

Bài 4: String

Chuỗi (mảng ký tự)

String - Chuỗi (mảng ký tự) là một thuật ngữ tin học chính xác dùng để chỉ *một dãy các ký tự*, đơn giản là như vậy! Một chuỗi ký tự được lưu trong bộ nhớ máy tính dưới dạng biến số. Nhờ vậy ta có thể lưu trữ tên của người dùng.

Và như bạn đã biết, máy tính chỉ có thể nhớ được những con số. Máy tính không hiểu chữ cái là gì. Vậy làm thế nào máy tính có thể nhớ được những dãy ký tự? 🤔

Biến kiểu char:

Trong phần này, chúng ta sẽ đặc biệt quan tâm đến biến kiểu char. Bạn có nhớ rằng biến kiểu char cho phép chứa các con số trong khoảng -128 và 127. 🤖



Liệu biến kiểu char có cho phép chứa những con số? Bạn cần biết rằng trong C người ta **rất hiếm khi** sử dụng chúng để làm điều đó. Bình thường, ngay cả đối với những con số thật sự nhỏ, người ta vẫn dùng int để lưu lại. Hẳn rằng tôi đã sử dụng nhiều bộ nhớ hơn so với char nhưng trong thời đại ngày nay, vấn đề bộ nhớ không còn đáng lo nữa. 😊

Thật ra biến kiểu char được tạo ra để chứa ... một ký tự! Chú ý là tôi nói rõ ràng « một ký tự ». Bộ nhớ máy tính chỉ có thể chứa những con số nên người ta đã tạo ra một bảng chuyển đổi giữa số và ký tự. Lấy ví dụ con số 65 sẽ được chuyển đổi thành chữ cái A. 🤖

Ngôn ngữ C cho phép chúng ta chuyển đổi dễ dàng giữa số và chữ cái tương ứng. Để nhận được một số ứng với chữ cái, người ta chỉ cần viết chúng giữa những dấu móc đơn, như sau: 'A'. Qua quá trình compilation, 'A' sẽ được thay thế bằng con số tương ứng.

Test thử nào:

C code:

```
int main (int argc, char *argv[ ])
{
    char letter = 'A';
    printf ("%ld\n", letter);
    return 0;
}
```

Console:

65

Chúng ta thấy ngay rằng chữ A viết hoa đã được thay bằng số 65. Tương tự như vậy B thay bằng 66, C bằng 67... Test thử với những chữ cái viết thường, giá trị của những chữ cái sẽ thay đổi. Và chữ 'a' không giống như 'A', máy tính phân biệt chữ cái viết hoa và viết thường.

Hầu hết các chữ cái thông thường được code giữa 0 và 127. Bảng chuyển đổi giữa số và chữ cái có tên là ASCII (cách đọc “át-xơ-ki”).

Trang web **AsciiTable.com** là địa chỉ khá nổi tiếng để tìm thấy bảng chuyển đổi này nhưng nó không phải là duy nhất, chúng ta có thể tìm thấy nó trên Wikipédia và một số trang web khác.

Cách hiển thị một ký tự:

Function printf, sẽ tiếp tục làm chúng ta ngạc nhiên, nó có khả năng hiển thị một chữ cái. Để làm điều đó, chúng ta cần phải sử dụng kí hiệu %c (c viết tắt của character):

C code:

```
int main (int argc, char *argv[ ])
{
    char letter = 'A';
    printf ("%c\n", letter);
    return 0;
}
```

Console:

```
A
```

Chúc mừng bạn đã biết cách hiển thị một chữ cái!

Chúng ta cũng có thể yêu cầu người dùng nhập vào một chữ cái bằng cách sử dụng %c trong scanf:

C code:

```
int main (int argc, char *argv[ ])
{
    char letter = 0;
    scanf ("%c", &letter);
    printf ("%c\n", letter);
    return 0;
}
```

Nếu tôi nhập vào đây chữ cái B, máy tính sẽ hiển thị:

Console:

```
B  
B
```

(chữ B thứ nhất là do tôi nhập từ bàn phím và chữ B thứ 2 do printf hiển thị)

Và đó là những gì bạn cần biết về biến kiểu char.



Cần nhớ:

- Type char cho phép nhập vào những giá trị từ -128 đến 127, unsigned char từ 0 đến 255.
- Có những bảng chuyển đổi cho phép máy tính chuyển từ chữ cái thành số và ngược lại.
- Chúng ta có thể sử dụng type char để chứa MỘT chữ cái.
- 'A' được chương trình dịch chuyển thành một số tương ứng (thông thường là 65). Chúng ta dùng những dấu ngoặc đơn '...' để có được giá trị của một chữ cái.

String - Chuỗi ký tự là một mảng các giá trị char!

Tiêu đề ở trên nói ra tất cả những gì tôi sẽ nói với bạn ở đây.

Một chuỗi ký tự (string) chỉ là một mảng các giá trị biến kiểu char. Đơn giản là một mảng, không hơn.

