LED (led nằm trên mạch) nối tiếp với một điện trở. LED\_BUILTIN là



một hằng số thay thế cho việc tuyên bố một biến có giá trị điều khiển on-board LED. Hầu hết trên các mạch Arduino, chúng có giá trị là 13 *true* là một hằng logic. Bạn cũng có thể HIỂU *true* là một hằng số nguyên mang giá trị là 1. Trong các biểu thức logic, một hằng số hay giá trị của một biểu thức khác 0 được xem như là mang giá trị *true*.

#### Lưu ý

- Không được viết "*true*" thành TRUE hay bất kì một dạng nào khác.
- Các giá trị sau là tương đương nhau: ***true*, HIGH, 1**

Trái lại với true, *false* là một hằng logic có giá trị là phủ định của *true* (và ngược lại), tức là  $(!true) = false$ . Bạn cũng có thể HIỂU *false* là một hằng số nguyên mang giá trị là 0. Trong các biểu thức logic, một hằng số hay giá trị của một biểu thức bằng 0 được xem như là bằng *false*.

**Lưu ý** Không được viết "*false*" thành FALSE hay bất kì một dạng nào khác.

- Các giá trị sau là tương đương nhau: ***false*, LOW, 0**
- **Mô tả**

"void" là một từ khóa chỉ dùng trong việc khai báo một function. Những function được khai báo với "void" sẽ không trả về bất kì dữ liệu nào khi được gọi.

**Ví dụ** led = 13;

```
void setup() {
```

```
pinMode(led, OUTPUT);
```



```
}  
void loop() {  
    blink();  
}  
void blink() {  
    digitalWrite(led, LOW);  
    delay(1000);  
    digitalWrite(led, HIGH);  
    delay(1000);  
}
```

### Giải thích

- "blink" là một function được định nghĩa với từ khóa "void", do đó nó không trả về một giá trị nào. Nhiệm vụ của "blink" chỉ là làm nhấp nháy đèn LED ở chân số 13 trên mạch Arduino.
- Bạn có thể thấy rằng những function kiểu này không dùng lệnh "return" để trả về giá trị của function

### Giới thiệu

**Một biến được khai báo kiểu boolean sẽ chỉ nhận một trong hai giá trị: true hoặc false. Và bạn sẽ mất 1 byte bộ nhớ cho điều đó.**

### Lưu ý

**Những cặp giá trị sau là tương đương nhau. Về bản chất, chúng đều là những hằng số nguyên với 2 giá trị 0 và 1:**

- true - false
- HIGH - LOW

- 1 - 0

### Ví dụ:

```
int led = 13;
```

```
boolean led_status;
```

```
void setup() {  
  pinMode(led, OUTPUT);  
  led_status = true;    // led ở trạng thái bật  
}  
  
void loop() {  
  digitalWrite(led, led_status);    // bật đèn, led_status = 1  
  delay(1000);  
  digitalWrite(led, !led_status);    // tắt đèn, !led_status = 0  
  delay(1000);  
}
```

### Kiểu dữ liệu

Kiểu dữ liệu này là kiểu dữ liệu biểu diễn cho 1 KÝ TỰ (nếu bạn cần biểu diễn một chuỗi trong chương trình Arduino - bạn cần sử dụng kiểu dữ liệu String). Kiểu dữ liệu này chiếm 1 byte bộ nhớ!

Kiểu char chỉ nhận các giá trị trong bảng mã [ASCII](#).

Kiểu char được lưu dưới dạng 1 số nguyên byte có số âm (có các giá trị từ -127 - 128), thay vì thiết đặt một biến kiểu char có giá trị là 'A', bạn có thể đặt là 65. Để hiểu rõ hơn bạn xem ví dụ dưới đây.

## Ví dụ

```
char myChar = 'A';
```

```
char myChar = 65;    // cả 2 cách khai báo đều hợp lệ
```

Giống hệt bài giới thiệu về kiểu [char](#). Tuy nhiên kiểu unsigned char lại biểu hiệu một số nguyên byte không âm (giá trị từ 0 - 255).

## Ví dụ

```
unsigned char myChar = 240;
```

Là một kiểu dữ liệu biểu diễn số nguyên nằm trong khoảng từ 0 đến 255. Bạn sẽ mất 1 byte bộ nhớ cho mỗi biến mang kiểu *byte*

## Ví dụ

```
byte a = 123;    //khai báo biến a mang kiểu byte, có giá trị là 123
```

Kiểu int là kiểu số nguyên chính được dùng trong chương trình Arduino. Kiểu int chiếm 2 byte bộ nhớ !

Trên mạch Arduino Uno, nó có đoạn giá trị từ -32,768 đến 32,767 ( $-2^{15}$  đến  $2^{15}-1$ ) (16 bit)

Trên mạch Arduino Due, nó có đoạn giá trị từ -2,147,483,648 đến 2,147,483,647 ( $-2^{31}$  đến  $2^{31}-1$ ) (32 bit) (lúc này nó chiếm 4 byte bộ nhớ)

## Ví dụ

```
int ledPin = 13;
```

## Cú pháp

```
int var = val;
```

var: tên biến

val: giá trị

## Một số thủ thuật lập trình

```
int x;
```

```
x = -32768;
```

```
x = x - 1; // x sẽ nhận giá trị là 32767
```

```
int x;
```

```
x = 32767;
```

```
x ++; // x sẽ nhận giá trị -32768
```

Kiểu `unsigned int` là kiểu số nguyên nằm trong khoảng từ 0 đến 65535 (0 đến  $2^{16} - 1$ ). Mỗi biến mang kiểu dữ liệu này chiếm 2 byte bộ nhớ.

### Lưu ý

Trên Arduino Due, *unsigned int* có khoảng giá trị từ 0 đến 4,294,967,295 ( $2^{32} - 1$ ) (lúc này nó chiếm 4 byte bộ nhớ).

Bạn có thể dễ dàng nhận ra rằng kiểu dữ liệu này không chứa các giá trị âm so với kiểu *int*.

### Cú pháp

```
unsigned int [tên biến] = [giá trị];
```

### Ví dụ

```
unsigned int ledPin = 13;
```

**Lưu ý đặc biệt** (nói chung cho các kiểu dữ liệu *unsigned*)

Khi một biến kiểu *unsigned int* được gán trị vượt ngoài phạm vi giá trị (bé hơn 0 hoặc lớn hơn 65525), giá trị của biến này sẽ tự động được đẩy lên giới hạn trên hoặc giới hạn dưới trong khoảng giá trị của nó.

### Ví dụ

```
unsigned int x = 0; // x nhận giá trị trong khoảng từ 0 đến 65535
```

```
x = x - 1          // x = 0 - 1 = 65535 (giới hạn trên của x)
```

```
x = x + 1          // x = 65535 + 1 = 0 (giới hạn dưới của x)
```

`word`

Giống như kiểu [unsigned int](#), kiểu dữ liệu này là kiểu số nguyên 16 bit không âm (chứa các giá trị từ 0 đến 65535), và nó chiếm 2 byte bộ nhớ!

### Ví dụ

```
word w = 10000;
```

*long* là một kiểu dữ liệu mở rộng của *int*. Những biến có kiểu *long* có thể mang giá trị 32bit từ -2,147,483,648 đến 2,147,483,647. Bạn sẽ mất 4 byte bộ nhớ cho một biến kiểu *long*.

Khi tính toán với số nguyên (biến kiểu *int*), bạn phải thêm hậu tố "L" phía sau các số nguyên kiểu *int* để chuyển chúng sang kiểu *long*. Việc tính toán (cộng, trừ, nhân,...) giữa 2 số thuộc 2 kiểu dữ liệu khác nhau là không được phép. Xem [Hậu tố U và L](#) để biết thêm chi tiết.

### Ví dụ

```
long a = 10;
```

```
long b = a + 10L // b = 20
```

Kiểu *unsigned long* là kiểu số nguyên nằm trong khoảng từ 0 đến 4,294,967,295 (0 đến  $2^{32} - 1$ ). Mỗi biến mang kiểu dữ liệu này chiếm 4 byte bộ nhớ.

### Ví dụ

```
unsigned long time;
```

```
void setup()
```

```
{
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop()
```

```
{
```

```
  Serial.print("Time: ");
```

```
  time = millis();
```

```
  Serial.println(time); //Xuất thời gian lúc chạy hết đoạn lệnh trên
```

```
  delay(1000); //Dừng chương trình trong 1 giây. Lúc này sẽ làm "tê liệt" hệ thống, bạn không thể chạy bất cứ lệnh gì trong thời gian delay
```

```
}
```

Giống hệt kiểu [int](#), tuy nhiên có điều trên mọi mạch Arduino nó đều chiếm 4 byte bộ nhớ và biểu thị giá trị trong khoảng [-32,768](#) đến [32,767](#) ( $-2^{15}$  đến  $2^{15}-1$ ) (16 bit).

### Ví dụ

```
short ledPin = 13;
```

### Cú pháp

```
short var = val;
```

var: tên biến

val: giá trị

Để định nghĩa 1 kiểu số thực, bạn có thể sử dụng kiểu dữ liệu float. Một biến dùng kiểu dữ liệu này có thể đặt một giá trị nằm trong khoảng - 3.4028235E+38 đến 3.4028235E+38. Nó chiếm 4 byte bộ nhớ.

Với kiểu dữ liệu float bạn có từ 6-7 chữ số có nghĩa nằm ở bên mỗi bên dấu ".". Điều đó có nghĩa rằng bạn có thể đặt một số thực dài đến 15 ký tự (bao gồm dấu .)

### **Lưu ý**

Để biểu diễn giá trị thực của một phép chia bạn phải 2 số thực chia cho lẫn nhau. Ví dụ: bạn xử lý phép tính  $5.0 / 2.0$  thì kết quả sẽ trả về là 2.5. Nhưng nếu mà bạn xử lý phép tính  $5 / 2$  thì kết quả sẽ là 2 (vì hai số nguyên chia nhau sẽ ra một số nguyên).

### **Ví dụ**

```
float myfloat;
```

```
float sensorCalbrate = 1.117;
```

### **Cú pháp**

```
float var = val;
```

var: tên biến

val: giá trị

### **Code tham khảo**

```
int x;
```

```
int y;
```

```
float z;
```

```
x = 1;
```

```
y = x / 2; // y sẽ trả về kết quả là 0
```

```
z = (float)x / 2.0; //z sẽ có kết quả là 0.5 (bạn nhập 2.0, chứ không phải là 2)
```

```
double
```

## Giới thiệu

Giống hết như kiểu [float](#). Nhưng trên mạch Arduino Due thì kiểu double lại chiếm đến 8 byte bộ nhớ (64 bit). Vì vậy hãy cẩn thận khi sử dụng kiểu dữ liệu này!

Array là mảng (tập hợp các giá trị có liên quan và được đánh dấu bằng những chỉ số). Array được dùng trên Arduino chính là Array [trong](#) ngôn ngữ lập trình C.

## Các cách khởi tạo một mảng

`int myInts[6];` // tạo mảng myInts chứa tối đa 6 phần tử (được đánh dấu từ 0-5), các phần tử này đều có kiểu là int => khai báo này chiếm  $2 \times 6 = 12$  byte bộ nhớ

`int myPins[] = {2, 4, 8, 3, 6};` // tạo mảng myPins chứa 5 phần tử (lần lượt là 2, 4, 8, 3, 6). Mảng này không giới hạn số lượng phần tử vì có khai báo là `[]`

`int mySensVals[6] = {2, 4, -8, 3, 2};` // tạo mảng mySensVals chứa tối đa 6 phần tử, trong đó 5 phần tử đầu tiên có giá trị lần lượt là 2, 4, -8, 3, 2

`char message[6] = "hello";` // tạo mảng ký tự (dạng chuỗi) có tối đa 6 ký tự!

## Truy cập các phần tử trong mảng

**Chú ý:** Phần tử đầu tiên trong mảng luôn được đánh dấu là **số 0**.

`mySensVals[0] == 2, mySensVals[1] == 4`, vâng vâng

Điều này có nghĩa rằng, việc khai báo một mảng có tối đa 10 phần tử, thì phần tử cuối cần (thứ 10) được đánh dấu là **số 9**

`int myArray[10] = {9, 3, 2, 4, 3, 2, 7, 8, 9, 11};`

`// myArray[9]` có giá trị là 11

`// myArray[10]` sẽ trả về một giá trị "hên xui" nằm trong khoảng giá trị của int

Vì vậy, hãy chú ý trong việc truy cập đến giá trị trong mảng, nếu bạn muốn truy cập đến phần tử cuối cùng thì hãy truy cập đến ô giới hạn của mảng - 1.

Hãy ghi nhớ rằng, trong trình biên dịch ngôn ngữ C, nó không kiểm tra bạn có truy cập đến một ô có nằm trong bộ nhớ hay không! Nên nếu không cẩn thận trong việc truy cập mảng, chương trình của bạn sẽ mắc lỗi logic và rất khó để tìm lỗi đấy!

### **Gán một giá trị cho một phần tử**

```
mySensVals[0] = 10;
```

### **Đọc một giá trị của một phần tử và gán cho một biến nào đó cùng kiểu dữ liệu**

```
x = mySensVals[0]; //10
```

### **Dùng mảng trong vòng lặp**

Mảng rất thường được dùng trong vòng lặp (chẳng hạn như dùng để lưu các chân digital quản lý đèn led). Trong đó, biến chạy của hàm for sẽ đi hết (hoặc một phần) của mảng, tùy thuộc vào yêu cầu của bạn mà thôi! Ví dụ về việc in 5 phần tử đầu của mảng myPins:

```
int i;
```

```
for (i = 0; i < 5; i = i + 1) {
```

```
    Serial.println(myPins[i]);
```

```
}
```

```
/*
```

```
shiftOut với 8 LED bằng 1 IC HC595
```

```
*/
```

```
//chân ST_CP của 74HC595
```

```
int latchPin = 8;
```

```
//chân SH_CP của 74HC595
```

```
int clockPin = 12;
```

```
//Chân DS của 74HC595
```

```
int dataPin = 11;
```



```
//Trạng thái của LED, hay chính là byte mà ta sẽ gửi qua shiftOut  
byte ledStatus;
```

```
void setup() {
```

```
    //Bạn BUỘC PHẢI pinMode các chân này là OUTPUT
```

```
    pinMode(latchPin, OUTPUT);
```

```
    pinMode(clockPin, OUTPUT);
```

```
    pinMode(dataPin, OUTPUT);
```

```
}
```

```
void loop() {
```

```
    /*
```

Trong tin học, ngoài các phép +, -, \*, / hay % mà bạn đã biết trên hệ cơ số 10.

Thì còn có nhiều phép tính khác nữa. Và một trong số đó là Bit Math (toán bit) trên hệ cơ số 2.

Để hiểu những gì tôi viết tiếp theo sau, bạn cần có kiến thức về Bit Math.

Để tìm hiểu về Bit Math, bạn vào mục Tài liệu tham khảo ở bảng chọn nằm phía trên cùng trang web và chạy xuống khi bạn kéo chuột trên trang Arduino.VN

```
    */
```

```
    //Sáng tuần tự
```

```
    ledStatus = 0;//mặc định là không có đèn nào sáng hết (0 = 0b00000000)
```

```
    for (int i = 0; i < 8; i++) {
```

```
        ledStatus = (ledStatus << 1) | 1;//Đẩy toàn bộ các bit qua trái 1 bit và cộng bit có giá trị là 1 ở bit 0
```

```
    /**
```

```
        Bắt buộc phải có để shiftOut
```

```
    **/
```

```
digitalWrite(latchPin, LOW); //các đèn LED sẽ không sáng khi bạn
digital LOW
```

```
//ShiftOut ra IC
```

```
shiftOut(dataPin, clockPin, MSBFIRST, ledStatus);
```

```
digitalWrite(latchPin, HIGH); //các đèn LED sẽ sáng với trạng thái vừa
được cập nhập
```

```
/**
```

```
    Kết thúc bắt buộc phải có
```

```
**/
```

```
delay(500); // Dừng chương trình khoảng 500 mili giây để thấy các
hiệu ứng của đèn LED
```

```
}
```

```
//Tắt tuần tự
```

```
for (int i = 0; i < 8; i++) {
```

```
    ledStatus <<= 1; //Đẩy tất cả các bit qua bên trái 1 bit
```

```
    digitalWrite(latchPin, LOW);
```

```
    shiftOut(dataPin, clockPin, MSBFIRST, ledStatus);
```

```
    digitalWrite(latchPin, HIGH);
```

```
    delay(500);
```

```
}
```

```
}
```

String Object - Đối tượng chuỗi cực mạnh trên Arduino

Nếu bạn muốn lưu một chuỗi nào đó, bạn có thể sử dụng kiểu [string](#) (mảng ký tự) hoặc sử dụng đối tượng String được giới thiệu ngày hôm nay. Vậy đối tượng String này có gì hay hơn kiểu string kia? Với object String bạn có thể làm nhiều việc hơn, chẳng hạn như cộng chuỗi, tìm kiếm chuỗi, xóa chuỗi,... Tuy nhiên, bạn cũng sẽ tốn nhiều bộ

nhớ hơn, nhưng ngược lại bạn sẽ có nhiều hàm hỗ trợ cho việc xử lý chuỗi của mình!

Để phân biệt giữa mảng chuỗi ([string](#)) và object String rất đơn giản. [string](#) là chuỗi với chữ "s" thường còn object String là chữ "S" hoa! Ngoài ra, chuỗi được định nghĩa trong hai dấu nháy kép (ví dụ như "Arduino.VN - Cong dong Arduino Viet Nam") được xem làm mảng chuỗi

### **hiện vụ**

Trả về ký tự thứ n (bắt đầu từ 0) của một kiểu String.

### **Cú pháp**

string.charAt(n)

### **Tham số**

string: một biến kiểu String.

n: vị trí cần lấy ký tự

### **Trả về**

Ký tự thứ n của một chuỗi

### **Ví dụ**

```
String text = "Arduino.vn";
```

```
void setup() {  
  Serial.begin(9600);  
  for (int i = 0; i < text.length(); i++) {  
    Serial.println(text.charAt(i)); //Xuất các ký tự của chuỗi  
  }  
  Serial.println();  
}  
void loop() {  
}
```

Kiểm tra thử chuỗi này đứng trước (bé hơn) hay đứng sau (lớn hơn) với chuỗi khác hay là không. Bạn tham khảo ví dụ cuối bài để rõ hơn.

### **Cú pháp**

string.compareTo(string2)

### **Tham số**

string: chuỗi này ([string](#))

string2: chuỗi kia ([string](#))

### **Trả về**

số dương: nếu chuỗi này bé hơn chuỗi kia

0: 2 chuỗi bằng nhau

số âm: nếu chuỗi này lớn hơn chuỗi kia

Nối 2 string lại thành 1 string. String thứ 2 được gắn sau string thứ 1.

