



Tổng quan về lập trình hướng đối tượng và C++

Nội dung

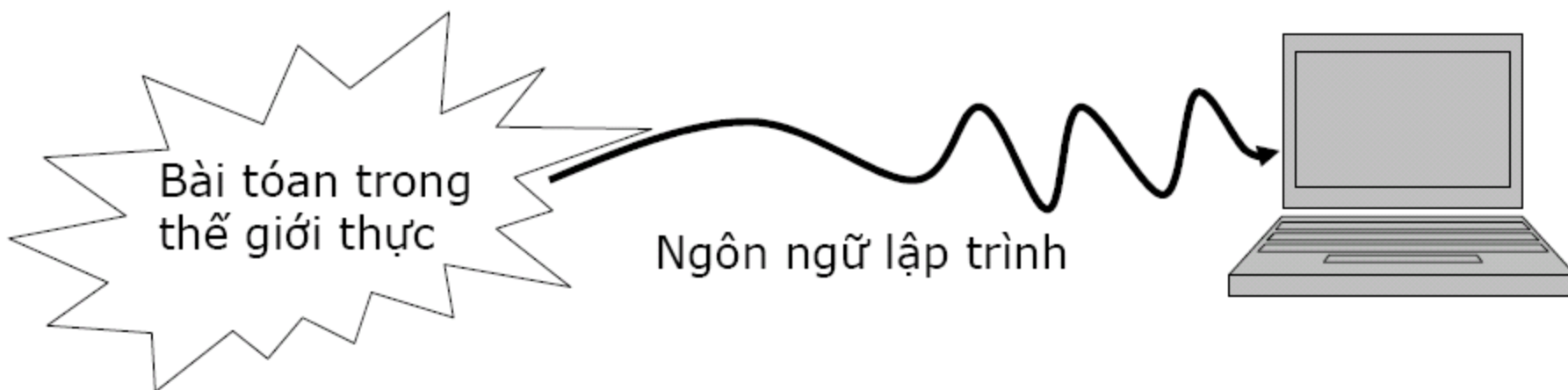


- ❖ Giới thiệu lập trình hướng đối tượng
- ❖ Ngôn ngữ C++, các điểm mới so với C
- ❖ Lớp trong C++

Mục tiêu của lập trình viên



- ❖ Tạo ra sản phẩm tốt một cách có hiệu quả
- ❖ Nắm bắt được công nghệ



Độ phức tạp và độ lớn ngày càng cao



- ❖ Một số hệ Unix chứa khoảng 4M dòng lệnh
- ❖ MS Windows chứa hàng chục triệu dòng lệnh
- ❖ Người dùng ngày càng đòi hỏi nhiều chức năng, đặc biệt là chức năng *thông minh*
- ❖ Phần mềm luôn cần được sửa đổi

Giải pháp



❖ Cần kiểm soát chi phí

- Chi phí phát triển
- Chi phí bảo trì

❖ Giải pháp chính là ***sử dụng lại (tái sử dụng)***

- Giảm chi phí và thời gian phát triển
- Nâng cao chất lượng

Để sử dụng lại (mã nguồn)



- ❖ Cần dễ hiểu
- ❖ Được coi là chính xác
- ❖ Có giao diện rõ ràng
- ❖ Tính module hóa
- ❖ *Không yêu cầu thay đổi khi sử dụng trong chương trình mới*

Mục tiêu của việc thiết kế 1 phần mềm



- ❖ Tính tái sử dụng (*reusability*): thiết kế các thành phần có thể được sử dụng trong nhiều phần mềm khác nhau
- ❖ Tính mở rộng (*extensibility*): hỗ trợ các plug-ins.
- ❖ Tính mềm dẻo (*flexibility*):
 - Có thể dễ dàng thay đổi khi thêm mới dữ liệu hay tính năng.
 - Các thay đổi không làm ảnh hưởng nhiều đến toàn bộ hệ thống

Các phương pháp lập trình



- ❖ Lập trình không có cấu trúc
- ❖ **Lập trình có cấu trúc** (lập trình thủ tục), hướng chức năng
- ❖ Lập trình logic, lập trình hàm
- ❖ **Lập trình hướng đối tượng**

Lập trình không có cấu trúc



❖ Là phương pháp xuất hiện đầu tiên

- các ngôn ngữ như Assembly, Basic
- sử dụng các biến toàn cục
- lạm dụng lệnh GOTO

❖ Các nhược điểm

- khó hiểu, khó bảo trì, hầu như không thể sử dụng lại
- chất lượng kém
- chi phí cao
- không thể phát triển các ứng dụng lớn

Ví dụ

10 k = 1

20 gosub 100

30 if y > 120 goto 60

40 k = k+1

50 goto 20

60 print k, y

70 stop

100 y = 3*k*k + 7*k-3

110 return

Lập trình có cấu trúc lập trình thủ tục



- ❖ Tổ chức thành **các chương trình con (hay các module)**
- ❖ Mỗi chương trình con đảm nhận xử lý một công việc nhỏ hay một nhóm công việc trong toàn bộ hệ thống
- ❖ Mỗi chương trình con này lại có thể chia nhỏ thành các chương trình con nhỏ hơn

Chương trình = Cấu trúc dữ liệu + Giải thuật

Lập trình có cấu trúc lập trình thủ tục



- ❖ sử dụng các lệnh có cấu trúc: for, do, while, if then else...
- ❖ các ngôn ngữ: Pascal, C, ...
- ❖ chương trình là tập các hàm/thủ tục
- ❖ Ưu điểm
 - chương trình được module hóa, do đó dễ hiểu, dễ bảo trì hơn
 - dễ dàng tạo ra các thư viện phần mềm





Ví dụ

```
struct Date {
```

```
    int year, mon, day;
```

```
};
```

```
...
```

```
void print_date(Date d) {
```

```
    printf("%d / %d / %d\n", d.day,d.mon,d.year);
```

```
}
```

