Tổng kết về ký tự và xâu ký tự

I.Kiểu ký tự char

• **Giá trị nguyên** biểu diễn dưới dạng một ký tự viết trong 2 dấu nháy
vd: 'z'=122 là giá trị nguyên của ký tự z (ký tự thứ 122 trong bảng mã ASCII)

• Các hàm liên quan đến kiểu char được định nghĩa trong ctype.h

Nhận dạng các ký tự:	
int isalnum (char c);	là số hoặc chữ
int isalpha (char c);	là chữ cái
int isascii (char c);	là một ký tự trên bàn phím (mã ASCII <=128)
int iscntrl (char c);	là một ký tự điều khiển (có mã quét bàn phím) Các kí tự điều khiển (Control Character) nằm từ 0x00 đến 0x1F và thêm 0x7F
int isdigit (char c);	là chữ số
int isgraph (char c);	là Graphical Character.Bất cứ kí tự nào có thể in ra được (printable character) đều gọi là Graphical Character, ngoại trừ kí tự <space>
int islower (char c);	là chữ viết thường
int isprint (char c);	là ký tự in được, bao gồm Graphical Character và kí tự trắng <space>
int ispunct (char c);	là dấu câu
int isspace (char c);	là ký tự phân cách (space,tab,enter...)
int isupper (char c);	là chữ viết hoa
int isxdigit(char c);	là chữ số thập lục phân ('0'..'9','A'..'F','a'..'f')

```
#include <iostream>
#include <ctype.h>
void main()
{
   char c;
   cout << "Nhap 1 ky tu:";cin >> c;
   if (isdigit(c)) cout << "mot so !";
   else if (isalpha(c) )
   {
      cout << "mot chu ";
      if ( isupper(c) ) cout << "viet hoa !";
      else cout << "viet thuong !";
   }
   getch();
}
```

Các ký tự điều khiển

Là những ký tự mà không thể được viết ở bất kì đâu khác trong chương trình như là mã xuống dòng (\n) hay tab (\t). Tất cả đều bắt đầu bằng dấu xổ ngược (\). Sau đây là danh sách các mã điều khiển và ý nghĩa của nó:

\n	xuống dòng
\r	lùi về đầu dòng
\t	kí tự tab
\v	căn thẳng theo chiều dọc
\b	backspace
\f	sang trang
\a	Kêu bíp
\'	dấu nháy đơn

\"	dấu nháy kép
\	dấu hỏi
\\	kí tự xổ ngược

Chuyển đổi case của kí tự (Character Case Conversion)

```
int tolower (char c);
//chuyen sang chu hoa
int toupper (char c);
//chuyen sang chu thuong
```

Ví dụ:

```
char c = 'A';
c=tolower(c);
cout << "\nAfter 1st-case-conversion : " << c;
cout << "\nAfter 2nd-case-conversion : " << (char)toupper(c) << endl;
//vi kieu tra ve la int nen phai ep kieu char de in ra ky tu
```

Kết quả:

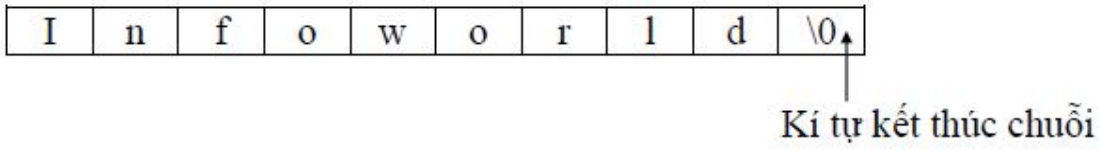
```
After 1st-case-conversion : a
After 2nd-case-conversion : A
```

II.Xâu ký tự (string)

•Là mảng 1 chiều gồm các phần tử có kiểu char như mẫu tự, con số và bất cứ ký tự đặc biệt như +, -, *, /, $, #...
•Viết trong cặp nháy kép, ví dụ: "I like C++"

–Theo quy ước, một xâu sẽ được kết thúc bởi ký tự **null** ('\0' : kí tự rỗng).
–Xâu là một con trỏ (pointer) trỏ đến ký tự đầu tiên của xâu (giống như với mảng)