### **Cú pháp**

string.concat(string,string2)

### **Tham số**

string, string2: biến kiểu String

### **Trả về**

Một kiểu string chứa nội dung của 2 string

### **Ví dụ**

### **Nối các chuỗi**

### **Mục tiêu**

**Chúng ta sẽ học cách nối chuỗi qua ví dụ dưới đây. Bài viết này mang nhiều tính chất về lập trình tin học hơn là phần cứng. Vì vậy bạn chỉ cần chỉ bị một mạch Arduino được gắn với máy tính qua cổng USB là ok.**

### **Lập trình**

```
String stringOne, stringTwo;
```

```
void setup() {
```

```
    // Mở cổng Serial để debug - xem thành quả của chúng ta
```

```
    Serial.begin(9600);
```

```
    while (!Serial) {
```

```
        ; // đợi cho cổng serial được bật (chỉ cần thiết với Arduino Leonardo)
```

```

}
stringOne = String("Cam bien");
stringTwo = String("Gia tri ");
//Gửi một đoạn thông báo là đã chạy xong hàm setup()
Serial.println("\n\nVi du ve noi chuoì:");
Serial.println();
}

void loop() {
    Serial.println(stringOne); // in chữ "cam bien"
    // chèn chuỗi thứ 1 vào sau chuỗi thứ 2
    stringTwo += stringOne;
    Serial.println(stringTwo); // in chữ "Gia tri cam bien"
    // thêm một chuỗi hằng vào
    stringTwo += " cho dau Analog ";
    Serial.println(stringTwo); // prints "Gia tri cam bien cho dau
    Analog "
    // Thêm một hằng ký tự vào
    stringTwo += 'A';
    Serial.println(stringTwo); // prints "Gia tri cam bien cho dau
    Analog A"
    // Thêm một số nguyên vào
    stringTwo += 0;
    Serial.println(stringTwo); // prints "Gia tri cam bien cho dau
    Analog A0"
    // thêm giá trị của một chân analog vào
    stringTwo += ": ";
    stringTwo += analogRead(A0);
    Serial.println(stringTwo); // prints "Gia tri cam bien cho dau
    Analog A0: 123" hoặc là một giá trị bất kỳ mà analogRead(A0) trả
    về

```

```

Serial.println("\n\nDoi gia tri của bien Chuoi");
stringOne = "Gia tri cua mot bien so nguyen kieu long: ";
stringTwo = "millis(): ";
stringOne += 123456789L;
Serial.println(stringOne); //Xuất "Gia tri cua mot bien so nguyen
kieu long: 123456789"
stringTwo.concat(millis());
Serial.println(stringTwo); //Xuất "millis(): "+thời điểm hiện tại
theo mili giây
// dừng hàm loop
while(true);
}

```

### Nhiệm vụ

So sánh 2 chuỗi có giống nhau hay không. Chuỗi trong Arduino phân biệt hoa thường nhé, vì vậy chuỗi "Arduino.vn" sẽ khác so với chuỗi "arduino.vn".

### Cú pháp

```
string.equals(string2)
```

### Tham số

string, string2: biến kiểu String

### Trả về

true: nếu 2 chuỗi bằng nhau

false: các trường hợp khác

So sánh chuỗi

### Giới thiệu

Chúng ta có nhiều cách để so sánh chuỗi trong môi trường lập trình Arduino. Trong đó, ta có thể dùng các toán tử như >=, <=, <, >, = hoặc sử dụng các hàm của thư viện String như [equals\(\)](#), [equalsIgnoreCase\(\)](#).

Việc so sánh chuỗi là việc so sánh mã của ký tự khác nhau đầu tiên của cả 2 chuỗi. Ví dụ, '1' < '2', 'a' < 'b', '999' > '1000' (vì ký tự đầu tiên khác

nhau của 2 chuỗi đó nằm ở vị trí thứ 0 và ký tự '9' trong mã ASCII có giá trị lớn hơn ký tự '1').

**Lưu ý:** Chúng ta rất dễ bị nhầm lẫn rằng việc so sánh chuỗi số cũng chính là so sánh số. Điều này không chính xác (ví dụ trên), và vì vậy ta cần phải cẩn thận khi so sánh các chuỗi số. Và điều đơn giản nhất đó là việc ta chuyển số các chuỗi số ấy thành số ở các kiểu như [int](#), [float](#), [long](#),...

**Bây giờ hãy gắn Arduino vào máy tính và upload đoạn code sau và rút kinh nghiệm của riêng mình!**

### Lập trình

```
String stringOne, stringTwo;

void setup() {
  // Mở cổng Serial với mức baudrate 9600
  Serial.begin(9600);
  while (!Serial) {
    ; // đợi khi nào cổng Serial được bật (chỉ cần thiết đối với mạch
    Leonardo)
  }
  stringOne = String("this");
  stringTwo = String("that");
}

void loop() {
  // so sánh 2 chuỗi có bằng nhau hay không (cách 1)
  if (stringOne == "this") {
    Serial.println("StringOne == \"this\"");
  }
  // nếu 2 chuỗi khác nhau
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " != " + stringTwo);
  }
}
```

// trường hợp 2 từ cùng 1 nghĩa nhưng cách viết khác nhau (hoa, thường)

```
stringOne = "This";
```

```
stringTwo = "this";
```

//Vì viết khác định dạng nên khi so sánh, 2 chuỗi này sẽ không bằng nhau

```
if (stringOne != stringTwo) {
```

```
    Serial.println(stringOne + " != " + stringTwo);
```

```
}
```

// Bạn có thể dùng hàm equals để kiểm tra 2 chuỗi có giống (bằng) nhau hay không

```
if (stringOne.equals(stringTwo)) {
```

Serial.println(stringOne + " equals " + stringTwo); //equals nghĩa là bằng

```
}
```

```
else {
```

Serial.println(stringOne + " does not equal " + stringTwo); // does not equal nghĩa là không bằng

```
}
```

// hoặc trong trường hợp bạn muốn kiểm tra 2 chuỗi có bằng nhau hay không mà không phân biệt hoa thường

```
if (stringOne.equalsIgnoreCase(stringTwo)) {
```

```
    Serial.println(stringOne + " equals (ignoring case) " + stringTwo);
```

//bằng (không phân biệt hoa thường)

```
} else {
```

Serial.println(stringOne + " does not equal (ignoring case) " + stringTwo); // không bằng (không phân biệt hoa thường)

```
}
```

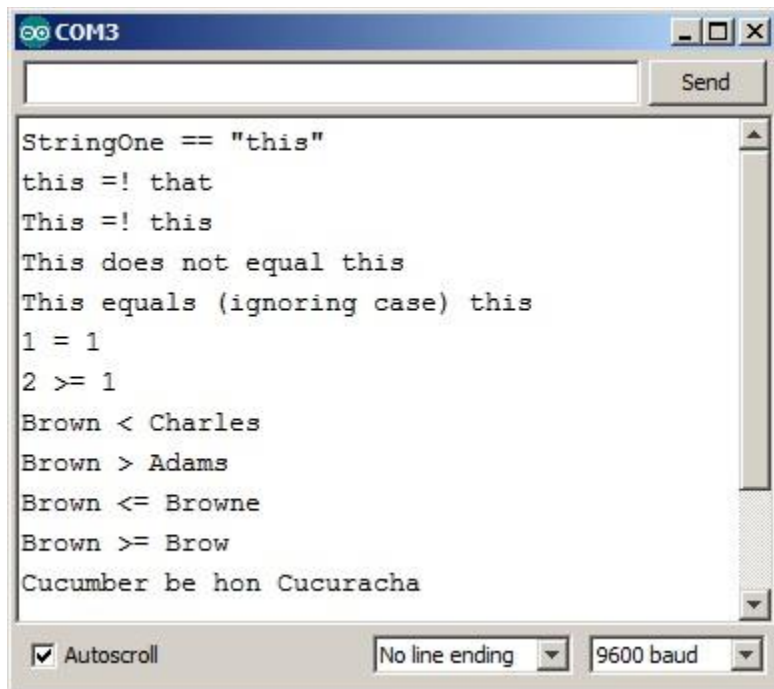
// một chuỗi số nguyên có thể so sánh với 1 số nguyên bằng cách...



```
stringOne = "1";  
int numberOne = 1;  
if (stringOne.toInt() == numberOne) {  
    Serial.println(stringOne + " = " + numberOne);  
}
```

```
// so sánh 2 chuỗi số nguyên  
stringOne = "2";  
stringTwo = "1";  
if (stringOne >= stringTwo) {  
    Serial.println(stringOne + " >= " + stringTwo);  
}  
// các toán tử so sánh cũng có thể dùng để so sánh 2 chuỗi  
stringOne = String("Brown");  
if (stringOne < "Charles") {  
    Serial.println(stringOne + " < Charles");  
}  
if (stringOne > "Adams") {  
    Serial.println(stringOne + " > Adams");  
}  
if (stringOne <= "Browne") {  
    Serial.println(stringOne + " <= Browne");  
}  
if (stringOne >= "Brow") {  
    Serial.println(stringOne + " >= Brow");  
}  
// hàm compareTo() dùng để so sánh 2 chuỗi
```

```
// nó sẽ trả về 1 số nguyên là vị trí đầu tiên (tính từ bên trái)
// khác nhau giữa 2 chuỗi.
// số nguyên nó là số âm nếu chuỗi A bé hơn chuỗi B
// là số dương nếu chuỗi A lớn hơn chuỗi B
// và bằng 0 khi 2 chuỗi bằng nhau.
stringOne = "Cucumber";
stringTwo = "Cucuracha";
if (stringOne.compareTo(stringTwo) < 0 ) {
    Serial.println(stringOne + " be hơn " + stringTwo);
}
else {
    Serial.println(stringOne + " lon hơn " + stringTwo);
}
while (true) {} //dừng chương trình không chạy tiếp nữa
}
```



## Nhiệm vụ

So sánh 2 chuỗi có giống nhau hay không. Khác với hàm [equals\(\)](#), hàm này không phân biệt hoa thường vì vậy chuỗi "Arduino.vn" sẽ bằng chuỗi "arduino.vn".

### **Cú pháp**

`string.equalsIgnoreCase(string2)`

### **Tham số**

`string, string2`: biến kiểu `String`

### **Trả về**

`true`: nếu 2 chuỗi bằng nhau

`false`: các trường hợp khác

### **Ví dụ**

So sánh chuỗi

`length()`

### **Nhiệm vụ**

Trả về số ký tự của chuỗi (đã bỏ qua ký tự null cảm cân).

### **Cú pháp**

`string.length()`

### **Tham số**

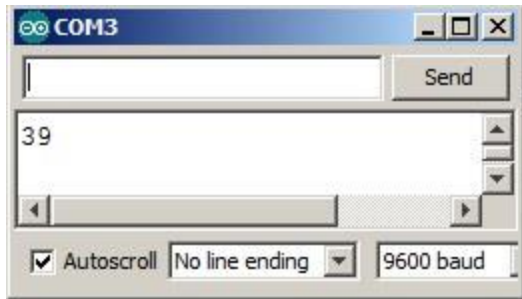
`string`: một biến kiểu `String`

### **Trả về**

[long](#): độ dài chuỗi

### **Ví dụ**

```
String text = "Arduino.VN - Cong dong Arduino Viet Nam";  
void setup() {  
    Serial.begin(9600); // Bật Serial ở baudrate 9600  
    Serial.println(text.length());  
}  
void loop() {  
}
```



## Mã thư viện:

Nhiệm vụ của hàm [remove\(\)](#) là xóa các ký tự trong đối tượng String. Nó sẽ xóa từ vị trí *index* đến hết chuỗi, hoặc từ vị trí *index* đến vị trí *index* cộng *length*.

## Cú pháp

1. `string.remove(index);`
2. `string.remove(index, length);`

## Tham số

string: đối tượng String

index: vị trí bắt đầu

length: độ dài sẽ bị cắt

## Trả về

không

## Ví dụ

Xóa chuỗi

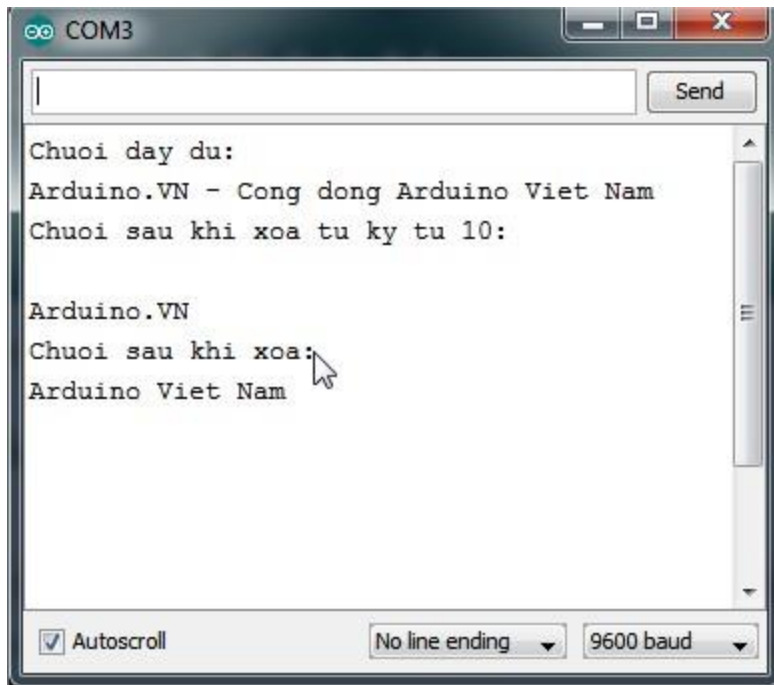
## Mục tiêu

Hàm [remove\(\)](#) cho phép chúng ta có xóa bất kỳ một đoạn ký tự trong một đối tượng String. Ta có thể dùng hàm này với 2 tham số khác nhau, mỗi cách dùng có một ý nghĩa riêng: dùng 1 tham số sẽ xóa từ vị trí bắt đầu (*index*) cho đến hết chuỗi; dùng 2 tham số sẽ xóa từ vị trí bắt đầu (*index*) cho đến khi xóa đủ ít nhất *length* ký tự. Các bạn hãy xem đoạn code lập trình ở dưới để rõ ràng hơn.

## Lập trình

1. `String example = "Arduino.VN - Cong dong Arduino Viet Nam";`
- 2.

```
3. void setup() {
4.   // Mở cổng Serial với mức baudrate là 9600
5.   Serial.begin(9600);
6.   while (!Serial) {
7.     ; // đợi cổng Serial được bật (Chỉ cần thiết với mạch Leonardo)
8.   }
9. }
10.
11. void loop() {
12.   // In ra chuỗi đầy đủ
13.   Serial.println("Chuoi day du:\n");
14.   Serial.println(example);
15.
16.   // Xóa từ ký tự 10 đến hết
17.   example.remove(10); // index = 10
18.   Serial.println("Chuoi sau khi xoa tu ky tu 10:\n");
19.   Serial.println(example); // Chỉ còn "Arduino.VN"
20.
21.   // Ví dụ khác, xóa từ vị trí thứ 7 đến khi xóa đủ 22 ký tự
22.   example = "Arduino.VN - Cong dong Arduino Viet Nam";
23.   example.remove(7, 23);
24.   Serial.println("Chuoi sau khi xoa:\n");
25.   Serial.println(example); // "Arduino Viet Nam"
26.
27.   while(1); // dừng vòng lặp
28. }
```



Tạo một bản copy kiểu chuỗi mảng char (đọc thêm tại [string](#)) từ kiểu String cho trước với độ dài xác định.

### Cú pháp

`string.toCharArray(buf, len)`

### Tham số

string: một biến kiểu String

buf: biến đệm dùng để lưu chuỗi kiểu mảng mới ([char\[\]](#))

len: độ dài chuỗi mới được tạo thành ([unsigned int](#)).

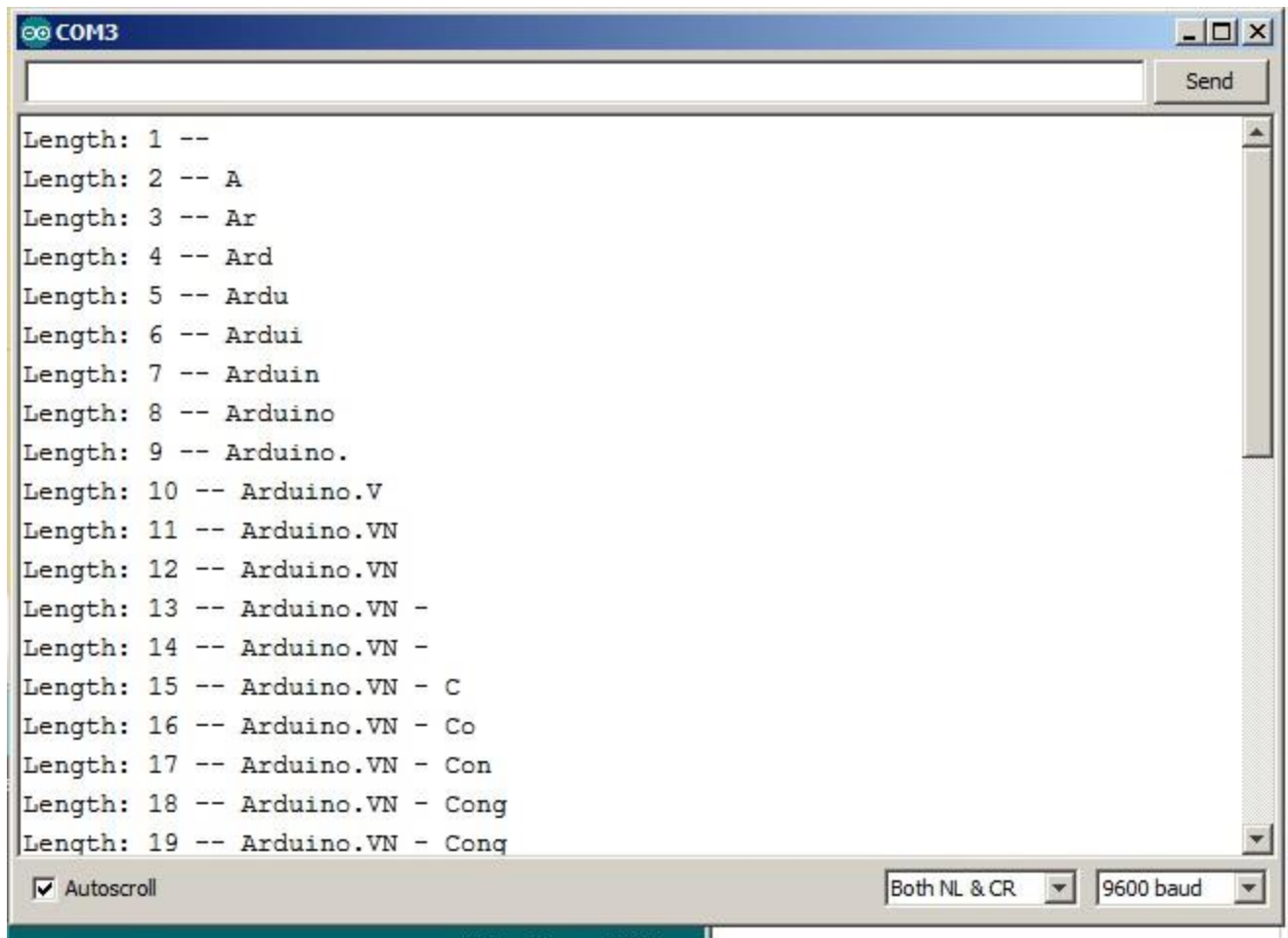
### Trả về

không

### Ví dụ

```
void setup() {  
  // mở Serial ở mức baudrate 9600  
  Serial.begin(9600);  
  while (!Serial) {  
    ; // đợi mở Serial, chỉ cần thiết với mạch Leonardo
```

```
}  
}  
char buf[100];  
void loop() {  
    // Hàm toCharArray sẽ tạo ra một chuỗi kiểu mảng mới với độ dài cho  
    trước  
    String text = "Arduino.VN - Cong dong Arduino Viet Nam";  
    //Bạn hãy thử bỏ +1  
    for (byte len = 1;len<=text.length()+1; len++){  
        Serial.print("Length: ");  
        Serial.print(len);  
        Serial.print(" -- ");  
        text.toCharArray(buf,len);  
        Serial.println(buf);  
        delay(100);  
    }  
    //Không làm gì nữa  
    while(true);  
}
```



## Mã thư viện:

### [String](#)

Chuyển một chuỗi thành một số. Hàm này sẽ trả về số nguyên đầu tiên nó tìm được với yêu cầu ký tự đầu tiên phải là một ký tự số.

### Cú pháp

`string.toInt()`

### Tham số

`string`: một biến kiểu `String`

### Trả về

[long](#): số nguyên được chuyển đổi. Hoặc là số 0 nếu không tìm thấy số thỏa mãn điều kiện trên.