Lập trình có cấu trúc

lập trình thủ tục

❖ Nhược điểm

- dữ liệu và mã xử lý là tách rời
- người lập trình phải biết cấu trúc dữ liệu (vấn đề này một thời gian dài được coi là hiển nhiên)
- khi thay đổi cấu trúc dữ liệu thì mã xử lý (thuật toán) phải thay đổi theo
- khó đảm bảo tính đúng đắn của dữ liệu
- không tự động khởi tạo hay giải phóng dữ liệu động
- **không mô tả được đầy đủ, trung thực hệ thống trong thực tế**



Lập trình Hướng đối tượng



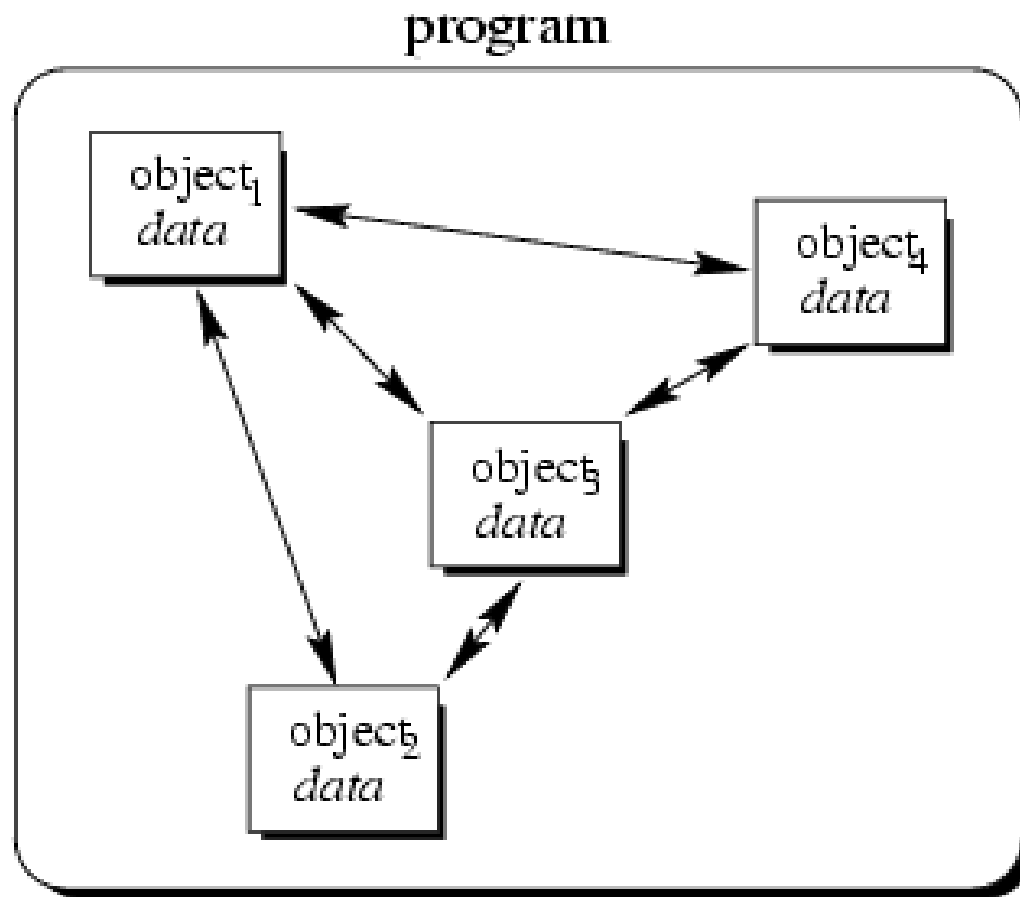
- ❖ Trong thế giới thực, chung quanh chúng ta là những đối tượng, đó là các thực thể có mối quan hệ với nhau. *Ví dụ: các phòng trong một công ty*
- ❖ Lập trình hướng đối tượng (OOP - *Object Oriented Programming*) là phương pháp lập trình lấy **đối tượng** làm nền tảng để xây dựng thuật giải, xây dựng chương trình

Lập trình Hướng đối tượng



Lập trình hướng đối tượng là phương pháp
lập trình dựa trên kiến trúc **lớp** (class)
và **đối tượng** (object)

Lập trình Hướng đối tượng



Một số khái niệm



❖ Đối tượng (object):

- Trong thế giới thực, khái niệm đối tượng được hiểu như là một thực thể: người, vật hoặc một bảng dữ liệu....
- Mỗi đối tượng sẽ tồn tại trong một hệ thống và có ý nghĩa nhất định trong hệ thống.
- Đối tượng giúp ***biểu diễn tốt hơn thế giới thực*** trên ***máy tính***
- Mỗi đối tượng bao gồm 2 thành phần: thuộc tính và thao tác (hành động)

Ví dụ: một người



- ❖ Một người có các thuộc tính: tên, tuổi, địa chỉ, màu mắt...
- ❖ Các hành động: đi, nói, thở...

**Một đối tượng là 1 thực thể bao gồm
thuộc tính và hành động**

Một số khái niệm



❖ Lớp:

- Các đối tượng có các đặc tính **tương tự nhau** được gom chung lại thành lớp đối tượng. Một lớp đối tượng được đặc trưng bằng các **thuộc tính**, và các **hoạt động** (hành vi, thao tác).
 - Ví dụ: Người là một lớp đối tượng.
- **Thuộc tính (*attribute*)** là một thành phần của đối tượng, có giá trị nhất định cho mỗi đối tượng tại mỗi thời điểm trong hệ thống.
 - Vd: Tên, Tuổi, Cân nặng là các thuộc tính của Người
- **Thao tác (*operation*)** thể hiện hành vi của một đối tượng tác động qua lại với các đối tượng khác hoặc với chính nó.