Ví dụ: xâu s="Infoworld"; được lưu trữ như sau:



Trong đó con trỏ s trỏ đến ký tự đầu tiên 'I'

Kết thúc bằng **null** như vậy rất khác so với các ngôn ngữ khác. Ví dụ, trong Pascal, mỗi chuỗi kí tự bao gồm một mảng kí tự và *length byte* chứa chiều dài chuỗi. Cấu trúc này giúp Pascal dễ dàng trả về độ dài chuỗi khi được yêu cầu. Khi đó, Pascal chỉ việc trả về giá trị *length byte*, trong khi C phải đếm cho tới khi nó gặp kí tự **'\0'**. Đây là lí do khiến C chậm hơn Pascal trong một vài tình huống nhất định.

1.Gán giá trị cho xâu (string assignment)

• Mảng của ký tự:

```
char color[] = "blue";
```

Biến con trỏ char*

```
char*colorPtr = "blue";
```

tạo con trỏ colorPtr trỏ đến chữ b trong xâu "blue" ("blue" ở trong bảng chuỗi hằng)

• Khởi tạo chuỗi như mảng:

```
char color[] = { 'b', 'l', 'u', 'e', '\0' };
```

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 sau là không hợp lệ:

```
//char mystring[6];
mystring = "Hello";
mystring[] = "Hello";
mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

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 về 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 (=).

Tuy nhiên C++ cho phép ta gán 2 mảng tĩnh có cùng kích thước như sau:

```
char a[]="Hello", b[6];
//hello và ký tự null tổng cộng 6 ký tự
//khai báo như trên thì 2 mảng tĩnh có cùng kích thước
 b=a;
```

Phép gán này tương đương đoạn chương trình sau:

```
int i=0;
while ( a[i] <= 6 )
      b[i]=a[i++];
```

Thiết lập n ký tự đầu của xâu s bằng ký tự c bằng 1 trong 2 hàm sau:

```
void strnset( char s[], char c, int n):
void memset(char *Des, int c, size_t n);
```

2.Những chuỗi hằng

Bạn hãy thử hai đoạn chương trình sau:

```
char *s="hello";
cout << s;
```

và:

```
char s[100];
strcpy(s,"hello");
cout << s;
```

Hai đoạn mã trên đưa ra cùng một kết qủa, nhưng cách hoạt động của chúng hòan tòan khác nhau. Trong đoạn 2, bạn không thể viết **s=”hello”;**. Để hiểu sự khác nhau, bạn cần phải biết hoạt động của *bảng chuỗi hằng* (string constant table) trong C.

Khi chương trình được thực thi, trình biên dịch tạo ra một file object, chứa mã máy và một bảng chứa tất cả các chuỗi hằng khai báo trong chương trình. Trong đoạn 1, lệnh **s = ”hello”;** xác định rằng **s** chỉ đến địa chỉ của chuỗi **hello** trong bảng chuỗi hằng. Bởi vì chuỗi này nằm trong bảng chuỗi hằng, và là một bộ phận trong mã exe, nên bạn không thể thay đổi được nó. Bạn chỉ có thể dùng nó theo kiểu chỉ-đọc (read-only).Để minh họa, bạn có thể chèn thêm câu lệnh strcpy(s,"modify"); vào sau lệnh gán ở ví dụ 1, trình biên dich sẽ báo lỗi ghi vào hằng.

Trong đọan 2, chuỗi **hello** cũng tồn tại trong bảng chuỗi hằng, do đó bạn có thể copy nó vào mảng kí tự tên là **s.** Bởi vì s không phải là một con trỏ, lệnh **s=”hello”;** sẽ không làm việc.

3.Đọc chuỗi

• Đọc dữ liệu cho mảng ký tự:

```
char word[20];
cin >> word;
```

- Đọc xâu không chấp nhận khoảng trống.
- Xâu có thể vượt quá kích thước mảng.

