Xâu ký tự trong C++

Khái niệm

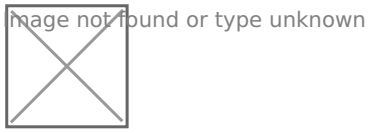
Xâu kí tự cho phép chúng ta biểu diễn các chuỗi kí tự như là các từ, câu, đoạn văn bản.

Trong C++ không có kiểu dữ liệu *cơ bản* để lưu các xâu kí tự. Để có thể thỏa mãn nhu cầu này, người ta sử dụng mảng có kiểu **`char`**. Hãy nhớ rằng kiểu dữ liệu này (**`char`**) chỉ có thể lưu trữ một kí tự đơn, bởi vậy nó được dùng để tạo ra xâu của các kí tự đơn.

Ví dụ, mảng sau (hay là xâu kí tự):

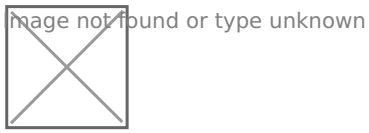
```
char jenny [20];
```

có thể lưu một xâu kí tự với độ dài cực đại là 20 kí tự. Bạn có thể tưởng tượng nó như sau:



Kích thước cực đại này không cần phải luôn luôn dùng đến. Ví dụ, **`jenny`** có thể lưu xâu `"Hello"` hay `"Merry christmas"`. Vì các mảng kí tự có thể lưu các xâu kí tự ngắn hơn độ dài của nó, trong C++ đã có một quy ước để kết thúc một nội dung của một xâu kí tự bằng một kí tự `null`, có thể được viết là `'\0'`.

Chúng ta có thể biểu diễn **`jenny`** (một mảng có 20 phần tử kiểu **`char`**) khi lưu trữ xâu kí tự `"Hello"` và `"Merry Christmas"` theo cách sau:



Chú ý rằng sau nội dung của xâu, một kí tự null (**`'\0'`**) được dùng để báo hiệu kết thúc xâu. Những ô màu xám biểu diễn những giá trị không xác định.

Khởi tạo các xâu kí tự

Vì những xâu kí tự là những mảng bình thường nên chúng cũng như các mảng khác. Ví dụ, nếu chúng ta muốn khởi tạo một xâu kí tự với những giá trị xác định chúng ta có thể làm điều đó tương tự như với các mảng khác:

```
char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

Tuy nhiên, chúng ta có thể khởi tạo giá trị cho một xâu kí tự bằng cách khác: sử dụng các hằng xâu kí tự.

Trong các biểu thức chúng ta đã sử dụng trong các ví dụ trong các chương trước các hằng xâu kí tự để xuất hiện vài lần. Chúng được biểu diễn trong cặp ngoặc kép ("), ví dụ:

```
"the result is: "
```

là một hằng xâu kí tự chúng ta sử dụng ở một số chỗ.

Không giống như dấu nháy đơn (') cho phép biểu diễn hằng kí tự, cặp ngoặc kép (") là hằng biểu diễn một chuỗi kí tự liên tiếp, và ở cuối chuỗi một kí tự null ('\0') luôn được tự động thêm vào.

Vì vậy chúng ta có thể khởi tạo xâu mystring theo một trong hai cách sau đây:

```
char mystring [] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char mystring [] = "Hello";
```

Trong cả hai trường hợp mảng (hay xâu kí tự) mystring được khai báo với kích thước 6 kí tự: 5 kí tự biểu diễn Hello cộng với một kí tự null.

Trước khi tiếp tục, tôi cần phải nhắc nhở bạn rằng việc gán nhiều hằng như việc sử dụng dấu ngoặc kép (") chỉ hợp lệ khi khởi tạo mảng, tức là lúc khai báo mảng. Các biểu thức trong chương trình như:

```
mystring = "Hello";
mystring[] = "Hello";
```

là không hợp lệ, cả câu lệnh dưới đây cũng vậy:

```
mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

Vậy hãy nhớ: Chúng ta chỉ có thể "gán" nhiều hằng cho một mảng vào lúc khởi tạo nó. Nguyên nhân là một thao tác gán (=) không thể nhận vế trái là cả một mảng mà chỉ có thể nhận một trong những phần tử của nó. Vào thời điểm khởi tạo mảng là một trường hợp đặc biệt, vì nó không thực sự là một lệnh gán mặc dù nó sử dụng dấu bằng (=).