### Ví dụ

Chuyển đổi chuỗi số thành số



Bài viết này có mục đích giải thích cách sử dụng hàm [toInt\(\)](#) và các ứng dụng trên Arduino. Bạn nên tham khảo để vận dụng thật tốt nhé. Đây là một ví dụ đơn giản liên quan đến lập trình hoàn toàn. Bạn chỉ cần một mạch Arduino và gắn vào máy tính là xong.

## Lập trình

```
String inString = ""; // biến inString dùng để lưu giá trị từ input
void setup() {
    // Mở cổng Serial với mức baudrate 9600
    Serial.begin(9600);
    while (!Serial) {
        ; // đợi cổng Serial được bật lên (chỉ cần với mạch Arduino Leonardo)
    }
    // Gửi một thông báo rằng hàm setup đã hoạt động xong
    Serial.println("\n\nChuyen doi chuoi so thanh so:");
    Serial.println();
}
void loop() {
    // Đọc giá trị từ Serial
    while (Serial.available() > 0) {
        int inChar = Serial.read(); //Xem thêm về Serial.read() tại
        http://arduino.vn/reference/library/serial/1/huong-dan-ham/read
        if (isDigit(inChar)) { // Hàm kiểm tra SỐ ĐÓ có là một ký tự số hay
            không (xem bảng ACSII http://arduino.vn/reference/bang-ma-ascii )
            // Chuyển đổi số đó thành ký tự
            // và thêm vào chuỗi. Nếu không thì bạn sẽ thêm một số nguyên vào
            đây :P
            inString += (char)inChar;
        }
        // Nếu inChar là một ký tự xuống dòng. Ta in kết quả ra
```

```
if (inChar == '\n' ) { //Một ký tự ta dùng dấu ' (nháy đơn) còn một chuỗi ta dùng " (dấu nháy kép)
```

```
    Serial.print("Gia tri :");
```

```
    Serial.println(inString.toInt());
```

```
    Serial.print("Chuoi: ");
```

```
    Serial.println(inString);
```

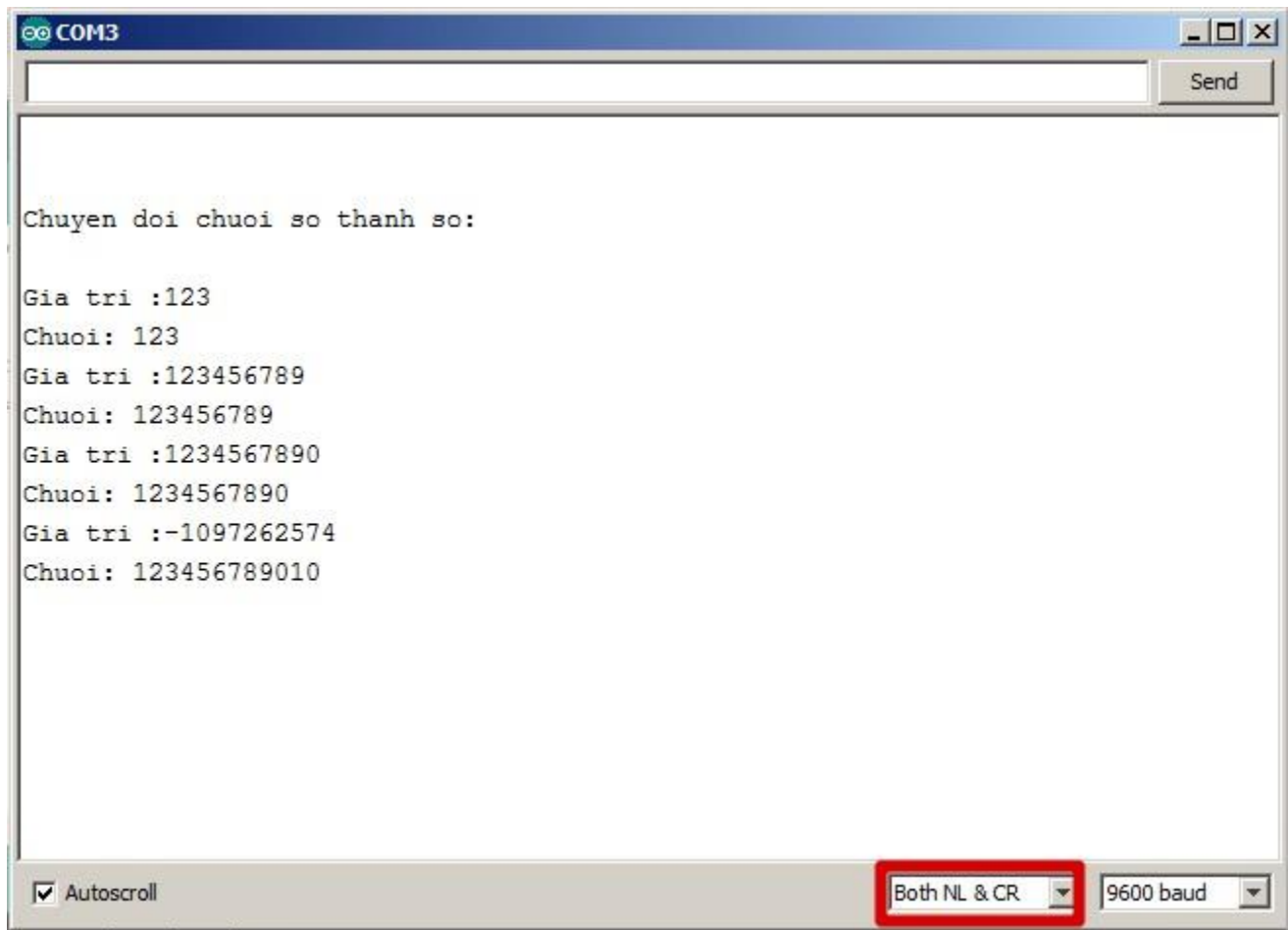
```
    // xóa giá trị của biến inString
```

```
    inString = "";
```

```
}
```

```
}
```

```
}
```



## Nhiệm vụ

Chuyển tất cả ký tự của chuỗi thành ký tự thường

## Cú pháp

`string.toLowerCase()`

## Tham số

`string`: một biến kiểu `String`

**Trả về** không

## Ví dụ

[Chuyển kiểu chữ \(hoa / thường\)](#)

Chuyển kiểu chữ (hoa / thường)

## Giới thiệu

Qua bài viết này, tớ sẽ hướng dẫn bạn cách chuyển kiểu chữ in hoa hoặc in thường từ một chuỗi cho trước. Đây là bài viết chỉ nói về ngôn ngữ lập trình Arduino, vì vậy bạn chỉ cần một mạch Arduino gắn vào máy tính là xong.

## Chương trình

```
void setup() {  
  // mở Serial ở mức baudrate 9600  
  Serial.begin(9600);  
  while (!Serial) {  
    ; // đợi mở Serial, chỉ cần thiết với mạch Leonardo  
  }  
}  
  
void loop() {  
  // toUpperCase() sẽ chuyển toàn bộ các ký tự trong chuỗi thành in hoa  
  String stringOne = "<html><head><body>";  
  Serial.println(stringOne);  
  stringOne.toUpperCase();  
  Serial.println(stringOne);  
}
```

// toLowerCase() sẽ chuyển toàn bộ các ký tự trong chuỗi thành in thường

```
String stringTwo = "</BODY></HTML>";
```

```
Serial.println(stringTwo);
```

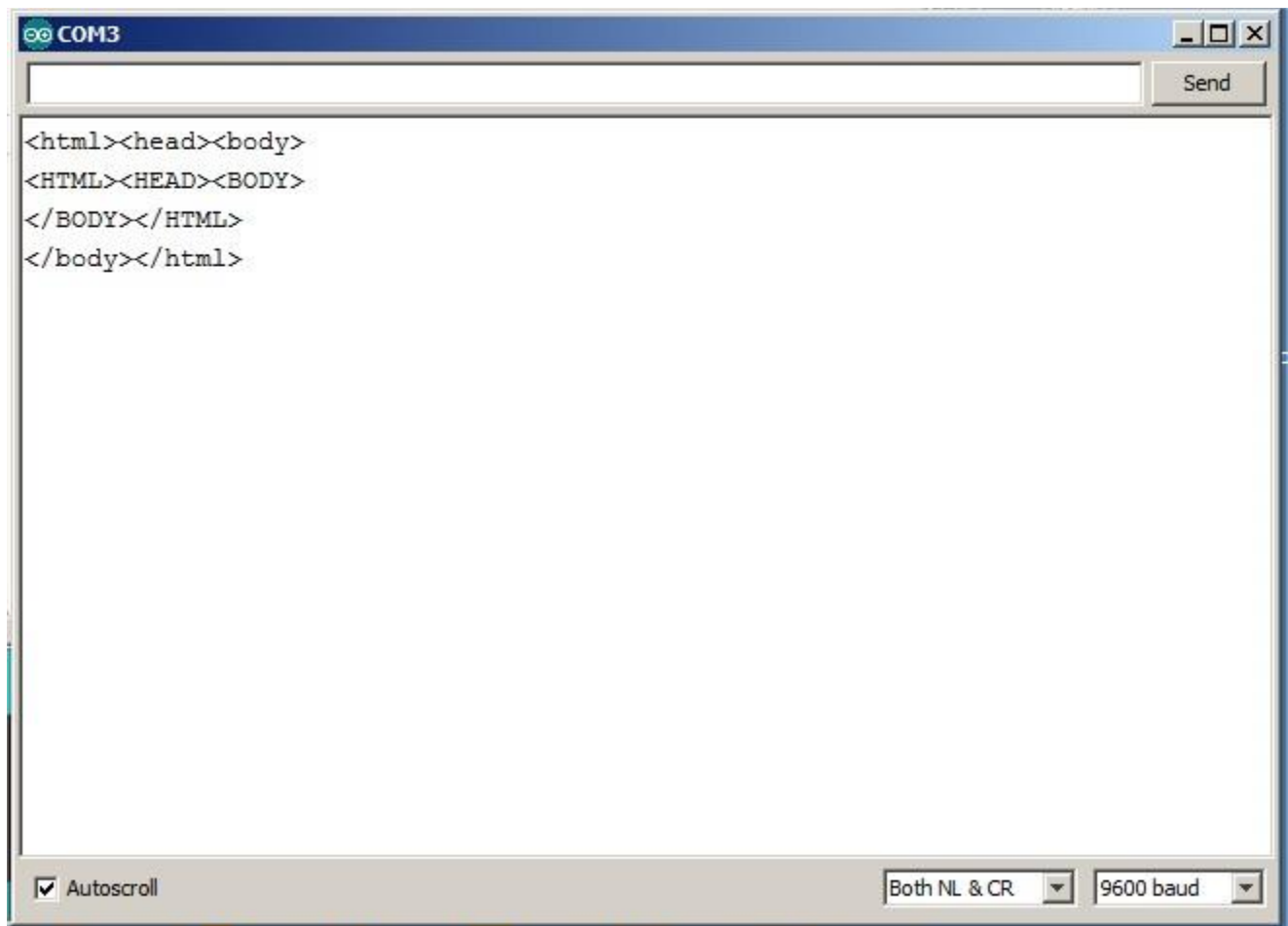
```
stringTwo.toLowerCase();
```

```
Serial.println(stringTwo);
```

//Không làm gì nữa

```
while(true);
```

```
}
```



## Nhiệm vụ

Chuyển tất cả ký tự của chuỗi thành ký tự hoa.

## Cú pháp

string.toUpperCase()

## Tham số

string: một biến kiểu String

Trả về không

## Ví dụ

[Chuyển kiểu chữ \(hoa / thường\)](#)

[String](#)

## Nhiệm vụ

Hàm này sẽ cắt bỏ hết những ký tự trống ở đầu và cuối câu. Các bạn xem ví dụ để rõ hơn.

**Cú pháp** string.trim()

**Tham số** string: một biến kiểu String

Trả về không

## Ví dụ

Độ dài và Chuỗi đã cắt gọn

## Mục đích

Qua ví dụ này bạn sẽ biết cách làm thế nào để lấy độ dài của một chuỗi và cắt gọn chuỗi đó. Bài này chỉ nói về kiến thức lập trình nên bạn chỉ cần duy nhất một mạch Arduino đã được kết nối tới máy tính là đủ.

## Lập trình

### Làm thế nào để lấy được độ dài một chuỗi?

```
String txtMsg = "";           // biến dùng để lưu chuỗi được gửi từ
máy tính
int lastStringLength = txtMsg.length(); // độ dài của chuỗi trước đó
void setup() {
  // Mở cổng Serial ở baudrate 9600
  Serial.begin(9600);
  while (!Serial) {
    ; // đợi mở cổng Serial. Chỉ cần thiết đối với mạch Arduino Leonardo
  }
```

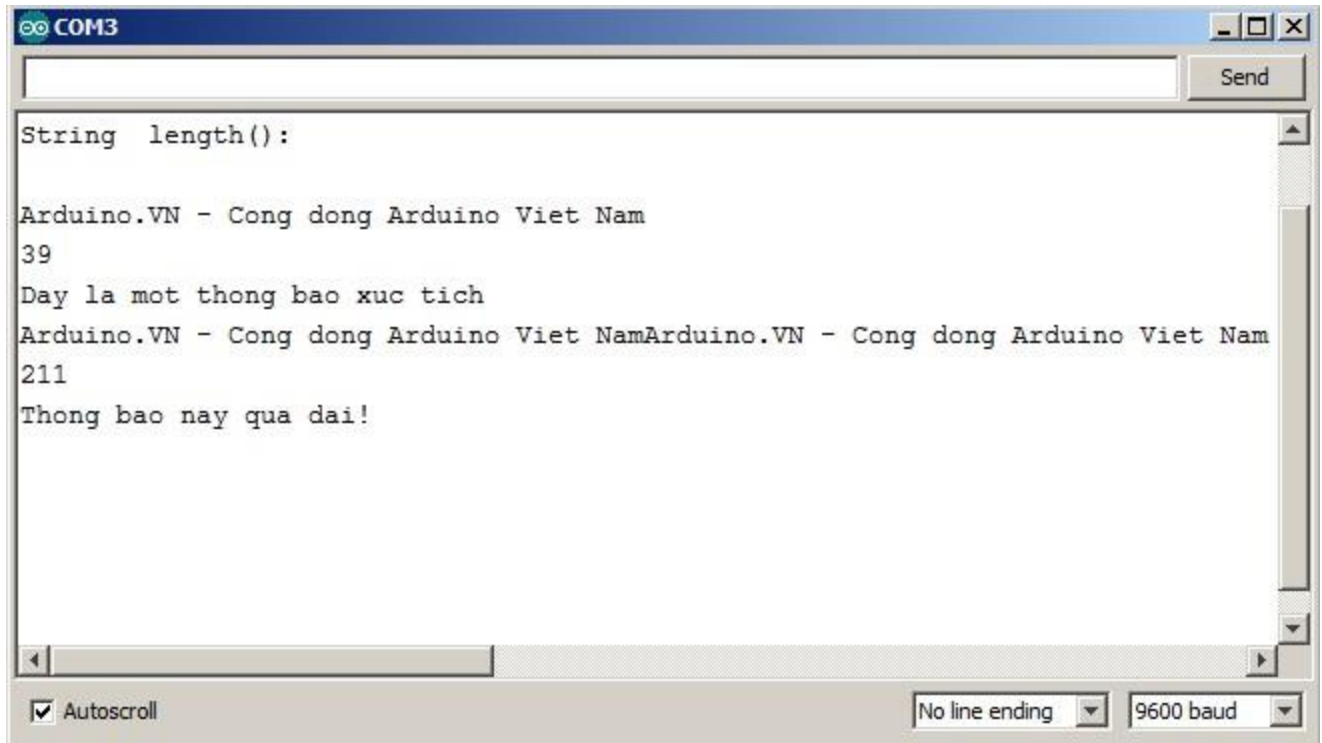
```

// Gửi một thông báo là phần setup đã hoạt động xong.
Serial.println("\n\nString length()");
Serial.println();
}

void loop() {
    // thêm tất cả những ký tự nhận được từ Serial Monitor
    while (Serial.available() > 0) {
        char inChar = Serial.read();
        txtMsg += inChar;
        delay(2); //Bạn hãy thử bỏ dòng này xem :).
        //Rồi sau đó thêm một bước nữa ở setup, bạn hãy nâng baudrate
        lên thành 115200!
    }

    // Xuất thông báo
    if (txtMsg.length() != lastStringLength) {
        Serial.println(txtMsg);
        Serial.println(txtMsg.length());
        // Nếu độ dài chuỗi bé hơn 140 thì xuất thông báo này
        if (txtMsg.length() < 140) {
            Serial.println("Day la mot thong bao xuc tich");
        }
        else {
            Serial.println("Thong bao nay qua dai!");
        }
        // lưu lại độ dài của chuỗi này để cho lần nhận dữ liệu tiếp theo
        lastStringLength = txtMsg.length();
    }
}

```

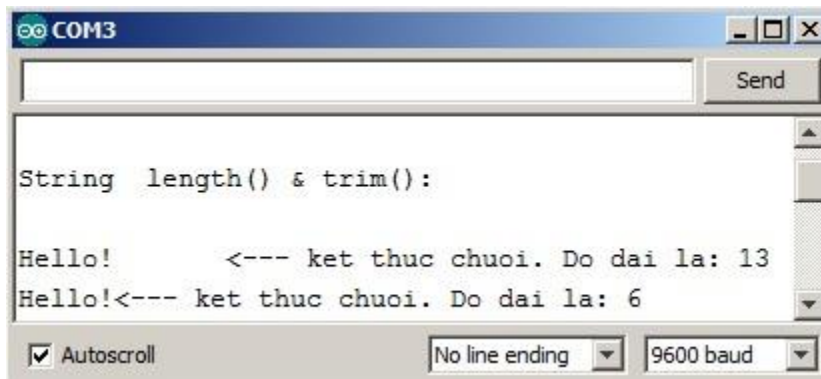


## Làm thế nào để cắt gọn chuỗi ấy?

Nói về thuật ngữ cắt gọn một tí nhé. Cắt gọn sẽ cắt bỏ những ký tự "trắng" ở đầu và cuối chuỗi. Các ký tự trắng bao gồm: Khoảng cách (Space bar) ([bảng ASCII 32](#)), tab ([bảng ASCII 9](#)), veritcal tab ([bảng ASCII 11](#)), form feed ([bảng ASCII 12](#)), carriage return ([bảng ASCII 12](#)), và newline (\n) ([bảng ASCII 10](#)).

```
void setup() {  
  //Mở cổng Serial với mức baudrate 9600  
  Serial.begin(9600);  
  while (!Serial) {  
    ; // đợi cổng Serial được mở. Chỉ cần ở mạch Arduino Leonardo  
  }  
  // gửi một thông báo  
  Serial.println("\n\nString length() & trim():");  
  Serial.println();  
}  
void loop() {
```

```
// đây là một chuỗi với những khoảng trắng
String stringOne = "Hello!    ";
Serial.print(stringOne);
Serial.print("<--- ket thuc chuoi. Do dai la: ");
Serial.println(stringOne.length());
// cắt gọn chuỗi
stringOne.trim();
Serial.print(stringOne);
Serial.print("<--- ket thuc chuoi. Do dai la: ");
Serial.println(stringOne.length());
// ngừng vòng loop
while(true);
}
```



## String

Phạm vi của biến và fan loại

Biến tĩnh là biến sẽ được tạo ra duy nhất một lần khi gọi hàm lần đầu tiên và nó sẽ không bị xóa đi để tạo lại khi gọi lại hàm ấy. Đây là sự khác biệt giữa biến tĩnh và biến cục bộ.

Biến tĩnh là loại biến lưỡng tính, vừa có tính chất của 1 biến toàn cục, vừa mang tính chất của 1 biến cục bộ:

- Tính chất 1 biến toàn cục: biến không mất đi khi chương trình con kết thúc, nó vẫn nằm trong ô nhớ của chương trình và được tự động cập nhật khi chương trình con được gọi lại. Giống như 1 biến toàn cục vậy.