```
cin >> setw( 20 ) >> word; // đọc 19 ky tu (1 đe danh cho '\0')
```

Đọc xâu với khoảng trống dùng 1 trong các cú pháp sau:

```
gets(array);// trong stdio.h, không được khuyến khích sử dụng
cin.get(array);
cin.get(array,size);
cin.getline(array,delimiter='\n');
//ky tu delimiter mac dinh la '\n' - xuong dong
cin.getline(array, size, delimiter='\n');
```

– Lưu input vào mảng array đến khi xảy ra một trong hai trường hợp
+ Kích thước dữ liệu đạt đến size –1
+ Ký tự delimiter được nhập vào
*Lưu ý :**delimiter='\n'** thì dấu = là tham số mặc định trong C++, tức là nếu không có tham số này thì trình biên dịch sẽ hiểu là để mặc định*
Ví dụ:

```
char sentence[ 80 ];
cin.getline( sentence, 80);
//dung delimiter mac dinh
```

Đối với các hàm get hay getline ta hoàn toàn có thể kết hợp với toán tử >> như thế này:

```
cout << " Nhap ten, tuoi, nghe nghiep" << endl;
//cin.ignore();
cin.getline( name ) >> age >> job;
```

Nếu một chương trình bị treo hay kết thúc bất thường khi làm việc với xâu thường là do một số ký tự vẫn còn trong vùng đệm. Kết quả là chương trình có vẻ kết thúc sớm hơn mong muốn.
Hàm fflush() hay cin.ignore() sẽ giải quyết vấn đề này. Nó sẽ làm sạch vùng đệm và chép tất cả những gì có trong vùng đệm ra ngoài (trong ví dụ ở trên thì nó không thật sự cần thiết lắm)

III.Thư viện xử lý xâu <string.h>

Cung cấp các hàm:

- Thao tác với dữ liệu kiểu xâu
- So sánh xâu
- Tìm kiếm trên xâu các ký tự hoặc xâu khác
- Chia xâu thành các từ tố (tokenize strings)

1.Một số hàm cơ bản

Chuyển chuỗi xâu sang chữ thường

```
char *strlwr(char *s);
```

Ví dụ:

```
char *s = "Borland C";
s = strlwr(s);      //ket qua s = "borland c"
```

Chuyển chuỗi xâu sang chữ hoa

```
char *strupr(char *s);
```

Ví dụ:
```
char *s = "Borland C";
s = strlwr(s);      //ket qua s = "BORLAND C"
```

Xác định độ dài xâu

```
size_t strlen( const char *s )
//tra ve so ky tu cua xau khong tinh đen ky tu '\0'
```

Ví dụ:
```
        char s[] = "This is a string";
        cout << "The string is include: " << strlen(s) << " characters\n";
        getch();
```

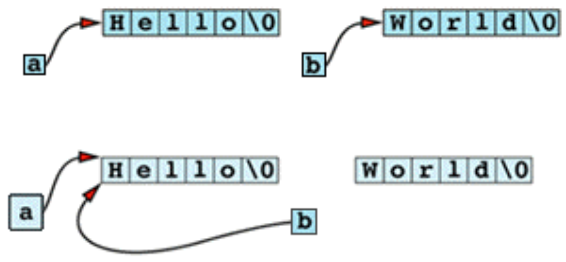
2.Copy xâu

Xét ví dụ sau:

```
#include <iostream.h>
void main()
{
        char a[]="Hello",*b="World";
        b=a;//vấn đề ở đây
        a[0] = '0';
        cout << a << " " << b;
        //dự định in ra “0ello Hello”
}
```
Kết quả:

0ello 0ello

Vấn đề của xâu ký tự: xâu thực chất là 1 con trỏ tham chiếu tới vùng nhớ có chứa nội dung nên sẽ xảy ra tình trạng như hình vẽ đối với phép gán:



Đó là lý do khi thay đổi xâu a ta lại nhận được sự thay đổi trên cả a và b.

Đây là điều mà chúng ta không muốn.Giải pháp đúng cho trường hợp này là sao chép nội dung bằng 1 vòng while()

```
//char *a="Hello",b[]="World";
int i=0;
while ( a[i] != NULL )
        b[i]=a[i++];
b[i]=a[i];//ky tu null
```

Nhưng tiện hơn là dùng hàm chuẩn:

```
char *strcpy( char *s1, const char *s2 );
```

-Copy tham số thứ hai vào tham số thứ nhất
–Tham số thứ nhất phải có kích thước đủ lớn để chứa xâu và ký tự null