Nếu tôi tạo một mảng:

C code:

```
char string[5];
```

... nếu tôi đặt vào string[0] chữ cái 'H', string[1] chữ cái 'e'... tôi sẽ tạo được một dãy ký tự , hoặc là một văn bản (text)

Đây là biểu đồ về cách bộ nhớ máy tính lưu trữ (Bạn cần chú ý kỹ phần này vì phần sau bài học sẽ phức tạp hơn rất nhiều):

Address	Value
18000	'H'
18001	'e'
18002	'l'
18003	'l'
18004	'o'

Giống như bạn thấy, đó là một bảng chứa 5 ô bộ nhớ để trình bày chữ « Hello ». Để có được giá trị số, tôi đặt chúng vào giữa những dấu ngoặc đơn, để nói với máy tính rằng đó là một số chứ không phải ký tự. Và trong thực tế, trong bộ nhớ máy tính sẽ lưu lại giá trị số tương ứng với những chữ cái được lưu.

Vâng, nhưng bạn cần lưu ý, một dãy ký tự không chỉ chứa các chữ cái! Biểu đồ trên chưa hoàn chỉnh.

Một dãy ký tự bắt buộc phải chứa một ký tự đặc biệt ở cuối cùng, gọi là « ký tự kết thúc chuỗi ». Ký tự đó được viết là `'\0'`.

? Tại sao ta cần phải kết thúc chuỗi bằng một `'\0'` ?

Đơn giản là vì máy tính cần biết khi nào kết thúc một chuỗi ký tự. Ký tự `'\0'` cho phép giải thích: « Dừng lại, không còn gì để đọc tiếp ở đây nữa! ».

Qua đó, để lưu từ « Hello » (gồm 5 chữ cái) vào bộ nhớ, ta cần dùng một string 6 char chứ không phải là một string 5 char!

Mỗi khi bạn tạo một chuỗi ký tự, bạn cần phải nghĩ đến trước đó vị trí dự phòng để đặt ký tự '\0', đây là điều bắt buộc!

Lỗi thường gặp trong lập trình ngôn ngữ C là việc bạn quên đi ký tự này, chính tôi cũng đã mắc lỗi này khá là nhiều lần

Để hiểu rõ hơn, đây chính là biểu đồ chính xác về từ « Hello » chứa trong bộ nhớ:

Address	Value
18000	'H'
18001	'e'
18002	'l'
18003	'l'
18004	'o'
18005	'\0'

Bạn thấy đấy, chuỗi ký tự chiếm 6 ô trong bộ nhớ chứ không phải 5.

Chuỗi ký tự sẽ kết thúc bằng '\0', ký tự kết thúc chuỗi cho phép máy tính biết rằng chuỗi ký tự sẽ kết thúc ở đây.

Bạn sẽ thấy rằng ký tự ‘\0’ sẽ là một lợi thế cho chúng ta. Nhờ nó mà bạn có thể biết được độ dài của chuỗi ký tự, vì nó nằm ở vị trí kết thúc của chuỗi. Bạn có thể đưa chuỗi ký tự vào một function mà không cần phải thêm vào một biến số chỉ độ dài của chuỗi.

Việc này chỉ có tác dụng đối với những chuỗi ký tự (có nghĩa là với type char*, hoặc char[]). Đối với những dạng mảng khác, các bạn bắt buộc phải lưu lại độ lớn của chuỗi ở đâu đó.

Khai báo và khởi tạo chuỗi ký tự

Nếu ta muốn khai báo một từ « Hello », ta có thể sử dụng phương pháp buồn chán sau:

C code:

```
char string[6]; // mang string gom 6 char de luu tru H-e-l-l-o va \0
string[0] = 'H';
string[1] = 'e';
string[2] = 'l';
string[3] = 'l';
string[4] = 'o';
string[5] = '\0';
```

Phương pháp này hoạt động, bạn có thể kiểm tra lại bằng cách sử dụng printf.

Ah, tôi quên mất ở printf: bạn cần phải biết thêm một ký tự đặc biệt khác đó là %s (s như là một string).

Và đây là đoạn code hoàn chỉnh tạo và hiển thị từ « Hello »:

C code:

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[ ])
{
    char string[6]; // mang string gom 6 char de luu tru H-e-l-l-o và \0
    // Khoi tao chuoi ky tu (Ta viet tung ky tu vao bo nho)
    string[0] = 'H';
    string[1] = 'e';
    string[2] = 'l';
    string[3] = 'l';
    string[4] = 'o';
    string[5] = '\0';
    // Hien thi chuoi ky tu bang printf nho %s
    printf ("%s", string);
    return 0;
}
```

Console:

Hello

Tất cả những dòng code trên chỉ để tạo và hiển thị mỗi từ « Hello ». Khá là mệt mỏi khi phải lặp đi lặp lại việc khai báo riêng biệt từng chữ cái trong mảng string. Để khởi tạo một chuỗi ký tự, may mắn thay ta còn một phương pháp khác đơn giản hơn:

C code:

```
int main (int argc, char *argv[ ])
{
    char string[ ] = "Hello"; // Do dài của chuỗi ký tự được máy tính tự động tính toán.
    printf ("%s", string);
    return 0;
}
```

Console:

Hello

Ở dòng đầu tiên trong main, tôi tạo một biến số dạng char[]. Tôi cũng có thể viết là char*, kết quả sẽ vẫn như nhau.

Sau đó bạn sẽ điền trong ngoặc kép từ bạn muốn lưu lại, bộ dịch của C sẽ tính toán tự động độ dài cần thiết. Có nghĩa là nó sẽ đếm số chữ cái và thêm vào ký tự '\0'. Sau đó nó sẽ điền từng chữ cái của từ « Hello » vào bộ nhớ giống như cách đầu tiên chúng ta làm ở trên.

Tóm lại, đơn giản và dễ sử dụng.

Khuyết điểm: Cách này chỉ hoạt động khi khởi tạo chuỗi, bạn sẽ không thể sử dụng tiếp ở những phần sau của chương trình, bạn không thể viết:

C code:

```
string = "Hello";
```

Sau phần khởi tạo, bạn chỉ có thể lưu một từ bằng cách viết riêng biệt từng chữ cái vào bộ nhớ.

Cách lưu trữ một chữ cái bằng scanf:

Các bạn có thể nhờ người sử dụng nhập vào một từ bằng scanf, bằng cách sử dụng tiếp ký tự %s.

Vấn đề duy nhất xảy ra, bạn sẽ không biết có bao nhiêu chữ cái người dùng sẽ nhập vào. Nếu bạn yêu cầu nhập tên, có thể người dùng chỉ nhập vào Nhu (3 chữ cái), và đôi lúc người đó sẽ nhập vào Superman (nhiều chữ cái hơn).

Để làm điều này, 36 ký của tên tử là không đủ cho bạn. Đôi khi, chúng ta cần khai báo một mảng char lớn hơn, đủ lớn để có thể chứa tên của người dùng. Chúng ta sẽ tạo một char[100] để chứa tên người dùng. Việc này khiến bạn có cảm giác chúng ta đang lãng phí bộ nhớ máy tính, nhưng vấn đề về bộ nhớ này không thật sự đáng để lưu tâm (sẽ có những chương trình làm lãng phí bộ nhớ máy tính hơn rất nhiều lần như thế này, từ từ bạn sẽ thấy).

C code:

```
int main (int argc, char *argv[ ])
{
    char ten[100];
    printf ("E ku, may ten gi vay? ");
    scanf ("%s", ten);
    printf ("Hello %s, tao rat vui vi duoc gap may!", ten);
    return 0;
}
```

Console:

```
E ku, may ten gi vay? M0N1M
Hello M0N1M, tao rat vui vi duoc gap may!"
```

Và đây là phần lớn những gì phải làm để yêu cầu người dùng nhập vào một từ.

Các thao tác sử dụng trên chuỗi ký tự:

Những chuỗi ký tự sẽ được sử dụng thường xuyên. Tất cả những câu, những từ hiển thị trên màn hình máy tính bạn thấy đều được tạo bởi các array type char trong bộ nhớ máy tính, chúng hoạt động theo cách tôi vừa hướng dẫn ở trên.

Tôi sẽ giới thiệu cho bạn một số thao tác để tùy chỉnh những chuỗi ký tự, người ta đã tạo trong thư viện string.h những function cần thiết.

Tôi sẽ không hướng dẫn bạn tất cả trong bài này, sẽ rất dài và có một số function thật sự không cần thiết.

Tôi sẽ hướng dẫn bạn những cái cần thiết đủ dùng trong thời điểm hiện tại.

Hãy nhớ khai báo thư viện string.h

Mặc dù điều này là hiển nhiên nhưng tôi muốn bạn xác định rõ: khi bạn sẽ phải sử dụng một thư viện mới (string.h), bạn cần phải khai báo ở đầu file .c mà bạn cần dùng đến:

```
#include <string.h>
```

Nếu bạn không làm việc này, máy tính sẽ không biết được những function mà tôi sắp hướng dẫn cho bạn, vì chương trình không có được những prototypes cần thiết, và việc dịch chương trình sẽ không hoàn thành được.

Tóm lại, bạn đừng quên phải khai báo thư viện này mỗi khi cần dùng đến những function thao tác trên chuỗi.

Strlen: tính độ dài một chuỗi

Strlen là một function dùng để tính toán độ dài một chuỗi ký tự (không tính ký tự ‘\0’).

Bạn chỉ cần đưa vào nó parameter: chuỗi ký tự của bạn! Function này sẽ trả về độ dài của nó.

Bây giờ bạn đã biết prototype là gì vì vậy tôi sẽ đưa bạn prototype của function tôi sắp hướng dẫn. Những người lập trình xem prototype giống như “hướng dẫn sử dụng trước khi dùng” của function (và những dòng chú thích bên cạnh sẽ không bao giờ thừa).

Prototype đây:

```
size_t strlen(const char* string);
```

⚠ size_t là một type đặc biệt, nó cho biết function sẽ trả về một số tương ứng với một kích thước nào đó. Đây không phải là một type cơ bản như int, long hay char, đây là một type được “chế” thêm. Chúng ta sẽ học cách tạo ra những type mới ở những bài học sau.

Trong thời điểm hiện tại, chúng ta tạm thời hài lòng với giá trị trả về của strlen được lưu lại trong một biến số type long (máy tính sẽ tự động chuyển **size_t** thành **long**). Để rõ ràng, chính xác, bạn cần lưu lại kết quả trong một biến số type size_t, nhưng thực tế thì một biến số type long là đủ.

Function này có parameter là một type const char. Const (có nghĩa là constant, bạn nhớ chứ?) có tác dụng ngăn không cho strlen thay đổi chuỗi ký tự của bạn. Khi bạn thấy một const, bạn biết rằng giá trị của biến số đó sẽ không thể thay đổi, chỉ có thể đọc giá trị của nó.

Test thử function strlen:

C code:

```
int main (int argc, char *argv[ ])
{
    char string[ ] = "Xinchao";
    long doDaiChuoi = 0;
    // giá trị do dài của chuỗi sẽ được lưu lại trong biến số doDaiChuoi
    doDaiChuoi = strlen (string);
    // hiển thị do dài chuỗi
    printf ("Chuoi %s có độ dài %ld kí tự", string, doDaiChuoi);
    return 0;
}
```

Console:

```
Chuoi Xinchao có độ dài 7 kí tự
```

Function strlen này được viết dễ dàng bằng cách sử dụng vòng lặp trên mảng kiểu char, và nó sẽ dừng lại khi gặp ký tự kết thúc chuỗi ‘\0’. Nó sẽ lần lượt truy cập vào các ô có chứa từng ký tự của chuỗi, số lượt truy cập sẽ tăng dần sau mỗi vòng lặp, và kết quả sẽ trả về giá trị cuối cùng của những lượt truy cập này.

Tôi sẽ viết function strlen của riêng mình. Việc này sẽ giúp bạn hiểu rõ hơn nó hoạt động như thế nào:

C code:

```
long doDaiChuoi(const char* string);

int main (int argc, char *argv[ ])
{
    char string[ ] = "Hello";
    long doDai = 0;
    doDai = doDaiChuoi(string);
    printf ("chuoi %s co do dai %ld ki tu", string, doDai);
    return 0;
}

long doDaiChuoi (const char* string)
{
    long soLuongKiTu = 0;
    char kiTuHienTai = 0;
    do
    {
        kiTuHienTai = string[soLuongKiTu];
        soLuongKiTu++;
    }
    while (kiTuHienTai != '\0'); // Vòng lặp tiếp tục nếu ki tu hiện tại không phải là \0
    soLuongKiTu--; // Do dài chuỗi giảm đi 1 vì ta không tính \0

    return soLuongKiTu;
}
```

Giải thích

1. Function *doDaiChuoi* tạo vòng lặp trên mảng string. Nó sẽ lưu lại từng ký tự trong biến *kiTuHienTai*. Khi *kiTuHienTai* tiến đến \0, vòng lặp sẽ dừng lại.
2. Tại mỗi vòng lặp, độ lớn sẽ tăng lên 1 sau mỗi lần truy cập ô chứa ký tự.
3. Khi kết thúc vòng lặp, số lượng ký tự sẽ bớt đi 1. Điều này có nghĩa là ta không tính ký tự kết thúc chuỗi '\0'.
4. Cuối cùng, kết quả sẽ được trả về *soLuongKiTu*.
5. Trò chơi kết thúc.

Strcpy: sao chép chuỗi này vào chuỗi khác

Function strcpy(có thể hiểu là « string copy ») cho phép sao chép một chuỗi ký tự này đặt vào trong một chuỗi ký tự khác.

Prototype của nó là:

```
char* strcpy(char* copyString, const char* stringCopy);
```

function này nhận 2 parameter:

copyString: là một pointer char* (mảng char). Chuỗi ký tự sẽ được chép vào trong mảng này.

stringCopy: là một pointer của một mảng char khác. Chuỗi ký tự này sẽ được dùng để chép vào copyString.

Function trả về pointer của copyString cũng không hữu dụng lắm. Thực tế, ta không cần sử dụng kết quả function này trả về. Cùng test thử nhé.

C code:

```
int main (int argc, char *argv[ ])
{
    /* Chung ta khai bao bien "string" kieu char trong do co chua 1 chuoi ky tu va mot bien "copy"
    voi kich thuc 100 ky tu de bao dam co du cho trong */

    char string[ ] = "Text", copy[100] = {0};

    strcpy(copy, string); // Chung ta se sao chep nhung ky tu tu "string" sang "copy"
    // Neu khong co gi sai sot thi "copy" bay gio se giong nhu "string"

    printf ("string is : %s\n", string);
    printf ("copy is : %s\n", copy);
    return 0;
}
```

Console:

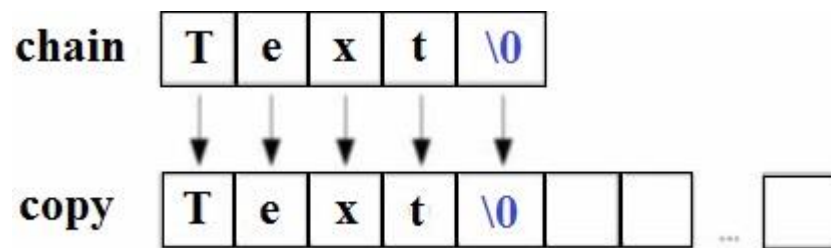
```
string is : Text
copy is : Text
```

Kết quả của string là « Text ». Điều đó là bình thường.

Nhưng, biến số copy cũng có kết quả tương tự, ban đầu biến số này không có giá trị nào, sau đó nó nhận nội dung của string. Vậy, string được sao chép lại vào trong copy.

Bạn cần chắc rằng độ lớn của chuỗi copy có đủ để nhận nội dung của string. Nếu trong ví dụ ở trên, tôi khai báo copy[4] (sẽ không đủ để chứa ký tự \0), function strcpy sẽ vượt giới hạn bộ nhớ, điều này sẽ làm chương trình bạn dừng lại. Bạn cần tránh điều này.

Biểu đồ sẽ như sau:



Mỗi ký tự trong string sẽ được điền vào copy.

Phần sau của copy còn nhiều ô nhớ không dùng đến, vì ở trên, tôi khai báo copy có độ dài là 100, nhưng trong ví dụ này, 5 là đủ để sử dụng. Lợi ích của việc tạo ra một mảng lớn hơn là để có thể sử dụng cho nhiều trường hợp khác nhau, có thể có một số chuỗi có độ dài lớn hơn trong phần sau của chương trình.

strcat: ghép nối 2 chuỗi

function này có tác dụng thêm nội dung một chuỗi phía sau một chuỗi khác. Gọi là concatenation. (sự xâu chuỗi)

Nếu ta có:

```
string1 = "Hello "
```

```
string2 = "M0N1M"
```

Nếu tôi nối string2 vào string1, string1 sẽ thành « Hello M0N1M ». Còn

string2 sẽ không thay đổi, string2 vẫn luôn là « M0N1M ». Chỉ mỗi string1 thay đổi.

Đó là cách strcat hoạt động, và đây là prototype của nó:

```
char* strcat(char* string1, const char* string2);
```

Như bạn thấy, string2 không thể thay đổi vì nó được định nghĩa là một constant trong prototype của function.

Function trả về pointer của string1, giống như strcpy, không có giá trị sử dụng nhiều nên ta có thể không cần quan tâm đến kết quả nó trả về.

Function thêm vào string1 nội dung của string2. Bạn có thể hiểu rõ hơn qua đoạn code sau:

C code:

```
int main (int argc, char *argv[ ])
{
    /* Chúng ta sẽ tạo ra 2 mảng ký tự, nhớ rằng string1 phải đủ lớn để chứa
    được những ký tự của string2. Nếu không chương trình sẽ báo lỗi. */

    char string1[100] = "Hello ", string2[] = "M0N1M";

    strcat (string1, string2); // Những ký tự của string2 sẽ được nối tiếp vào string1
    // Nếu mọi thứ diễn ra tốt đẹp thì kết quả string1 sẽ là "Hello M0N1M"

    printf ("string1 is : %s\n", string1);
    // string2 vẫn không bị thay đổi :

    printf ("string2 is always : %s\n", string2);
    return 0;
}
```

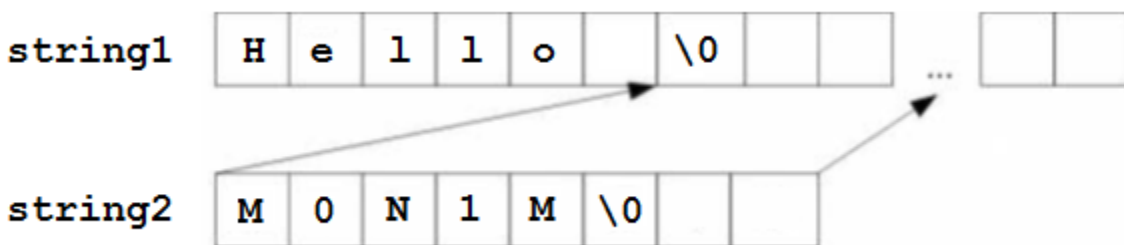
Console:

```
string1 is : Hello M0N1M
string2 is always : M0N1M
```

Cần chắc chắn là string1 phải đủ lớn để chứa thêm nội dung của string2, nếu không bạn sẽ vượt qua giới hạn bộ nhớ cho phép, điều đó sẽ khiến chương trình dừng lại.

Vì vậy tôi khai báo string1 với độ lớn là 100. Trong string2, tôi để máy tính tự tính độ lớn của chuỗi (tôi không cần phải suy nghĩ nhiều về điều này) vì chuỗi này không bị thay đổi. Ta không cần thiết phải khai báo.

Biểu đồ sẽ như sau:



Mảng của string2 sẽ được thêm vào phía sau của string1 (nó sẽ chiếm thêm một vài ô bộ nhớ)

Ký tự '\0' của chuỗi ở string1 sẽ bị xóa đi (hay được thay thế bằng M của M0N1M). Thực tế, không thể điền '\0' vào giữa chuỗi nếu không nó sẽ cắt chuỗi ra làm 2 phần! ta chỉ đặt '\0' vào vị trí cuối của chuỗi, khi chuỗi kết thúc.

Strcmp: so sánh độ dài 2 chuỗi

Function Strcmp dùng để so sánh độ dài 2 chuỗi với nhau.

Và đây là prototype:

C code:

```
int strcmp(const char* string1, const char* string2);
```

Những biến số của string1 và string2 được so sánh với nhau. Như bạn thấy, không có chuỗi nào bị thay đổi vì chúng được khai báo như những constants.

Kết quả trả về của function này cần được giữ lại.

Thực tế, strcmp trả về:

- Giá trị 0 nếu hai chuỗi giống nhau.
- Một giá trị khác 0 (lớn hơn hay nhỏ hơn 0) nếu hai chuỗi khác nhau.

Về logic, function trả về 1 nếu 2 chuỗi bằng nhau để nói là « TRUE » thì hợp lí hơn (những boolean). Nhưng function này không do tôi viết ra...

Nói rõ hơn, function sẽ so sánh giá trị của từng ký tự của mỗi chuỗi với nhau. Nếu tất cả những ký tự giống nhau, nó sẽ trả về 0.

Nếu các ký tự của string1 lớn hơn string2, nó sẽ trả về một số dương. Ngược lại, function sẽ trả về một số âm.

Trong thực tiễn, người ta thường dùng strcmp để so sánh 2 chuỗi nếu chúng giống nhau.

Đây là đoạn mã để test.

C code:

```
int main (int argc, char *argv[ ])
{
    char string1[ ] = "Text for test", string2[ ] = "Text for test";
    if (strcmp(string1, string2) == 0) // Neu 2 chuoi giiong nhau
    {
        printf ("Hai chuoi giiong nhau!\n");
    }
    else
    {
        printf ("Hai chuoi khac nhau!\n");
    }
    return 0;
}
```

Console:

```
Hai chuoi giiong nhau!
```

Nếu hai chuỗi giống nhau, kết quả sẽ trả về 0.

Lưu ý thêm là tôi có thể lưu kết quả của strcmp lại bằng một biến số kiểu int. Tuy nhiên chúng ta không bắt buộc phải làm điều này, bạn có thể dùng trực tiếp if như tôi đã làm.

Chúng ta sẽ không nói nhiều hơn về function này. Nó khá đơn giản để sử dụng, và điều bạn cần nhớ đó là giá trị 0 có nghĩa là « giống nhau » và một giá trị khác 0 có nghĩa là « khác nhau ».

Đây chính là nguyên nhân lỗi thường xuất hiện khi sử dụng function này.

Strchr: tìm kiếm một ký tự

Chức năng của function strchr là tìm kiếm một ký tự trong chuỗi.

Prototype của function strchr đây:

```
char* strchr(const char* string, int characterSearch);
```

function nhận 2 parameters:

- *string*: chúng ta sẽ tìm kiếm ký tự trong biến này.
- *characterSearch*: ký tự bạn muốn tìm kiếm trong biến string.

Bạn thấy rằng characterSearch là biến kiểu int chứ không phải char. Điều này cũng bình thường thôi vì về cơ bản, ký tự chính là số.

Tuy nhiên, người ta thường dùng char hơn là int để chứa một ký tự trong bộ nhớ.

Function sẽ trả về pointer của chữ cái đầu tiên tìm thấy, có nghĩa là trả về địa chỉ của ký tự đó trong bộ nhớ. Kết quả sẽ trả về NULL nếu không tìm thấy ký tự bạn muốn tìm.

Trong ví dụ sau, tôi sẽ lưu pointer này trong subString:

C code:

```
int main (int argc, char *argv[ ])
{
    char string[ ] = "Text for test", *subString = NULL;
    subString = strchr(string, 'f');
    if (subString != NULL) // NULL là ko tìm thấy, vậy dòng này nghĩa là “neu ta tìm dc gi do”
    {
        printf ("Bat dau tu ky tu dau tien duoc tim thay, string duoc in ra la : %s \n", subString);
    }
    return 0;
}
```

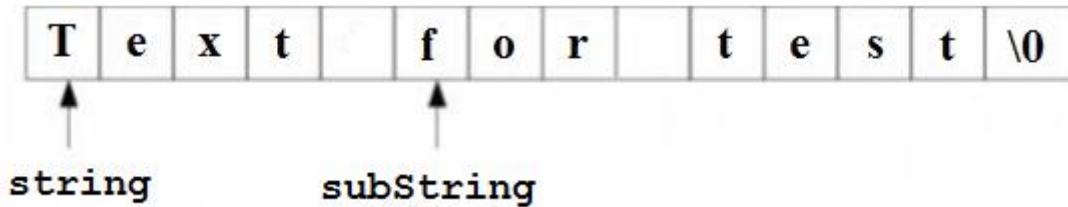
Console:

```
Bat dau tu ky tu dau tien duoc tim thay, string duoc in ra la : for test
```

Bạn hiểu điều gì đã xảy ra ở đây không? Có một sự khác biệt nhỏ ở đây .

Thực tế, subString cũng là một pointer như string. Khác biệt ở chỗ string chỉ vào chữ cái đầu tiên (T viết hoa), còn subString chỉ vào chữ 'f' đầu tiên được tìm thấy trong string.

Biểu đồ sau chỉ ra cho bạn thấy nó hoạt động như thế nào :



String bắt đầu ở ký tự đầu ('T' viết hoa) còn subString thì chỉ vào ký tự 'f'

Khi tôi viết một printf lên subString, thông thường nó sẽ hiển thị kết quả là «for test». Function printf sẽ hiển thị tất cả những chữ cái nó gặp ('f', 'o', 'r', ' ', 't', 'e', 's', 't') đến khi gặp ký tự \0 để thông báo là string kết thúc ở đây.

Biến thể:

Ngoài ra còn có một function **strrchr** với cách hoạt động hoàn toàn giống như **strchr**, khác ở chỗ nó sẽ chỉ vào chữ cái **cuối cùng** được tìm thấy trong string thay vì chữ cái **đầu tiên**.

Lưu ý:

- Function strchr tìm kiếm và chỉ vào ký tự đầu tiên nó tìm thấy trong chuỗi.
- Function strrchr tìm kiếm và chỉ vào ký tự cuối cùng nó tìm thấy trong chuỗi.

VD: Nếu bạn muốn tìm ký tự 'r' trong chuỗi Program:

- Khi dùng strchr: hàm sẽ chỉ vào ký tự 'r' đầu tiên giữa ký tự 'P' và 'o'.
- Khi dùng strrchr: hàm sẽ chỉ vào ký tự 'r' cuối cùng giữa ký tự 'g' và 'a'.

Strpbrk: chữ cái đầu tiên trong danh sách.

Function này có cách hoạt động khá giống với function trước. Nó sẽ tìm một trong những chữ cái trong danh sách bạn cho dưới dạng một dãy ký tự, ngược lại với strchr chỉ có thể tìm kiếm mỗi lần duy nhất một ký tự.

Ví dụ, nếu danh sách tìm kiếm đưa vào có dạng « xfs » cho dãy ký tự « Text for test », Function sẽ cho kết quả là một pointer chỉ vào chữ cái đầu tiên tìm được của một trong những chữ cái trong danh sách. Nói rõ hơn, chữ cái đầu tiên trong « xfs » nó tìm thấy trong « Text for test » là x, nên strpbrk sẽ trả về pointer lên 'x'.

Prototype của function này là:

```
char* strpbrk (const char* string, const char* charactersSearch);
```

Test thử xem nào:

C code:

```
int main (int argc, char *argv[ ])
{
    char *subString;
    // Chúng ta tìm kiếm sự xuất hiện đầu tiên của 1 trong 3 ký tự x, f hoặc s trong "Text for test"

    subString = strpbrk("Text for test", "xfs");

    if (subString != NULL)
    {
        printf ("Sau khi tìm một trong ba ký tự: x, f, s trong "Text for test"\n");
        printf ("Bắt đầu từ ký tự đầu tiên được tìm thấy.\n");
        printf ("Biên string được in ra là: %s", subString);
    }
    return 0;
}
```

Console:

```
Sau khi tìm một trong ba ký tự: x, f, s trong "Text for test"
Bắt đầu từ một trong những ký tự đầu tiên được tìm thấy.
Biên string được in ra là: xt for test
```

Trong ví dụ này, tôi đưa trực tiếp các giá trị vào hàm function (trong ngoặc kép). Chúng ta không bị bắt buộc phải tạo một biến số, việc viết trực tiếp như thế này khá tiện lợi.

Các bạn phải lưu ý những cách viết sau:

Nếu các bạn sử dụng ngoặc kép “ ” có nghĩa là chuỗi (mảng ký tự).

Nếu các bạn sử dụng móc đơn ‘ ’ có nghĩa là ký tự

Strstr: tìm kiếm chuỗi ký tự trong một chuỗi ký tự khác.

Function này sẽ giúp bạn *tìm kiếm chuỗi ký tự đầu tiên* tìm thấy trong một chuỗi ký tự khác.

Prototype của nó là:

```
char* strstr(const char* string, const char* stringSearch);
```

Prototype của function này khá giống với **strpbrk**, nhưng chú ý đừng nhầm lẫn, **strpbrk** tìm kiếm một chữ cái trong danh sách, còn **strstr** tìm kiếm nguyên cả dãy ký tự

Ví dụ:


C code:

```
int main (int argc, char *argv[ ])
{
    char *subString;
    // Chúng ta sẽ tìm kiếm chuỗi "test" trong "Text for test" :
    subString = strstr("Text for test", "test");
    if (subString != NULL)
    {
        printf ("Chuỗi ký tự đầu tiên mà bạn muốn tìm trong chuỗi Text for test là: %s\n", subString);
    }
    return 0;
}
```

Chuỗi ký tự đầu tiên mà bạn muốn tìm trong chuỗi Text for test là : test

Function strstr tìm kiếm dãy ký tự « test » trong “Test de test”.

Và trả lại kết quả, giống như những function khác, một pointer tại vị trí nó tìm thấy. Kết quả là NULL nếu nó không tìm thấy gì.

 Hãy nhớ kiểm tra trường hợp function tìm kiếm không trả về kết quả NULL. Nếu bạn không làm điều này trước, khi bạn muốn hiển thị một dãy ký tự có pointer là NULL, chương trình sẽ bị lỗi. Hệ điều hành sẽ dừng chương trình của bạn ngay tức khắc vì nó cố gắng xâm nhập vào một địa chỉ NULL mà nó không được phép.

Trong vd trên, tôi hài lòng với kết quả mà function này đã trả về. Trong thực tiễn, việc này thật sự **không cần thiết**. Bạn chỉ cần viết một đoạn code `if (result != NULL)` để biết rằng việc tìm kiếm tìm được một thứ gì đó. Trường hợp không tìm thấy gì, hiển thị thông báo “Không tìm được kết quả cần tìm”.

Tất cả đều tùy theo chương trình bạn viết, nhưng trong mọi trường hợp, những function này là cơ bản để bạn thực hiện các thao tác tìm kiếm liên quan tới văn bản.

sprintf: viết trong một chuỗi

Khác với các function trước đó, function này được tìm thấy trong **stdio.h**.

Tên của function này khiến bạn nhớ đến thứ gì quen thuộc.

Function này khá giống với function printf mà bạn đã được biết, nhưng thay vì in ra màn hình, sprintf sẽ viết vào một dãy ký tự! Bạn để ý có thể thấy tên của function này bắt đầu bằng “s” của từ “string”

Đây là một function rất tiện lợi để tạo ra một chuỗi, sau đây là một ví dụ nhỏ:

C code:

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[ ])
{
    char string[100];
    long age = 18;
    // Chương trình sẽ viết "You are 18 years old !"
    sprintf (string, "You are %ld years old !", age);
    // Chúng ta sẽ in chuỗi ký tự này ra màn hình để đảm bảo chương trình hiển thị đúng
    printf ("%s", string);
    return 0;
}
```

Console:

```
You are 18 years old !
```

Nó được sử dụng tương tự như printf, chỉ khác ở điểm bạn phải thêm vào một parameter ngay vị trí đầu tiên, đó là một pointer chỉ đến chuỗi bạn cần viết vào.