- Tính chất 1 biến cục bộ: biến chỉ có thể được sử dụng trong chương trình con mà nó được khai báo.

Để khai báo bạn chỉ cần thêm từ khóa "static" trước khai báo biến. Xem ví dụ để rõ hơn.

### Ví dụ

```
void setup(){
    Serial.begin(9600); // Khởi tạo cổng Serial ở baudrate 9600
}

void loop() {
    testStatus();// Chạy hàm testStatus
    delay(500); // dừng 500 giây để bạn thấy được sự thay đổi
}

void testStatus() {
    static int a = 0;// Khi khai báo biến "a" là biến tĩnh
    // thì duy nhất chỉ có 1 lần đầu tiên khi gọi hàm testStatus
    // là biến "a" được tạo và lúc đó ta gán "a" có giá trị là 0

    a++;
    Serial.println(a);
    // Biến a sẽ không bị mất đi khi chạy xong hàm testStatus
    // Đó là sự khác biệt giữa biến tĩnh và biến cục bộ!
}
```

Với một từ khóa "const" nằm trước một khai báo biến, bạn sẽ làm cho biến này thành một biến chỉ có thể đọc "read-only". Nếu bạn có "lỡ làm" thay đổi giá trị của một biến hằng thì đừng lo lắng, chương trình dịch sẽ báo lỗi cho bạn!

Các biến có từ khóa const vẫn tuân theo [phạm vi hiệu lực của biến](#).

Ngoài cách sử dụng const để khai báo một biến hằng, ta còn có thể sử dụng [#define](#) để khai báo một **hằng số** hoặc **hằng chuỗi**. Tuy nhiên sử dụng const được ưa chuộng hơn trong lập trình, vì khả năng "tuân theo"

phạm vi hiệu lực của biến! Còn [#define](#) hoạt động như thế nào thì bạn có thể xem thêm bài viết của có [tại đây](#).

### Ví dụ

```
const float pi = 3.14;
```

```
float x;
```

```
// ....
```

```
x = pi * 2; // bạn có thể dùng hằng số pi trong tính toán - vì đơn giản  
bạn chỉ đọc nó
```

```
pi = 7; // lỗi ! bạn không thể thay đổi giá trị của một hằng số
```

### Dùng const hay dùng #define ?

Để khai báo một biến hằng số (nguyên / thực) hoặc hằng chuỗi thì bạn có thể dùng cả 2 cách đều được. Tuy nhiên, để khai báo một biến mảng ([array](#)) là một hằng số bạn chỉ có thể sử dụng từ khóa const. Và đây là một lý do nữa khiến const được dùng nhiều và được ưa chuộng hơn #define!

Nếu bạn cần tìm hiểu về biến volatile, hãy tham khảo ở bài viết tại [Wikipedia](#)

### Sơ lược

Một biến được nên khai báo với từ khóa **volatile** nếu giá trị của nó có thể bị tác động bởi các yếu tố nằm ngoài sự kiểm soát của chương trình đang chạy, nói đơn giản hơn là giá trị của biến thay đổi một cách không xác định. Các biến kiểu **volatile** thường là:

1. *Memory-mapped peripheral registers* (thanh ghi ngoại vi có ánh xạ đến ô nhớ)
2. *Biến toàn cục được truy xuất từ các tiến trình con xử lý ngắt* (Interrupt Service Routine - ISR)
3. *Biến toàn cục được truy xuất từ nhiều tác vụ trong một ứng dụng thực thi đa luồng* (Concurrently Executing Thread).

Trong lập trình Arduino mà cụ thể hơn là trong việc giao tiếp với các chip Atmega, nơi duy nhất xảy ra sự tác động không dự báo trước đến giá trị các biến là những phần chương trình có sự giao tiếp với các **ngắt**

(*interrupts*). Người ta ngăn sự thay đổi bất thường này bằng cách khai báo từ khóa ***volatile*** khi khai báo biến - cách mà sẽ đưa biến lên lưu trữ ở RAM để xử lý tạm thời thay vì lưu trữ ở các thanh ghi (register) bị ảnh hưởng bởi *interrupts*. Những biến này thuộc kiểu **số 2** kể trên.

### **Khai báo ví dụ**

```
volatile int state;
```

### **Chương trình mẫu**

Chương trình sau sẽ thực hiện bật tắt đèn LED ở chân Digital 13 khi chân Interrupt 0 trên Arduino thay đổi trạng thái

```
int pin = 13;
volatile int state = LOW;
void setup() {
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}
void loop() {
    digitalWrite(pin, state);
}
void blink() {
    state = !state;
}
```

Hàm hỗ trợ

### **Giới thiệu**

Hàm sizeof() có nhiệm vụ trả về số byte bộ nhớ của một biến, hoặc là trả về tổng số byte bộ nhớ của một mảng [array](#).

### **Cú pháp**

```
sizeof(variable)
```

### **Tham số**

variable: mọi kiểu dữ liệu hoặc mọi biến (thuộc bất cứ kiểu dữ liệu nào) hoặc một mảng.

## Ví dụ

Hàm sizeof() tỏ ra rất hiệu quả trong việc kiểm tra độ dài [chuỗi](#), nhưng bạn cần lưu ý cho về ký tự "cần cân" của Arduino. Sau đây là một ví dụ về việc đọc từng giá trị của một chuỗi cho trước. Để thấy được hiệu quả chương trình bạn hãy thử thay chuỗi trong ví dụ bằng một chuỗi khác xem.

```
char myStr[] = "this is a test";
int i;
void setup(){
  Serial.begin(9600);
}
void loop() {
  for (i = 0; i < sizeof(myStr) - 1; i++){
    Serial.print(i, DEC);
    Serial.print(" = ");
    Serial.write(myStr[i]);
    Serial.println();
  }
  delay(5000); // làm chậm chương trình để bạn thấy được chương trình
  này muốn nói lên điều gì
}
```

## Lưu ý

Vì hàm sizeof sẽ trả về số byte bộ nhớ của một biến hay một mảng nào đó, vì vậy nếu bạn muốn ĐẾM SỐ phần tử của một mảng số nguyên có kiểu dữ liệu > 1 byte (như là: [int](#), [word](#), [float](#),...) thì bạn cần chia số bộ nhớ của mảng cho số bộ nhớ của kiểu dữ liệu của mảng đó. Ví dụ một mảng có kiểu int.

```
for (i = 0; i < (sizeof(myInts)/sizeof(int)) - 1; i++) {
  // hàm làm gì đó với biến myInts[i]
}
```

Hàm và thủ tục  
Nhập xuất digital  
digitalWrite()

## Giới thiệu

Xuất tín hiệu ra các chân digital, có 2 giá trị là [HIGH](#) hoặc là [LOW](#). Nếu một pin được thiết đặt là OUTPUT bởi [pinMode\(\)](#). Và bạn dùng digitalWrite để xuất tín hiệu thì điện thế tại chân này sẽ là 5V (hoặc là 3,3 V trên mạch 3,3 V) nếu được xuất tín hiệu là [HIGH](#), và 0V nếu được xuất tín hiệu là [LOW](#).

Nếu một pin được thiết đặt là INPUT bởi pinMode(). Lúc này digitalWrite sẽ bật (HIGH) hoặc tắt (LOW) hệ thống điện trở pullup nội bộ. Chúng tôi khuyên bạn nên dùng [INPUT\\_PULLUP](#) nếu muốn bật hệ thống điện trở pullup nội bộ.

## Cú pháp

digitalWrite(pin,value)

**Thông số pin:** Số của chân digital mà bạn muốn thiết đặt

value: [HIGH](#) hoặc [LOW](#)

**Trả về** không

## Ví dụ

```
int ledPin = 13;           // đèn LED được kết nối với chân digital 13

void setup()
{
    pinMode(ledPin, OUTPUT); // thiết đặt chân ledPin là OUTPUT
}

void loop()
{
    digitalWrite(ledPin, HIGH); // bật đèn led
    delay(1000);                // dừng trong 1 giây
    digitalWrite(ledPin, LOW);  // tắt đèn led
    delay(1000);                // dừng trong 1 giây
}
```

```
}  
digitalRead()
```

### **Giới thiệu**

Đọc tín hiệu điện từ một chân digital (được thiết đặt là [INPUT](#)). Trả về 2 giá trị [HIGH](#) hoặc [LOW](#).

**Cú pháp** digitalRead(pin)

**Thông số** pin: giá trị của digital muốn đọc

**Trả về**

[HIGH](#) hoặc [LOW](#)

### **Ví dụ**

Ví dụ này sẽ làm cho đèn led tại pin 13 nhận giá trị như giá trị tại pin 2

```
int ledPin = 13; // chân led 13
```

```
int inPin = 2; // button tại chân 2
```

```
int val = 0; // biến "val" dùng để lưu tín hiệu từ digitalRead
```

```
void setup()
```

```
{
```

```
  pinMode(ledPin, OUTPUT); // đặt pin digital 13 là output
```

```
  pinMode(inPin, INPUT); // đặt pin digital 2 là input
```

```
}
```

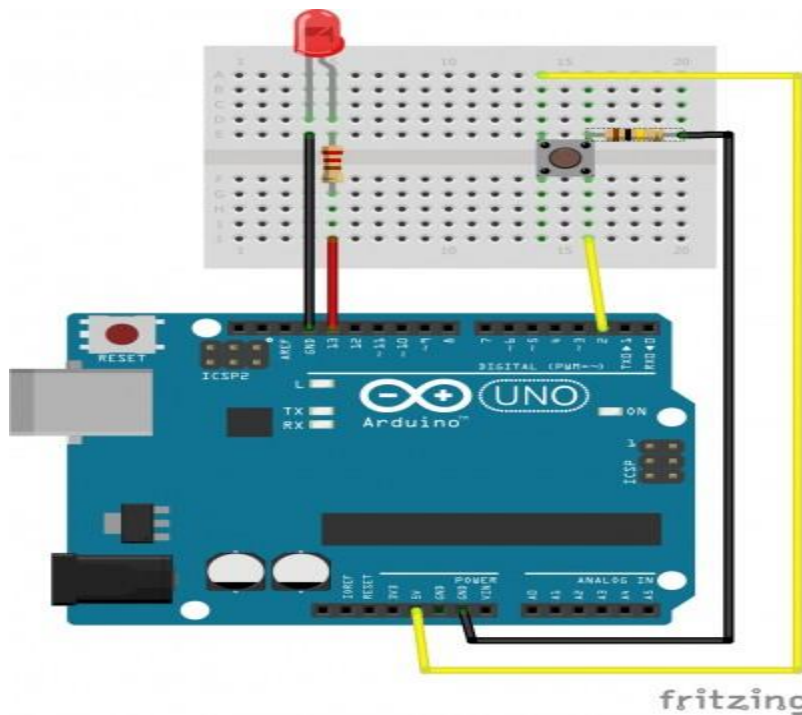
```
void loop()
```

```
{
```

```
  val = digitalRead(inPin); // đọc tín hiệu từ digital2
```

```
  digitalWrite(ledPin, val); // thay đổi giá trị của đèn LED là giá trị của  
  digital 2
```

```
}
```



## Chú ý

Nếu chân input không được kết nối với bất kỳ một thứ gì thì hàm `digitalRead()` sẽ trả về tín hiệu HIGH hoặc LOW một cách "hên xui"

Các chân Analog cũng có thể dùng được `digitalRead` với các cổng pin có tên như là: A0, A1,...

`pinMode()`

## Giới thiệu

Cấu hình 1 pin quy định hoạt động như là một đầu vào (INPUT) hoặc đầu ra (OUTPUT). Xem mô tả kỹ thuật số (datasheet) để biết chi tiết về các chức năng của các chân.

Như trong phiên bản Arduino 1.0.1, nó có thể kích hoạt các điện trở pullup nội bộ với chế độ `INPUT_PULLUP`. Ngoài ra, chế độ INPUT vô hiệu hóa một cách rõ ràng điện trở pullups nội bộ.

## Cú pháp

`pinMode(pin, mode)`

## Thông số

pin: Số của chân digital mà bạn muốn thiết đặt

mode: [INPUT](#), [INPUT\\_PULLUP](#) hoặc [OUTPUT](#)

## Trả về

không

## Ví dụ

```
int ledPin = 13;           // đèn LED được kết nối với chân digital 13

void setup()
{
    pinMode(ledPin, OUTPUT); // thiết đặt chân ledPin là OUTPUT
}

void loop()
{
    digitalWrite(ledPin, HIGH); // bật đèn led
    delay(1000);                // dừng trong 1 giây
    digitalWrite(ledPin, LOW);  // tắt đèn led
    delay(1000);                // dừng trong 1 giây
}
```



## Ghi chú

Các chân Analog cũng có thể được sử dụng dưới dạng Digital I/O. Ví dụ: A0, A1, ...

Nhập xuất analog

Hàm `analogReference()` có nhiệm vụ đặt lại mức (điện áp) tối đa khi đọc tín hiệu `analogRead`. Ứng dụng như sau, giả sử bạn đọc một tín hiệu dạng analog có hiệu điện thế từ 0-1,1V. Nhưng mà nếu dùng mức điện áp tối đa mặc định của hệ thống (5V) thì khoảng giá trị sẽ ngắn hơn => độ chính xác kém hơn => hàm này ra đời để giải quyết việc đó!

## Cú pháp

`analogReference(type)`



type: một trong các kiểu giá trị sau: DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, hoặc EXTERNAL

Kiểu	Nhiệm vụ đảm nhiệm	Ghi chú
DEFAULT	Đặt mức điện áp tối đa là 5V (nếu trên mạch dùng nguồn 5V làm nuôi chính) hoặc là 3,3V (nếu trên mạch dùng nguồn 3,3V làm nguồn nuôi chính)	
INTERNAL	Đặt lại mức điện áp tối đa là 1,1 V (nếu sử dụng vi điều khiển ATmega328 hoặc ATmega168) Đặt lại mức điện áp tối đa là 2,56V (nếu sử dụng vi điều khiển ATmega8)	
INTERNAL1V1	Đặt lại mức điện áp tối đa là 1,1 V	Chỉ có trên Arduino Mega
INTERNAL2V56	Đặt lại mức điện áp tối đa là 2,56 V	Chỉ có trên Arduino Mega
EXTERNAL	<b>Đặt lại mức điện áp tối đa BẰNG với mức điện áp được cấp vào chân AREF</b>	<b>Chỉ được cấp vào chân AREF một điện áp nằm trong khoảng 0-5V</b>

**Trả về**

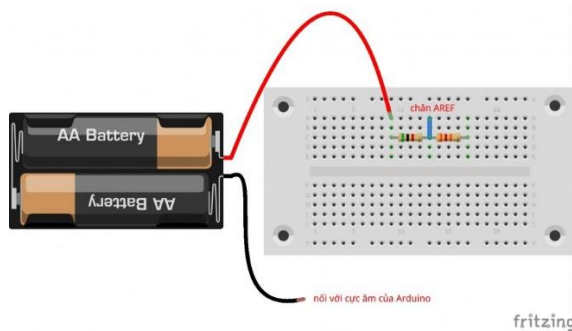
không

**Cảnh báo**

NẾU bạn sử dụng kiểu EXTERNAL cho hàm analogReference thì bạn **BUỘC** phải cấp nó một nguồn nằm trong khoảng từ 0-5V, và nếu bạn đã cấp một nguồn điện thỏa mãn điều kiện trên vào chân AREF thì bạn **BUỘC** phải gọi dòng lệnh analogReference(EXTERNAL) trước khi sử dụng [analogRead\(\)](#) [NẾU KHÔNG MẠCH BẠN SẼ "die"]

Ngoài ra, bạn có thể sử dụng một điện trở 5kΩ đặt trước chân AREF rồi đặt nguồn điện ngoài (điện áp bạn muốn cấp vào chân AREF). Vì sao lại làm như vậy? Bởi vì sao chỗ gắn chân AREF có một nội điện trở (điện trở có sẵn trong mạch) khoảng 32kΩ => sẽ tạo ra mạch giảm áp phân bản dễ nhất => giảm điện thế gắn vào chân AREF => không hư nếu bạn

có lẽ gần nguồn hơn 5V 😊. Nếu bạn chưa hiểu rõ, bạn có thể xem hình sau.



Nhiệm vụ của `analogRead()` là đọc giá trị điện áp từ một chân Analog (ADC). Trên mạch Arduino UNO có 6 chân Analog In, được kí hiệu từ A0 đến A5. Trên các mạch khác cũng có những chân tương tự như vậy với tiền tố "A" đứng đầu, sau đó là số hiệu của chân.

`analogRead()` **luôn** trả về 1 số nguyên nằm trong khoảng từ 0 đến 1023 tương ứng với thang điện áp (mặc định) từ 0 đến 5V. Bạn có thể điều chỉnh thang điện áp này bằng hàm [analogReference\(\)](#).

Hàm `analogRead()` cần 100 micro giây để thực hiện.

Khi người ta nói "đọc tín hiệu analog", bạn có thể hiểu đó chính là việc đọc giá trị điện áp.

### Cú pháp

```
analogRead([chân đọc điện áp]);
```

### Ví dụ

```
int voltage = analogRead(A0);
```

Trong đó A0 là chân dùng để đọc điện áp.

Nếu bạn chưa kết nối chân đọc điện áp, hàm `analogRead()` sẽ trả về một giá trị ngẫu nhiên trong khoảng từ 0 đến 1023. Để khắc phục điều này, bạn phải mắc thêm một điện trở có trị số lớn (khoảng 10k ohm trở lên) hoặc một tụ điện 104 từ chân đọc điện áp xuống GND.

### Reference Tags:

*Bạn cần tìm hiểu về [xung PWM](#) trước khi đọc bài viết này !*

### Giới thiệu

`analogWrite()` là lệnh xuất ra từ một chân trên mạch Arduino một mức tín hiệu analog (phát xung PWM). Người ta thường điều khiển mức sáng tối của đèn LED hay hướng quay của động cơ servo bằng cách phát xung PWM như thế này.

Bạn không cần gọi hàm `pinMode()` để đặt chế độ OUTPUT cho chân sẽ dùng để phát xung PWM trên mạch Arduino.

### Cú pháp

`analogWrite([chân phát xung PWM], [giá trị xung PWM]);`

Giá trị mức xung PWM nằm trong khoảng từ 0 đến 255, tương ứng với mức duty cycle từ 0% đến 100%

### Ví dụ

```
int led = 11;
void setup() {
}
void loop() {
  for (int i = 0; i <= 255; i++) {
    analogWrite(led,i);
    delay(20);
  }
}
```

Đoạn code trên có chức năng làm sáng dần một đèn LED được kết nối vào chân số 11 trên mạch Arduino.

### Xung PWM

#### Kiến thức cơ bản

Xung là các trạng thái cao / thấp (HIGH/LOW) về mức điện áp được lặp đi lặp lại. Đại lượng đặc trưng cho 1 xung PWM (Pulse Width Modulation) bao gồm **tần số**(frequency) và **chu kì xung** (duty cycle).

Tần số là gì?

Tần số là số lần lặp lại trong 1 đơn vị thời gian. Đơn vị tần số là Hz, tức là số lần lặp lại dao động trong 1 giây.

Lấy ví dụ,  $1\text{Hz} = 1$  dao động trong 1 giây.  $2\text{Hz} = 2$  dao động trong 1 giây.  $16\text{MHz} = 16$  triệu dao động trong 1 giây.

Như vậy theo quy tắc tam suất: 16 triệu dao động - 1 giây --> 1 dao động tốn  $1/16.000.000$  (giây) =  $0,0625$  (micro giây)

Cách xác định 1 dao động như thế nào? Đa phần các bạn mới nghiên cứu điện tử thường mắc sai lầm ở việc xác định 1 dao động. Dao động được xác định từ trạng thái bắt đầu và kết thúc ngay trước khi trạng thái bắt đầu được lặp lại.