```
char *strncpy( char *s1, const char *s2, size_t n );
```

- Xác định rõ số ký tự được copy từ xâu vào mảng
- Không nhất thiết copy ký tự null

```
#include <iostream>
#include <conio.h>
#include <string>
//chua prototypes (khai bao) strcpy & strncpy

void main()
{
   char x[] = "Happy Birthday to You";
      char y[ 25 ];
      char z[ 15 ];
   strcpy( y, x );  // copy contents of x into y
   cout << "The string in array x is: " << x<< "\nThe string in array y is: " << y << '\n';
   // copy 14 ky tu dau tien tu x vao z
   strncpy( z, x, 14 );  // khong copy ky tu '\0'
   z[ 14 ] = '\0'; //ghi them '\0'
   cout << "The string in array z is: " << z << endl;
   getch();
}
```

Kết quả:

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

3.Nối xâu (Concatenating strings)

```
char *strcat( char *s1, const char *s2 )
```

- Nối xâu thứ hai vào sau xâu thứ nhất
- Ký tự đầu tiên của tham số thứ hai thay thế ký tự null của tham số thứ nhất
- Phải chắc chắn rằng tham số thứ nhất có kích thước đủ lớn để chứa thêm phần nối vào và ký tự null kết thúc xâu.

```
char *strncat( char *s1, const char *s2, size_t n )
```

- Thêm n ký tự của tham số thứ hai vào sau tham số thứ nhất
- Thêm ký tự null vào kết quả

```
void main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";
   cout << "s1 = " << s1 << "\ns2 = " << s2;
   strcat( s1, s2 );  // them s2 vao sau s1
   cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;
   strncat( s3, s1, 6 );
   // them 6 ky tu dau tien cua s1 vao sau s3, ghi '\0' vao cuoi
   cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1 << "\ns3 = " << s3;
   strcat( s3, s1 );  // them s1 vao sau s3
   cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1 << "\ns3 = " << s3
   << endl;
   getch();
```

```
}
```

Kết quả:

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

4.So sánh xâu (comparing strings)

- Các ký tự được biểu diễn bằng mã dạng số (numeric code), các mã đó được dùng để so sánh các xâu ký tự
- Các hàm so sánh xâu:

```
int strcmp( const char *s1, const char *s2 )
//phan biet chu hoa chu thuong
int strcmpi( const char *s1, const char *s2)
//khong phan biet chu hoa chu thuong
```

+So sánh từng ký tự một, theo thứ tự từ điển
+Trả về
(+) 0 nếu xâu bằng nhau
(+) giá trị âm nếu xâu thứ nhất nhỏ hơn xâu thứ hai
(+) giá trị dương nếu xâu thứ nhất lớn hơn xâu thứ hai

```
int strncmp( const char *s1, const char *s2, size_t n );
int strncmpi( const char *s1, const char *s2, size_t n );
```

- So sánh n ký tự đầu tiên
- Dừng so sánh nếu gặp ký tự null của 1 trong 2 tham số

```
void main()
{
	char *s1 = "Happy New Year";
	char *s2 = "Happy New Year";
	char *s3 = "Happy Holidays";
	cout << "s1 = " << s1 << "\ns2 = " << s2
	<< "\ns3 = " << s3 << "\n\nstrcmp(s1, s2) = "
	<< setw( 2 ) << strcmp( s1, s2 )
	<< "\nstrcmp(s1, s3) = " << setw( 2 )
	<< strcmp( s1, s3 ) << "\nstrcmp(s3, s1) = "
	<< setw( 2 ) << strcmp( s3, s1 );

   cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
   << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = "
   << setw( 2 ) << strncmp( s1, s3, 7 )
   << "\nstrncmp(s3, s1, 7) = "
   << setw( 2 ) << strncmp( s3, s1, 7 ) << endl;
   getch();
}
```

Kết quả:

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

5.Tìm kiếm nội dung xâu ký tự

Hàm strchr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của ký tự c trong chuỗi str.
Hàm strstr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1.

char *strchr(const char *str, int c)

- Nếu ký tự đã chỉ định không có trong chuỗi, kết quả trả về là NULL.
- Kết quả trả về của hàm là một con trỏ **trỏ đến ký tự c được tìm thấy đầu tiên** trong chuỗi str.

char*strstr(const char *s1, const char *s2)

Kết quả trả về của hàm là một con trỏ **trỏ đến phần tử đầu tiên của chuỗi s1 có chứa chuỗi s2** hoặc giá trị NULL nếu chuỗi s2 không có trong chuỗi s1.

Ví dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi “hoc”.

