

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

MÔN
NGÔN NGỮ LẬP TRÌNH C++

CHƯƠNG 5
CON TRỎ VÀ XÂU KÝ TỰ

Giảng viên	:	ThS. PHAN NGHĨA HIỆP
Khoa	:	CÔNG NGHỆ THÔNG TIN 2
Bộ môn	:	AN TOÀN THÔNG TIN
Điện thoại/ Email	:	hieppn@ptithcm.edu.vn

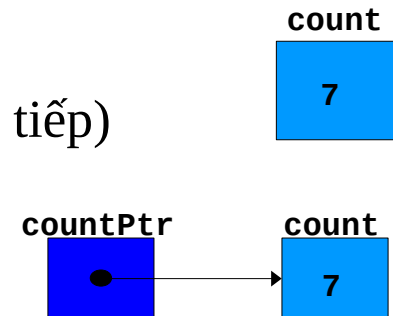
Chương 5 – Con trỏ và Xâu ký tự

Đề mục

- 5.1 Giới thiệu
- 5.2 Khai báo và khởi tạo biến con trỏ
- 5.3 Các thao tác trên con trỏ
- 5.4 Gọi hàm bằng tham chiếu
- 5.5 Sử dụng const với con trỏ
- 5.6 Sắp xếp nổi bọt sử dụng Pass-by-Reference
- 5.7 Các phép toán trên con trỏ
- 5.8 Quan hệ giữa con trỏ và mảng
- 5.9 Mảng con trỏ
- 5.10 Ví dụ: giả lập tráo và chia bài
- 5.11 Con trỏ tới hàm
- 5.12 Giới thiệu về xử lý ký tự và xâu
 - 5.12.1 Tổng quát về ký tự và xâu
 - 5.12.2 Các hàm xử lý xâu

5.1 Giới thiệu

- Con trỏ (Pointer)
 - Mạnh, nhưng khó làm chủ
 - Có tác dụng như truyền tham chiếu (pass-by-reference)
 - Có liên quan chặt chẽ đến mảng và chuỗi
- Biến con trỏ (Pointer variable)
 - Chứa địa chỉ vùng nhớ thay vì chứa giá trị
 - Thông thường, biến chứa giá trị (tham chiếu trực tiếp)
 - Con trỏ chứa địa chỉ của biến mang giá trị cụ thể (tham chiếu gián tiếp)



5.2 Khai báo và khởi tạo biến con trỏ

- Khai báo con trỏ
 - * cho biết biến là con trỏ
`int *myPtr;`
dữ liệu kiểu `int` có địa chỉ là `myPtr`, con trỏ kiểu `int *`
 - Mỗi con trỏ cần một dấu sao
`int *myPtr1, *myPtr2;`
 - Có thể khai báo con trỏ tới bất cứ kiểu dữ liệu nào
- Khởi tạo con trỏ (Pointer initialization)
 - Khởi tạo về `0`, `NULL`, hoặc địa chỉ
 - `0` hoặc `NULL` không trỏ đến đâu cả

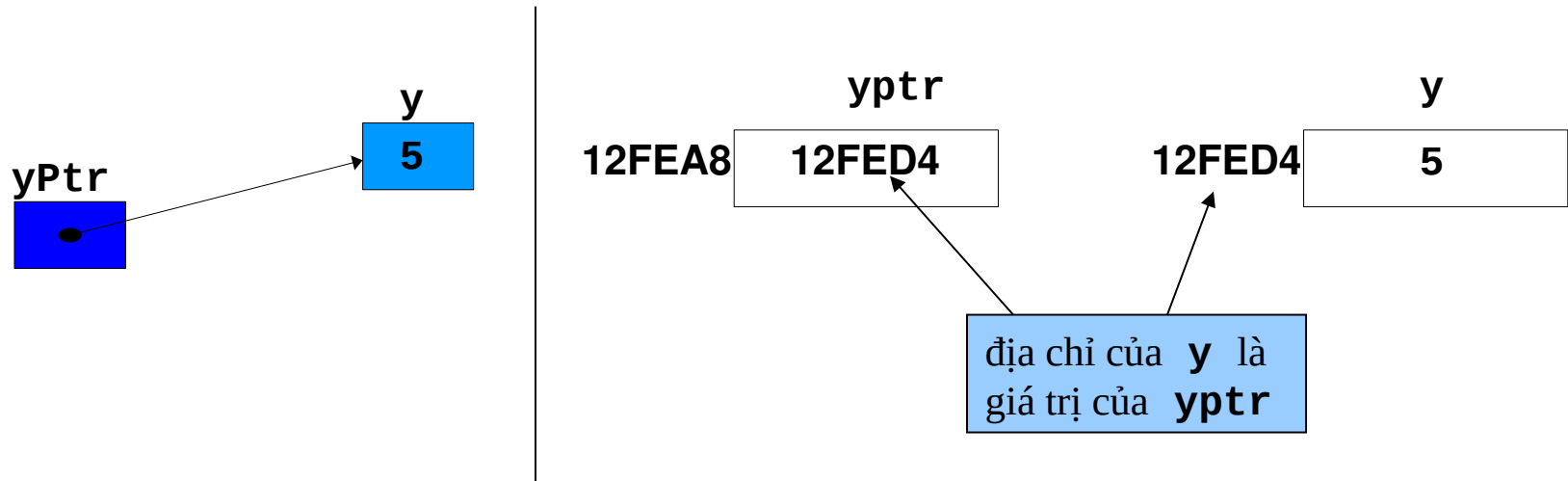
5.3 Các thao tác đối với con trỏ

- **&** Toán tử địa chỉ (address operator)

- Trả về địa chỉ vùng nhớ của toán hạng
- Ví dụ

```
int y = 5;  
int *yPtr;  
yPtr = &y;    // yPtr chứa địa chỉ của y
```

- **yPtr** “trỏ đến” **y**

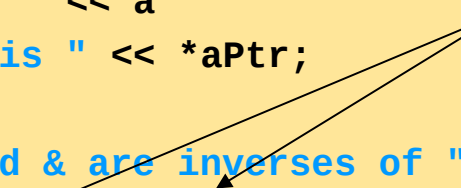


5.3 Các thao tác đối với con trỏ

- * phép thâm nhập (indirection/dereferencing)
 - Trả về đối tượng mà con trỏ trỏ tới
 - ***yPtr** trả về **y** (vì **yPtr** trỏ đến **y**).
 - con trỏ khi bị thâm nhập (dereferenced) là giá trị trái (lvalue)
***yptr = 9; // assigns 9 to y**
- * và & ngược nhau

```
1 // Fig. 5.4: fig05_04.cpp
2 // Using the & and * operators.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     int a;        // a is an integer
11     int *aPtr;    // aPtr is a pointer to an integer
12
13     a = 7;
14     aPtr = &a;    // aPtr assigned address of a
15
16     cout << "The address of a is " << &a
17          << "\nThe value of aPtr is " << aPtr;
18
19     cout << "\n\nThe value of a is " << a
20          << "\nThe value of *aPtr is " << *aPtr;
21
22     cout << "\n\nShowing that * and & are inverses of "
23          << "each other.\n&*aPtr = " << &*aPtr
24          << "\n*&aPtr = " << *&aPtr << endl;
25 }
```

* và & ngược nhau



```
26     return 0;  // indicates successful termination
27
28 } // end main
```

fig05_04.cpp
(2 of 2)

The address of a is 0012FED4
The value of aPtr is 0012FED4

fig05_04.cpp
output (1 of 1)

The value of a is 7
The value of *aPtr is 7

Showing that * and & are inverses of each other.

