Ngôn ngữ lập trình

Bài 2:
Các cấu trúc điều khiển, mảng và con trỏ

Giảng viên: Lê Nguyễn Tuấn Thành

Email: thanhlnt@tlu.edu.vn

Bộ Môn Công Nghệ Phần Mềm – Khoa CNTT

Trường Đại Học Thủy Lợi

Nội dung

1. Cấu trúc rẽ nhánh
2. Cấu trúc lặp
3. Mảng (Array)
4. Con trỏ (Pointer)

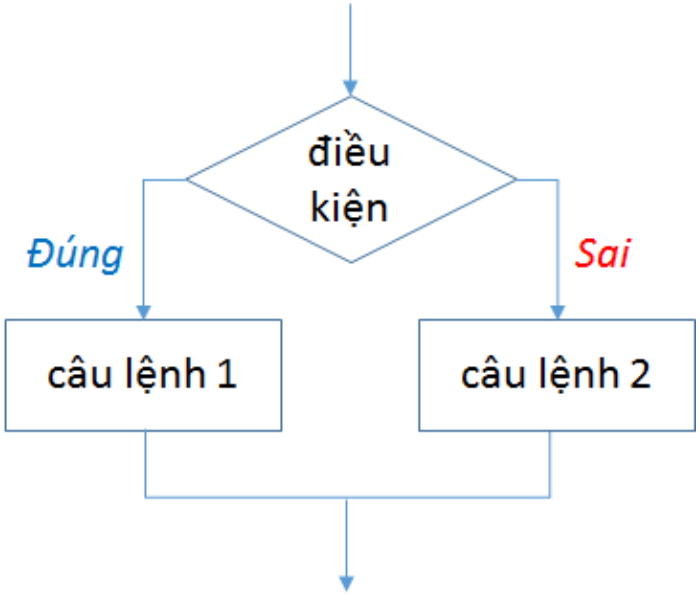
Bài giảng có sử dụng hình vẽ trong cuốn sách "*Absolute C++*. W. Savitch, Addison Wesley, 2002"

1. CẤU TRÚC RẼ NHÁNH

1.1. Cấu trúc rẽ nhánh với **if-else**

- **Mục đích**
  - Diễn đạt sự lựa chọn một trong nhiều nhánh, phụ thuộc vào giá trị của câu điều kiện
- Cú pháp:

```
if (<boolean_expression>)
        <yes_statement>
else
        <no_statement>
```



- Ví dụ:

```
if (hrs > 40)
        grossPay = rate*40 + 1.5*rate*(hrs-40);
else
        grossPay = rate*hrs;
```

Câu lệnh phức hợp

- Mỗi nhánh trong if-else ở slide trước chỉ có một câu lệnh
- Để ghép nhiều câu lệnh trong một nhánh, sử dụng { }. Tập lệnh khi đó được gọi là một khối (block)
- Ví dụ:

```
if (myScore > yourScore)
{
        cout << "I win!\n";
        wager = wager + 100;
}
else
{
        cout << "I wish these were golf scores.\n";
        wager = 0;
}
```

Một vài lưu ý

- **Toán tử "=" khác toán tử "==" như thế nào?**
  - "=" dùng để gán giá trị cho các biến
  - "==" dùng để so sánh hai biểu thức
- **Mệnh đề else có bắt buộc không?**
  - Ví dụ:

```
if (sales >= minimum)
    salary = salary + bonus;
cout << "Salary = %" << salary;
```

Câu lệnh lồng nhau (nested)

- Chúng ta có thể lồng một cặp if-else trong một nhánh của cặp if-else khác
- Ví dụ:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```

Đa rẽ nhánh (if - else if - else)

Multiway if–else Statement

SYNTAX

```
if (Boolean_Expression_1)
     Statement_1
else if (Boolean_Expression_2)
     Statement_2
           .
           .
           .
else if (Boolean_Expression_n)
     Statement_n
else
     Statement_For_All_Other_Possibilities
```

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature < -10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first `true` Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.

Bài tập với cấu trúc rẽ nhánh if-else

Bài 1: Viết một chương trình C++ để nhắc người dùng nhập 3 số nguyên và tìm giá trị lớn nhất.

Bài 2: Nhập vào một số nguyên tương ứng với một tháng trong năm và in ra màn hình số ngày trong tháng đó.

ví dụ:

input: 1

output: tháng 1 có 31 ngày

Câu hỏi:

Nếu có quá nhiều nhánh rẽ thì ngoài sử dụng if-else, C++ còn cung cấp cách nào nữa không?

1.2. Rẽ nhánh với lệnh witch (1/2)

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
                    .
                    .
                    .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

You need not place a **break** *statement in each case. If you omit a* **break**, *that case continues until a* **break** *(or the end of the* **switch** *statement) is reached.*

Rẽ nhánh với lệnh witch (2/2)

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
  }
```

If you forget this **break**, *then passenger cars will pay $1.50.*

Lệnh switch: câu hỏi

```
switch (aChar)
{
    case "A":
    case "a":
        cout << "Excellent: you got an "A"!\n";
        break;
    case "B":
    case "b":
        cout << "Good: you got a "B"!\n";
        break;
}
```

Nếu giá trị của aChar là "A" hoặc "B" thì kết quả in ra là gì ?

Toán tử điều kiện (Conditional/ternary operator)

- Thay thế cho mệnh đề if-else đơn giản với hai toán tử "?" và ":"
- Cấu trúc:

if (condition)
if_true;
else
if_false;

Có thể thay bằng một lệnh

(condition) ? (if_true) : (if_false)

Bài tập: viết hàm trả lại số lớn nhất trong hai số

- *#define MAX(a, b) ((a > b) ? a : b)*
- *#define MIN(a, b) ((a < b) ? a : b)*
- Giá trị của a trong câu lệnh sau (với x > 0) ***a = x ? : y;***

2. CẤU TRÚC LẶP

2. Cấu trúc lặp (loop)

Các cấu trúc lặp trong C++

1. *While*
2. *do-while*
3. *for*

Cấu trúc lặp với while

A while STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)
    Statement
```