```
void main()
{
char Chuoi[255],*s;
cout << "Nhap chuoi: ";
cin.get(Chuoi);
s=strstr(Chuoi,”hoc”);
cout << “Chuoi trich ra: ” << s;
getch();
}
```

size_t strcspn(const char *str1, const char *str2);

Trả về vị trí đầu tiên của bất cứ kí tự nào trong 'str1' tìm thấy trong 'str2'
Nếu không tìm thấy thì trả về độ dài chuỗi str1

```
void main()
{
   const char *str="Xcross87 ";
   char *key="123456789";
   int pos = strcspn(str,key);
   cout << "tim thay o vi tri" << pos;
   getch();
}
```

size_t strspn(const char *str1, const char *str2);

Trả về vị trí của kí tự đầu tiên trong chuỗi str1 mà không khớp str2

char* strpbrk(const char* str1, const char* str2);

Trả về con trỏ trỏ đến kí tự xuất hiện đầu tiên trong 'str1' xuất hiện trong 'str2'.
Trả về NULL nếu không tìm thấy.

```
void main()
{
   const char *str1="con ga trong ko phai la con ga mai";
   const char *str2="1234956za";
   char* found;

   found = strpbrk(str1,str2);

   if (NULL != found) cout << " Tim thay ki tu dau tien: " << *found;

   getch();
}
```

6.Đảo ngược xâu : char *strrev(char *s);

```
char s[]="1234956za";
cout << strrev(s);
```

Kết quả:

```
az6594321
```

7.Phân tích từ tố (tokenizing)

– Chia xâu thành các từ tố, phân tách bởi các ký tự ngăn cách (delimiting character)
vd: "This is my string" có 4 từ tố (tách nhau bởi các dấu trống)

char *strtok(char *s1, const char *s2)

+Cần gọi nhiều lần
(+)Lần gọi đầu cần 2 tham số,xâu cần phân tích từ tố và xâu chứa các ký tự ngăn cách
(+)Những lời gọi tiếp theo sử dụng đối số thứ nhất là NULL, tiếp tục phân tích từ tố trên xâu đó
+Kết quả trả về của hàm là một xâu - từ tố hoặc giá trị NULL nếu đã duyệt hết.

```
void main()
{
   char sentence[] = "This is a sentence with 7 tokens";
   char *tokenPtr;
   cout << "The string to be tokenized is:\n" << sentence
   << "\n\nThe tokens are:\n\n";
   // Loi goi strtok dau tien khoi dau viec phan tich tu
   tokenPtr = strtok( sentence, " " );
   // Cac loi goi strtok tiep sau
   //voi NULL là doi so thu nhat de tiep tuc viec phan tich tu to
   while ( tokenPtr != NULL )
   {
      cout << tokenPtr << '\n';
      tokenPtr = strtok( NULL, " " );  //lay token tiep theo
   } // end while
   cout << "\nAfter strtok, sentence = " << sentence << endl;
   getch();
}
```

Kết quả:

```
The string to be tokenized is:
This is a sentence with 7 tokens
```

```
The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

IV.Các chức năng khác

Thao tác hàng loạt với xâu

Hàm transform() trong thư viện <algorithm> của C++

```
void*transform(void*vị trí bắt đầu,
               void* vị trí kết thúc,
               void*mảng đích lưu kết quả,
               con trỏ hàm chuyển đổi);
```

Ví dụ:

```
#include <ctype.h>      // for toupper
#include <algorithm>    // for transform
#include <iostream>

using namespace std;

char alphabet(char c)
{
   static char ch = 'a';
   return ch++;
}
int main()
{
   char s[] = "this is a lower case string";

   transform(s, s + sizeof(s), s, toupper);
   cout << s << endl;

   transform(s, s + sizeof(s), s, alphabet);
   cout << s;
   return 0;
}
```