&*aPtr = 0012FED4
*&aPtr = 0012FED4

* và & ngược nhau; cùng kết quả khi
cùng sử dụng cả 2 với **aPtr**

5.4 Gọi hàm bằng tham chiếu

- 3 cách truyền tham số cho hàm
 - Truyền giá trị (Pass-by-value)
 - Truyền tham chiếu với đối số là tham chiếu (Pass-by-reference with reference arguments)
 - Truyền tham chiếu với đối số là con trỏ (Pass-by-reference with pointer arguments)

5.4 Gọi hàm bằng tham chiếu

- Truyền tham chiếu với đối số là tham chiếu
 - Thay đổi giá trị gốc của tham số
 - hàm có thể “trả về” nhiều hơn một giá trị
- Truyền tham chiếu bằng đối số là con trỏ
 - Tương tự pass-by-reference
 - Sử dụng con trỏ và toán tử *
 - Truyền địa chỉ của đối số bằng toán tử &
 - Truyền mảng không cần toán tử & vì tên mảng chính là con trỏ
 - Toán tử thâm nhập * được dùng cùng con trỏ để tạo một tên khác cho biến được truyền vào

```
1 // Fig. 5.6: fig05_06.cpp
2 // Cube a variable using pass-by-value.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int cubeByValue( int ); // prototype
9
10 int main()
11 {
12     int number = 5;
13
14     cout << "The original value of number is " << number << endl;
15
16     // pass number by value to cubeByValue
17     number = cubeByValue( number );
18
19     cout << "\nThe new value of number is " << number << endl;
20
21     return 0; // indicates successful termination
22
23 } // end main
24
```

Truyền number bằng giá trị;
kết quả được trả về bởi
cubeByValue

```
25 // calculate and return cube of integer argument
26 int cubeByValue( int n )
27 {
28     return n * n * n; // cube local variable
29 }
30 // end function cubeByValue
```

cubeByValue nhận tham số
passed-by-value

Tính lập phương và trả về biến
địa phương (local variable) **n**

ig05_06.cpp
2 of 2)

ig05_06.cpp
output (1 of 1)

The original value of number is 5
The new value of number is 125

```
1 // Fig. 5.7: fig05_07.cpp
2 // Cube a variable using pass-by-reference
3 // with a pointer argument.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 void cubeByReference( int * ); // prototype
10
11 int main()
12 {
13     int number = 5;
14
15     cout << "The original value of number is " << number;
16
17     // pass address of number to cubeByReference
18     cubeByReference( &number );
19
20     cout << "\nThe new value of number is " << number << endl;
21
22     return 0; // indicates successful termination
23
24 } // end main
25
```

Prototype cho biết tham số là
con trỏ trỏ đến dữ liệu kiểu **int**

Dùng toán tử địa chỉ **&** để
truyền địa chỉ của **number** tới
cubeByReference

cubeByReference
thay đổi biến **number**

```

26 // calculate cube of *nPtr; modifies variable number in main
27 void cubeByReference( int *nPtr )
28 {
29     *nPtr = *nPtr * *nPtr * *nPtr; // cube
30
31 } // end function cubeByReference

```

cubeByReference nhận địa chỉ của biến kiểu **int**, tức là con trỏ trỏ đến một số **int**

The original value of number is 5
The new value of number is 125

Thay đổi và truy nhập biến kiểu **int** sử dụng toán tử thâm nhập *****

07.cpp

07.cpp

output (1 of 1)

5.5 Sử dụng **const** với con trỏ

- Tính chất của **const**
 - Giá trị của biến không thay đổi
 - **const** được sử dụng cho một biến khi hàm không cần thay đổi biến đó.
- Nguyên tắc quyền ưu tiên tối thiểu
 - Chỉ cho hàm đủ quyền truy nhập để thực hiện nhiệm vụ của mình, không cho nhiều quyền hơn.
- Bốn cách truyền con trỏ cho hàm
 - Con trỏ thường trỏ đến dữ liệu thường
 - Khả năng truy cập cao nhất
 - Con trỏ thường trỏ đến hằng dữ liệu
 - Hằng con trỏ trỏ đến dữ liệu thường
 - Hằng con trỏ trỏ đến hằng dữ liệu
 - Ít quyền truy cập nhất

fig05_10.cpp
(1 of 2)

```
1 // Fig. 5.10: fig05_10.cpp
2 // Converting lowercase letters to uppercase letters
3 // using a non-constant pointer to non-constant data.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 #include <cctype> // prototypes for islower and toupper
10
11 void convertToUppercase( char * );
12
13 int main()
14 {
15     char phrase[] = "characters and $32.98";
16
17     cout << "The phrase before conversion is: " << phrase;
18     convertToUppercase( phrase );
19     cout << "\nThe phrase after conversion is: "
20         << phrase << endl;
21
22     return 0; // indicates successful termination
23
24 } // end main
25
```

Con trỏ thường
đến dữ liệu thường

convertToUppercase
thay đổi biến **phrase**


```

26 // convert string to uppercase letters
27 void convertToUppercase( char *sPtr )
28 {
29     while ( *sPtr != '\0' ) { // current character
30
31         if ( islower( *sPtr ) ) // if character is lowercase,
32             *sPtr = toupper( *sPtr );
33
34         ++sPtr; // move sPtr to next character in string
35
36     } // end while
37
38 } // end function convertToUppercase

```

sPtr là con trỏ thường trỏ đến dữ liệu thường (2012)

Hàm **islower** trả về **true** nếu ký tự là chữ thường

Hàm **toupper** trả về chữ hoa nếu ký tự ban đầu là chữ thường; nếu không **toupper** trả về ký tự đó (chữ hoa)

Khi dùng toán tử **++** cho con trỏ trỏ đến mảng, địa chỉ vùng nhớ lưu trong con trỏ sẽ được sửa để con trỏ trỏ đến phần tử tiếp theo của mảng.

fig05_10.cpp
output (1 of 1)

The phrase before conversion is: characters and \$32.98
The phrase after conversion is: CHARACTERS AND \$32.98

```
1 // Fig. 5.11: fig05_11.cpp
2 // Printing a string one character at a time using
3 // a non-constant pointer to constant data.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 void printCharacters( const char * );
10
11 int main()
12 {
13     char phrase[] = "print characters of a string";
14
15     cout << "The string is:\n";
16     printCharacters( phrase );
17     cout << endl;
18
19     return 0; // indicates successful termination
20
21 } // end main
22
```

Tham số là con trỏ thường trỏ
đến hằng dữ liệu

Truyền con trỏ **phrase** cho
hàm **printCharacters**.

```
23 // sPtr cannot modify the character to which it points,  
24 // i.e., sPtr is a "read-only" pointer  
25 void printCharacters( const char *sPtr )  
26 {  
27     for ( ; *sPtr != '\\0'; sPtr++ )    // no initialization  
28         cout << *sPtr;  
29  
30 } // end function printCharacters
```

sPtr là con trỏ thường trỏ đến hằng dữ liệu; không thể thay đổi ký tự mà **sPtr** trỏ đến.