A while STATEMENT WITH A MULTISTATEMENT BODY

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
}
```

```
int count = 0;                          // Initialization
while (++count < 3)             // Loop Condition
{
        cout << "Hi ";                  // Loop Body
}
```

Chuỗi "Hi" sẽ được in ra màn hình bao nhiêu lần?

Cấu trúc lặp với do-while (1/2)

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do
    Statement
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTATEMENT BODY

```
do
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
} while (Boolean_Expression);
```

Do not forget the final semicolon.

Cấu trúc lặp với do-while (2/2)

```
int count = 0;                  // Initialization
do
{
        cout << "Hi "; // Loop Body
} while (++count < 3);          // Loop Condition
```

Chuỗi "Hi" sẽ được in ra màn hình bao nhiêu lần?

So sánh **while** và **do-while**

- Khá giống nhau, nhưng một khác biệt quan trọng
  - **while**: kiểm tra điều kiện logic TRƯỚC KHI thực thi lệnh bên trong
  - **do-while**: kiểm tra điều kiện logic SAU KHI đã thực thi lệnh bên trong

Cấu trúc lặp với **for**

- Cú pháp

  for (Init_Action; Bool_Expression; Update_Action)

  Body_Statement

- Ví dụ:

```
for (count=0; count<3; count++)
{
   cout << "Hi ";        // Loop Body
}
```

- Chuỗi "Hi" sẽ được in ra màn hình bao nhiêu lần?
- Điều gì xảy ra với câu lệnh sau:
  - *for (; ;) { cout << "Hi"; }*

Một vài chú ý với cấu trúc lặp (1/2)

- Biểu thức điều kiện của vòng lặp có thể là BẤT KỲ biểu thức logic nào
- Ví dụ:

```
while (count<3 && done!=0)
{
    // Do something
}
-----
for (index=0; index<10 && entry!=99)
{
    // Do something
}
```

Một vài chú ý với cấu trúc lặp (2/2)

- Vòng lặp vô hạn
- Ví dụ:

```
while (1)
{
    cout << "Hello ";
}
---
for ( ;;)
{
    cout << “Hello”;
}
```

Lệnh **break** và **continue**

- Lệnh *break*: ép buộc thoát khỏi vòng lặp ngay lập tức
- Lệnh *continue*: bỏ qua phần còn lại trong thân vòng lặp (loop body)
- Hai lệnh này vi phạm luồng chạy tự nhiên => chỉ dùng khi thật cần thiết

Minh họa lệnh **continue**

```
#include <iostream>
using namespace std;

int main ()
{
   // Khai bao bien cuc bo:
   int a = 10;

   // vong lap do...while
   do
   {
       if( a == 15)
       {
          // nhay qua buoc lap.
          a = a + 1;
          continue;
       }
       cout << "Gia tri cua a la: " << a << endl;
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

```
Gia tri cua a la: 10
Gia tri cua a la: 11
Gia tri cua a la: 12
Gia tri cua a la: 13
Gia tri cua a la: 14
Gia tri cua a la: 16
Gia tri cua a la: 17
Gia tri cua a la: 18
Gia tri cua a la: 19
```

Cấu trúc lặp lồng nhau

- ***Nhớ lại***: bất kỳ mệnh đề hợp lệ nào trong C++ có thể được đặt bên trong vòng lặp
- Có thể dùng "{ }" hoặc thụt lề (indent) để biểu diễn vòng lặp lồng nhau (nested loops)
- Ví dụ:

```
for (outer=0; outer<5; outer++)
    for (inner=7; inner>2; inner--)
        cout << outer << inner;
```

hoặc

```
for (outer=0; outer<5; outer++)
{
    for (inner=7; inner>2; inner--)
    {
        cout << outer << inner;
    }
}
```

Bài tập với cấu trúc lặp

- Viết chương trình tìm TẤT CẢ các số nguyên tố nhỏ hơn một số nguyên dương nhập vào từ bàn phím

  Input: N – số nguyên dương

  Output: in ra tất cả các số nguyên tố nhỏ hơn N

3. MẢNG

3. Mảng (Array)

- Giới thiệu về mảng
  - Mảng khai báo (*declaring arrays*) và mảng tham chiếu (*referencing arrays*)
  - Vòng lặp for và mảng
  - Mảng trong bộ nhớ
- Mảng trong hàm
  - Sử dụng mảng như tham số của hàm hoặc giá trị trả lại
- Lập trình với mảng
  - Tìm kiếm (*searching*), sắp xếp (*sorting*)
- Mảng nhiều chiều (*multidimensional arrays*)

3.1. Giới thiệu về mảng

- Định nghĩa: một tập giá trị có CÙNG KIỂU
- Mảng là một cơ chế lưu trữ phổ biến
- Được sử dụng như danh sách các phần tử:
  - Tránh khai báo nhiều biến đơn giản
  - Có thể thao tác danh sách như một thực thể (entity)