Gán giá trị cho xâu kí tự

Vì vế trái của một lệnh gán chỉ có thể là một phần tử của mảng chứ không thể là cả mảng, chúng ta có thể gán một xâu kí tự cho một mảng kiểu char sử dụng một phương pháp như sau:

```
mystring[0] = 'H';
mystring[1] = 'e';
```

```
mystring[2] = 'l';
mystring[3] = 'l';
mystring[4] = 'o';
mystring[5] = '\0';
```

Nhưng rõ ràng đây không phải là một phương pháp thực tế. Để gán giá trị cho một xâu kí tự, chúng ta có thể sử dụng loạt hàm kiểu strcpy (string copy), hàm này được định nghĩa trong string.h và có thể được gọi như sau:

```
strcpy (string1, string2);
```

Lệnh này copy nội dung của string2 sang string1. string2 có thể là một mảng, con trỏ hay một hằng xâu kí tự, bởi vậy lệnh sau đây là một cách đúng để gán xâu hằng "Hello" cho mystring:

```
strcpy (mystring, "Hello");
```

Ví dụ:

```
// setting value to string
#include <iostream.h>
#include <string.h>

int main ()
{
  char szMyName [20];
  strcpy (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

Kết quả:

```
J. Soulie
```

Để ý rằng chúng ta phải include file `<string.h>` để có thể sử dụng hàm `strcpy`.

Mặc dù chúng ta luôn có thể viết một hàm đơn giản như hàm `setstring` dưới đây để thực hiện một thao tác giống như `strcpy`:

```
// setting value to string
#include <iostream.h>
```

```
void setstring (char szOut [], char szIn [])
{
  int n=0;
  do {
    szOut[n] = szIn[n];
    n++;
  } while (szIn[n] != 0);
}

int main ()
{
  char szMyName [20];
  setstring (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

Kết quả:

```
J. Soulie
```

Một phương thức thường dùng khác để gán giá trị cho một mảng là sử dụng trực tiếp dòng nhập dữ liệu (**cin**). Trong trường hợp này giá trị của xâu kí tự được gán bởi người dùng trong quá trình chương trình thực hiện.

Khi **cin** được sử dụng với các xâu kí tự nó thường được dùng với phương thức **getline** của nó, phương thức này có thể được gọi như sau:

```
cin.getline ( char buffer[], int length, char delimiter = '\n');
```

trong đó ***buffer*** (bộ đệm) là địa chỉ nơi sẽ lưu trữ dữ liệu vào (như là một mảng chẳng hạn), ***length*** là độ dài cực đại của bộ đệm (kích thước của mảng) và ***delimiter*** là kí tự được dùng để kết thúc việc nhập, mặc định - nếu chúng ta không dùng tham số này - sẽ là kí tự xuống dòng (**'\n'**).

Ví dụ sau đây lặp lại tất cả những gì bạn gõ trên bàn phím. Nó rất đơn giản nhưng là một ví dụ cho thấy bạn có thể sử dụng **cin.getline** với các xâu kí tự như thế nào:

```
// cin with strings
#include <iostream.h>

int main ()
{
```

```
  char mybuffer [100];
  cout << "What's your name? ";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
  cout << "Which is your favourite team? ";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << " too.\n";
  return 0;
}
```

Kết quả:

```
What's your name? Juan
Hello Juan.
Which is your favourite team? Inter Milan
I like Inter Milan too.
```

Chú ý trong cả hai lời gọi **cin.getline** chúng ta sử dụng cùng một biến xâu (**mybuffer**). Những gì chương trình làm trong lời gọi thứ hai đơn giản là thay thế nội dung của **buffer** trong lời gọi cũ bằng nội dung mới.