Một số khái niệm



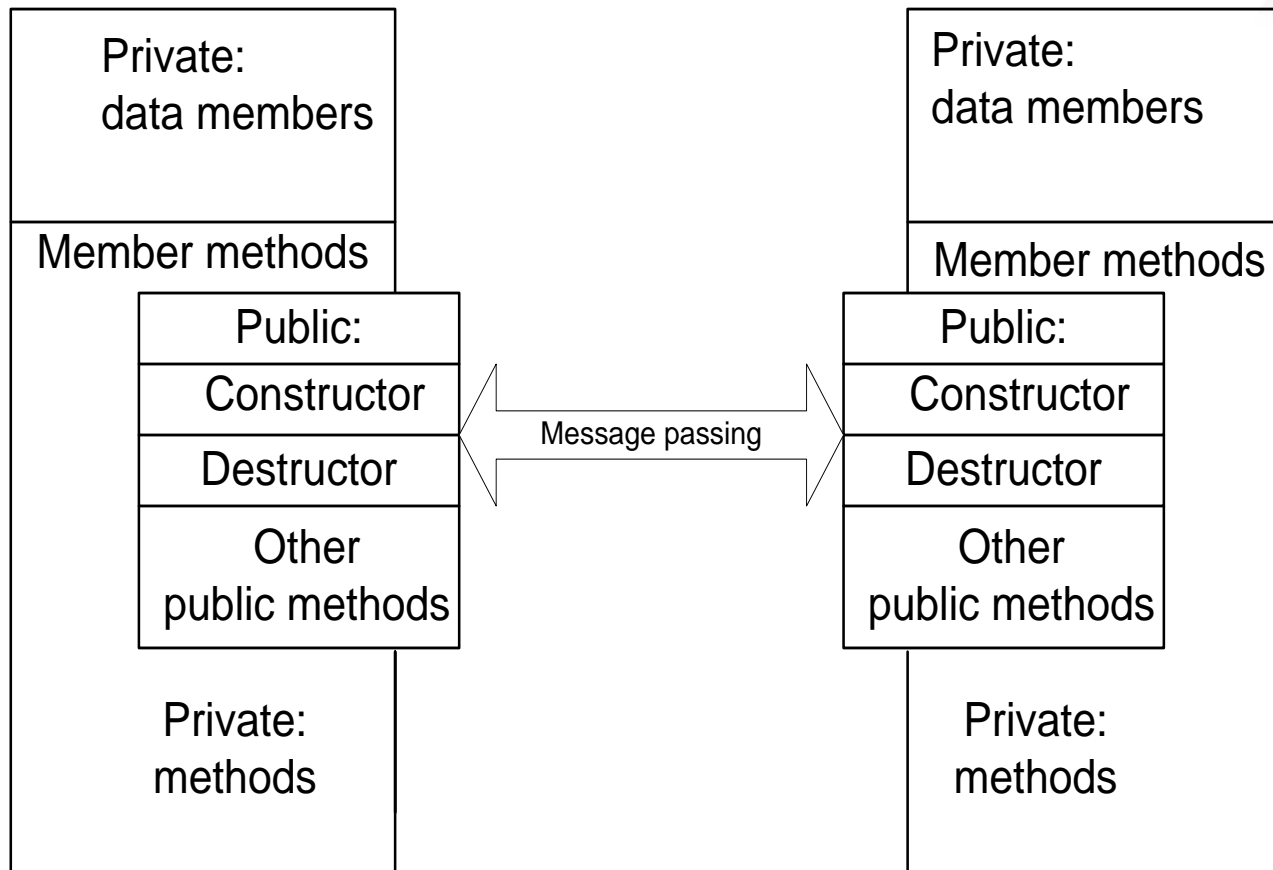
- ❖ Mỗi thao tác trên một lớp đối tượng cụ thể tương ứng với một cài đặt cụ thể khác nhau. Một cài đặt như vậy được gọi là một **phương thức (*method*)**.
- ❖ Cùng một thao tác (phương thức) có thể được áp dụng cho nhiều lớp đối tượng khác nhau, một thao tác như vậy được gọi là có **tính đa hình (*polymorphism*)**.
- ❖ Một đối tượng cụ thể thuộc một lớp được gọi là **một thể hiện (*instance*)** của lớp đó.
 - Joe Smith, 25 tuổi, nặng 58kg, là một thể hiện của lớp người.