Tăng **sPtr** để trỏ đến ký tự tiếp theo.

The string is:
print characters of a string

```

1  // Fig. 5.12: fig05_12.cpp
2  // Attempting to modify data through a
3  // non-constant pointer to constant data.
4
5  void f( const int * ); // prototype
6
7  int main()
8  {
9      int y;
10
11     f( &y ); // f attempts illegal modification
12
13     return 0; // indicates successful
14
15 } // end main
16
17 // xPtr cannot modify the value of the variable
18 // to which it points
19 void f( const int *xPtr )
20 {
21     *xPtr = 100; // error: cannot modify a const object
22
23 } // end function f

```

Tham số là con trỏ thường trỏ đến hằng dữ liệu.

Truyền địa chỉ của biến **y** để thử thay đổi một cách không hợp lệ.

Cố thay đổi đối tượng hằng (const object) mà **xPtr** trỏ đến.

Lỗi sinh ra khi biên dịch.

fig05_12.cpp
(1 of 1)

fig05_12.cpp
output (1 of 1)

d:\cpphttp4_examples\ch05\Fig05_12.cpp(21) : error C2166:
l-value specifies const object

5.5 Sử dụng const với con trỏ

- **const** pointers - hằng con trỏ
 - Luôn trỏ đến vùng nhớ cố định
 - là mặc định cho tên mảng
 - Phải được khởi tạo khi khai báo

_13.cpp

1)

_13.cpp

it (1 of 1)

```
1 // Fig. 5.13: fig05_13.cpp
2 // Attempting to modify a constant pointer to
3 // non-constant data.
4
5 int main()
6 {
7     int x, y;
8
9     // ptr is a constant
10    // be modified through ptr
11    // same memory location.
12    int * const ptr = &x;
13
14    *ptr = 7; // allowed: *ptr is not const
15    ptr = &y; // error: ptr is const; cannot assign new address
16
17    return 0; // indicates successful termination
18
19 } // end main
```

ptr là hằng con trỏ trỏ tới số nguyên.

Có thể thay đổi x (trỏ bởi ptr) vì x không phải là hằng

Không thể cho ptr trỏ đến địa chỉ mới vì ptr là hằng

Dòng 15 sinh ra lỗi biên dịch vì thay đổi địa chỉ mới cho constant pointer.

d:\cpphttp4_examples\ch05\Fig05_13.cpp(15) : error C2166: l-value specifies const object

```
1 // Fig. 5.14: fig05_14.cpp
2 // Attempting to modify a constant pointer to constant data.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     int x = 5, y;
11
12     // ptr is a constant pointer to a constant integer.
13     // ptr always points to the same location; the integer
14     // at that location cannot be modified
15     const int *const ptr = &x;
16
17     cout << *ptr << endl;
18
19     *ptr = 7; // error: *ptr is const; cannot assign new value
20     ptr = &y; // error: ptr is const; cannot assign new address
21
22     return 0; // indicates successful termination
23
24 }
```

ptr là hằng con trỏ trỏ tới hằng số nguyên.

Không thể thay đổi **x** (trỏ bởi **ptr**) vì khai báo ***ptr** là hằng.

Không thể cho **ptr** trỏ đến địa chỉ mới vì **ptr** được khai báo là hằng.

5.6 Sắp xếp nổi bọt sử dụng truyền tham chiếu

- **bubbleSort** dùng con trỏ
 - Hàm **swap** truy nhập các phần tử của mảng
 - Các phần tử đơn của mảng: dữ liệu vô hướng (scalars)
 - Mặc định là pass by value
 - Truyền tham chiếu bằng toán tử địa chỉ **&**


```

1  // Fig. 5.15: fig05_15.cpp
2  // This program puts values into an array, sorts the values into
3  // ascending order, and prints the resulting array.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 void bubbleSort( int *, const int );    // prototype
14 void swap( int * const, int * const ); // prototype
15
16 int main()
17 {
18     const int arraySize = 10;
19     int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
20
21     cout << "Data items in original order\n";
22
23     for ( int i = 0; i < arraySize; i++ )
24         cout << setw( 4 ) << a[ i ];
25

```

fig05_15.cpp
 of 3)

```
26 bubbleSort( a, arraySize ); // sort the array
27
28 cout << "\nData items in ascending order\n";
29
30 for ( int j = 0; j < arraySize; j++ )
31     cout << setw( 4 ) << a[ j ];
32
33 cout << endl;
34
35 return 0; // indicates successful
36
37 } // end main
38
39 // sort an array of integers using bubble sort algorithm
40 void bubbleSort( int *array, const int size )
41 {
42     // loop to control passes
43     for ( int pass = 0; pass < size - 1; pass++ )
44
45         // loop to control comparisons during each pass
46         for ( int k = 0; k < size - 1; k++ )
47
48             // swap adjacent elements if they are out of order
49             if ( array[ k ] > array[ k + 1 ] )
50                 swap( &array[ k ], &array[ k + 1 ] );
```

Khai báo là **int *array** (thay vì **int array[]**) để cho hàm **bubbleSort** nhận mảng 1 chiều. Hai cách khai báo này là như nhau.

Nhận tham số kích thước của mảng; khai báo là **const** để chắc chắn rằng **size** sẽ không bị thay đổi.

```

51
52 } // end function bubbleSort
53
54 // swap values at memory locations to which
55 // element1Ptr and element2Ptr point
56 void swap( int * const element1Ptr, int * const element2Ptr )
57 {
58     int hold = *element1Ptr;
59     *element1Ptr = *element2Ptr;
60     *element2Ptr = hold;
61
62 } // end function swap

```

fig05_15.cpp
(3 of 3)

fig05_15.cpp
output (1 of 1)

Truyền tham chiếu, cho phép
hàm trao giá trị tại vùng nhớ.

```

Data items in original order
  2   6   4   8  10  12  89  68  45  37
Data items in ascending order
  2   4   6   8  10  12  37  45  68  89

```

5.6 Sắp xếp nổi bọt sử dụng truyền tham chiếu

- **sizeof**

- Toán tử trả về kích thước byte của toán hạng
- Với mảng, **sizeof** trả về giá trị
(kích thước 1 phần tử) * (số phần tử)
- Nếu **sizeof(int) = 4**, thì

```
int myArray[10];  
cout << sizeof(myArray);
```

sẽ in ra 40

- **sizeof** có thể dùng với

- Tên biến `cout << "sizeof c = " << sizeof c`
- Tên kiểu dữ liệu `cout << sizeof(char)`
- Hằng số

```
1 // Fig. 5.16: fig05_16.cpp
2 // Sizeof operator when used on an array name
3 // returns the number of bytes in the array.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 size_t getSize( double * ); // prototype
10
11 int main()
12 {
13     double array[ 20 ];
14
15     cout << "The number of bytes in the array is "
16          << sizeof( array );
17
18     cout << "\nThe number of bytes returned by getSize is "
19          << getSize( array ) << endl;
20
21     return 0; // indicates successful termination
22 } // end main
23
24
```

sizeof trả về tổng số byte của mảng.

Hàm **getSize** trả về số byte được dùng để lưu địa chỉ mảng **array**.

```
25 // return size of ptr
26 size_t getSize( double *ptr )
27 {
28     return sizeof( ptr );
29
30 } // end function getSize
```

sizeof trả về số byte của con trỏ.