Mảng khai báo

- Khai báo mảng là cấp phát một dải vùng nhớ (allocate memory)
- Cấu trúc: *Kiểu Tên_Mảng[Kích_Thước]*
- Ví dụ: *int score[5];*
  - Khai báo một mảng gồm 5 số nguyên tên là "score"
  - Giống như khai báo 5 biến: *int score[0], score[1], score[2], score[3], score[4];*
- Số nguyên dương ở giữa hai dấu [ ] được gọi là *chỉ số*, nằm trong khoảng từ 0 đến (*size-1*)
- Truy cập các phần tử trong mảng thông qua chỉ số. Ví dụ: *cout << score[3];*

Khởi tạo mảng

- Giống như các biến có thể được khởi tạo lúc khai báo.
  Ví dụ: *int price = 0;* // 0 là giá trị khởi tạo
- Khai báo mảng cũng như thế.
- Ví dụ: *int children[3] = {2, 12, 1};* tương đương

```
int children[3];
children[0] = 2;
children[1] = 12;
children[2] = 1;
```

- Nếu số lượng giá trị nhỏ hơn kích thước mảng thì:
  - Khởi tạo giá trị từ đầu
  - Phần còn lại được gán trị 0
- Nếu thiếu kích thước mảng (vd: *int b[] = {5, 12, 11};*) tự động khai báo mảng với kích thước dựa trên số lượng giá trị khởi tạo

Vòng lặp **for** và mảng

- Vòng lặp đếm tự nhiên duyệt qua tất cả các phần tử của mảng
- Ví dụ:

```
for(int i = 0; i < 5; i++)
        cout << a[i] << "\t";
```

Bài tập (1/3)

- Viết một chương trình chấp nhận một mảng số nguyên *score* có tối đa 10 phần tử. Tìm phần tử lớn nhất của mảng và in ra khoảng cách từ mỗi phần tử đến phần tử lớn nhất.

  Input: một mảng N phần tử (3 < N <=10)

  Output: số lớn nhất của mảng này

Bài tập (2/3)

Display 5.1 **Program Using an Array**

```
1  //Reads in five scores and shows how much each
2  //score differs from the highest score.
3  #include <iostream>
4  using namespace std;
5  int main( )
6  {
7      int i, score[5], max;
8      cout << "Enter 5 scores:\n";
9      cin >> score[0];
10     max = score[0];
11     for (i = 1; i < 5; i++)
12     {
13         cin >> score[i];
14         if (score[i] > max)
15             max = score[i];
16         //max is the largest of the values score[0],..., score[i].
17     }
```

Bài tập (3/3)

```
18      cout << "The highest score is " << max << endl
19           << "The scores and their\n"
20           << "differences from the highest are:\n";
21      for (i = 0; i < 5; i++)
22          cout << score[i] << " off by "
23               << (max - score[i]) << endl;
24      return 0;
25  }
```

SAMPLE DIALOGUE

Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 4

Lưu ý

- Phần tử đầu tiên có chỉ số là 0
- Lỗi: *Out of rang*e, trình biên dịch không báo lỗi nhưng lúc chạy có thể sẽ dẫn đến kết quả sai !
- Dùng hằng số (constant) để khai báo kích thước mảng.Ví dụ:

```
const int NUMBER_OF_STUDENTS = 5;
int score[NUMBER_OF_STUDENTS];
```

Trong vòng lặp:

```
for (idx = 0; idx < NUMBER_OF_STUDENTS; idx++)
{
    // Manipulate array
}

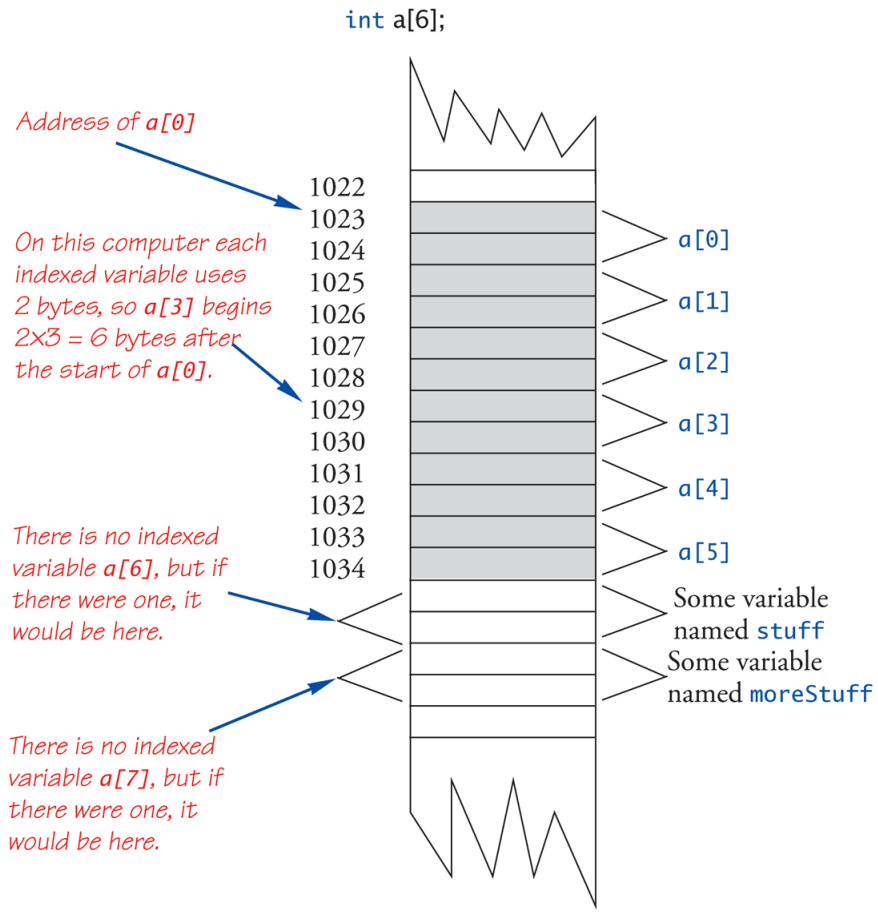
lastIndex = (NUMBER_OF_STUDENTS – 1);
```