Nếu bạn còn nhớ phần nói về giao tiếp với, bạn sẽ nhớ rằng chúng ta đã sử dụng toán tử **>>** để nhận dữ liệu trực tiếp từ đầu vào chuẩn. Phương thức này có thể được dùng với các xâu kí tự thay cho **cin.getline**. Ví dụ, trong chươn trình của chúng ta, khi chúng ta muốn nhận dữ liệu từ người dùng chúng ta có thể viết:

```
cin >> mybuffer;
```

lệnh này sẽ làm việc như nó có những hạn chế sau mà **cin.getline** không có:

- Nó chỉ có thể nhận những từ đơn (không nhận được cả câu) vì phương thức này sử dụng kí tự trống(bao gồm cả dấu cách, dấu tab và dấu xuống dòng) làm dấu hiệu kết thúc..
- Nó không cho phép chỉ định kích thước cho bộ đệm. Chương trình của bạn có thể chạy không ổn định nếu dữ liệu vào lớn hơn kích cỡ của mảng chứa nó.

Vì những nguyên nhân trên, khi muốn nhập vào các xâu kí tự bạn nên sử dụng **cin.getline** thay vì **cin >>**.

Chuyển đổi xâu kí tự sang các kiểu khác

Vì một xâu kí tự có thể biểu diễn nhiều kiểu dữ liệu khác như dạng số nên việc chuyển đổi nội dung như vậy sang dạng số là rất hữu ích. Ví dụ, một xâu có thể mang giá trị "1977"nhưng đó là một chuỗi gồm 5 kí tự (kể cả kí tự null) và không dễ gì chuyển thành một số nguyên. Vì vậy thư viện cstdlib (stdlib.h) đã cung cấp 3 macro/hàm hữu ích sau:

- atoi: chuyển xâu thành kiểu int.
- atol: chuyển xâu thành kiểu long.
- atof: chuyển xâu thành kiểu float.

Tất cả các hàm này nhận một tham số và trả về giá trị số (int, long hoặc float). Các hàm này khi kết hợp với phương thức `getline` của `cin` là một cách đáng tin cậy hơn phương thức `cin>>` cổ điển khi yêu cầu người sử dụng nhập vào một số:

```
// cin and ato* functions
#include <iostream.h>
#include <stdlib.h>

int main ()
{
  char mybuffer [100];
  float price;
  int quantity;
  cout << "Enter price: ";
  cin.getline (mybuffer,100);
  price = atof (mybuffer);
  cout << "Enter quantity: ";
  cin.getline (mybuffer,100);
  quantity = atoi (mybuffer);
  cout << "Total price: " << price*quantity;
  return 0;
}
```

Kết quả:

```
Enter price: 2.75
Enter quantity: 21
Total price: 57.75
```

Các hàm để thao tác trên chuỗi

Thư viện cstring (string.h) không chỉ có hàm strcpy mà còn có nhiều hàm khác để thao tác trên chuỗi. Dưới đây là giới thiệu lướt qua của các hàm thông dụng nhất:

- strcat:

```
strcat: char* strcat (char* dest, const char* src);
// G□n thêm chu□i src vào phía cu□i c□a dest. Tr□ v□ dest.
```

- strcmp:

```
strcmp: int strcmp (const char* string1, const char* string2);
// So sánh hai xâu string1 và string2. Trả về 0 nếu hai xâu là bằng nhau.
```

- strcpy:

```
strcpy: char* strcpy (char* dest, const char* src);
// Copy nội dung của src cho dest. Trả về dest.
```

- strlen:

```
strlen: size_t strlen (const char* string);
// Trả về độ dài của string.
```

Chú ý: char* hoàn toàn tương đương với char[]

Revision #1
Created 5 October 2019 11:00:24 by Laptrinh.vn
Updated 5 October 2019 11:10:55 by Laptrinh.vn