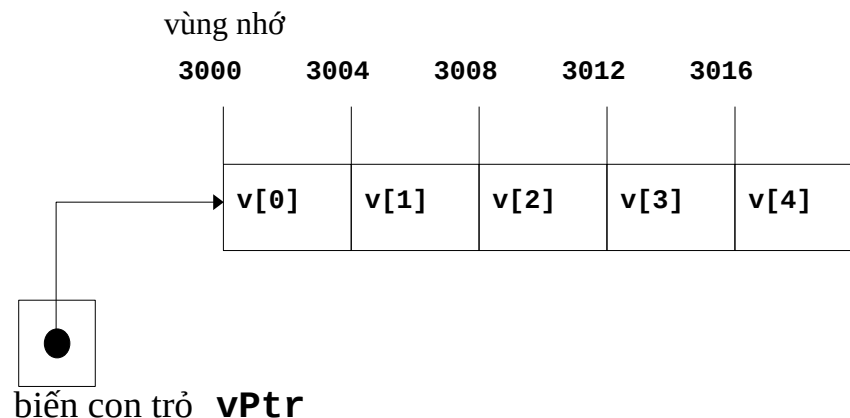
fig05_16.cpp
(2 of 2)

fig05_16.cpp
output (1 of 1)

The number of bytes in the array is 160
The number of bytes returned by getSize is 4

5.7 Các phép toán đối với con trỏ

- Các phép toán con trỏ
 - Tăng/giảm con trỏ (**++** hoặc **--**)
 - Cộng/trừ 1 số nguyên với 1 con trỏ (**+** hoặc **+=** , **-** hoặc **-=**)
 - Con trỏ có thể trừ lẫn nhau
 - Cộng trừ với con trỏ là vô nghĩa trừ khi dùng cho con trỏ mảng
- Ví dụ: Mảng 5 phần tử **int** trên máy dùng kiểu **int** 4 byte
 - **vPtr** trỏ đến phần tử thứ nhất **v[0]**, tại địa chỉ 3000
vPtr = 3000
 - **vPtr += 2**; trỏ **vPtr** tới **3008**
vPtr trỏ tới **v[2]**



5.7 Các phép toán đối với con trỏ

- Trừ con trỏ (Subtracting pointers)

- Trả về số phần tử giữa 2 địa chỉ

```
vPtr2 = v[ 2 ];  
vPtr  = v[ 0 ];  
vPtr2 - vPtr == 2
```

- Gán con trỏ (Pointer assignment)

- Một con trỏ có thể được gán cho con trỏ khác nếu cả hai cùng kiểu
- Nếu không cùng kiểu thì phải đổi kiểu (cast)
- Ngoại lệ: con trỏ tới **void** (kiểu **void ***)
 - con trỏ tổng quát, đại diện cho kiểu bất kỳ
 - không cần đổi kiểu để chuyển sang con trỏ sang dạng **void pointer**
 - Không thể (dùng *) lấy dữ liệu của con trỏ kiểu **void**

5.7 Các phép toán đối với con trỏ

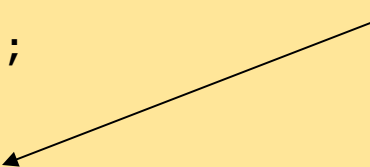
- So sánh con trỏ (Pointer comparison)
 - Sử dụng các toán tử quan hệ để so sánh địa chỉ chứa trong con trỏ
 - Ví dụ: có hai con trỏ trỏ đến hai phần tử của một mảng, chỉ ra con trỏ trỏ đến phần tử được đánh số thứ tự cao
 - So sánh là vô nghĩa trừ khi các con trỏ trỏ đến các phần tử của cùng một mảng
 - Thường dùng để xác định khi con trỏ có giá trị bằng 0 (null) (không trỏ đến đâu cả)

5.8 Quan hệ giữa Con trỏ và Mảng

- Mảng và con trỏ có quan hệ chặt chẽ
 - Tên mảng cũng như hằng con trỏ (constant pointer)
 - Có thể dùng chỉ số đối với các con trỏ
- Dùng con trỏ để truy nhập các phần tử mảng
 - Phần tử **b[n]** có thể truy nhập bởi ***(bPtr + n)**
 - ký hiệu pointer/offset
 - Địa chỉ
 - **&b[3]** tương đương **bPtr + 3**
 - Tên mảng có thể coi như con trỏ
 - **b[3]** tương đương ***(b + 3)**
 - Con trỏ có thể viết với cặp ngoặc vuông (ký hiệu pointer/subscript)
 - **bPtr[3]** tương đương **b[3]**

```
1  // Fig. 5.20: fig05_20.cpp
2  // Using subscripting and pointer notations with arrays.
3
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int main()
10 {
11     int b[] = { 10, 20, 30, 40 };
12     int *bPtr = b;    // set bPtr to point to array b
13
14     // output array b using array subscript notation
15     cout << "Array b printed with:\n"
16          << "Array subscript notation\n";
17
18     for ( int i = 0; i < 4; i++ )
19         cout << "b[" << i << "] = " << b[ i ] << '\n';
20
21     // output array b using the array name and
22     // pointer/offset notation
23     cout << "\nPointer/offset notation where "
24          << "the pointer is the array name\n";
25
```

Sử dụng ký hiệu chỉ số mảng.



```
26 for ( int offset1 = 0; offset1 < 4; offset1++ )
27     cout << "(b + " << offset1 << ") = "
28         << *( b + offset1 ) << '\n';
```

```
30 // output array b using bPtr and array subscript notation
```

```
31 cout << "\nPointer subscript notation\n";
```

Sử dụng tên mảng và ký hiệu pointer/offset.

```
33 for ( int j = 0; j < 4; j++ )
34     cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
```

```
36 cout << "\nPointer/offset notation\n";
```

Sử dụng ký hiệu chỉ số cho con trỏ.

```
38 // output array b using bPtr and pointer/offset notation
```

```
39 for ( int offset2 = 0; offset2 < 4; offset2++ )
40     cout << "(bPtr + " << offset2 << ") = "
41         << *( bPtr + offset2 ) << '\n';
```

```
43 return 0; // indicates successful termination
```

Sử dụng **bPtr** và ký hiệu pointer/offset.

```
45 } // end main
```

Array b printed with:

Array subscript notation

b[0] = 10

b[1] = 20

b[2] = 30

b[3] = 40

Pointer/offset notation where the pointer is the array name

*(b + 0) = 10

*(b + 1) = 20

*(b + 2) = 30

*(b + 3) = 40

Pointer subscript notation

bPtr[0] = 10

bPtr[1] = 20

bPtr[2] = 30

bPtr[3] = 40

Pointer/offset notation

*(bPtr + 0) = 10

*(bPtr + 1) = 20

*(bPtr + 2) = 30

*(bPtr + 3) = 40

fig05_20.cpp
output (1 of 1)

```

1  // Fig. 5.21: fig05_21.cpp
2  // Copying a string using array notation
3  // and pointer notation.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  void copy1( char *, const char * ); // prototype
10 void copy2( char *, const char * ); // prototype
11
12 int main()
13 {
14     char string1[ 10 ];
15     char *string2 = "Hello";
16     char string3[ 10 ];
17     char string4[] = "Good Bye";
18
19     copy1( string1, string2 );
20     cout << "string1 = " << string1 << endl;
21
22     copy2( string3, string4 );
23     cout << "string3 = " << string3 << endl;
24
25     return 0; // indicates successful termination

```

fig05_21.cpp
(1 of 2)

```

26
27 } // end main
28
29 // copy s2 to s1 using array notation
30 void copy1( char *s1, const char *s2 )
31 {
32     for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
33         ; // do nothing in body
34
35 } // end function copy1
36
37 // copy s2 to s1 using pointer notation
38 void copy2( char *s1, const char *s2 )
39 {
40     for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
41         ; // do nothing in body
42
43 } // end function copy2

```

Sử dụng chỉ số mảng để copy
xâu tại **s2** vào mảng ký tự **s1**.

fig05_21.cpp
(2 of 2)

fig05_21.cpp
output (1 of 1)

Sử dụng ký hiệu con trỏ để copy xâu
tại **s2** vào mảng ký tự **s1**.