Trong ví dụ của tôi, tôi viết trong chuỗi “You are %ld years old !”, tại đó %ld sẽ được thay thế bằng giá trị của biến số age. Tất cả các nguyên tắc của printf đều có ở đây, bạn có thể sử dụng %s để điền vào một chuỗi ký tự khác vào đây.

Như thường lệ, hãy kiểm tra trước việc chuỗi của bạn có đủ lớn để nhận những ký tự mà sprintf sẽ gửi vào.

Nếu không, chúuuuuuu... Bùm

Phải nói là các thao tác trên chuỗi ký tự trong C cần được thi hành một cách rất tỉ mỉ.

Bạn cần biết rằng tôi cũng không thể biết được tất cả những function có trong string.h. Tôi cũng không yêu cầu bạn phải học thuộc lòng. Nhưng bạn cần phải biết cách dãy ký tự hoạt động với \0 và những điều ở trên.

Tổng kết !!!

- Máy tính không thể làm việc với các ký tự, nó chỉ biết đến những con số. Chúng ta có những con số để giải quyết vấn đề liên quan đến những ký tự trong bảng chữ cái của mình bằng bảng mã tên là ASCII.
- Mỗi biến kiểu char chỉ có thể chứa một và chỉ một ký tự duy nhất. Chúng thường được lưu lại tại một ô địa chỉ bộ nhớ ngẫu nhiên trên máy tính, máy tính sẽ tự động sắp xếp và biên dịch những biến này.
- Để có thể tạo ra một từ hoặc một cụm từ chúng ta phải tạo ra một chuỗi. Trong trường hợp này, chúng ta sẽ sử dụng mảng ký tự.
- Tất cả các chuỗi đều được kết thúc bởi một ký tự đặc biệt, đó là ký tự kết thúc chuỗi ‘\0’
- Có rất nhiều functions dùng để xử lý chuỗi đã được viết sẵn trong thư viện string.h. Vì vậy đừng quên khai báo thư viện này ở đầu chương trình trước khi bạn muốn thao tác với chuỗi ký tự nhé.

Bạn cần nhớ rằng ngôn ngữ C có bậc “tương đối thấp”, có nghĩa là các thao tác của bạn rất gần với cách hoạt động của máy tính.

Lợi ích khác là hiện giờ bạn đã biết phương thức hoạt động của máy tính trên chuỗi ký tự. Những kiến thức tôi dạy bạn ở đây sẽ phát huy tác dụng trong tương lai, tôi có thể chắc đảm bảo với bạn. Ngược lại, việc lập trình trên Java hay trên Basic không cần thiết phải hiểu cách thức hoạt động của máy tính, bạn hiện giờ đã bắt đầu hiểu làm thế nào máy tính hoạt động, điều này theo tôi là rất quan trọng.

Nghe có vẻ hay đấy nhưng có một điều đáng ngại là chương này hơi phức tạp. Bạn phải dự đoán trước độ lớn của array, nghĩ đến lưu lại \0... Các thao tác trên dãy ký tự không dễ dàng nắm vững bởi người mới bắt đầu, phải cần thêm một ít thực hành để có thể đạt được.

Nói về phương diện thực hành một cách chính xác thì tôi có công việc dành cho bạn.

Tôi khuyến khích bạn cố gắng thực hành thật nhiều. Còn điều gì tốt hơn bằng cách làm việc trên những chuỗi ký tự? Nếu chưa đủ phê, hãy cùng lúc làm việc trên chuỗi ký tự, mảng và pointer...

Và đây là việc tôi muốn bạn làm: Bạn vừa học một số functions trong thư viện string.h, nhưng bạn đã có đủ khả năng để tự viết lại các function cho riêng mình.

(?)Nhưng có cần thiết ko? Nếu các function đã được viết rồi, tại sao phải phí công viết lại?

Thật sự thì không cần thiết tí nào, và trong tương lai chắc chắn bạn sẽ sử dụng các function trong string.h chứ không phải những function của riêng bạn. Nhưng việc này sẽ giúp bạn luyện tập, tôi thấy rằng đây là một bài tập khá hay. Tôi đã chỉ cho bạn cách hoạt động của function strlen, điều đó có thể giúp bạn thực hiện các function khác.

Nhưng, đừng cố gắng viết lại những function như sprintf, đây là một function với độ phức tạp tương đối cao. Hãy tạm chấp nhận với các function có trong string.h.

Nếu bạn bị kẹt ở đâu đó, đừng ngại đặt câu hỏi trên các diễn đàn.

Nào, làm việc thôi!