### Cách xác định 1 dao động

Như vậy thông thường, 1 dao động sẽ bao gồm 2 trạng thái điện: mức cao (x giây) và mức thấp (y giây). Tỷ lệ phần trăm thời gian giữa 2 trạng thái điện này chính là chu kỳ xung.

Với  $x/y = 0\%$  ta có xung chứa toàn bộ điện áp thấp (khái niệm xung nên hiểu mở rộng)

Với  $x/y = 50\%$  thì 50% thời gian đầu, xung có điện áp cao, 50% sau xung có điện áp thấp.

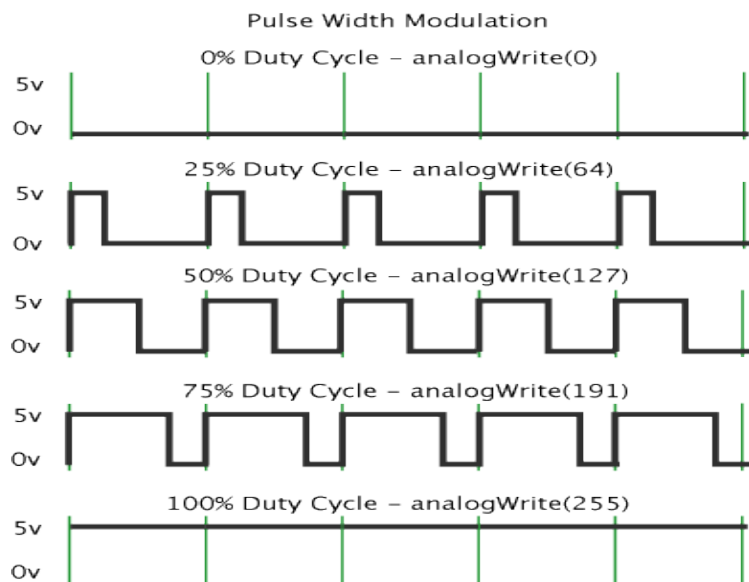
Với  $x/y = 100\%$  ta có xung chứa toàn bộ điện áp cao.

Tóm lại, với 1 xung ta có:

1. Tần số: để tính toán ra được thời gian của 1 xung
2. Chu kỳ xung: bao nhiêu thời gian xung có mức áp cao, bao nhiêu thời gian xung có mức áp thấp.

### Liên hệ với Arduino

Với kiến thức cơ bản về xung, các bạn sẽ hiểu rõ hơn về xung trong thực tế như thế nào.



## Xung khi sử dụng với hàm `analogWrite` trong Arduino

Giữa 2 vạch màu xanh lá cây là 1 xung.

<code>analogWrite</code>	tỉ lệ	chu kì xung
<code>analogWrite(0)</code>	0/255	0%
<code>analogWrite(64)</code>	64/255	25%
<code>analogWrite(127)</code>	127/255	50%
<code>analogWrite(191)</code>	191/255	75%
<code>analogWrite(255)</code>	255/255	100%

Hàm [analogWrite\(\)](#) trong Arduino giúp việc tạo 1 xung dễ dàng hơn. Hàm này truyền vào tham số cho phép thay đổi chu kì xung, bạn có thể tính toán ra được chu kì xung như ở bảng trên. Tần số xung được Arduino thiết lập mặc định.

Đối với board Arduino Uno, xung trên các chân 3,9,10,11 có tần số là 490Hz, xung trên chân 5,6 có tần số 980Hz. Làm thế nào để tạo ra các xung có tần số nhanh hơn? Bạn có thể tham khảo thêm các thư viện riêng hỗ trợ việc này. Trong mã nguồn Arduino gốc không hỗ trợ phần này.

**Lưu ý:** xung điều khiển servo có tên gọi [PPM](#) (Pulse Position Modulation) khác với xung PWM.

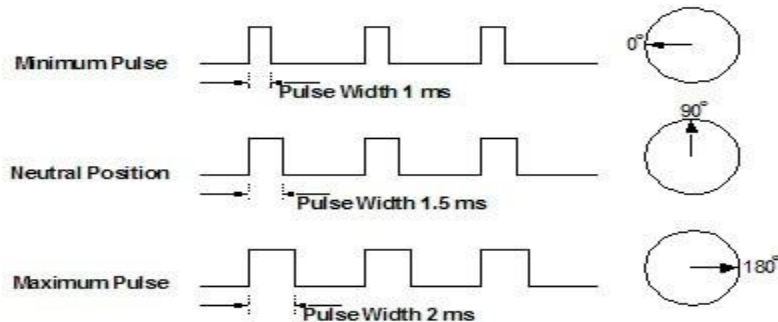
Xung PPM

Bài này giới thiệu về xung PPM (Pulse Position Modulation) được sử dụng để điều khiển servo. Về bản chất PPM cũng là một xung, do vậy bạn cần tham khảo về xung trong bài [xung PWM](#) trước khi đọc tiếp nội dung của bài này.

Xung PPM khác với PWM ở chỗ:

1. tần số thông thường có giá trị trong khoảng 50Hz (20 mili giây), không quan trọng
2. thời gian xung ở mức cao chỉ từ 1ms đến 2ms, rất quan trọng.
3. có thể có nhiều hơn 1 sự thay đổi trạng thái điện cao/thấp

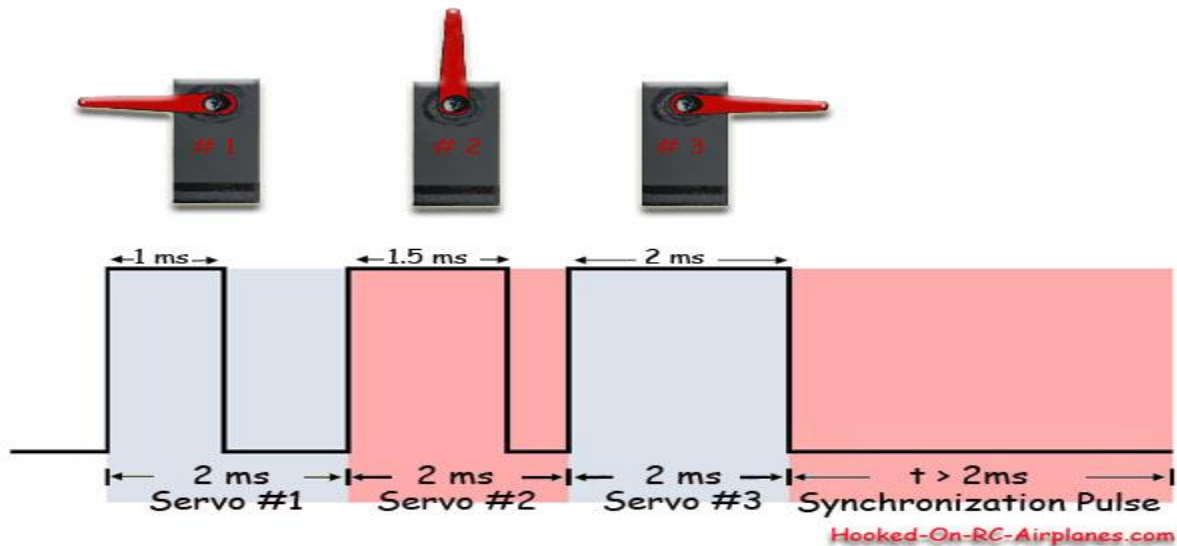
Nắm bắt được 2 ý trên ta đã có thể phân biệt được xung PPM và xung PWM giống nhau và khác nhau như thế nào.



Thời gian xung ở mức cao quy định góc quay của RC servo.

Với thời gian 1ms mức cao, góc quay của servo là 0, 1.5ms góc quay 90 và 2ms góc quay là 180. Các góc khác từ 0-180 được xác định trong khoảng thời gian 1-2ms.

Lưu ý: có thể ghép nhiều xung trong cùng 1 thời gian là 20ms để xác định vị trí góc của nhiều servo cùng 1 lúc. Tối đa là 10 servo.



Điều khiển 3 servo cùng lúc

Hàm thời gian

millis() có nhiệm vụ trả về một số - là thời gian (tính theo mili giây) kể từ lúc mạch Arduino bắt đầu chương trình của bạn. Nó sẽ tràn số và quay số 0 (sau đó tiếp tục tăng) sau 50 ngày.

**Tham số** không

**Trả về** một số nguyên kiểu [unsigned long](#) là thời gian kể từ lúc thương trình Arduino được khởi động

**Ví dụ**

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.print("Time: ");
  time = millis();
  // in ra thời gian kể từ lúc chương trình được bắt đầu
  Serial.println(time);
}
```

```
// đợi 1 giây trước khi tiếp tục in  
delay(1000);  
}
```

### **Lưu ý quan trọng:**

Các hàm về thời gian trong Arduino gồm millis() và [micros\(\)](#) sẽ bị tràn số sau 1 thời gian sử dụng. Với hàm millis() là khoảng 50 ngày. Tuy nhiên, do là kiểu số nguyên không âm (unsigned long) nên ta dễ dàng khắc phục điều này bằng cách sử dụng hình thức ép kiểu.

```
unsigned long time;  
byte ledPin = 10;  
void setup()  
{  
    // khởi tạo giá trị biến time là giá trị hiện tại  
    // của hàm millis();  
    time = millis();  
    pinMode(ledPin, OUTPUT);  
    digitalWrite(ledPin, LOW);  
}  
void loop()  
{  
    // Lưu ý các dấu ngoặc khi ép kiểu  
    // đoạn chương trình này có nghĩa là sau mỗi 1000 mili giây  
    // đèn Led ở chân số 10 sẽ thay đổi trạng thái  
    if ( (unsigned long) (millis() - time) > 1000)  
    {  
        // Thay đổi trạng thái đèn led  
        if (digitalRead(ledPin) == LOW)  
        {  
            digitalWrite(ledPin, HIGH);  
        }  
    }  
}
```



```

    } else {
        digitalWrite(ledPin, LOW);
    }
    // cập nhật lại biến time
    time = millis();
}
}

```

Thông thường, nếu ta có 2 số A, B và B lớn hơn A ( $B > A$ ) thì phép trừ thu được  $A-B$  là một số âm. Nhưng khi ép kiểu [unsigned long](#) là kiểu số nguyên dương, không có số âm nên giá trị trả về là 1 số nguyên dương lớn.

Ví dụ: kết quả của phép trừ:

```
unsigned long ex = (unsigned long) (0 - 1);
```

là 4294967295, con số này chính là giá trị lớn nhất của kiểu số [unsigned long](#). Giống như bạn đạp xe 1 vòng và quay về vạch xuất phát vậy.

`micros()` có nhiệm vụ trả về một số - là thời gian (tính theo micro giây) kể từ lúc mạch Arduino bắt đầu chương trình của bạn. Nó sẽ tràn số và quay số 0 (sau đó tiếp tục tăng) sau 70 phút. Tuy nhiên, trên mạch Arduino 16MHz (ví dụ Duemilanove và Nano) thì giá trị của hàm này tương đương 4 đơn vị micro giây. Ví dụ `micros()` trả về giá trị là 10 thì có nghĩa chương trình của bạn đã chạy được 40 microgiây. Tương tự, trên mạch 8Mhz (ví dụ LilyPad), hàm này có giá trị tương đương 8 micro giây.

Lưu ý:  $10^6$  micro giây = 1 giây

### **Tham số không**

**Trả về** một số nguyên kiểu [unsigned long](#) là thời gian kể từ lúc thương trình Arduino được khởi động

### **Ví dụ**

```

unsigned long time;
void setup(){
    Serial.begin(9600);

```

```

}
void loop(){
  Serial.print("Time: ");
  time = micros();
  // in ra thời gian kể từ lúc chương trình được bắt đầu
  Serial.println(time);
  // đợi 1 giây trước khi tiếp tục in
  delay(1000);
}

```

delay có nhiệm vụ dừng chương trình trong thời gian mili giây. Và cứ mỗi 1000 mili giây = 1 giây.

### Cú pháp

delay(ms)

**Thông số** ms: thời gian ở mức mili giây. ms có kiểu dữ liệu là [unsigned long](#)

**Trả về** không

**Ví dụ** Bạn xem bài viết: [Bài 2: Cách làm đèn LED nhấp nháy theo yêu cầu](#)

delayMicroseconds có nhiệm vụ dừng chương trình trong thời gian micro giây. Và cứ mỗi 1000000 micro giây = 1 giây.

**Cú pháp** delayMicroseconds(micro);

**Thông số** micro: thời gian ở mức micro giây. micro có kiểu dữ liệu là [unsigned int](#). micro phải  $\leq 16383$ . Con số này là mức tối đa của hệ thống Arduino, và có thể sẽ được điều chỉnh tăng trong tương lai. Và nếu bạn muốn dừng chương trình lâu hơn thì bạn cần dùng hàm [delay](#)

**Trả về** không

Ví dụ

```

int outPin = 8;           // digital pin 8
void setup()
{

```

```

pinMode(outPin, OUTPUT);    // đặt là output
}
void loop()
{
  digitalWrite(outPin, HIGH); // xuất 5V
  delayMicroseconds(50);      // đợi 50 micro giây
  digitalWrite(outPin, LOW);   // xuất 0V
  delayMicroseconds(50);      // đợi 50 micro giây
}

```

Ví dụ cho ta một cách để tạo một xung PWM tại chân số 8.

Hàm toán học

Hàm min có nhiệm vụ trả về giá trị nhỏ nhất giữa hai biến.

**Cú pháp** min(x, y);

**Tham số** x: số thứ nhất, mọi kiểu dữ liệu đều được chấp nhận.

y: số thứ hai, mọi kiểu dữ liệu đều được chấp nhận.

**Trả về** Số nhỏ nhất trong 2 số.

**Gợi ý** Hàm min được dùng để lấy chặn trên (không để giá trị vượt quá một mức quy định nào đó).

**Cảnh báo cú pháp**

min(a++, 100); // nếu bạn nhập như thế này thì sẽ bị lỗi đây

a++;

min(a, 100); // nhưng nếu nhập như thế này thì ổn! Và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này, bạn nhé

Hàm max có nhiệm vụ trả về giá trị lớn nhất giữa hai biến.

**Cú pháp** max(x, y);

**Tham số** x: số thứ nhất, mọi kiểu dữ liệu đều được chấp nhận.

y: số thứ hai, mọi kiểu dữ liệu đều được chấp nhận.

**Trả về** Số lớn nhất trong 2 số.

**Gợi ý** Hàm max được dùng để lấy chặn dưới (không để giá trị tụt xuống quá một mức quy định nào đó).

### **Cảnh báo cú pháp**

max(a--, 0); // nếu bạn nhập như thế này thì sẽ bị lỗi đấy

a--;

min(a, 0); // nhưng nếu nhập như thế này thì ổn! Và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này, bạn nhé

Hàm abs có nhiệm vụ trả về giá trị tuyệt đối của một số.

### **Cú pháp abs(x);**

**Tham số** x: một số bất kỳ

**Trả về** Nếu  $x \geq 0$ , thì trả về x còn ngược lại là trả về -x

### **Cảnh báo cú pháp**

abs(a--); // nếu bạn nhập như thế này thì sẽ bị lỗi đấy

a--;

abs(a); // nhưng nếu nhập như thế này thì ổn! Và hãy ghi nhớ là không được để bất cứ phép tính nào bên trong hàm này, bạn nhé

map() là hàm dùng để chuyển một giá trị từ thang đo này sang một giá trị ở thang đo khác. Giá trị trả về của hàm map() luôn là một số nguyên.

### **Cú pháp map(val,A1,A2,B1,B2);**

#### **Trong đó:**

- val là giá trị cần chuyển đổi
- A1, A2 là giới hạn trên và dưới của thang đo hiện tại
- B1,B2 là giới hạn trên và dưới của thang đo cần chuyển tới

### **Ví dụ**

//Chuyển đổi 37 độ C sang độ F

int C\_deg = 37;

int F\_deg = map(37,0,100,32,212); //F\_deg = 98

pow() là hàm dùng để tính lũy thừa của một số bất kì (có thể là số nguyên hoặc số thực tùy ý). pow() trả về kết quả tính toán này.

### **Cú pháp pow([cơ số], [lũy thừa]);**

### Ví dụ

```
int luythua1 = pow(2,3);  
float luythua2 = pow(1.2,2.3);  
double luythua3 = pow(1.11111,1.11111);  
//luythua1 = 8    (=23)  
//luythua2 = 1.52  (=1.22.3)  
//luythua3 = 1.12  (=1.111111.11111)
```

**Chú ý** Cả 2 tham số đưa vào hàm pow() đều được định nghĩa là kiểu số thực float. Kết quả trả về của pow() được định nghĩa là kiểu số thực double

sqrt() là hàm dùng để tính căn bậc 2 của một số bất kì (có thể là số nguyên hoặc số thực tùy ý) và trả về kết quả này.

**Cú pháp** sqrt([số cần tính căn bậc 2]);

### Ví dụ

```
int v1 = sqrt(9);  
float v2 = sqrt(6.4);  
double v3 = sqrt(6.5256);  
int v4 = sqrt(-9);  
float v5 = sqrt(-6.4);  
//v1 = 3  
//v2 = 2.53  
//v3 = 2.55  
//v4 = 0  
//v5 = NaN    (tham khảo hàm isnan\(\))
```

### Chú ý

Tham số đưa vào hàm sqrt() có thể là bất kì kiểu dữ liệu biểu diễn số nào. Kết quả trả về của sqrt() được định nghĩa là kiểu số thực double hoặc NaN nếu tham số đưa vào là số thực bé hơn 0.

Hàm `sq()` được dùng để tính bình phương của một số bất kì, số này có thể thuộc bất kì kiểu dữ liệu biểu diễn số nào. `sq()` trả về giá trị mà nó tính được với kiểu dữ liệu giống như kiểu dữ liệu của tham số ta đưa vào.

**Cú pháp** `sq([số cần tính bình phương]);`

**Ví dụ**

```
int binhphuong1 = sq(5);
int binhphuong2 = sq(-5);
float binhphuong3 = sq(9.9);
float binhphuong4 = sq(-9.9);
//binhphuong1 = 25
//binhphuong2 = 25
//binhphuong3 = 98.01
//binhphuong4 = 98.01
```

snan sẽ trả về là [true](#) nếu giá trị cần kiểm tra không phải là một biểu thức toán học đúng đắn. Chữ nan có nghĩa là Not-A-Number.

**Cú pháp** `isnan(double x);`

**Trả về** [true](#) hoặc [false](#)

**Ví dụ**

```
isnan(sqrt(-2)); //true
isnan(sqrt(2)); // false
```

**Giới thiệu** Bắt buộc giá trị nằm trong một khoảng cho trước.

**Cú pháp** `constrain(x, a, b)`

**Tham số**

x: giá trị cần xét

a: chặn dưới (a là giá trị nhỏ nhất của khoảng)

b: chặn trên (b là giá trị lớn nhất của khoảng)

**Trả về**

x: nếu  $a \leq x \leq b$

a: nếu  $x < a$

b: nếu  $x > b$

### Ví dụ

```
int sensVal = analogRead(A2);
```

```
sensVal = constrain(sensVal, 10, 150);
```

//Giới hạn giá trị sensVal trong khoảng [10,150]

Hàm lượng giác

Hàm này có nhiệm vụ tính cos một góc (đơn vị radian). Giá trị chạy trong đoạn  $[-1,1]$ .

**Cú pháp** cos(rad)

**Tham số** rad: góc ở đơn vị radian (kiểu [float](#))

**Trả về** cos của góc rad (kiểu [double](#))

Hàm này có nhiệm vụ tính sin một góc (đơn vị radian). Giá trị chạy trong đoạn  $[-1,1]$ .

**Cú pháp** sin(rad)

**Tham số** rad: góc ở đơn vị radian (kiểu [float](#))

**Trả về** sin của góc rad (kiểu [double](#))

Hàm này có nhiệm vụ tính tan một góc (đơn vị radian). Giá trị chạy trong khoảng từ âm vô cùng đến dương vô cùng.

**Cú pháp** tan(rad)

**Tham số** rad: góc ở đơn vị radian (kiểu [float](#))

**Trả về** tan của góc rad (kiểu [double](#))

sinh số ngẫu nhiên

Hàm [random\(\)](#) luôn trả về một số ngẫu nhiên trong phạm vi cho trước.

Giả sử mình gọi hàm này 10 lần, nó sẽ trả về 10 giá trị số nguyên ngẫu nhiên. Nếu gọi nó **n** lần, [random\(\)](#) sẽ trả về **n** số. Tuy nhiên những giá trị mà nó trả về luôn được biết trước (cố định).

Bạn hãy chạy thử chương trình sau