Tăng cả hai con trỏ để trỏ đến
phần tử tiếp theo trong mảng
tương ứng.

```

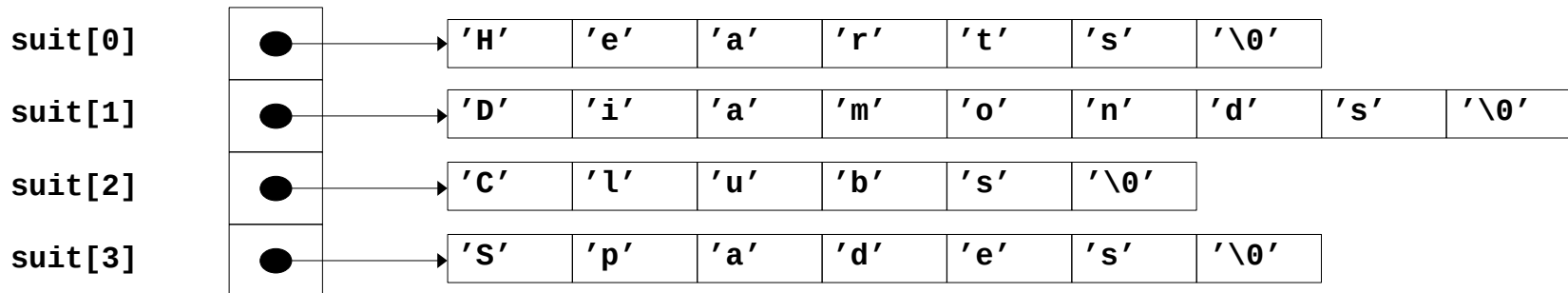
string1 = Hello
string3 = Good Bye

```

5.9 Mảng con trỏ

- Mảng chứa con trỏ
 - Thường dùng để lưu mảng của xâu

```
char *suit[ 4 ] = {"Hearts", "Diamonds",  
                  "Clubs", "Spades" };
```
 - Mỗi phần tử của **suit** trỏ đến **char *** (1 xâu)
 - Mảng không chứa xâu, chỉ trỏ đến xâu



- Mảng **suit** có kích thước cố định, nhưng xâu thì không

5.10 Ví dụ: Giải lập tráo bài và chia bài Tú-lơ-khơ

- Chương trình tráo bài (Card shuffling program)
 - Dùng một mảng gồm các con trỏ trỏ đến xâu để lưu trữ tên các chất (suit), i.e. cơ (hearts), rô (diamonds), pích (spades), tép (clubs)
 - Sử dụng một mảng hai chiều (hàng: chất, cột: giá trị)

		Ace	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten	Jack	Queen	King
		0	1	2	3	4	5	6	7	8	9	10	11	12
Hearts	0													
Diamonds	1													
Clubs	2													
Spades	3													

`deck[2][12]` biểu diễn K-tép

Clubs King

- Ghi các số từ 1-52 vào mảng để làm thứ tự chia các con bài

5.10 Ví dụ: Giải lập tráo bài và chia bài Tú-lơ-khơ

- Thuật toán tráo (shuffle) và chia (deal) bài

Ban đầu

Làm mịn

Làm mịn lần hai

Initialize the suit array
Initialize the face array
Initialize the deck array

Shuffle the deck
(tráo bài)

Deal 52 cards
(chia bài)

For each of the 52 cards

Place card number in randomly
selected unoccupied slot of deck
(Đặt chỉ số quân bài vào một ô
ngẫu nhiên còn trống trong desk)

For each of the 52 cards

Find card number in deck array
and print face and suit of card
(Tìm chỉ số quân bài trong mảng
desk và in ra số hiệu và chất của
quân bài)

Choose slot of deck randomly

While chosen slot of deck has been
previously chosen (Trong khi ô
vừa chọn đã bị chọn từ trước)

Choose slot of deck randomly
(chọn ngẫu nhiên một ô)

Place card number in chosen slot
of deck (đặt chỉ số quân bài vào ô
được chọn)

For each slot of the deck array

If slot contains card number
Print the face and suit of the
card

```
1  // Fig. 5.24: fig05_24.cpp
2  // Card shuffling dealing program.
3  #include <iostream>
4
5  using std::cout;
6  using std::left;
7  using std::right;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 #include <cstdlib> // prototypes for rand and srand
14 #include <ctime>  // prototype for time
15
16 // prototypes
17 void shuffle( int [][] [ 13 ] );
18 void deal( const int [][] [ 13 ], const char *[], const char
*[] );
19
20 int main()
21 {
22     // initialize suit array
23     const char *suit[ 4 ] =
24         { "Hearts", "Diamonds", "Clubs", "Spades" };
25
```

mảng **suit** chứa các con trỏ
trỏ đến các mảng **char**.

```
26 // initialize face array
27 const char *face[ 13 ] =
28     { "Ace", "Deuce", "Three", "Four",
29       "Five", "Six", "Seven", "Eight",
30       "Nine", "Ten", "Jack", "Queen", "King" };
31
32 // initialize deck array
33 int deck[ 4 ][ 13 ] = { 0 };
34
35 srand( time( 0 ) );           // seed random number generator
36
37 shuffle( deck );
38 deal( deck, face, suit );
39
40 return 0; // indicates successful termination
41
42 } // end main
43
```

mảng **face** chứa các con trỏ
trỏ đến các mảng **char**.

```
44 // shuffle cards in deck
45 void shuffle( int wDeck[][ 13 ] )
46 {
47     int row;
48     int column;
49
50     // for each of the 52 cards, choose slot of deck randomly
51     for ( int card = 1; card <= 52; card++ ) {
52
53         // choose new random location until unoccupied slot found
54         do {
55             row = rand() % 4;
56             column = rand() % 13;
57         } while ( wDeck[ row ][ column ] != 0 ); // end do/while
58
59         // place card number in chosen slot of deck
60         wDeck[ row ][ column ] = card;
61
62     } // end for
63
64 } // end function shuffle
65
```

← Vị trí hiện tại có dòng và cột được chọn ngẫu nhiên.

```
66 // deal cards in deck
67 void deal( const int wDeck[][ 13 ], const char *wFace[],
68           const char *wSuit[] )
69 {
70     // for each of the 52 cards
71     for ( int card = 1; card <= 52; card++ )
72
73         // loop through rows of wDeck
74         for ( int row = 0; row <= 3; row++ )
75
76             // loop through columns of wDeck for current row
77             for ( int column = 0; column <= 12; column++ )
78
79                 // if slot contains current card, display card
80                 if ( wDeck[ row ][ column ] == card ) {
81                     cout << setw( 5 ) << right << wFace[ column ]
82                         << " of " << setw( 8 ) << left
83                         << wSuit[ row ]
84                         << ( card % 2 == 0 ? '\n' : '\t' );
85
86                 } // end if
87
88 } // end function deal
```

Căn lề phải trong một vùng
gồm 5 ký tự.