Các hàm chuyển đổi dữ liệu với xâu

•Chuyển đổi 1 chuỗi sang giá trị int.

```
int atoi(const char *s);
int atol(const char *s);
//tuong tu atoi() nhung su dung voi kieu long
//phai khai bao stdlib.h
```

Ví dụ:

```
int i;
char *str = "12345.67";
i = atoi(str);
//ket qua i = 12345
```

• Chuyển đổi 1 chuỗi sang giá trị long

long strtol(const char *start, char **end, int base);

start là chuỗi đầu vào của convert, end là phần còn lại của chuỗi sau khi convert, base là cơ số của chuỗi nhập vào
convert thành công thì trả về số double đã convert
nếu thất bại thì trả về: 0.0

```
void main ()
{
   char *start="12f.a3sygjlg";
   char *end;
   long result;

   result = strtol(start,&end,16);

   cout << "Sau khi convert: ";
   cout << "\n start = " << start;
   cout << "\n end = " << end;
   cout << "\n result = " << result;

   getch();
}
```

```
unsigned long strtoul( const char *start, char **end, int base );
//giong nhu tren nhung la kieu long khong dau
```

• Chuyển đổi 1 chuỗi sang giá trị double

double atof(const char *s);
Phải khai báo math.h hoặc stdlib.h

Ví dụ:

```
float f;
char *str = "12345.67";
f = atof(str);
//ket qua f = 12345.67;
```

```
double strtod( const char *start, char *end );
//su dung giong strtol()
```

nếu thất bại thì trả về: 0.0
nếu convert thành công mà kết quả lại quá to so với giới hạn của double thì gới hạn được trả về.

• Chuyển đổi số nguyên value sang chuỗi string theo cơ số radix.

```
char *itoa(int value, char *string, int radix);
```

Phải khai báo stdlib.h

Ví dụ:

```
int number = 12345;
char string[25];
itoa(number, string, 10);
//chuyen doi number sang chuoi theo co so 10
//ket qua string = "12345"
itoa(number, string, 2);
//chuyen doi number sang chuoi theo co so 2
//ket qua string = "11000000111001"
```

V.Chú ý về sử dụng xâu với cấp phát động

1.Giả sử có chương trình sau:

```
int main()
{
   char *s;

   s= new char[100];
   s="hello";
   delete[] s;
   return 0;
}
```

Đoạn trên dịch tốt, nhưng khi chạy thì bị lỗi segmentation fault ở dòng delete[].Toán tử new cấp phát một khối 100 bytes và trỏ s tới đó, nhưng sau đó, lệnh s="hello"; gây ra lỗi. Về cú pháp thì lệnh này không sai, vì s là một con trỏ; tuy nhiên khi lệnh s="hello"; được thực thi, s trỏ tới một chuỗi trong bảng chuỗi hằng (string constant table) và khối bộ nhớ cấp phát trước đó "bơ vơ" (memory leak).Vì s trỏ đến một chuỗi trong bảng chuỗi hằng, mà chuỗi này không thể thay đổi, nên delete không thực hiện được việc giải phóng bộ nhớ.

Chương trình trên cần phải sửa lại thành:

```
int main()
{
   char *s;
   s= new char[100];
   strcpy(s,"hello");
   delete[] s;
   return 0;
}
```

2.Xem đoạn mã sau đây:

```
char *p, buf[ 256 ];
gets( buf );
p = new char[strlen(buf)];
strcpy( p, buf );
```

Đoạn mã trên sai ở chỗ: Hàm strlen() không tính đến kí tự null ở cuối xâu, trong khi hàm strcpy vẫn sao chép kí tự null ở cuối xâu nguồn sang xâu đích. Kết quả là strcpy ghi kí tự null ra ngoài vùng nhớ được cấp phát cho p.Sửa lại là:

```
p = new char[ strlen( buf ) + 1 ];
strcpy( p, buf );
```

Hầu hết các môi trường phát triển C và C++ đều cung cấp một hàm có tên là strdup(). Hàm này sử dụng malloc() và strcpy() để tạo ra một bản sao của xâu, nhờ đó tránh được lỗi nói trên. Thật không may, strdup() lại không phải là một hàm chuẩn của ANSI C.