Mảng trong bộ nhớ (1/2)

- Nhớ lại: các biến được cấp phát bộ nhớ với địa chỉ (address) xác định
- Mảng khai báo cấp phát bộ nhớ cho toàn mảng theo kiểu tuần tự (sequentially-allocated)

Mảng trong bộ nhớ (2/2)

Display 5.2 **An Array in Memory**



Sử dụng mảng trong hàm

- Như tham số của hàm
  - Phần tử của mảng: giống như một biến đơn giản. Ví dụ: *void myFunction(double par1);*

    int i; double n, a[10];

    myFunction(i); *// i is converted to double*
    myFunction(a[3]); *// a[3] is double*
    myFunction(n); *// n is double*
  - Toàn bộ mảng
- Như giá trị trả lại của hàm (sẽ học sau)

Truyền toàn bộ mảng vào hàm

Display 5.3 Function with an Array Parameter

SAMPLE DIALOGUEFUNCTION DECLARATION

```
void fillUp(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

SAMPLE DIALOGUEFUNCTION DEFINITION

```
void fillUp(int a[], int size)
{
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    cout << "The last array index used is " << (size - 1) << endl;
}
```

int score[5], numberOfScores = 5;
fillup(score, numberOfScores);

Lưu ý: không có [ ] khi gọi hàm

Mảng như tham số: cách hoạt đông?

- Điều gì thực sự xảy ra?
- Xem mảng gồm 3 thành phần:
  - Địa chỉ của phần tử đầu tiên (arrName[0])
  - Kiểu của các phần tử trong mảng
  - Kích thước của mảng
- Chỉ thành phần thứ nhất được truyền vào hàm
  - Là địa chỉ của phần tử đầu tiên của mảng
  - Tương tự như truyền tham chiếu (pass-by-reference)

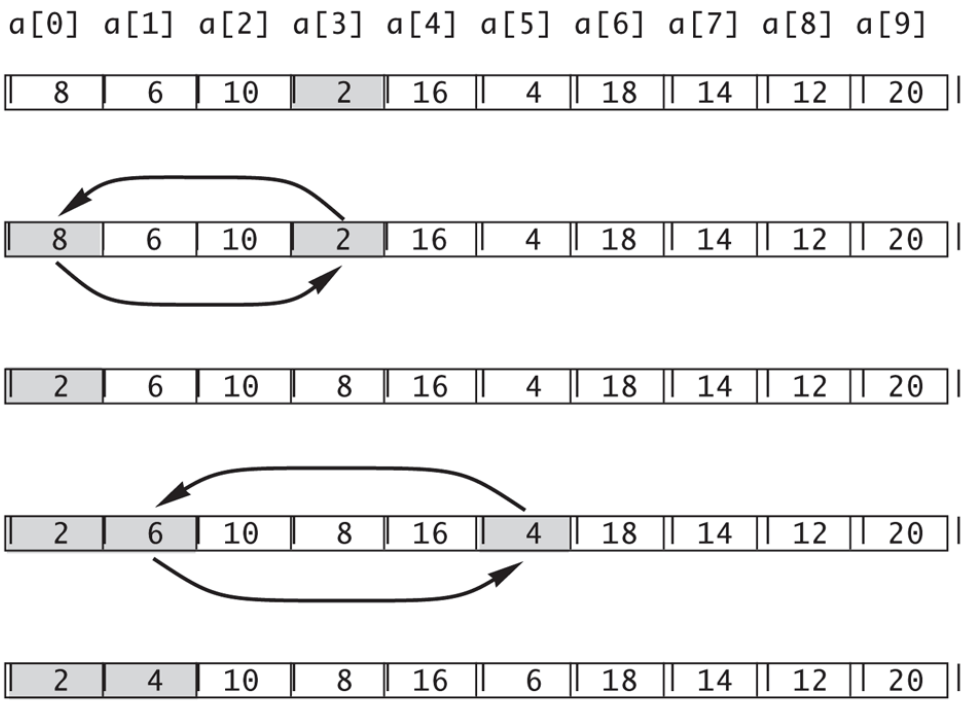
Lưu ý

- Phải có tham số là kích thước của mảng
- Không cần [ ] (brackets) khi gọi hàm
- Trong hàm có thể thay đổi giá trị của mảng nên đôi khi phải bảo vệ tránh sự thay đổi này bằng cách dùng **const**

Ứng dụng của mảng

- Tìm kiếm (searching)
- Sắp xếp (sorting)

Display 5.7 Selection Sort



Bài tập

Viết một chương trình chấp nhận một mảng số nguyên có tối đa 200 phần tử.

Hiển thị các phần tử của mảng, sắp xếp mảng theo chiều tăng dần và hiển thị mảng sau khi sắp xếp ra màn hình.

Input: mảng số nguyên N phần tử

Output: in ra mảng này sau khi sắp xếp

Mảng nhiều chiều

- Mảng có thể có nhiều hơn 1 chỉ số.

  Ví dụ: *char page[30][100];*
- Truy cập vào từng phần tử *page[i][j]*

Bài tập

- **Bài 1**: Cho một mảng hai chiều số nguyên dương với tối đa 100 hàng và 100 cột, tính tổng các phần tử chẵn trong mảng và hiển thị ra màn hình.

  Input: ma trận số nguyên dương kích thước tối đa 100x100

  Output: tổng của các phần tử chẵn trong ma trận

- **Bài 2**: Cho một mảng hai chiều số nguyên với đối đa 100 hàng và 100 cột, xét xem mảng có đối xứng qua đường chéo chính hay không ?

4. CON TRỎ

4. CON TRỎ (POINTER)

- Con trỏ
  - Biến con trỏ (pointer variables)
  - Quản lý bộ nhớ
- Mảng động (dynamic arrays)
  - Tạo và sử dụng mảng động
  - Phép tính với con trỏ (pointer arithmetic)

Định nghĩa Con trỏ

- *Con trỏ là địa chỉ trong bộ nhớ của một biến*
- Tham số tham chiếu (*call-by-reference*) của hàm chính là con trỏ: địa chỉ của tham số được truyền vào trong hàm.

Biến con trỏ

- Biến con trỏ *là biến kiểu con trỏ (không phải kiểu int, double, …) dùng để trỏ đến một vùng nhớ đã được khởi tạo.*
- Cú pháp khai báo: **Kiểu *Biến_Con_Trỏ;**
- Ví dụ:

  *double *p;* // biến p được khai báo là một biến có thể trỏ đến bất kỳ một vùng nhớ kiểu double (mà không phải kiểu int hay float)

  *int *p1, *p2, v1, v2;*

Địa chỉ và số (Addresses & numbers)

- Con trỏ là một địa chỉ
- Địa chỉ (trong vùng nhớ) là một số nguyên
- NHƯNG: Con trỏ KHÔNG PHẢI là một số nguyên !!!
- C++ ép buộc con trỏ được sử dụng như một địa chỉ
  - Không được dùng như một số nguyên
  - Mặc dù nó giống như một số nguyên

Toán tử & và * (1/2)

- **Toán tử &**
  - Khi đặt trước một biến sẽ trả về địa chỉ của biến đó (cũng được xem là một con trỏ trỏ đến biến).
  - Thường được gọi là *toán tử địa chỉ* ("*address of*" *operator*)
  - *p = &v;* nghĩa là con trỏ p được gán bằng địa chỉ của biến v hay con trỏ p trỏ đến biến v
- **Toán tử ***
  - Khi đặt trước một biến con trỏ sẽ dùng để chỉ định biến mà con trỏ đấy đang trỏ đến.
  - Toán tử * với cách dùng như trên được gọi là *toán tử giải tham chiếu* (***dereference operator***)
  - **p* nghĩa là "*lấy dữ liệu mà p đang trỏ tới*"

Toán tử & và * (2/2)

- **Kết quả in ra của chương trình sau?**:

```
int *p1, v1 = 0;
p1 = &v1;     // biến con trỏ p1 được gán cho địa chỉ vùng nhớ của biến v1
*p1 = 42;     // giải tham chiếu p1 và gán 42
cout << v1 << "\t";
cout << *p1 << endl;
```

- Có 2 cách để tham chiếu đến biến *v1*:
  - Sử dụng chính bản thân biến v1
  - Thông qua con trỏ p1

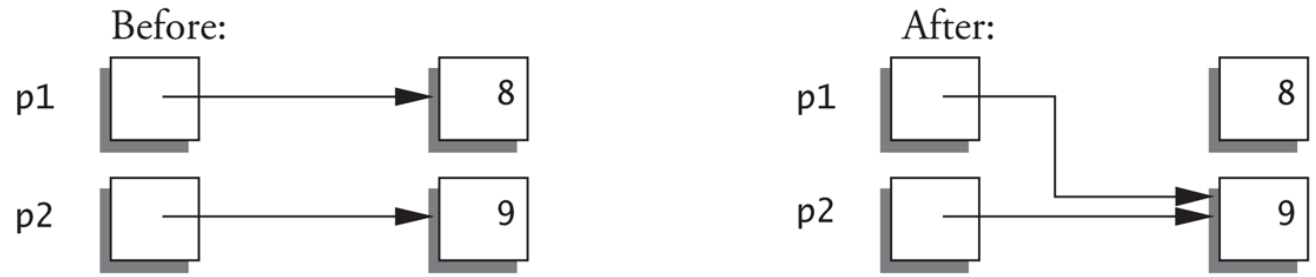
Gán con trỏ (Pointer assignments)

- *int *p1, *p2;*
  ***p2 = p1* khác **p2 = *p1* như thế nào?**
- **p2 = p1**
  - thay đổi địa chỉ vùng nhớ của con trỏ p2, trỏ tới nơi mà p1 đang trỏ tới. p2, p1 lúc này trỏ tới cùng 1 địa chỉ.
- ***p2 = *p1**
  - gán giá trị của biến mà p2 đang trỏ tới bằng giá trị của biến mà p1 đang trỏ tới. p2, p1 vẫn trỏ tới hai địa chỉ khác nhau nhưng giá trị tại hai địa chỉ này giờ bằng nhau

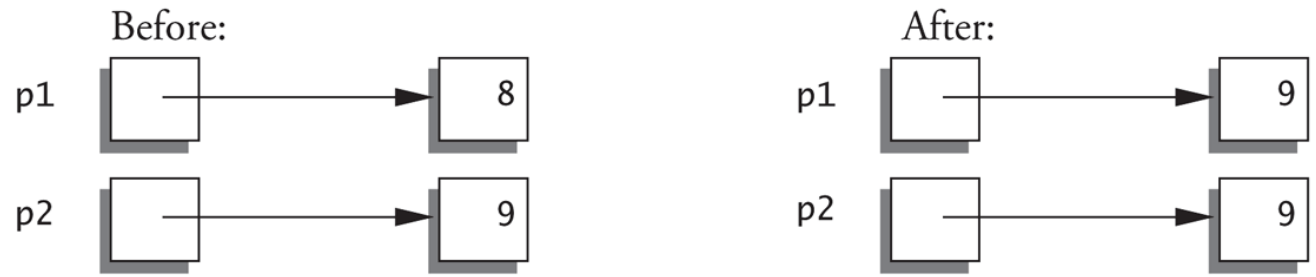
Minh họa gán con trỏ

Display 10.1 Uses of the Assignment Operator with Pointer Variables