Interacting Objects



Class A

Class B

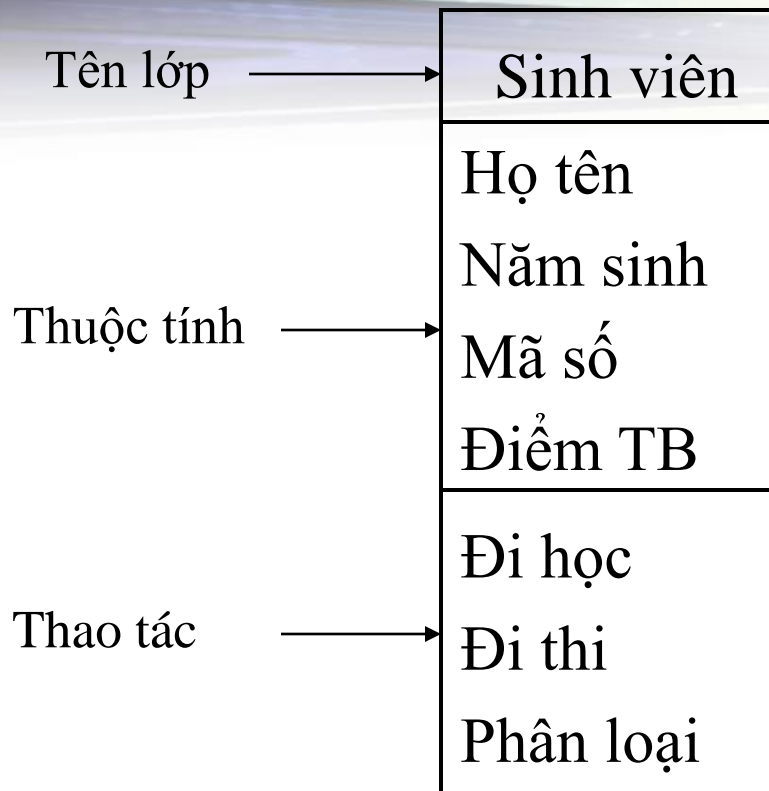


Sơ đồ đối tượng

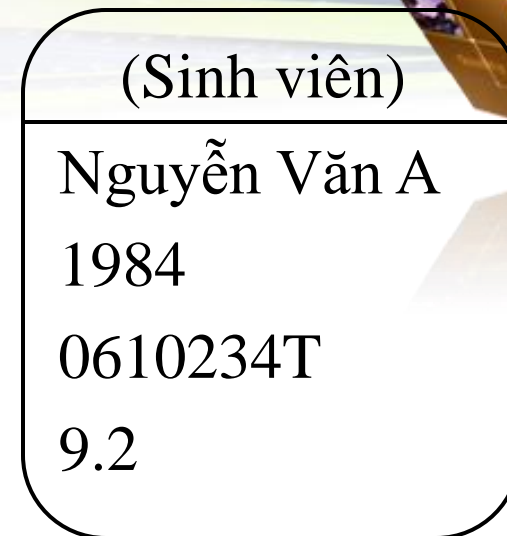


- ❖ Sơ đồ đối tượng dùng để mô tả các lớp đối tượng. Sơ đồ đối tượng bao gồm sơ đồ lớp và sơ đồ thể hiện
- ❖ Sơ đồ lớp mô tả các lớp đối tượng trong hệ thống, một lớp đối tượng được diễn tả bằng một hình chữ nhật có 3 phần:
 - phần đầu chỉ tên lớp,
 - phần thứ hai mô tả các thuộc tính
 - phần thứ ba mô tả các thao tác của các đối tượng trong lớp đó.

Sơ đồ lớp và sơ đồ thể hiện



Sơ đồ lớp



Sơ đồ thể hiện

Đối tượng = Dữ liệu + Phương thức

Thiết kế theo hướng đối tượng



- ❖ Trừu tượng hóa dữ liệu và các hàm/thủ tục liên quan
- ❖ Chia hệ thống ra thành các lớp/đối tượng
- ❖ Mỗi lớp/đối tượng có các tính năng và hành động chuyên biệt
- ❖ Các lớp có thể được sử dụng để tạo ra nhiều đối tượng cụ thể