Căn lề trái trong một vùng gồm
8 ký tự.

Nine of Spades	Seven of Clubs
Five of Spades	Eight of Clubs
Queen of Diamonds	Three of Hearts
Jack of Spades	Five of Diamonds
Jack of Diamonds	Three of Diamonds
Three of Clubs	Six of Clubs
Ten of Clubs	Nine of Diamonds
Ace of Hearts	Queen of Hearts
Seven of Spades	Deuce of Spades
Six of Hearts	Deuce of Clubs
Ace of Clubs	Deuce of Diamonds
Nine of Hearts	Seven of Diamonds
Six of Spades	Eight of Diamonds
Ten of Spades	King of Hearts
Four of Clubs	Ace of Spades
Ten of Hearts	Four of Spades
Eight of Hearts	Eight of Spades
Jack of Hearts	Ten of Diamonds
Four of Diamonds	King of Diamonds
Seven of Hearts	King of Spades
Queen of Spades	Four of Hearts
Nine of Clubs	Six of Diamonds
Deuce of Hearts	Jack of Clubs
King of Clubs	Three of Spades
Queen of Clubs	Five of Clubs
Five of Hearts	Ace of Diamonds

fig05_24.cpp
output (1 of 1)

5.11 Con trỏ tới hàm (Function Pointer)

- Con trỏ tới hàm
 - chứa địa chỉ của hàm
 - Tên mảng có giá trị là địa chỉ của phần tử đầu tiên của mảng
 - Tương tự, tên hàm có giá trị là địa chỉ bắt đầu của đoạn mã định nghĩa hàm
- Các con trỏ tới hàm có thể
 - được truyền vào trong hàm
 - được trả về từ hàm
 - được lưu trong mảng
 - được gán cho các con trỏ hàm khác

5.11 Con trỏ tới hàm

- Gọi hàm bằng con trỏ tới hàm
 - giả sử `compare` được khai báo là con trỏ tới hàm có kiểu tham số và kiểu trả về như sau:
 - **`bool (*compare) (int, int)`**
 - gọi hàm bằng một trong hai cách
 - **`(*compare) (int1, int2)`**
 - thêm nhập con trỏ để chạy hàm được con trỏ trỏ tới
- HOẶC
 - **`compare(int1, int2)`**
 - dễ nhầm lẫn
 - người dùng có thể tưởng **`compare`** là tên của hàm thực trong chương trình

fig05_25.cpp
(1 of 5)

```
1  // Fig. 5.25: fig05_25.cpp
2  // Multipurpose sorting program using function pointers.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 // prototypes
14 void bubble( int [], const int, bool (*)( int, int ) );
15 void swap( int * const, int * const );
16 bool ascending( int, int );
17 bool descending( int, int );
18
19 int main()
20 {
21     const int arraySize = 10;
22     int order;
23     int counter;
24     int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37
25 }
```

Tham số thứ ba là con trỏ tới một hàm nhận 2 tham số **int** và trả về kết quả kiểu **bool**.

```
26  cout << "Enter 1 to sort in ascending order,\n"
27      << "Enter 2 to sort in descending order: ";
28  cin >> order;
29  cout << "\nData items in original order\n";
30
31  // output original array
32  for ( counter = 0; counter < arraySize; counter++ )
33      cout << setw( 4 ) << a[ counter ];
34
35  // sort array in ascending order; pass function ascending
36  // as an argument to specify ascending sorting order
37  if ( order == 1 ) {
38      bubble( a, arraySize, ascending );
39      cout << "\nData items in ascending order\n";
40  }
41
42  // sort array in descending order; pass function descending
43  // as an argument to specify descending sorting order
44  else {
45      bubble( a, arraySize, descending );
46      cout << "\nData items in descending order\n";
47  }
48
```

```
49 // output sorted array
50 for ( counter = 0; counter < arraySize; counter++ )
51     cout << setw( 4 ) << a[ counter ];
52
53 cout << endl;
54
55 return 0; // indicates successful termination
56
57 } // end main
58
59 // multipurpose bubble sort; parameter compare is a pointer to
60 // the comparison function that determines sorting order
61 void bubble( int work[], const int size,
62             bool (*compare)( int, int ) )
63 {
64     // loop to control passes
65     for ( int pass = 1; pass < size; pass++ )
66     {
67         // loop to control number of comparisons per pass
68         for ( int count = 0; count < size - 1; count++ )
69         {
70             // if adjacent elements are out of order, swap them
71             if ( (*compare)( work[ count ], work[ count + 1 ] ) )
72                 swap( &work[ count ], &work[ count + 1 ] );
73         }
74     } // end function bubble
```

compare là con trỏ tới một hàm nhận 2 tham số kiểu **int** và trả về giá trị kiểu **bool**.

Dùng ngoặc để chỉ rõ đây là con trỏ tới hàm

gọi hàm **compare** được truyền vào; thêm nhập con trỏ để chạy hàm.

```

75
76 // swap values at memory locations to which
77 // element1Ptr and element2Ptr point
78 void swap( int * const element1Ptr, int * const element2Ptr )
79 {
80     int hold = *element1Ptr;
81     *element1Ptr = *element2Ptr;
82     *element2Ptr = hold;
83
84 } // end function swap
85
86 // determine whether elements are out of order
87 // for an ascending order sort
88 bool ascending( int a, int b )
89 {
90     return b < a; // swap if b is less than a
91
92 } // end function ascending
93
94 // determine whether elements are out of order
95 // for a descending order sort
96 bool descending( int a, int b )
97 {
98     return b > a; // swap if b is greater than a
99
100 } // end function descending

```

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in ascending order

2 4 6 8 10 12 37 45 68 89

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in descending order

89 68 45 37 12 10 8 6 4 2

5.11 Con trỏ tới hàm

- Mảng gồm các con trỏ hàm
 - Thường dùng cho các hệ thống điều khiển bằng thực đơn (menu-driven system)
 - Các con trỏ đến từng hàm được lưu trong mảng con trỏ hàm
 - các hàm đều phải có kiểu dữ liệu trả về giống nhau, và kiểu dữ liệu của tham số như nhau
 - Ánh xạ
(lựa chọn thực đơn → chỉ số trong mảng con trỏ tới hàm)

```
1  // Fig. 5.26: fig05_26.cpp
2  // Demonstrating an array of pointers to functions.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function prototypes
10 void function1( int );
11 void function2( int );
12 void function3( int );
13
14 int main()
15 {
16     // initialize array of 3 pointers to functions that each
17     // take an int argument and return void
18     void (*f[ 3 ])( int ) = { function1, function2, function3 };
19
20     int choice;
21
22     cout << "Enter a number between 0 and 2, 3 to end: ";
23     cin >> choice;
24
```

Mảng được khởi tạo với tên của ba hàm, tên của hàm chính là con trỏ.

```
25 // process user's choice
26 while ( choice >= 0 && choice < 3 ) {
27
28     // invoke function at location choice in array f
29     // and pass choice as an argument
30     (*f[ choice ])( choice );
31
32     cout << "Enter a number between 0 and 2, 3 to end: ";
33     cin >> choice;
34 }
35
36 cout << "Program execution completed." << endl;
37
38 return 0; // indicates successful termination
39
40 } // end main
41
42 void function1( int a )
43 {
44     cout << "You entered " << a
45         << " so function1 was called\n\n";
46
47 } // end function1
48
```

Gọi hàm được chọn bằng cách thâm nhập vào (dereferencing) phần tử tương ứng trong mảng.


```

49 void function2( int b )
50 {
51     cout << "You entered " << b
52         << " so function2 was called\n\n";
53
54 } // end function2
55
56 void function3( int c )
57 {
58     cout << "You entered " << c
59         << " so function3 was called\n\n";
60
61 } // end function3

```

fig05_26.cpp
(3 of 3)

fig05_26.cpp
output (1 of 1)

Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.