```
p1 = p2;
```




```
*p1 = *p2;
```



Toán tử **new**

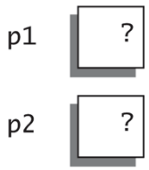
- Do con trỏ có thể dùng để tham chiếu đến một biến => Không cần thiết phải có một định danh (không nhất thiết phải có tên)
- Có thể cấp phát động các biến bằng toán tử *new*, sẽ tạo ra một biến
  - Không có định danh tham chiếu đến nó
  - Chỉ là một con trỏ
- Ví dụ: *int *p; p = new int;*
  - Tạo ra một biến mới vô danh (không được đặt tên) và gán con trỏ p trỏ tới biến đó
  - Có thể truy cập biến đó thông qua *p và sử dụng như một biến thông thường
- *int *n; n = new int(17);* //Khởi tạo *n to 17

Minh họa Toán tử new

Display 10.3 Explanation of Display 10.2

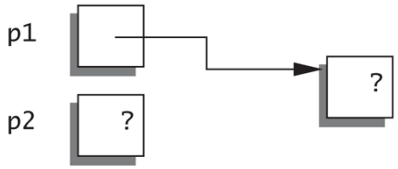
(a)

```
int *p1, *p2;
```



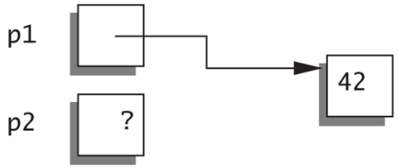
(b)

```
p1 = new int;
```



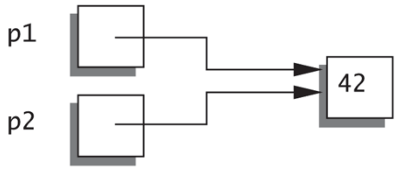
(c)

```
*p1 = 42;
```



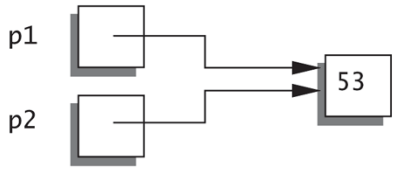
(d)

```
p2 = p1;
```



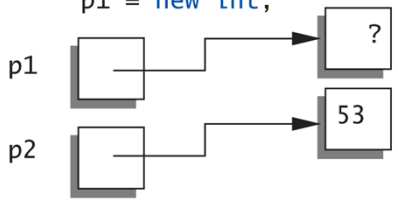
(e)

```
*p2 = 53;
```



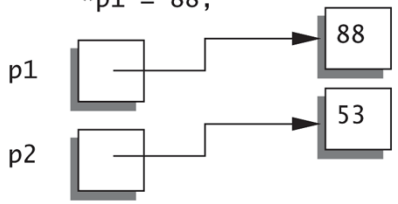
(f)

```
p1 = new int;
```



(g)

```
*p1 = 88;
```



Chương trình với **new** (1/2)

Display 10.2 **Basic Pointer Manipulations**

```
1   //Program to demonstrate pointers and dynamic variables.
2   #include <iostream>
3   using std::cout;
4   using std::endl;


5   int main( )
6   {
7       int *p1, *p2;


8       p1 = new int;
9       *p1 = 42;
10      p2 = p1;
11      cout << "*p1 == " << *p1 << endl;
12      cout << "*p2 == " << *p2 << endl;


13      *p2 = 53;
14      cout << "*p1 == " << *p1 << endl;
15      cout << "*p2 == " << *p2 << endl;
```

Chương trình với **new** (2/2)

```
16        p1 = new int;
17        *p1 = 88;
18        cout << "*p1 == " << *p1 << endl;
19        cout << "*p2 == " << *p2 << endl;


20        cout << "Hope you got the point of this example!\n";
21        return 0;
22    }
```

SAMPLE DIALOGUE

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

Con trỏ và hàm

- Con trỏ có thể được sử dụng:
  - như tham số của hàm
  - như giá trị trả lại của hàm
- *int* findOtherPointer(int* p);*

  Hàm này khai báo:
  - Một tham số kiểu con trỏ có thể trỏ tới biến kiểu int
  - Giá trị trả lại là một kiểu con trỏ có thể trỏ tới biến kiểu int

Quản lý bộ nhớ (Memory management)

- Heap (đống)
  - Cũng được gọi là "***freestore***"
  - Được dành riêng cho các biến cấp phát động (dynamically-allocated) với toán tử *new*
  - Tất cả các biến được cấp phát động sẽ sử dụng bộ nhớ trong freestore (heap) => nếu có quá nhiều biến động, có thể dẫn đến hết bộ nhớ freestore
- Thao tác cấp phát động (với toán tử *new*) có thể không thực hiện được nếu *freestore* bị đầy

Kiểm tra kết quả cấp phát bộ nhớ (1/2)

- Với những trình biên dịch cũ, thực hiện 2 bước
  1. Kiểm tra liệu giá trị *null* có được trả về sau khi gọi new hay không