Các đặc điểm quan trọng của OO

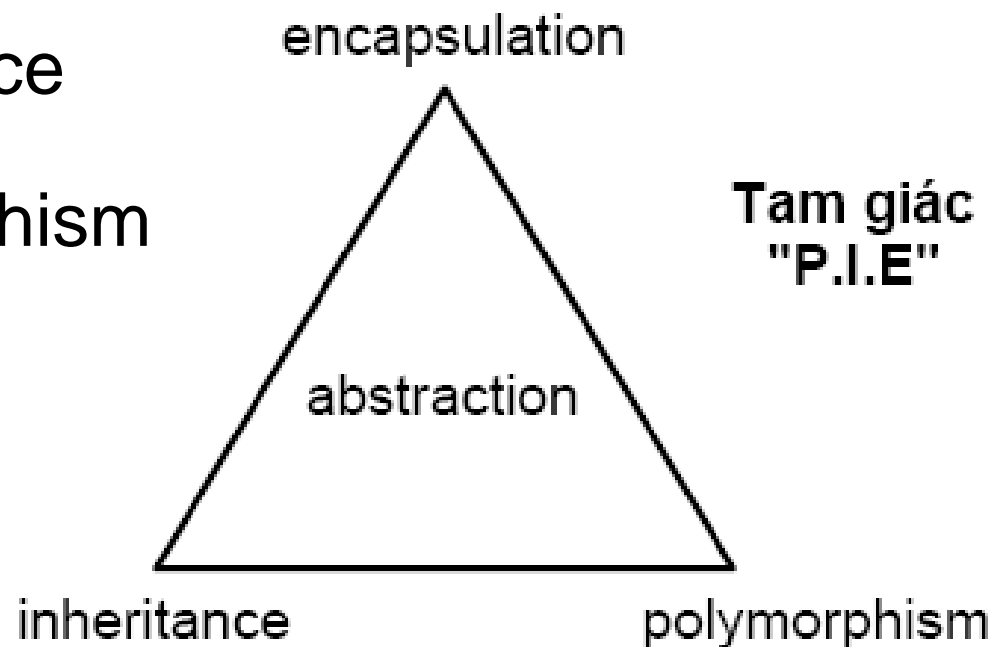


❖ Các lớp đối tượng - Classes

❖ Đóng gói – Encapsulation

❖ Thừa kế - Inheritance

❖ Đa hình - Polymorphism



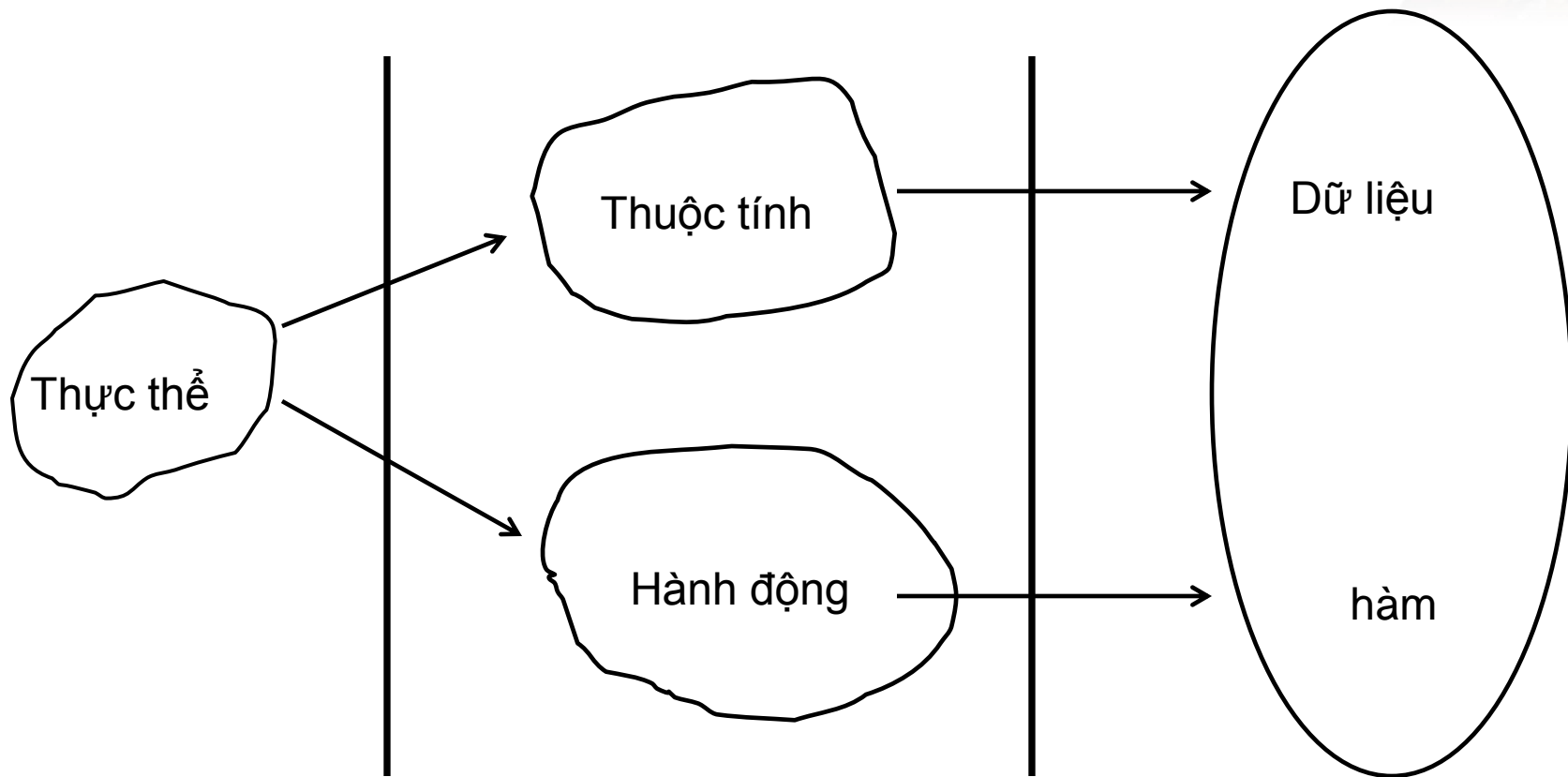
Trừu tượng hóa



Thế giới thực

Trừu tượng hóa

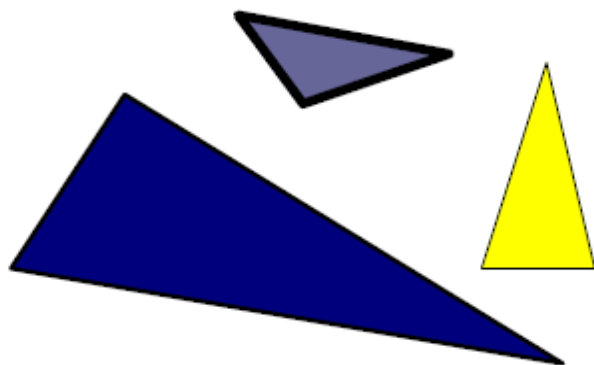
Phần mềm





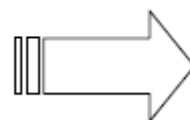
❖ Method / Behavior / Function

Trừu tượng hóa



các đối tượng

trừu tượng hóa



Tam giác:

cạnh1, cạnh2, cạnh3

màu nền, màu biên,

độ đậm biên

vẽ, tính diện tích, tính chu vi

lớp/kiểu dữ liệu

Cách nhìn **khái quát hóa** về một tập các đối tượng có chung các đặc điểm được **quan tâm** (và bỏ qua những chi tiết không cần thiết).

Đóng gói – Che dấu thông tin

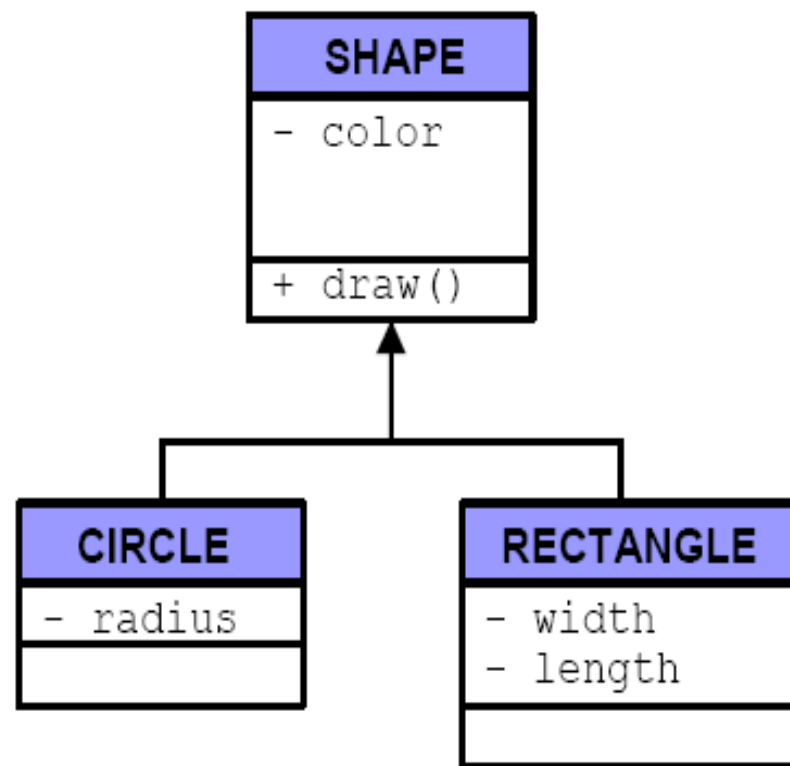


- ❖ Đóng gói: Nhóm những gì có liên quan với nhau vào làm một, để sau này có thể dùng một cái tên để gọi đến
 - Các đối tượng đóng gói dữ liệu của chúng và các thủ tục có liên quan
- ❖ Che dấu thông tin: đóng gói để che một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không nhìn thấy
 - che giấu những gì mà người dùng không cần
 - che giấu những gì mà mình cần giữ bí mật

Thừa kế



- ❖ Là cơ chế cho phép một lớp **D** có được các thuộc tính và thao tác của lớp **C**, như thể các thuộc tính và thao tác đó đã được định nghĩa tại lớp **D**.
- ❖ Cho phép cài đặt nhiều quan hệ giữa các đối tượng: đặc biệt hóa (“là”), khái quát hóa



Đa hình



- ❖ Là cơ chế cho phép một tên thao tác hoặc thuộc tính có thể được định nghĩa tại nhiều lớp và có thể có nhiều cài đặt khác nhau tại mỗi lớp trong các lớp đó

Các ưu điểm của OOP



- ❖ Nguyên lý kế thừa: tránh lặp, tái sử dụng.
- ❖ Nguyên lý đóng gói hay che dấu thông tin: chương trình an toàn không bị thay đổi bởi những đoạn chương trình khác.
- ❖ Dễ mở rộng, nâng cấp.
- ❖ Mô phỏng thế giới thực tốt hơn.