```
void setup(){
```

```
  Serial.begin(9600);
```

```

}
void loop(){
  Serial.println(random(100));
  delay(200);
}

```

Tôi có thể khẳng định rằng 10 giá trị "ngẫu nhiên" đầu tiên bạn nhận được là: 7, 49, 73, 58, 30, 72, 44, 78, 23, 9,... Điều này nghe có vẻ không được "ngẫu nhiên" cho lắm.

Bạn hãy thử chạy chương trình này:

```

void setup(){
  Serial.begin(9600);
  randomSeed(10);
}
void loop(){
  Serial.println(random(100));
  delay(200);
}

```

Nhận thấy rằng: chuỗi giá trị mà hàm **random()** trả về đã có sự thay đổi. Tuy nhiên chuỗi này vẫn là chuỗi cố định. Thử thay đổi tham số của lệnh **randomSeed()** từ **10** sang một số khác, bạn sẽ thấy chuỗi số trả về cũng thay đổi theo nhưng giá trị xuất ra thì vẫn cố định, dù cho bạn có bấm nút reset trên Arduino thì chuỗi số được in ra những lần sau đều y hệt như lần đầu tiên chúng được in ra. Để ý rằng tham số của hàm **random()** vẫn cố định, dĩ nhiên nếu bạn thay đổi tham số này thì chuỗi ngẫu nhiên trả về sẽ thay đổi theo, nhưng chúng cũng vẫn là một chuỗi số cố định.

Với cùng khoảng giá trị truyền vào hàm **random()**, hàm **randomSeed()** quyết định trật tự các giá trị mà **random()** trả về. Trật tự này phụ thuộc vào tham số mà ta truyền vào **randomSeed()**.

**Cú pháp** **randomSeed(number);**

Với **number** là một số nguyên bất kì.



Lưu ý: nếu bạn gọi hàm **random()** mà không chạy lệnh **randomSeed()** trước đó, chương trình sẽ mặc định chạy sẵn lệnh **randomSeed(0)** (tham số là 0).

### Ví dụ

Nếu chạy **randomSeed(0)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 7, 49, 73, 58, 30, 72, 44, 78, 23, 9, ...

Nếu chạy **randomSeed(10)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 70, 43, 1, 92, 65, 26, 40, 98, 48, 67, ...

Nếu chạy **randomSeed(-46)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 15, 50, 82, 36, 36, 37, 25, 59, 93, 74, ...

Nếu chạy **randomSeed(159)**, hàm **random(100)** sẽ trả về 10 giá trị đầu tiên là: 13, 51, 67, 38, 22, 50, 67, 73, 81, 75, ...

Nếu chạy **randomSeed(159)**, hàm **random(99)** sẽ trả về 10 giá trị đầu tiên là: 67, 42, 70, 34, 53, 6, 42, 38, 29, 64, ...

### Mẹo nhỏ

Nếu bạn cần một chuỗi số ngẫu nhiên một cách thực sự, tức là giá trị của chuỗi số trả về không thể xác định được, hãy chạy lệnh **randomSeed()** với một tham số ngẫu nhiên truyền vào.

Tham khảo chương trình sau

```
void setup() {  
  Serial.begin(9600);  
  randomSeed(analogRead(A0));  
}  
  
void loop() {  
  Serial.println(random(100));  
  delay(300);  
}
```

Ở đây tham số ngẫu nhiên truyền vào **randomSeed()** là giá trị trả về của [analogRead\(A0\)](#) với chân A0 trên Arduino không được sử dụng. Ngoài ra, bạn cũng có thể sử dụng tham số truyền vào **randomSeed()** là thời gian trên đồng hồ hệ thống.

Trả về một giá trị nguyên ngẫu nhiên trong khoảng giá trị cho trước.

**Cú pháp** random(max+1);

random(min,max+1);

Trong đó: **min** và **max** là giá trị đầu và cuối của khoảng giá trị mà *random()* trả về.

**Trả về**

Giá trị nguyên ngẫu nhiên nằm trong khoảng từ **min** đến **max**. Nếu giá trị **min** không được đưa vào thì nó được hiểu ngầm là 0.

**Ví dụ**

```
int a = random(100); //giá trị a nằm trong khoảng từ 0 đến 99
```

```
int b = random(0,11); //giá trị b nằm trong khoảng từ 0 đến 10
```

## **Nhập xuất nâng cao(*Advanced I/O*)**

tone()

**Giới thiệu**

Hàm này sẽ tạo ra một sóng vuông ở tần số được định trước (chỉ nửa chu kỳ) tại một pin digital bất kỳ (analog vẫn được). Thời hạn của quá trình tạo ra sóng âm có thể được định trước hoặc nó sẽ phát ra âm thanh liên tục cho đến khi Arduino IDE chạy hàm [noTone\(\)](#). Chân digital đó cần được kết nối tới một [buzzer](#) hoặc một [loa](#) để có thể phát được âm thanh.

Lưu ý rằng, chỉ có thể sử dụng duy nhất một hàm tone() trong cùng một thời điểm. Nếu hàm tone() đang chạy trên một pin nào đó, bây giờ bạn lại tone() thêm một lần nữa thì hàm tone() sau sẽ không có hiệu lực. Nếu bạn tone() lên pin đang được tone() thì hàm tone() sau sẽ thay đổi tần số sóng của pin đó.

Trên mạch Arduino Mega, sử dụng hàm tone() thì sẽ can thiệp đến đầu ra PWM tại các chân digital 3 và digital 11.

Hàm tone() sẽ không thể phát ra âm thanh có tần số < 31 Hz. Để biết thêm về kĩ thuật này, [hãy xem trang này](#).

**Chú ý:** Nếu bạn muốn chơi nhiều cao độ khác nhau trên nhiều pin. Thì trước khi chơi trên một pin khác thì bạn phải noTone() trên pin đang được sử dụng.

**Cú pháp** tone(pin, frequency)

tone(pin, frequency, duration)

**Tham số** pin: cổng digital / analog mà bạn muốn chơi nhạc (nói cách khác là pin được kết nối tới loa)

frequency: tần số của sóng vuông (sóng âm) - [unsigned int](#)

duration: thời gian phát nhạc, đơn vị là mili giây (tùy chọn) - [unsigned long](#)

**Trả về** không

**Ví dụ**

[Bài 12: Phát nhạc bằng Arduino với một cái loa hoặc buzzer](#)

Hàm này có nhiệm vụ kết thúc một sự kiện tone() trên một pin nào đó (đang chạy lệnh [tone\(\)](#)). Nếu không có bất kỳ hàm tone() nào đang hoạt động thì hàm này sẽ không gây bất kỳ ảnh hưởng gì đến chương trình.

**Chú ý:** Nếu bạn muốn chơi nhiều cao độ khác nhau trên nhiều pin. Thì trước khi chơi trên một pin khác thì bạn phải noTone() trên pin đang được sử dụng.

**Cú pháp** noTone(pin)

**Tham số** pin: cổng digital / analog mà bạn muốn chơi nhạc (nói cách khác là pin được kết nối tới loa)

**Trả về** không

shiftOut() có nhiệm vụ chuyển 1 [byte](#) (gồm 8 bit) ra ngoài từng bit một. Bit được chuyển đi có thể được bắt đầu từ bit nằm bên trái nhất (leftmost) hoặc từ bit nằm bên phải nhất (rightmost). Các bit này được xuất ra tại chân dataPin sau khi chân clockPin được pulsed (có mức điện thế là HIGH, sau đó bị đẩy xuống LOW).

**Lưu ý:** Nếu bạn đang giao tiếp với một thiết bị mà chân clock của nó có giá trị được thay đổi từ mức điện thế LOW lên HIGH (rising edge) khi

shiftOut, thì bạn cần chắc chắn rằng chân clockPin cần được chạy lệnh này: digitalWrite(clockPin,LOW);

**Cú pháp** shiftOut(dataPin, clockPin, bitOrder, value)

**Tham số** dataPin: pin sẽ được xuất ra tín hiệu ([int](#))

clockPin: pin dùng để xác nhận việc gửi từng bit của **dataPin** ([int](#))

bitOrder: một trong hai giá trị **MSBFIRST** hoặc **LSBFIRST**.

(Bắt đầu từ bit bên phải nhất hoặc Bắt đầu từ bit bên trái nhất)

value: dữ liệu cần được shiftOut. ([byte](#))

**Chú ý** shiftOut() chỉ xuất được dữ liệu kiểu byte. Nếu bạn muốn xuất một kiểu dữ liệu lớn hơn thì bạn phải shiftOut 2 lần (hoặc nhiều hơn), mỗi lần là 8 bit.

**Trả về** không

**Ví dụ**

[Điều khiển 8 đèn LED sáng theo ý muốn của bạn, dễ hay khó ?](#)

Đọc một xung tín hiệu digital ([HIGH/LOW](#)) và trả về chu kỳ của xung tín hiệu, tức là thời gian tín hiệu chuyển từ mức HIGH xuống LOW hoặc ngược lại (LOW -> HIGH). Một số cảm biến như cảm biến màu sắc như TCS3200D hay cảm biến siêu âm dòng HC-SRxx phải giao tiếp qua xung tín hiệu nên ta phải kết hợp giữa 2 hàm [digitalWrite\(\)](#) để xuất tín hiệu và [pulseIn\(\)](#) để đọc tín hiệu.

**Cú pháp** pulseIn(pin, value);

pulseIn(pin, value, timeout);

**Trong đó:**

**pin** là chân được chọn để đọc xung. **pin** có kiểu dữ liệu là [int](#).

Nếu đặt **value** là HIGH, hàm **pulseIn()** sẽ đợi đến khi tín hiệu đạt mức HIGH, khởi động bộ đếm thời gian. Khi tín hiệu nhảy xuống LOW, bộ đếm thời gian dừng lại. **pulseIn()** sẽ trả về thời gian tín hiệu nhảy từ mức **HIGH** xuống LOW này. Nếu đặt **value** là LOW, hàm **pulseIn()** sẽ làm ngược lại, đó là đo thời gian tín hiệu nhảy từ mức LOW lên HIGH. **value** có kiểu dữ liệu là [int](#).

Nếu tín hiệu luôn ở một mức HIGH/LOW cố định thì sau khoảng thời gian ***timeout***, hàm ***pulseIn()*** sẽ dừng bộ đếm thời gian và trả về giá trị 0. ***timeout*** được tính bằng đơn vị micro giây. Giá trị mặc định của ***timeout*** là  $60 \cdot 10^6$  tương ứng với 1 phút. Giá trị tối đa là  $180 \cdot 10^6$  tương ứng với 3 phút. ***timeout*** có kiểu dữ liệu là [unsigned long](#).

### Trả về

Một số nguyên kiểu [unsigned long](#), đơn vị là micro giây. ***pulseIn()*** trả về 0 nếu thời gian nhảy trạng thái HIGH/LOW vượt quá ***timeout***

### Ví dụ

```
int pin = 7;
unsigned long duration;
void setup() {
    Serial.begin(9600);
    pinMode(pin, INPUT);
}
void loop() {
    duration = pulseIn(pin, HIGH);
    //Hãy nối chân 7 của Arduino vào đường tín hiệu
    //bạn muốn đọc xung
    Serial.println(duration);
}
```

## Bits và Bytes

***lowByte()*** là hàm trả về byte cuối cùng (8 bit cuối cùng) của một chuỗi các bit. Một số nguyên bất kỳ cũng được xem như là một chuỗi các bit, vì bất kỳ số nguyên nào cũng có thể biểu diễn ở hệ nhị phân dưới dạng các bit "0" và "1".

**Lưu ý:** ***lowByte()*** không nhận giá trị thuộc kiểu dữ liệu số thực. Bạn sẽ gặp lỗi biên dịch nếu cố làm điều này.

**Cú pháp** ***lowByte()***([giá trị cần lấy ra 8 bit cuối]);

## Trả về [byte](#)

### Ví dụ

```
int A = lowByte(0B11110011001100); //A = 0B11001100 = 204;  
int B = lowByte(511);                //B = lowByte(0B11111111) = 255;  
int C = lowByte(5);                   //C = lowByte(0B00000101) = 0B101 = 5;
```

*highByte()* là hàm trả về một chuỗi 8 bit kè với 8 bit cuối cùng của một chuỗi các bit. Như vậy, nếu dữ liệu đưa vào một chuỗi 16bit thì *highByte()* sẽ trả về 8 bit đầu tiên, nếu dữ liệu đưa vào là một chuỗi 8bit hoặc nhỏ hơn, *highByte()* sẽ trả về giá trị 0. Một số nguyên bất kỳ cũng được xem như là một chuỗi các bit, vì bất kỳ số nguyên nào cũng có thể biểu diễn ở hệ nhị phân dưới dạng các bit "0" và "1".

### Lưu ý:

*highByte()* không nhận giá trị thuộc kiểu dữ liệu số thực. Bạn sẽ gặp lỗi biên dịch nếu cố làm điều này.

**Cú pháp** *highByte*([giá trị đưa vào]);

## Trả về [byte](#)

### Ví dụ

```
int A = highByte(0B1111111100000000); //A = 0B11111111 = 255;  
int B = highByte(0B10101010);          //B = 0  
int C = highByte(0B110000000011111111) //C = 0B00000000 = 0  
int D = highByte(1023);                 //D = highByte(0B11111111) =  
0B11 = 3
```

hàm *bitRead()* sẽ trả về giá trị tại một bit nào đó được xác định bởi người lập trình của một số nguyên.

**Cú pháp** *bitRead*(x, n)

### Tham số

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: bit cần đọc. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0

**Trả về** Giá trị của 1 bit (1 hoặc là 0)

### **Ví dụ**

`bitRead(B11110010,0); // trả về 0`

`bitRead(B11110010,1); // trả về 1`

`bitRead(B11110010,2); // trả về 0`

`//Hàm bitRead có thể viết như sau`

`B11110010 >> 0 & 1 // = 0`

`B11110010 >> 1 & 1 // = 1`

`B11110010 >> 2 & 1 // = 0`

`bitWrite()` sẽ ghi đè bit tại một vị trí xác định của số nguyên.

**Cú pháp** `bitWrite(x, n, b)`

### **Tham số**

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

b: 1 hoặc 0

**Trả về** không

### **Ví dụ**

`bitWrite(B11110010,0,1); // B11110011`

`bitWrite(B11110010,1,0); // B11110000`

`bitWrite(B11110010,2,1); // B11110110`

`//Hàm bitWrite có thể viết như sau`

`B11110010 | (1 << 0) // = B11110011`

`B11110010 & ~(1 << 1) // = B11110000`

`B11110010 | (1 << 2) // = B11110110`

`bitSet()` sẽ thay giá trị tại một bit xác định của một số nguyên thành 1.

**Cú pháp** `bitSet(x, n)`

**Tham số** x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

**Trả về không**

**Ví dụ**

```
bitSet(B11110010,0); // B11110011
```

```
bitSet(B11110010,2); // B11110110
```

//Hàm bitSet có thể viết như sau

```
B11110010 | (1 << 0) // = B11110011
```

```
B11110010 | (1 << 2) // = B11110110
```

bitClear() sẽ thay giá trị tại một bit xác định của một số nguyên thành 0.

**Cú pháp** bitClear(x, n)

**Tham số**

x: một số nguyên thuộc bất cứ kiểu số nguyên nào

n: vị trí bit cần ghi. Các bit sẽ được tính từ phải qua trái, và số thứ tự đầu tiên là số 0.

**Trả về không**

**Ví dụ**

```
bitClear(B11110010,1); // B11110000
```

//Hàm bitClear có thể viết như sau

```
B11110010 & ~(1 << 1) // = B11110000
```

Trả về một số nguyên dạng  $2^n$  (2 mũ n).

**Cú pháp** bit(n)

**Tham số** n: số nguyên

**Trả về một số nguyên**

**Ví dụ**

```
bit(0); //  $2^0 = 1$ 
```

```
bit(1); //  $2^1 = 2$ 
```

```
bit(2); //  $2^2 = 4$ 
```

```
bit(8); //  $2^8 = 256$ 
```

// cũng có thể viết như sau

```
1 << 0 // = 1
```



1 << 1 // = 2  
1 << 2 // = 4  
1 << 8 // = 256

## Ngắt (interrupt)

attachInterrupt()

### Giới thiệu

Ngắt (interrupt) là những lời gọi hàm tự động khi hệ thống sinh ra một sự kiện. Những sự kiện này được nhà sản xuất vi điều khiển thiết lập bằng phần cứng và được cấu hình trong phần mềm bằng những tên gọi cố định.

Vì ngắt hoạt động độc lập và tự sinh ra khi được cấu hình nên chương trình chính sẽ đơn giản hơn. Một ví dụ điển hình về ngắt là hàm `millis()`. Hàm này tự động chạy cùng với chương trình và trả về 1 con số tăng dần theo thời gian mặc dù chúng ta không cài đặt nó. Việc cài đặt hàm `millis()` sử dụng đến ngắt và được cấu hình tự động bên trong mã chương trình Arduino.

Vì sao cần phải dùng đến ngắt?

Ngắt giúp chương trình gọn nhẹ và xử lý nhanh hơn. Chẳng hạn, khi kiểm tra 1 nút nhấn có được nhấn hay không, thông thường bạn cần kiểm tra trạng thái nút nhấn bằng hàm `digitalRead()` trong đoạn chương trình `loop()`. Với việc sử dụng ngắt, bạn chỉ cần nối nút nhấn đến đúng chân có hỗ trợ ngắt, sau đó cài đặt ngắt sẽ sinh ra khi trạng thái nút chuyển từ HIGH->LOW. Thêm 1 tên hàm sẽ gọi khi ngắt sinh ra. Vậy là xong, biến trong đoạn chương trình ngắt sẽ cho ta biết trạng thái nút nhấn.

Số lượng các ngắt phụ thuộc vào từng dòng vi điều khiển. Với Arduino Uno bạn chỉ có 2 ngắt, Mega 2560 có 6 ngắt và Leonardo có 5 ngắt.

Board	int.0	int.1	int.2
Uno, Ethernet	2	3	

Mega2560	2	3	21
Leonardo	3	2	0

**Cú pháp** `attachInterrupt(interrupt, ISR, mode);`

### Thông số

***interrupt***: Số thứ tự của ngắt. Trên Arduino Uno, bạn có 2 ngắt với số thứ tự là 0 và 1. Ngắt số 0 nối với chân digital số 2 và ngắt số 1 nối với chân digital số 3. Muốn dùng ngắt bạn phải gắn nút nhấn hoặc cảm biến vào đúng các chân này thì mới sinh ra sự kiện ngắt. Nếu dùng ngắt số 0 mà gắn nút nhấn ở chân digital 4 thì không chạy được rồi.

***ISR***: tên hàm sẽ gọi khi có sự kiện ngắt được sinh ra.

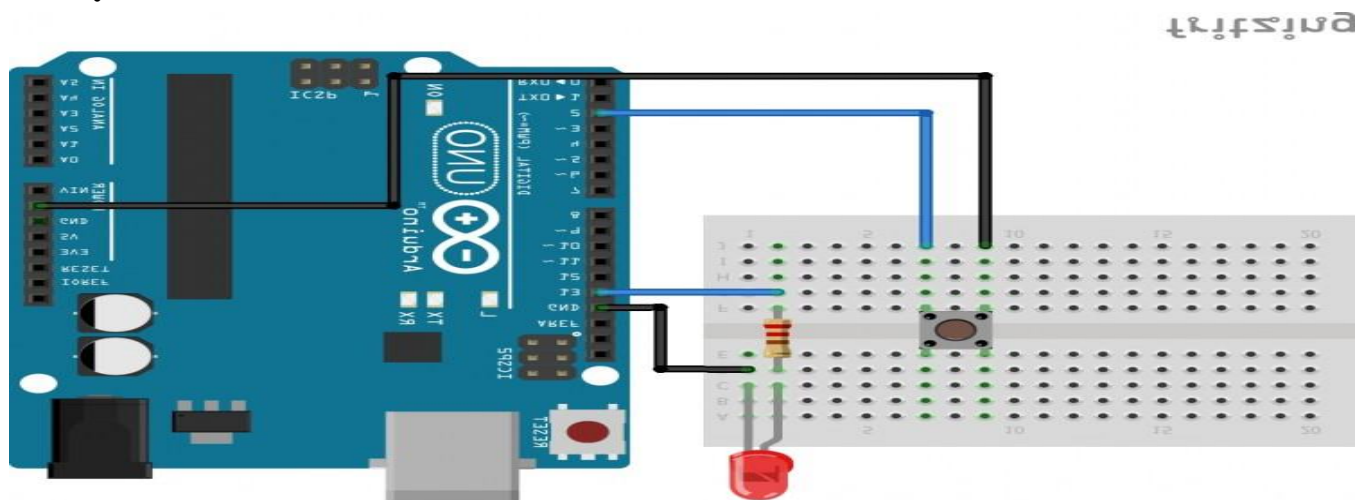
***mode***: kiểu kích hoạt ngắt, bao gồm

- **LOW**: kích hoạt liên tục khi trạng thái chân digital có mức thấp
- **HIGH**: kích hoạt liên tục khi trạng thái chân digital có mức cao.
- **RISING**: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp thấp sang mức điện áp cao.
- **FALLING**: kích hoạt khi trạng thái của chân digital chuyển từ mức điện áp cao sang mức điện áp thấp.

Lưu ý: với mode LOW và HIGH, chương trình ngắt sẽ được gọi liên tục khi chân digital còn giữ mức điện áp tương ứng.

**Trả về** không

### Ví dụ



Đoạn chương trình dưới đây sẽ làm sáng đèn led khi không nhấn nút và làm đèn led tắt đi khi người dùng nhấn nút, nếu vẫn giữ nút nhấn thì đèn led vẫn còn tắt. Sau khi thả nút nhấn, đèn led sẽ sáng trở lại.