```
int *p;
p = new int;
if (p == NULL)
{
      cout << "Error: Insufficient memory.\n";
      exit( 1 );
}
```

  2. Nếu cấp phát thành công thì mới tiếp tục chương trình

Kiểm tra kết quả cấp phát bộ nhớ (2/2)

- Với những trình biên dịch mới, nếu hoạt động cấp phát với *new* bị lỗi:
  - Chương trình sẽ tự động ngừng ngay lập tức
  - Thông báo lỗi

Toán tử **delete**

- Toán tử *delete*: giải phóng (de-allocate) vùng nhớ động đang được trỏ bởi một biến con trỏ.
  - Sử dụng khi biến con trỏ không còn cần thiết
  - Trả vùng nhớ này về freestore, để sau đó có thể được dùng để cấp phát cho biến khác
- VD: *delete p;* // giải phóng vùng nhớ được trỏ bởi con trỏ p

```
int *p;
p = new int(5);
…
// Một vài xử lý
…
delete p;
```

Con trỏ treo (Dangling pointers)

- *delete p;* giải phóng vùng nhớ động nhưng p vẫn trỏ tới đó !
  - Dẫn đến con trỏ treo (dangling pointer)
  - Điều gì xảy ra khi gọi **p* sau đó?
- Tránh con trỏ treo
  - Nên gán con trỏ bằng *null* sau khi *delete p*

```
delete p;
p = NULL;
```

Biến động và Biến tự động (Dynamic vs automatic variables)

- Biến động (dynamic variables)
  - Được tạo với toán tử *new*
  - Được tạo và hủy (destroy) trong lúc chạy chương trình
- Biến địa phương (local variables)
  - Được khai báo bên trong định nghĩa hàm
  - Không phải là biến động (not dynamic)
    - Được tạo khi hàm được gọi
    - Bị hủy khi lời gọi hàm hoàn tất
  - Thường được gọi là các biến tự động (automatic variables). Chương trình sẽ kiểm soát các biến này cho bạn

Định nghĩa lại tên cho kiểu con trỏ

- Giúp bạn tránh việc thêm dấu "*" mỗi khi khai báo con trỏ
- *typedef int* IntPtr;*
  - Định nghĩa một kiểu biệt danh mới (alias)
  - *IntPtr p;* tương đương với *int *p;*

Tham số con trỏ call-by-value (1/3)

Display 10.4 **A Call-by-Value Pointer Parameter**

```
1  //Program to demonstrate the way call-by-value parameters
2  //behave with pointer arguments.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;

7  typedef int* IntPointer;

8  void sneaky(IntPointer temp);

9  int main( )
10 {
11     IntPointer p;

12     p = new int;
13     *p = 77;
14     cout << "Before call to function *p == "
15          << *p << endl;
```

Tham số con trỏ call-by-value (2/3)

```
16        sneaky(p);

17        cout << "After call to function *p == "
18             << *p << endl;

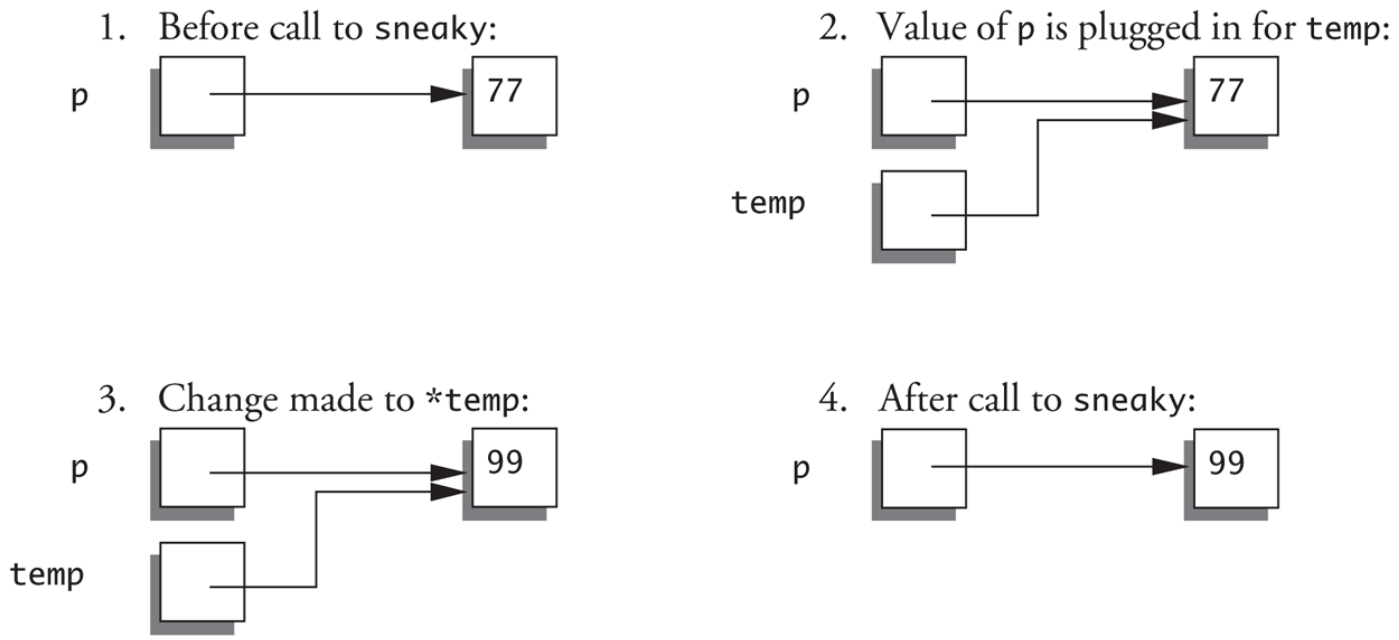
19        return 0;
20   }
21   void sneaky(IntPointer temp)
22   {
23        *temp = 99;
24        cout << "Inside function call *temp == "
25             << *temp << endl;
26   }
```

SAMPLE DIALOGUE