OOP có các đặc tính chủ yếu sau



- ❖ Chương trình được chia thành các đối tượng.
- ❖ Các cấu trúc dữ liệu được thiết kế sao cho đặc tả được đối tượng.
- ❖ Các hàm thao tác trên các vùng dữ liệu của đối tượng được gắn với cấu trúc dữ liệu đó.

OOP có các đặc tính chủ yếu sau



- ❖ Dữ liệu được đóng gói lại, được che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
- ❖ Các đối tượng tác động và trao đổi thông tin với nhau qua các hàm.
- ❖ Có thể dễ dàng bổ sung dữ liệu và các hàm mới vào đối tượng nào đó khi cần thiết.
- ❖ Chương trình được thiết kế theo cách tiếp cận từ dưới lên (bottom-up).

Một số thuật ngữ hướng đối tượng



- ❖ **OOM** (Object Oriented Methodology): Phương pháp luận hướng đối tượng
- ❖ **OOA** (Object Oriented Analysis): Phân tích hướng đối tượng
- ❖ **OOD** (Object Oriented Design): Thiết kế hướng đối tượng
- ❖ **OOP** (Object Oriented Programming): Lập trình hướng đối tượng
- ❖ **Inheritance**: Kế thừa
- ❖ **Polymorphism**: Đa hình
- ❖ **Encapsulation**: Tính đóng gói.

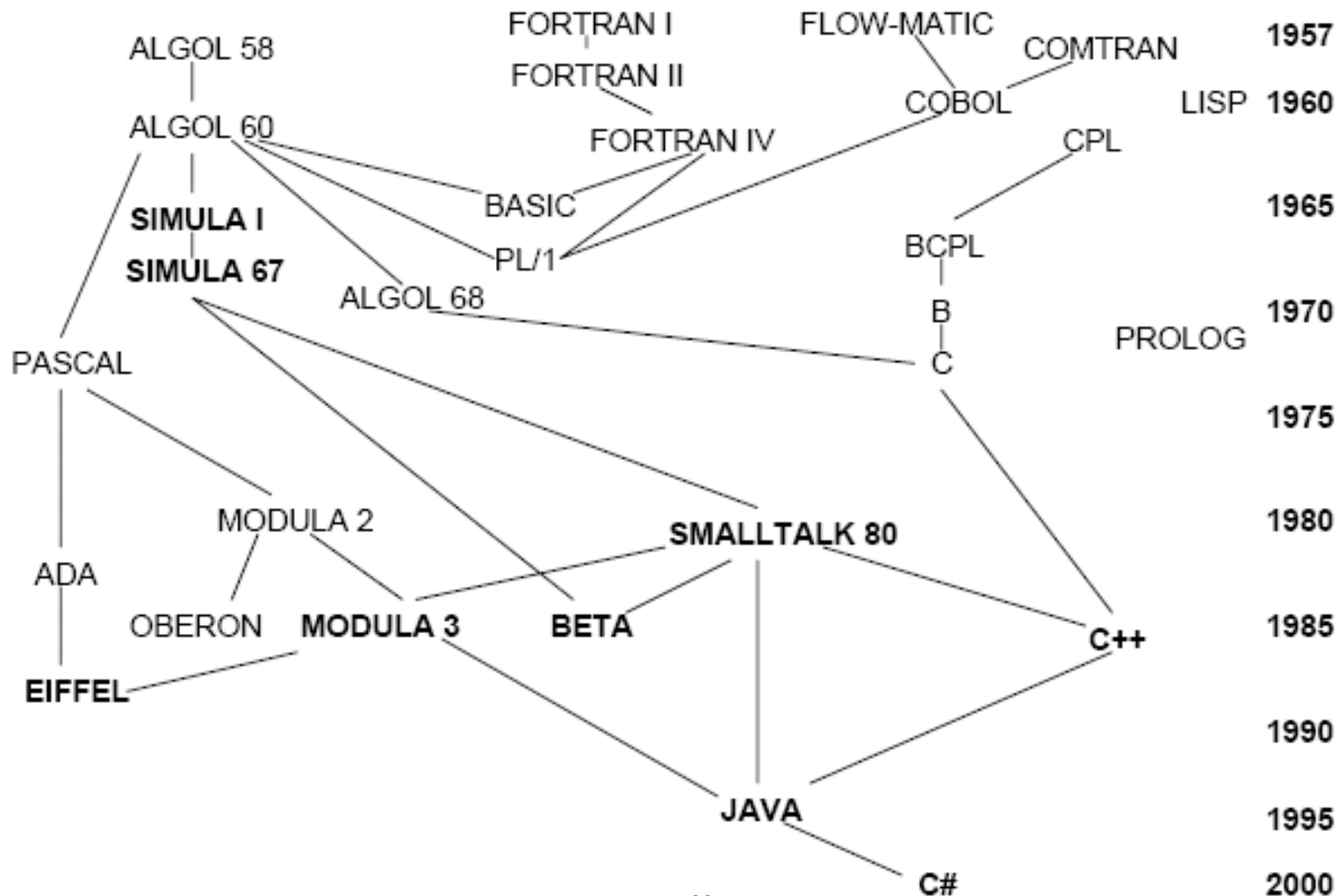
Ngôn ngữ lập trình hướng đối tượng



❖ Cung cấp được những khả năng lập trình hướng đối tượng:

- cung cấp khả năng kiểm soát truy cập
- kế thừa
- đa hình

Lịch sử ngôn ngữ lập trình



Lịch sử C++



- Mở rộng của C
- Đầu thập niên 1980: Bjarne Stroustrup (Bell Laboratories)
- Cung cấp khả năng lập trình hướng đối tượng
 - Objects
 - Object-oriented programs
- Ngôn ngữ lai
 - C-like style
 - Object-oriented style
 - Both

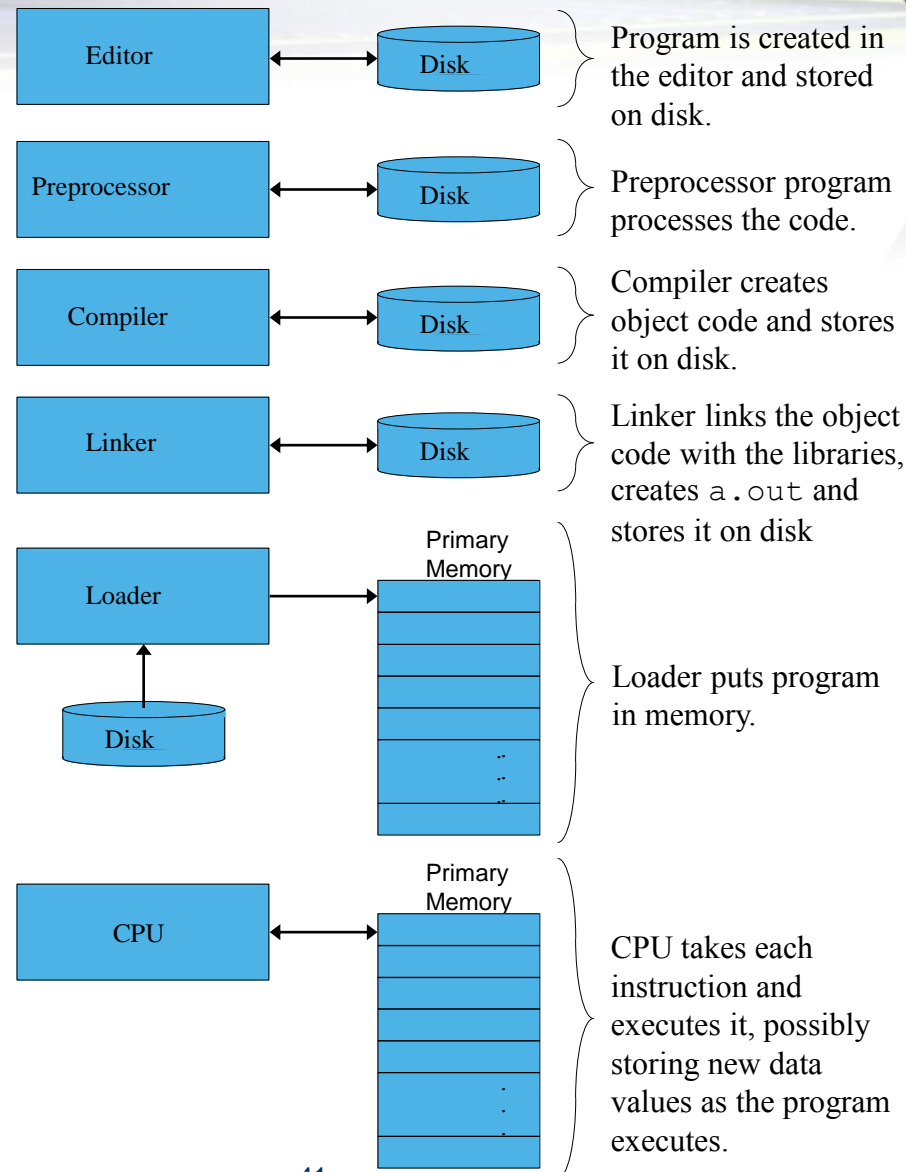
C++ Environment



Phases of C++

Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Khác biệt đối với C



- ❖ Chú thích
- ❖ Các kiểu dữ liệu
- ❖ Kiểm tra kiểu, đổi kiểu
- ❖ Phạm vi và khai báo
- ❖ Không gian tên
- ❖ Hằng
- ❖ Quản lý bộ nhớ
- ❖ Tham chiếu

C++



❖ Input/output

■ **cin**

- Standard input stream
- Normally keyboard

■ **cout**

- Standard output stream
- Normally computer screen

■ **cerr**

- Standard error stream
- Display error messages



❖ comments

```
/* You can still use the old comment style, */  
/* but you must be // very careful about mixing them */  
// It's best to use this style for 1 line or partial lines  
/* And use this style when your comment  
consists of multiple lines */
```

❖ cin and cout (and #include <iostream.h>)

```
cout << "hey";  
char name[10];  
cin >> name;  
cout << "Hey " << name << ", nice name." << endl;  
cout << endl;      // print a blank line
```

❖ declaring variables almost anywhere

```
// declare a variable when you need it  
for (int k = 1; k < 5; k++){  
    cout << k;  
}
```


Ví dụ



❖ Comments

- Document programs
- Improve program readability
- Ignored by compiler
- Single-line comment
 - Begin with //

❖ Preprocessor directives

- Processed by preprocessor before compiling
- Begin with #

```
1 // Fig. 1.2: fig01_02.cpp
2 // A first program in C++
3 #include <iostream>
4
5 // function main
6 int main()
7 {
8     cout << "Welcome to C++!"
9
10     return 0; // successfully
11 } // end function main
```

Single-line

Function main returns an

Left brace { begins function body.

or directive to

appears

Statements end with a semicolon ;.

exactly once in every C++ program..

Corresponding right brace } ends function.

Name

Stream insertion operator.

namespace std

Welcome to C++!

Keyword return is one of several means to exit function; value 0 indicates program terminated successfully.

Ví dụ

```
1 // Fig. 1.6: fig01_06.cpp
2 // Addition program.
3 #include <iostream.h>
4
5 // function main begins program execution
6 int main()
7 {
8     int integer1; // first number to be input by user
9     int integer2; // second number to be input by user
10    int sum; // variable to hold the sum
11
12    cout << "Enter first integer: ";
13    cin >> integer1; // read an integer
14
15    cout << "Enter second integer\n"; // prompt
16    cin >> integer2;
17
18    sum = integer1 + integer2;
19
20    cout << "Sum is " << sum << endl; // print sum
21
22    return 0; // indicate that program ended successfully
23
24 } // end function main
```

Declare integer variables.