5.12.1 Tổng kết về ký tự và xâu ký tự

- Hằng ký tự - Character constant
 - Giá trị nguyên biểu diễn dưới dạng một ký tự viết trong 2 dấu nháy
 - 'z' là giá trị nguyên của ký tự z
 - Mã **122** trong bảng mã ASCII
- Xâu ký tự - String
 - Chuỗi các ký tự được coi như là một single unit
 - Có thể bao gồm chữ cái, chữ số, ký tự đặc biệt +, -, * ...
 - Hằng xâu ký tự - String literal (string constants)
 - Viết trong cặp nháy kép, ví dụ: **"I like C++"**
 - Mảng của các ký tự, kết thúc với ký tự rỗng (null character) **'\0'**
 - Xâu là một hằng con trỏ (constant pointer)
 - Trỏ đến ký tự đầu tiên của xâu
 - Giống như với mảng

5.12.1 Tổng kết về ký tự và xâu ký tự

- Gán giá trị cho xâu - String assignment
 - Mảng của ký tự
 - **char color[] = "blue";**
 - Tạo mảng **color** 5 phần tử kiểu **char**
 - phần tử cuối cùng là **'\0'**
 - Biến kiểu **char ***
 - **char *colorPtr = "blue";**
 - Tạo con trỏ **colorPtr** trỏ đến chữ **b** trong xâu **"blue"**
 - **"blue"** ở đâu đó trong bộ nhớ
 - Một cách khác cho mảng ký tự
 - **char color[] = { 'b', 'l', 'u', 'e', '\0' };**

5.12.1 Tổng kết về ký tự và xâu ký tự

- Đọc xâu
 - Đọc dữ liệu cho mảng ký tự **word[20]**
cin >> word
 - Đọc các ký tự cho đến khi gặp ký tự trắng hoặc EOF
 - Xâu có thể vượt quá kích thước mảng
cin >> setw(20) >> word;
 - Đọc 19 ký tự (để lại chỗ cho '**\0**')
- **cin.getline**
 - Đọc 1 dòng văn bản
 - **cin.getline(array, size, delimiter);**
 - 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
 - Ví dụ
char sentence[80];
cin.getline(sentence, 80, '\n');

5.12.2 Các hàm xử lý chuỗi ký tự

- Thư viện xử lý chuỗi **<cstring>** cung cấp các hàm
 - thao tác với dữ liệu kiểu chuỗi
 - so sánh chuỗi
 - tìm kiếm trên chuỗi các ký tự hoặc chuỗi khác
 - chia chuỗi thành các từ tố (tokenize strings)

5.12.2 Các hàm xử lý chuỗi ký tự

char *strcpy(char *s1, const char *s2);	Copy chuỗi s2 vào chuỗi s1 . Trả về giá trị của s1 .
char *strncpy(char *s1, const char *s2, size_t n);	Copy nhiều nhất n ký tự của chuỗi s2 vào chuỗi s1 . Trả về giá trị của s1 .
char *strcat(char *s1, const char *s2);	Thêm chuỗi s2 vào sau chuỗi s1 . Ký tự đầu tiên của s2 ghi đè lên ký tự null của s1 . Trả về giá trị của s1 .
char *strncat(char *s1, const char *s2, size_t n);	Thêm chuỗi nhiều nhất là n ký tự của s2 vào sau chuỗi s1 . Ký tự đầu tiên của s2 ghi đè lên ký tự null của s1 . Trả về giá trị của s1 .
int strcmp(const char *s1, const char *s2);	So sánh chuỗi s1 và chuỗi s2 . Hàm trả về giá trị 0, nhỏ hơn 0, hoặc lớn hơn 0 nếu s1 bằng, nhỏ hơn hoặc lớn hơn s2 .

5.12.2 Các hàm xử lý chuỗi ký tự

<pre>int strncmp(const char *s1, const char *s2, size_t n);</pre>	<p>So sánh n ký tự chuỗi s1 và chuỗi s2. Hàm trả về giá trị 0, nhỏ hơn 0 hoặc lớn hơn 0 nếu s1 bằng, nhỏ hơn hoặc lớn hơn s2.</p>
<pre>char *strtok(char *s1, const char *s2);</pre>	<p>Một chuỗi lời gọi đến strtok chia chuỗi s1 thành các “tokens”—từ tố, chẳng hạn các từ trong một dòng văn bản—phân tách nhau bởi các ký tự chứa trong chuỗi s2. Lời gọi đầu tiên lấy s1 làm tham số thứ nhất, các lời gọi tiếp sau (với NULL là tham số thứ nhất) tiếp tục lấy các từ tố từ chính chuỗi đó. Mỗi lời gọi trả về một con trỏ tới từ tố vừa nhận được. Nếu không còn từ tố nào, hàm sẽ trả về giá trị NULL.</p>
<pre>size_t strlen(const char *s);</pre>	<p>Xác định độ dài của chuỗi s. Trả về số ký tự của chuỗi (không tính ký tự null).</p>

5.12.2 Các hàm xử lý chuỗi ký tự

- Copy chuỗi
 - **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 chuỗi và ký tự null
 - **char *strncpy(char *s1, const char *s2, size_t n)**
 - Xác định rõ số ký tự được copy từ chuỗi vào mảng
 - Không nhất thiết copy ký tự null

fig05_28.cpp
(1 of 2)

```
1 // Fig. 5.28: fig05_28.cpp
2 // Using strcpy and strncpy.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstring> // prototypes for strcpy and strncpy
9
10 int main()
11 {
12     char x[] = "Happy Birthday to You";
13     char y[ 25 ];
14     char z[ 15 ];
15
16     strcpy( y, x ); // copy contents of x into y
17
18     cout << "The string in array x is: " << x
19          << "\nThe string in array y is: " << y << '\n';
20
21     // copy first 14 characters of x into z
22     strncpy( z, x, 14 ); // does not copy null character
23     z[ 14 ] = '\0'; // append '\0' to z's contents
24
25     cout << "The string in array z is: " << z << endl;
```

`<cstring>` chứa prototype
cho `strcpy` và `strncpy`.

Copy toàn bộ xâu trong mảng
`x` vào mảng `y`.

Copy 14 ký tự đầu tiên của mảng
`x` vào mảng `y`. Chú ý rằng lệnh
này không viết ký tự null.

Thêm ký tự null.

```
26
27     return 0; // indicates successful termination
28
29 }
```

fig05_28.cpp
(2 of 2)

fig05_28.cpp
output (1 of 1)

Xâu gốc.