```
Before call to function *p == 77
Inside function call *temp == 99
After call to function *p == 99
```

Tham số con trỏ call-by-value (3/3)

Display 10.5 **The Function Call `sneaky(p);`**



Mảng động (Dynamic arrays)

- Biến kiểu mảng thực sự là một kiểu con trỏ
- Mảng chuẩn với kích thước cố định
- Mảng động:
  - Kích thước của mảng không được xác định khi lập trình
  - Được quyết định trong lúc chạy chương trình

Biến kiểu mảng (1/2) (Array variables)

- Mảng được lưu trữ trong bộ nhớ theo địa chỉ tuần tự
  - Biến mảng tham chiếu đến giá trị đầu tiên của mảng
  - Do đó biến mảng cũng là một kiểu con trỏ
- Ví dụ:

  int a[10];
  *int * p;*

  a và *p* đều là con trỏ

Biến kiểu mảng (2/2) (Array variables)

- *a* và *p* đều là con trỏ => có thể thực hiện việc gán
  - *p* = *a*; // Hợp lệ. *p* bây giờ trỏ tới nơi mà *a* đang trỏ (tới vị trí đầu tiên của mảng *a*)
  - *a*= *p*; // KHÔNG HỢP LỆ. Do con trỏ mảng là một CON TRỎ HẰNG SỐ (constant pointer)
- *a* có kiểu ***const int****
  - Mảng đã được cấp phát trong bộ nhớ
  - Biến *a* PHẢI LUÔN trỏ tới đó! Không thể thay đổi!
- Đối lập với con trỏ bình thường: có thể thay đổi địa chỉ trỏ tới

Mảng động

- Những hạn chế của mảng chuẩn
  - Phải khai báo kích thước trước, ước lượng kích thước tối đa cần thiết
  - Có thể lãng phí bộ nhớ
- Mảng động
  - Có thể co giãn khi cần thiết
- Tạo mảng động với toán tử *new*. Cấp phát động với biến con trỏ

  *typedef double * DoublePtr;*
  DoublePtr d;
  d = new double[10]; // Tạo một biến mảng *d* được cấp phát động với 10 phần tử kiểu double

Xóa mảng động

- Được cấp phát động lúc chạy (run-time), vì thế nên được hủy lúc chạy
- *d = new double[10];*
  … //Xử lý
  delete [] d;
  - Giải phóng tất cả bộ nhớ của mảng động
  - Dấu ngoặc vuông (brackets) [] ám chỉ đây là một mảng
  - Tuy nhiên *d* vẫn chỉ tới vùng nhớ vừa được giải phóng => nên đặt lại *d = NULL;*

Phép tính với con trỏ (Pointer arithmetic)

- Có thể thực hiện phép tính trên con trỏ: tính toán với các địa chỉ
- *typedef double* DoublePtr;*
  DoublePtr d;
  d = new double[10];
  - d chứa địa chỉ của d[0]
  - d + 1 là địa chỉ của d[1]
  - d + 2 là địa chỉ của d[2]
- *for (int i = 0; i < arraySize; i++)*
  *cout << *(d + i) << " ";*
- Chỉ thực hiện phép **+ -** trên con trỏ (không thực hiện các phép tính *** /** với con trỏ)

Con trỏ động nhiều chiều (Multidimensional dynamic arrays)

- Nhớ lại: *Mảng của Mảng*
- ```
  typedef int* IntArrayPtr;
  IntArrayPtr *m = new IntArrayPtr[3];
  for (int i = 0; i < 3; i++)
      m[i] = new int[4];
  ```
  - Tạo mảng của 3 con trỏ
  - Cấp phát cho mỗi con trỏ một mảng 4 phần tử kiểu int
- Kết quả: một mảng động 3x4
  ```

Tóm tắt (1/3)

- Mệnh đề rẽ nhánh: *if-else*, *switch*
- Vòng lặp:
  - *while*,
  - *do-while*: luôn thực hiện phần thân vòng lặp ít nhất 1 lần
  - *for*
- Vòng lặp có thể bị ngắt quãng đột ngột với hai lệnh: *break*, *continue*

Tóm tắt (2/3)

- Mảng là tập hợp của dữ liệu cùng kiểu
- Phần tử trong mảng được sử dụng như những biến đơn giản khác
- Vòng lặp for là cách tự nhiên để duyệt mảng
- Chú ý lỗi out-of-range
- Các phần tử trong mảng được lưu trữ tuần tự
- Mảng nhiều chiều

Tóm tắt (3/3)

- Con trỏ là một địa chỉ trong bộ nhớ. Cung cấp một tham chiếu không trực tiếp đến các biến
- Biến động: được tạo ra và hủy trong lúc chạy chương trình
- Freestore: bộ nhớ dành cho các biến động
- Mảng cấp phát động: kích thước được quyết định khi chạy chương trình

Tham khảo

- **Giáo trình chính: W. Savitch, *Absolute C++*, Addison Wesley, 2002**
- Tham khảo:
  - A. Ford and T. Teorey, *Practical Debugging in C++*, Prentice Hall, 2002
  - Nguyễn Thanh Thủy, *Kĩ thuật lập trình C++*, NXB Khoa học và Kĩ Thuật, 2006