```
int ledPin = 13;

void tatled()
{
    digitalWrite(ledPin, LOW); // tắt đèn led
}

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân
    số 2, ngắt 0
    attachInterrupt(0, tatled, LOW); // gọi hàm tatled liên tục khi còn nhấn
    nút
}

void loop()
{
    digitalWrite(ledPin, HIGH); // bật đèn led
}
```

Một ví dụ khác khi sử dụng ngắt, các bạn có thể thoát khỏi các hàm delay để xử lý 1 đoạn chương trình khác

```
int ledPin = 13;

void tatled()
{
    // tắt đèn led khi nhấn nút, nhả ra led nhấp nháy trở lại
    digitalWrite(ledPin, LOW);
}

void setup()
```

```

{
    pinMode(ledPin, OUTPUT);
    pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân
    số 2, ngắt 0
    attachInterrupt(0, tatled, LOW);
}
void loop()
{
    // đoạn chương trình này nhấp nháy led sau 500ms
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
}

```

## **Đi xa hơn**

### **Ngắt trên AVR. (Arduino sử dụng vi điều khiển AVR)**

Interrupts, thường được gọi là ngắt, là một tín hiệu khẩn cấp gửi đến bộ xử lý, yêu cầu bộ xử lý tạm ngừng tức khắc các hoạt động hiện tại để “nhảy” đến một nơi khác thực hiện một nhiệm vụ khẩn cấp nào đó, nhiệm vụ này gọi là trình phục vụ ngắt – isr (interrupt service routine). Sau khi kết thúc nhiệm vụ trong isr, bộ đếm chương trình sẽ được trả về giá trị trước đó để bộ xử lý quay về thực hiện tiếp các nhiệm vụ còn dang dở. Như vậy, ngắt có mức độ ưu tiên xử lý cao nhất, ngắt thường được dùng để xử lý các sự kiện bất ngờ nhưng không tốn quá nhiều thời gian. Các tín hiệu dẫn đến ngắt có thể xuất phát từ các thiết bị bên trong chip (ngắt báo bộ đếm timer/counter tràn, ngắt báo quá trình gửi dữ liệu bằng RS232 kết thúc...) hay do các tác nhân bên ngoài (ngắt báo có 1 button được nhấn, ngắt báo có 1 gói dữ liệu đã được nhận...).

Ngắt là một trong 2 kỹ thuật “bắt” sự kiện cơ bản là hỏi vòng (Polling) và ngắt. Hãy tưởng tượng bạn cần thiết kế một mạch điều khiển hoàn chỉnh thực hiện rất nhiều nhiệm vụ bao gồm nhận thông tin từ người

dùng qua các button hay keypad (hoặc keyboard), nhận tín hiệu từ cảm biến, xử lý thông tin, xuất tín hiệu điều khiển, hiển thị thông tin trạng thái lên các LCD...(bạn hoàn toàn có thể làm được với AVR), rõ ràng trong các nhiệm vụ này việc nhận thông tin người dùng (start, stop, setup, change,...) rất hiếm xảy ra (so với các nhiệm vụ khác) nhưng lại rất “khẩn cấp”, được ưu tiên hàng đầu. Nếu dùng Polling nghĩa là bạn cần viết 1 đoạn chương trình chuyên thăm dò trạng thái của các button (tôi tạm gọi đoạn chương trình đó là Input()) và bạn phải chèn đoạn chương trình Input() này vào rất nhiều vị trí trong chương trình chính để tránh trường hợp bỏ sót lệnh từ người dùng, điều này thật lãng phí thời gian thực thi. Giải pháp cho vấn đề này là sử dụng ngắt, bằng cách kết nối các button với đường ngắt của chip và sử dụng chương trình Input() làm trình phục vụ ngắt - isr của ngắt đó, bạn không cần phải chèn Input() trong lúc đang thực thi và vì thế không tốn thời gian cho nó, Input() chỉ được gọi khi người dùng nhấn các button. Đó là ý tưởng sử dụng ngắt. Nếu bạn chưa biết Ngắt (interrupt) là gì, vui lòng tham khảo thêm tại bài [attachInterrupt\(\)](#).

Hàm detachInterrupt() sẽ tắt các ngắt đã được kích hoạt tương ứng với thông số truyền vào. Giả sử sau khi nhấn nút bấm lần đầu tiên đèn led sẽ tắt nhưng nhấn lần thứ 2 đèn sẽ không tắt nữa. Lúc này cần dùng đến detachInterrupt() để tắt ngắt chúng ta đã tạo ra.

### **Cú pháp**

detachInterrupt(interrupt);

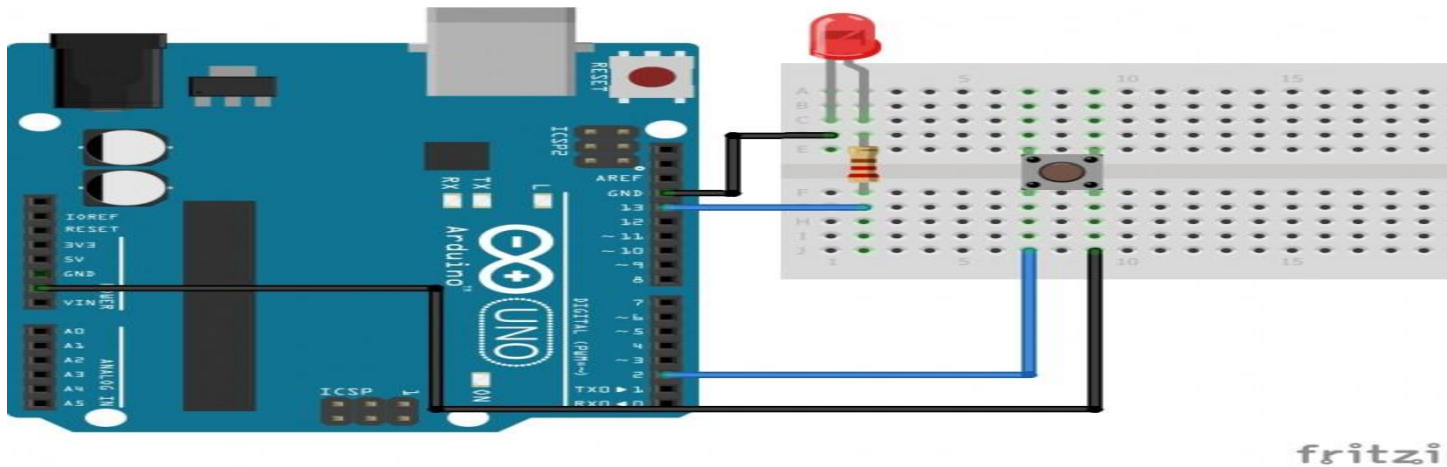
### **Thông số**

**interrupt:** số thứ tự ngắt (xem thêm ở bài [attachInterrupt\(\)](#) )

### **Trả về**

không

### **Ví dụ**



Đoạn chương trình dưới đây sẽ bật sáng đèn led và chỉ tắt nó khi nhấn lần đầu tiên, thả ra đèn sẽ sáng lại. Nếu tiếp tục nhấn nữa thì đèn vẫn sáng mà không bị tắt đi.

```
int ledPin = 13;      // đèn LED được kết nối với chân digital 13
boolean daNhan = false; // lưu giữ giá trị cho biết đã nhấn nút hay chưa
void tatled()
{
    digitalWrite(ledPin, LOW); // tắt đèn led khi còn nhấn nút
    daNhan = true; // lúc này đã nhấn nút
}
void setup()
{
    pinMode(ledPin, OUTPUT); // thiết đặt chân ledPin là OUTPUT
    pinMode(2, INPUT_PULLUP); // sử dụng điện trở kéo lên cho chân số
    2, ngắt 0
    attachInterrupt(0, tatled, LOW); // cài đặt ngắt gọi hàm tatled
}
void loop()
{
    digitalWrite(ledPin, HIGH); // bật đèn led
    if (daNhan == true)
```

```

{
    // Nếu đã nhấn nút thì tắt ngắt đi
    detachInterrupt(0);
}
}

```

Nếu bạn chưa biết Ngắt (interrupt) là gì, vui lòng tham khảo thêm tại bài [attachInterrupt\(\)](#).

Mặc định, Arduino luôn bật các ngắt nên trong phần setup(), bạn không cần gọi hàm này để bật các ngắt. Hàm interrupts() sẽ bật toàn bộ các ngắt đã được cài đặt. Nếu vì lý do nào đó bạn tắt các ngắt bằng hàm [noInterrupts\(\)](#), bạn sử dụng hàm này để bật lại các ngắt.

**Cú pháp** interrupts();

**Thông số** không

**Trả về** không

**Ví dụ**

```

void setup() { }
void loop()
{
    noInterrupts();
    // tắt các ngắt để chạy
    // đoạn chương trình yêu cầu cao về thời gian
    interrupts();
    // bật lại các ngắt, các ngắt hoạt động
    // bình thường trở lại
}

```

Nếu bạn chưa biết Ngắt (interrupt) là gì, vui lòng tham khảo thêm tại bài [attachInterrupt\(\)](#).

Khi cần chạy các đoạn chương trình yêu cầu chính xác về thời gian, bạn cần tắt các ngắt để Arduino chỉ tập trung vào xử lý các tác vụ cần thiết

và chỉ duy nhất các tác vụ này. Các ngắt chạy nền sẽ không được thực thi sau khi gọi hàm noInterrupts().

**Cú pháp** noInterrupts();

**Thông số** không


**Trả về** không

**Ví dụ**

```
void setup() { }  
void loop()  
{  
  noInterrupts();  
  // tắt các ngắt để chạy  
  // đoạn chương trình yêu cầu cao về thời gian  
  interrupts();  
  // bật lại các ngắt, các ngắt hoạt động  
  // bình thường trở lại  
}
```

**Serial** - Thư viện giao tiếp giữa các mạch Arduino dễ học nhất

Thư viện Serial được dùng trong việc giao tiếp giữa các board mạch với nhau (hoặc board mạch với máy tính hoặc với các thiết bị khác). Tất cả các mạch Arduino đều có ít nhất 1 cổng **Serial** (hay còn được gọi là UART hoặc USART). Giao tiếp Serial được thực hiện qua 2 cổng digital 0 (RX) và 1 (TX) hoặc qua cổng USB tới máy tính. Vì vậy, nếu bạn đang sử dụng các hàm của thư viện Serial này, bạn không thể sử dụng các chân digital 0 và digital 1 để làm việc khác được!

Bạn có thể sử dụng bảng Serial monitor có sẵn trong Arduino IDE để giao tiếp với Arduino qua giao thức Serial. Kích vào biểu tượng Serial Monitor () hoặc nhấn tổ hợp phím Ctrl+Shift+M để mở bảng Serial Monitor, sau đó bạn kích chuột vào bảng chọn như hình dưới để chọn baudrate giống với baudrate được dùng trong quá trình lập trình của bạn. Mặc định là 9600.





Một số mạch khác có nhiều cổng Serial hơn (Ví dụ như Mega,...), để rõ hơn, bạn vào địa chỉ [này](#).

### Mã thư viện:

#### [Serial](#)

available()

### Giới thiệu

Trả về số byte (ký tự) tối đa mà ta có thể đọc qua Serial. Các dữ liệu đến được lưu vào một bộ nhớ đệm có dung lượng 64KB.

**Cú pháp** Serial.available()

// Chỉ hoạt động trên Arduino Mega

Serial1.available()

Serial2.available()

Serial3.available()

**Tham số** Không

**Trả về** một số là số byte (ký tự) ta có thể đọc

### Ví dụ

```
int incomingByte = 0; // lưu tín hiệu đến
```

```
void setup() {
```

```
    Serial.begin(9600); // mở cổng serial với mức baudrate là 9600
```

```
}
```

```
void loop() {
```

```
    // xuất tín hiệu khi nhận được tín hiệu
```

```
    if (Serial.available() > 0) {
```

```
        // đọc các giá trị nhận được
```

```
        incomingByte = Serial.read();
```

```

        // xuất ra những gì nhận được
        Serial.print("Toi nghe duoc: ");
        Serial.println(incomingByte, DEC);
    }
}

```

### **Ví dụ dành cho Arduino Mega**

```

void setup() {
    Serial.begin(9600);
    Serial1.begin(9600);
}

void loop() {
    //đọc tín hiệu từ Serial rồi gửi qua Serial1
    if (Serial.available()) {
        int inByte = Serial.read();
        Serial1.print(inByte, BYTE);
    }
    //Đọc tín hiệu từ Serial1 rồi gửi qua Serial0
    if (Serial1.available()) {
        int inByte = Serial1.read();
        Serial.print(inByte, BYTE);
    }
}

begin()

```

### **Giới thiệu**

Khởi động một cổng Serial với một baudrate cho trước có trên Arduino. Để giao tiếp với máy tính, bạn phải dùng một trong các mức baudrate sau: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, hoặc 115200. Ngoài ra, bạn có thể thay thế mức baudrate khác những mức trên trong trường hợp giao tiếp với một mạch nào đó có sẵn mức baudrate xác định và không thay đổi được.

## Cú pháp

Serial.begin(speed)

Serial.begin(speed, config)

//Chỉ dành cho Arduino Mega

Serial1.begin(speed)

Serial2.begin(speed)

Serial3.begin(speed)

Serial1.begin(speed, config)

Serial2.begin(speed, config)

Serial3.begin(speed, config)

## Tham số

speed: bits / giây (baud) - [long](#)

config: tập hợp dữ liệu, bit chẵn lẻ, và stop bits. Những giá trị khả dụng là:

- SERIAL\_5N1
- SERIAL\_6N1
- SERIAL\_7N1
- SERIAL\_8N1 (mặc định)
- SERIAL\_5N2
- SERIAL\_6N2
- SERIAL\_7N2
- SERIAL\_8N2
- SERIAL\_5E1
- SERIAL\_6E1
- SERIAL\_7E1
- SERIAL\_8E1
- SERIAL\_5E2
- SERIAL\_6E2
- SERIAL\_7E2

- SERIAL\_8E2
- SERIAL\_5O1
- SERIAL\_6O1
- SERIAL\_7O1
- SERIAL\_8O1
- SERIAL\_5O2
- SERIAL\_6O2
- SERIAL\_7O2
- SERIAL\_8O2

**Trả về** không

**Ví dụ** void setup() {

Serial.begin(9600); // mở port ở mức 9600

}

void loop() {}

**Ví dụ trên Arduino Mega**

// Nếu bạn sử dụng cùng lúc nhiều cổng Serial

// (Serial, Serial1, Serial2, Serial3),

// thì bạn phải đặt các mức baudrate khác nhau cho nó

void setup(){

Serial.begin(9600);

Serial1.begin(38400);

Serial2.begin(19200);

Serial3.begin(4800);

Serial.println("Hello Computer");

Serial1.println("Hello Serial 1");

Serial2.println("Hello Serial 2");

Serial3.println("Hello Serial 3");

}

```
void loop() { }  
end()
```

### **Giới thiệu**

Đóng cổng Serial, để dùng các chân digital 0, 1 hoặc các chân tương tự trên (Arduino Mega) cho công tác [pinMode](#). Để bật lại, bạn dùng [Serial.begin\(\)](#).

### **Cú pháp Serial.end()**

//Đóng cổng Serial trên Arduino Mega

```
Serial1.end()
```

```
Serial2.end()
```

```
Serial3.end()
```

**Tham số** không

**Trả về** không

### **Ví dụ**

```
void setup() {  
    Serial.begin(9600); // mở port ở mức 9600  
    Serial.end();// Đóng cổng  
    Serial.println("I can't post");  
}  
void loop() { }
```

### **Giới thiệu**

Serial.find() sẽ đọc tín hiệu từ bộ nhớ đệm đến khi nào tìm được ký tự cho phép. Nó sẽ trả về true nếu tìm ra hoặc false nếu không tìm thấy hoặc hết hạn (timeout).

### **Cú pháp Serial.find(target)**

**Tham số** *target* : chuỗi cần tìm kiểu (kiểu [string](#))

**Trả về** boolean

### **Giới thiệu**

Serial.findUntil() sẽ đọc tín hiệu từ bộ nhớ đệm đến khi nào tìm được ký tự cho phép hoặc là dừng quá trình tìm đến khi nhận được ký tự thoát

(terminator). Nó sẽ trả về true nếu tìm ra hoặc false nếu không tìm thấy hoặc hết hạn (timeout).

**Cú pháp** Serial.findUntil(target,terminator)

**Tham số** *target* : chuỗi cần tìm kiểu (kiểu [string](#))

terminator: chuỗi dừng tìm (terminator)

**Trả về** boolean

### **Giới thiệu**

Hàm này có nhiệm vụ chờ đợi quá trình gửi thông tin qua Serial kết thúc rồi mới cho chạy tiếp tục chương trình.

### **Cú pháp**

Serial.flush()

//Chỉ cho Arduino Mega

Serial1.flush()

Serial2.flush()

Serial3.flush()

**Tham số** không

**Trả về** không

f (Serial)

### **Giới thiệu**

Nó có nhiệm vụ cho biết các cổng kết nối Serial đã sẵn sàng hay chưa.

**Cú pháp** //Tất cả các loại mạch Arduino

if (Serial)

//Arduino Leonardo

if (Serial1)

//Arduino Mega

if (Serial1)

if (Serial2)

if (Serial3)

**Tham số** Không

## Trả về

boolean: là true nếu cổng Serial đã được bật và bạn có thể giao tiếp qua Serial, ngược lại là false

## Ví dụ