Copy xâu bằng **strcpy**.

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

Copy 14 ký tự đầu tiên
bằng **strncpy**.

5.12.2 Các hàm xử lý chuỗi ký tự

- Nối chuỗi - Concatenating strings
 - **char *strcat(char *s1, const char *s2)**
 - Nối chuỗi thứ hai vào sau chuỗi 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 chuỗi.
 - **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

fig05_29.cpp
(1 of 2)

```
1 // Fig. 5.29: fig05_29.cpp
2 // Using strcat and strncat.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstring> // prototypes for strcat and strncat
9
10 int main()
11 {
12     char s1[ 20 ] = "Happy ";
13     char s2[] = "New Year ";
14     char s3[ 40 ] = "";
15
16     cout << "s1 = " << s1 << "\ns2 = " << s2;
17
18     strcat( s1, s2 ); // concatenate s2 to s1
19
20     cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1
21         << "\ns2 = " << s2;
22
23     // concatenate first 6 characters of s1 to s3
24     strncat( s3, s1, 6 ); // places '\0' after last character
25
```

`<cstring>` chứa prototype
cho **strcat** và **strncat**.

Thêm **s2** vào sau **s1**.

Thêm 6 ký tự đầu tiên của **s1** vào sau **s3**.

```

26  cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
27      << "\ns3 = " << s3;
28
29  strcat( s3, s1 ); // concatenate s1 to s3
30  cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
31      << "\ns3 = " << s3 << endl;
32
33  return 0; // indicates successful termination
34
35 } // end main

```

Thêm **s1** vào sau **s3**.

fig05_29.cpp
(2 of 2)

fig05_29.cpp
output (1 of 1)

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

5.12.2 Các hàm xử lý chuỗi ký tự

- So sánh chuỗi - Comparing strings
 - Các ký tự được biểu diễn bằng mã dạng số (numeric code)
 - các mã đó được dùng để so sánh các chuỗi ký tự
 - Các bộ mã ký tự (Character codes / character sets)
 - ASCII “American Standard Code for Information Interchange”
 - EBCDIC “Extended Binary Coded Decimal Interchange Code”
- Các hàm so sánh chuỗi
 - **int strcmp(const char *s1, const char *s2)**
 - So sánh từng ký tự một, theo thứ tự từ điển
 - Trả về
 - 0 nếu chuỗi bằng nhau
 - Giá trị âm nếu chuỗi thứ nhất nhỏ hơn chuỗi thứ hai
 - Giá trị dương nếu chuỗi thứ nhất lớn hơn chuỗi thứ hai
 - **int strncmp(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

```
1 // Fig. 5.30: fig05_30.cpp
2 // Using strcmp and strncmp.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 #include <cstring> // prototypes for strcmp and strncmp
13
14 int main()
15 {
16     char *s1 = "Happy New Year";
17     char *s2 = "Happy New Year";
18     char *s3 = "Happy Holidays";
19
20     cout << "s1 = " << s1 << "\ns2 = " << s2
21         << "\ns3 = " << s3 << "\n\nstrcmp(s1, s2) = "
22         << setw( 2 ) << strcmp( s1, s2 )
23         << "\nstrcmp(s1, s3) = " << setw( 2 )
24         << strcmp( s1, s3 ) << "\nstrcmp(s3, s1) = "
25         << setw( 2 ) << strcmp( s3, s1 );
```

`<cstring>` chứa prototype
cho `strcmp` và `strncmp`.

So sánh **s1** với **s2**.

So sánh **s1** với **s3**.

So sánh **s3** với **s1**.

```

26
27     cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
28         << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = "
29         << setw( 2 ) << strncmp( s1, s3, 7 )
30         << "\nstrncmp(s3, s1, 7) = "
31         << setw( 2 ) << strncmp( s3, s1, 7 ) << endl;
32
33     return 0; // indicates successful termination
34
35 } // end main

```

So sánh 6 ký tự đầu tiên của **s1** với **s3**.

fig05_30.cpp
(2 of 2)

fig05_30.cpp
(1 of 1)

So sánh 7 ký tự đầu tiên của **s1** với **s3**.

So sánh 7 ký tự đầu tiên của **s3** với **s1**.

```

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

```


5.12.2 Các hàm xử lý chuỗi ký tự

- Phân tích từ tổ - Tokenizing
 - Chia chuỗi thành các từ tổ, phân tách bởi các ký tự ngăn cách (delimiting character)
 - Các từ tổ thường là các đơn vị logic (logical units), chẳng hạn các từ (tách nhau bởi các dấu trống)
 - **"This is my string"** có 4 từ tổ (tách nhau bởi các dấu trống)
 - **char *strtok(char *s1, const char *s2)**
 - Cần gọi nhiều lần
 - Lần gọi đầu cần 2 tham số, chuỗi cần phân tích từ tổ và chuỗi chứa các ký tự ngăn cách
 - Tìm ký tự ngăn cách tiếp theo và thay bằng ký tự null
 - Những lời gọi tiếp theo tiếp tục phân tích từ tổ trên chuỗi đó
 - Gọi hàm với tham số thứ nhất là **NULL**

fig05_31.cpp
(1 of 2)

```
1 // Fig. 5.31: fig05_31.cpp
2 // Using strtok.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstring> // prototype for strtok
9
10 int main()
11 {
12     char sentence[] = "This is a sentence with 7 tokens";
13     char *tokenPtr;
14
15     cout << "The string to be tokenized is:\n" << sentence
16          << "\n\nThe tokens are:\n\n";
17
18     // begin tokenization of sentence
19     tokenPtr = strtok( sentence, " " );
20
```

<cstring> chứa prototype
cho **strtok**.

Lời gọi **strtok** đầu tiên
khởi đầu việc phân tích từ tổ.

```
21 // continue tokenizing sentence until tokenPtr becomes NULL
22 while ( tokenPtr != NULL ) {
23     cout << tokenPtr << '\n';
24     tokenPtr = strtok( NULL, " " ); // get next token
25
26 } // end while
27
28 cout << "\nAfter strtok, sentence = " << sentence << endl;
29
30 return 0; // indicates successful termination
31
32 } // end main
```

Các lời gọi **strtok** tiếp sau với **NULL** là tham số đầu để tiếp tục việc phân tích từ tố trên xâu **sentence**.

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

5.12.2 Các hàm xử lý chuỗi ký tự

- Xác định độ dài chuỗi
 - **size_t strlen(const char *s)**
 - Trả về số ký tự của chuỗi
 - Không tính đến ký tự null

```
1 // Fig. 5.32: fig05_32.cpp
2 // Using strlen.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstring> // prototype for strlen
9
10 int main()
11 {
12     char *string1 = "abcdefghijklmnopqrstuvwxyz";
13     char *string2 = "four";
14     char *string3 = "Boston";
15
16     cout << "The length of \"" << string1
17          << "\" is " << strlen( string1 )
18          << "\nThe length of \"" << string2
19          << "\" is " << strlen( string2 )
20          << "\nThe length of \"" << string3
21          << "\" is " << strlen( string3 ) << endl;
22
23     return 0; // indicates success
24
25 } // end main
```

<cstring> chứa prototype
cho **strlen**.

Sử dụng **strlen** để xác định
độ dài chuỗi.

The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6