```
void setup() {  
  //Bật Serial  
  Serial.begin(9600);  
  while (!Serial) {  
    ; //Đợi đến khi nào cổng Serial đã được bật. Chỉ cần thiết với mạch  
    Leonardo  
  }  
}  
  
void loop() {  
}
```

## Giới thiệu

Hàm này có nhiệm vụ lấy ra giá trị số thực đầu tiên có trong bộ nhớ đệm serial. Nó sẽ bỏ qua các ký tự không phải dùng để biểu diễn một số thực (ngoại trừ dấu trừ "-", dấu chấm ".", chữ "e" hoặc "E"). Nó sẽ dừng đọc khi thấy ký tự nó đang đọc không phải là một ký tự biểu diễn số thực.

**Cú pháp** Serial.parseFloat()

**Tham số** không

**Trả về** [float](#)

## Giới thiệu

Hàm này sẽ trả về số nguyên đầu tiên nó lấy được trong bộ nhớ đệm Serial. Nó sẽ dừng quá trình tìm nếu quá 1000 mili giây. Bạn có thể thay đổi thời gian chờ bằng hàm [Serial.setTimeout\(\)](#). Nếu bị dừng nó sẽ về giá trị là 0.

**Cú pháp** Serial.parseInt()

**Tham số** không

**Trả về** [int](#)

## Giới thiệu

Chúng ta sẽ sử dụng hàm này để "nhìn trộm" xem thử byte (ký tự) tiếp theo trong bộ nhớ đệm mà ta sẽ được đọc là gì. Và khi nhìn trộm, ta sẽ không xóa ký tự đó khỏi bộ nhớ đệm. Điều này rất phù hợp với việc xem trước gói tin là gì, sau đó chúng ta sẽ đọc hoặc bỏ qua.

### Cú pháp Serial.peek()

//Chỉ trên Arduino Mega

Serial1.peek()

Serial2.peek()

Serial3.peek()

### Tham số không

**Trả về** [int](#): byte đầu tiên trong bộ nhớ đệm (hoặc -1 nếu không tồn tại dữ liệu trong bộ nhớ đệm)

## Giới thiệu

Hàm này sẽ xuất dữ liệu ra cổng Serial dưới dạng chuỗi con người có thể đọc được. Hàm này có thể được sử dụng dưới nhiều dạng khác nhau. Các chữ số của một số (nguyên hoặc thực) được chuyển thành chuỗi và xuất ra màn hình. Ví dụ:

- Serial.print(78) cho ta "78"
- Serial.print(1.23456) cho ta "1.23"
- Serial.print('N') cho ta "N"
- Serial.print("Hello world.") cho ta "Hello world."

Tham số thứ 2 (có thể có hoặc không) sẽ giúp hệ thống Arduino in dữ liệu dưới dạng mà bạn muốn (thường là dùng để debug). Các giá trị hợp lệ là:

- BIN: in dữ liệu dưới dạng hệ nhị phân (hệ cơ số 2)
- DEC: in dữ liệu dưới dạng hệ thập phân (hệ cơ số 10)
- OCT: in dữ liệu dưới dạng hệ bát phân (hệ cơ số 8)
- HEX: in dữ liệu dưới dạng hệ thập lục phân (hệ cơ số 16)



- Còn đối với số thực, thì giá trị nhập vào là một số nguyên bất kỳ (âm hoặc dương), hệ thống sẽ dùng giá trị này để làm tròn số thực của bạn. Xem ví dụ để rõ hơn.
- `Serial.print(78, BIN)` cho ta "1001110"
- `Serial.print(78, OCT)` cho ta "116"
- `Serial.print(78, DEC)` cho ta "78"
- `Serial.print(78, HEX)` cho ta "4E"
- `Serial.println(1.23456, 0)` cho ta "1"
- `Serial.println(1.23456, 2)` cho ta "1.23"
- `Serial.println(1.23456, 4)` cho ta "1.2346"

**Cú pháp** `Serial.print(val)`

`Serial.print(val, format)`

**Tham số** `val`: bất kỳ giá trị ở bất kỳ kiểu dữ liệu nào

`format`: Xem ở trên

**Trả về**

`size_t`: [int](#) - Số byte (sau khi đã chuyển thành chuỗi) được gửi vào cổng Serial. Nó sẽ được trả về trước khi có bất kỳ giá trị nào được gửi đi vào Serial trong các phiên bản Arduino 1.0 trở lên.

**Ví dụ**

```
int x = 0;    // Biến
```

```
void setup() {
```

```
    Serial.begin(9600);    // Khởi động serial ở mức baudrate 9600
```

```
}
```

```
void loop() {
```

```
    // Gửi các giá trị
```

```
    Serial.print("NO FORMAT");    // in giá trị nhận
```

```
    Serial.print("\t");    // in một phím Tab
```

```
    Serial.print("DEC");
```

```

Serial.print("\t");
Serial.print("HEX");
Serial.print("\t");
Serial.print("OCT");
Serial.print("\t");
Serial.print("BIN");
Serial.print("\t");
for(x=0; x< 64; x++){ // chỉ các ký tự nằm trong bảng ASCII được
xuất ra
    //Xuất ra dưới nhiều định dạng
    Serial.print(x);    // xuất số x dưới dạng chuỗi (Giống với format :
DEC)
    Serial.print("\t"); // in một phím tab
    Serial.print(x, DEC); // số x được chuyển hệ dạng thập phân rồi xuất
dưới dạng chuỗi
    Serial.print("\t"); // in một phím tab
    Serial.print(x, HEX); // số x được chuyển hệ dạng thập lục phân rồi
xuất dưới dạng chuỗi
    Serial.print("\t"); // in một phím tab
    Serial.print(x, OCT); // số x được chuyển hệ dạng bát phân rồi xuất
dưới dạng chuỗi
    Serial.print("\t"); // prints a tab

    Serial.println(x, BIN); // số x được chuyển hệ dạng nhị phân rồi xuất
dưới dạng chuỗi
    //                                sau đó thêm một ký tự xuống dòng. xem hàm
Serial.println() để rõ!
    delay(200);    // đợi 200 mili giây
}
Serial.println(""); //Xuống dòng một lần nữa!

```

```
}
```

## Giới thiệu

Giống hệt hàm [Serial.print\(\)](#), nhưng nó sẽ gửi thêm một dấu xuống dòng sau khi gửi những gì bạn yêu cầu.

**Cú pháp**    `Serial.println(val)`

`Serial.println(val, format)`

## Tham số

val: bất kỳ giá trị ở bất kỳ kiểu dữ liệu nào.

format: Xem ở trên

## Trả về

size\_t: [int](#) - Số byte (sau khi đã chuyển thành chuỗi) được gửi vào cổng Serial. Nó sẽ được trả về trước khi có bất kỳ giá trị nào được gửi đi vào Serial trong các phiên bản Arduino 1.0 trở lên.

## Ví dụ

```
int analogValue = 0;  // biến lưu giá trị analog
void setup() {
  // mở serial với baudrate 9600
  Serial.begin(9600);
}
void loop() {
  // đọc giá trị analog trên chân A0
  analogValue = analogRead(0);
  //Xuất giá trị
  Serial.println(analogValue);    // xuất giá trị dưới dạng chuỗi
  Serial.println(analogValue, DEC); // đổi giá trị sang hệ thập phân rồi
  xuất dưới dạng chuỗi
  Serial.println(analogValue, HEX); // đổi giá trị sang hệ thập lục phân
  rồi xuất dưới dạng chuỗi
  Serial.println(analogValue, OCT); // đổi giá trị sang hệ bát phân rồi
  xuất dưới dạng chuỗi
```

```
Serial.println(analogValue, BIN); // đổi giá trị sang hệ nhị phân rồi  
xuất dưới dạng chuỗi
```

```
// dừng 1 giây để xem rồi đọc tiếp
```

```
delay(1000);
```

```
}
```

**Giới thiệu** Dùng để đọc từng ký tự trong bộ nhớ đệm của Serial.

**Cú pháp** Serial.read()

```
//Chỉ có trên Arduino Mega
```

```
Serial1.read()
```

```
Serial2.read()
```

```
Serial3.read()
```

**Tham số** không

**Trả về** [int](#) : byte đầu tiên trong bộ nhớ đệm

**Ví dụ** int incomingByte = 0; // dùng để lưu giá trị được gửi

```
void setup() {
```

```
    Serial.begin(9600); // mở serial với baudrate 9600
```

```
}
```

```
void loop() {
```

```
    // nếu còn có thể đọc được
```

```
    if (Serial.available() > 0) {
```

```
        // đọc chữ liệu
```

```
        incomingByte = Serial.read();
```

```
        // trả về những gì nhận được
```

```
        Serial.print("Toi nhan duoc: ");
```

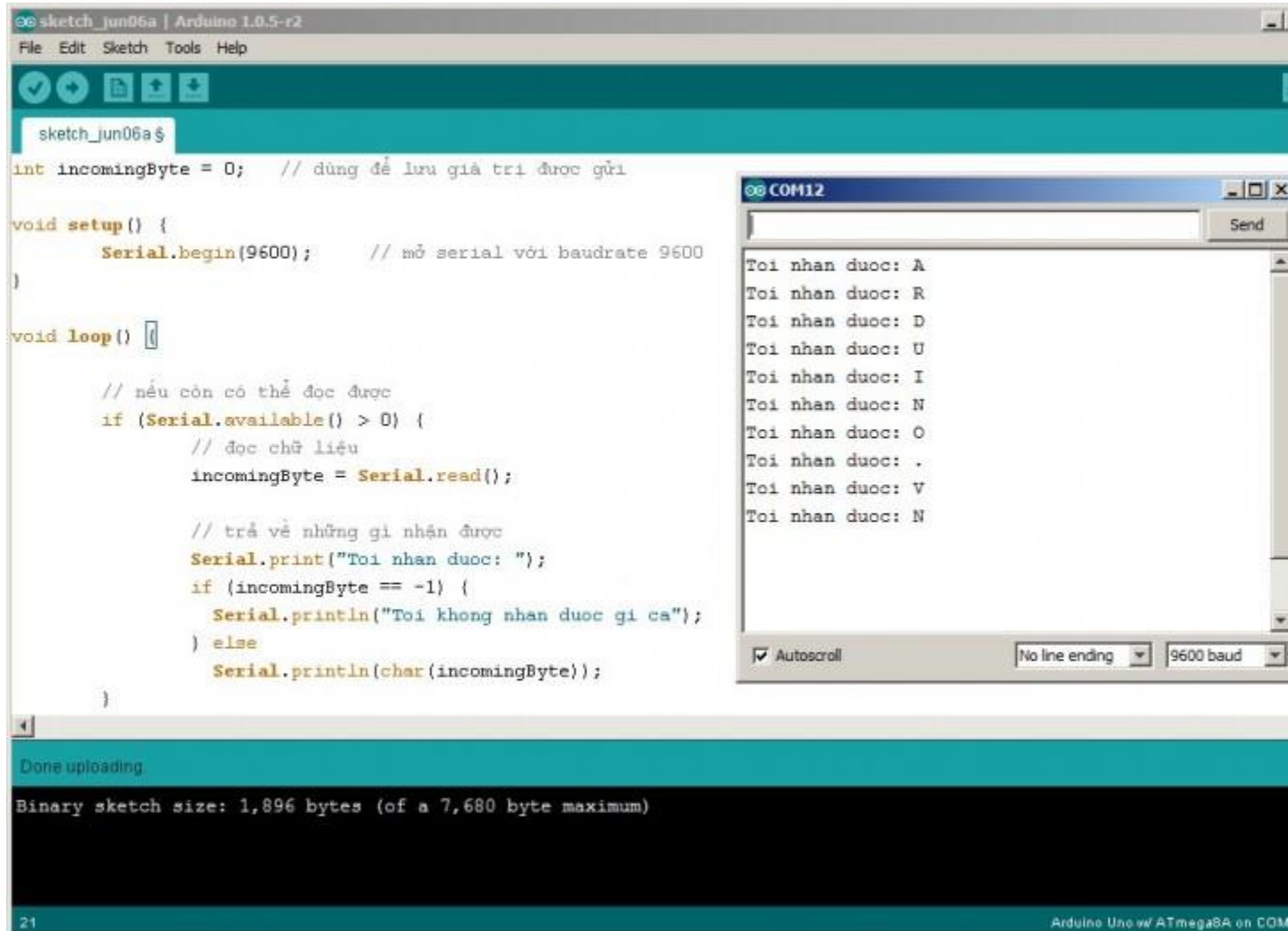
```
        if (incomingByte == -1) {
```

```
            Serial.println("Toi khong nhan duoc gi ca");
```

```

    } else
        Serial.println(char(incomingByte));
    }
}

```



**Mã thư viện:**

**Giới thiệu**

Dùng để đọc từ ký tự trong bộ nhớ đệm của `Serial`. Nó sẽ đọc cho đủ số lượng byte mà người dùng yêu cầu. Và nếu không đọc đủ thì nó sẽ dừng khi hết hạn (xem [Serial.setTimeout\(\)](#)).

**Cú pháp** `Serial.readBytes(buffer, length)`

**Tham số**

buffer: biến đệm dùng để lưu lại các byte (char[] hoặc byte[])  
length : số byte cần lưu ([int](#))

**Trả về** [byte](#)

### **Ví dụ**

```
char buffer[4]= { }; // Biến đệm lưu giá trị
```

```
void setup() {
```

```
    Serial.begin(9600);    // mở serial với baudrate 9600
```

```
}
```

```
void loop()
```

```
    // nếu còn có thể đọc được
```

```
    if (Serial.available() >= 3) {
```

```
        // đọc chữ liệu
```

```
        Serial.readBytes(buffer,3);
```

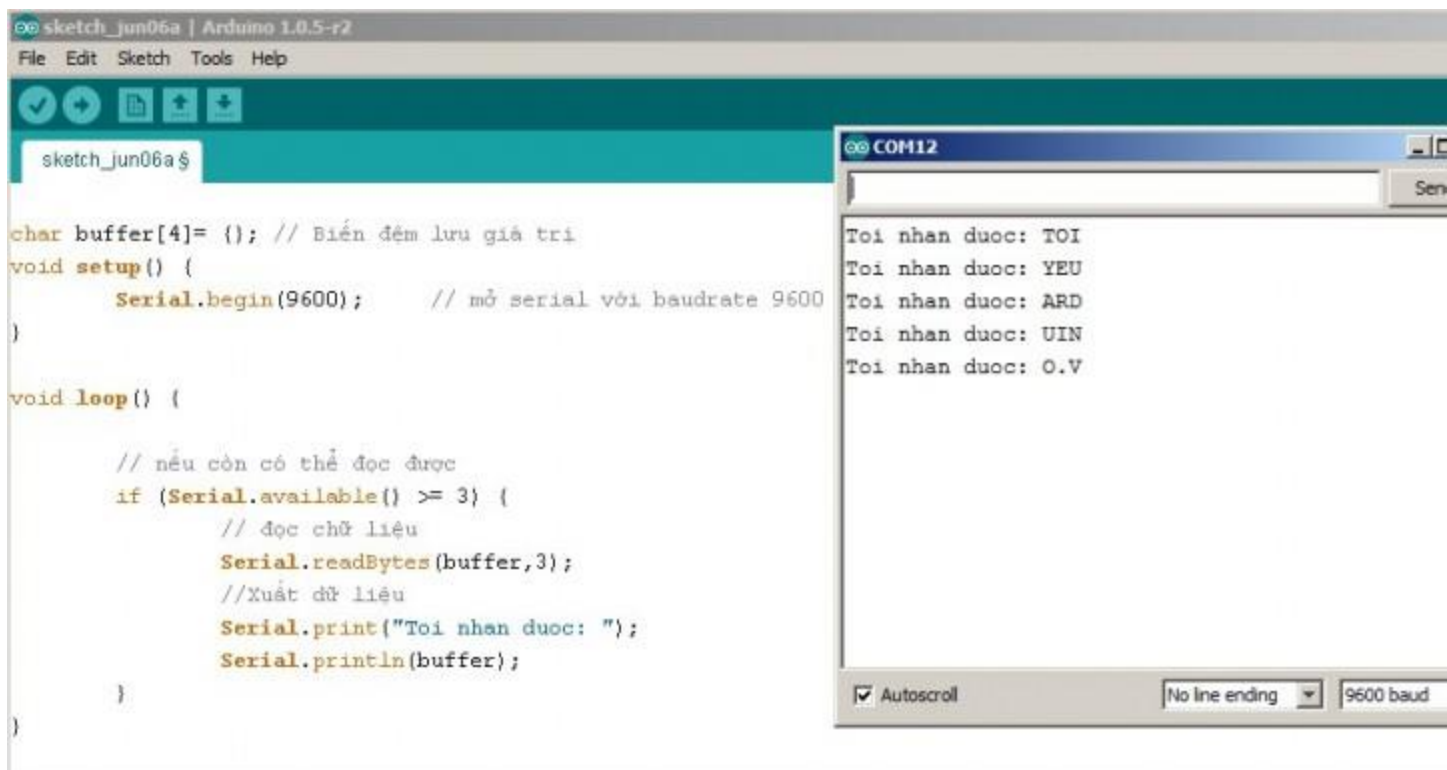
```
        //Xuất dữ liệu
```

```
        Serial.print("Toi nhan duoc: ");
```

```
        Serial.println(buffer);
```

```
    }
```

```
}
```



## Mã thư viện:

### [Serial](#)

## Giới thiệu

Có nhiệm vụ như [Serial.readBytes\(\)](#). Nhưng nó sẽ kết thúc quá trình đọc nếu gặp trùng ký tự kết thúc (kiểu [char](#))

**Cú pháp** `Serial.readBytesUntil(character, buffer, length)`

## Tham số

character: ký tự kết thúc (kiểu [char](#))

buffer: biến đếm dùng để lưu lại các byte (char[] hoặc byte[])

length : số byte cần lưu ([int](#))

**Trả về** [byte](#)

## Giới thiệu

Hàm `serialEvent()` sẽ được gọi khi nào có tín hiệu từ cổng Serial. Và sau đó bạn sẽ dùng hàm [Serial.read\(\)](#) hoặc tương tự để đọc dữ liệu.

Hiện tại thì hàm này không khả dụng trên các mạch *Esplora*, *Leonardo*, hoặc *Micro*

**Cú pháp** void serialEvent(){

//câu lệnh

}

//Chỉ có trên Arduino Mega

void serialEvent1(){

//câu lệnh

}

void serialEvent2(){

//câu lệnh

}

void serialEvent3(){

//câu lệnh

}

## **Giới thiệu**

Đặt thời gian tối đa cho việc xử lý dữ liệu Serial. Mặc định là 1000.

**Cú pháp** Serial.setTimeout(time)

**Tham số** time: thời gian chờ đợi ở đơn vị mili giây ([long](#))

**Trả về** không

## **Giới thiệu**

Gửi dữ liệu tới cổng Serial. Dữ liệu được gửi bằng kiểu [byte](#) hoặc một dãy các byte Cũng như hàm Serial.print(). Các byte được mã hóa thành các ký tự kiểu char.

**Cú pháp** Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

## **Tham số**

val: Một giá trị kiểu byte

str: Một giá trị kiểu [string](#).



buf: Kiểu byte[] (mảng byte), len: độ dài các phần tử sẽ được xuất ra Serial, sẽ được tính từ phần tử đầu tiên đến phần tử len - 1

**Trả về** [byte](#): số lượng byte được gửi đi

### **Ví dụ**

```
void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.write(45); // gửi một kiểu byte có giá trị 45
  int bytesSent = Serial.write("hello"); // gửi chuỗi "hello" và trả về số
byte đã gửi
}
```