Use stream extraction operator with standard input stream to obtain user input.

Calculations can be performed in output statements: alternative for lines 18 and 20:

output buffer.

Concatenating, chaining or cascading stream insertion operations.

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Vi dụ

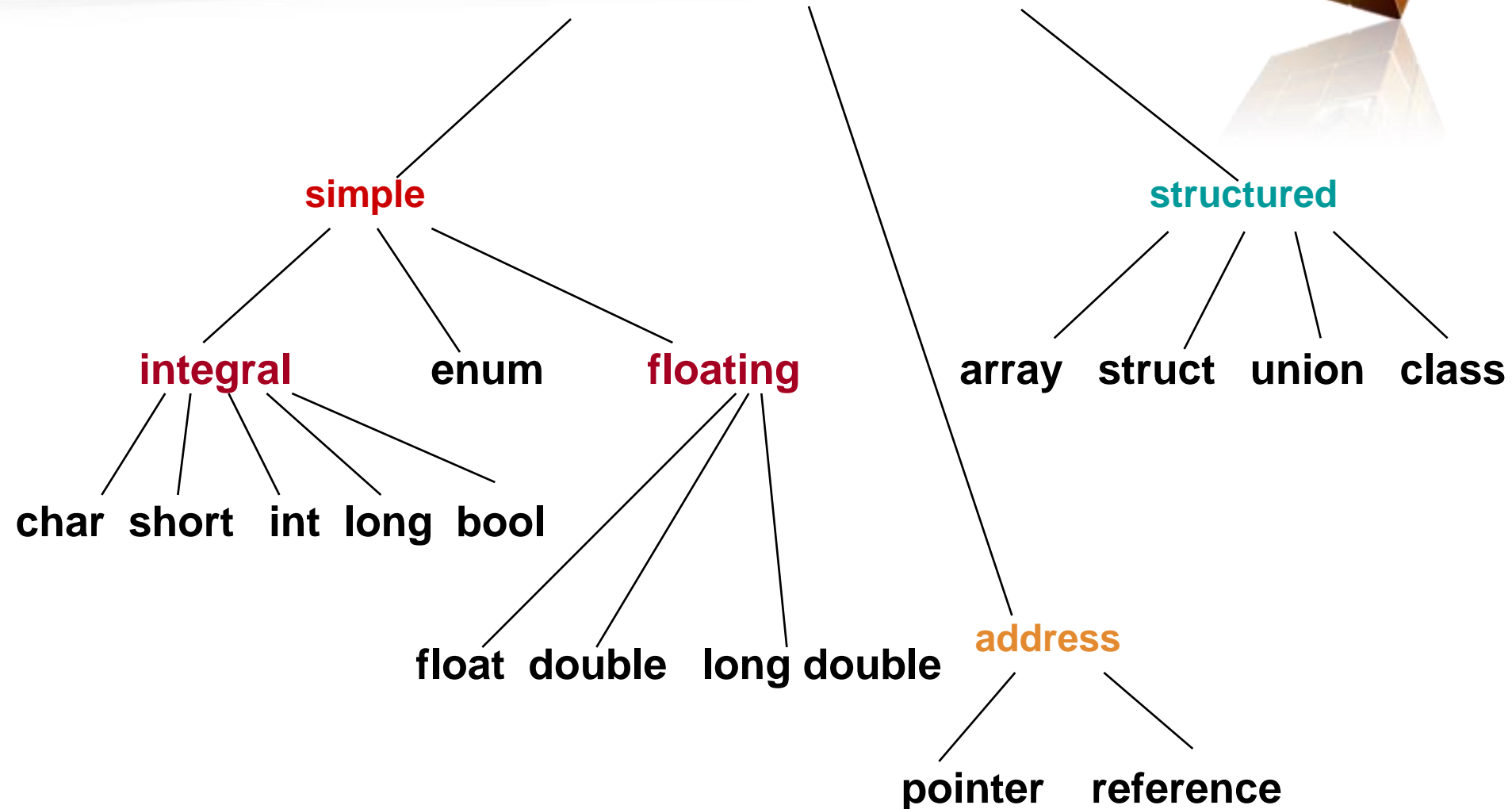
Hàm cin/cout



```
#include <iostream.h>
void main() {
    int n;
    double d;
    char s[100];

    cout << "Input an int, a double and a
    string.";
    cin >> n >> d >> s;
    cout << "n = " << n << "\n";
    cout << "d = " << d << "\n";
    cout << "s = " << s << "\n";
}
```

C++ Data Types



Phạm vi biến

```
1 // Fig. 3.12: fig03_12.cpp
```

```
2 // A scoping example.
```

```
3 #include <iostream.h>
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8 void useLocal( void );
```

```
9 void useStaticLocal( void );
```

```
10 void useGlobal( void );
```

```
11
```

```
12 int x = 1; // global variable
```

```
13
```

```
14 int main()
```

```
15 {
```

```
16     int x = 5; // local variable to main
```

```
17
```

```
18     cout << "local x in main's outer scope is " << x << endl;
```

```
19
```

```
20     { // start new scope
```

```
21
```

```
22         int x = 7;
```

```
23
```

```
24         cout << "local x in main's inner scope is " << x << endl;
```

```
25
```

```
26     } // end new scope
```

Declared outside of
function; global variable
with file scope.

Local variable with
function scope.

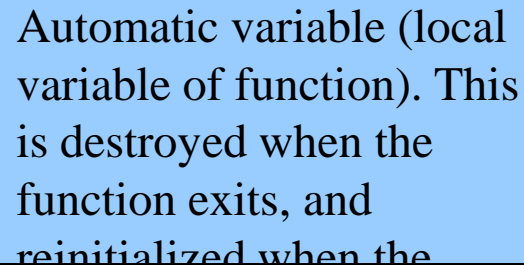
Create a new block, giving
x block scope. When the
block ends, this x is
destroyed.

```
27
28     cout << "local x in main's outer scope is " << x << endl;
29
30     useLocal();           // useLocal has local x
31     useStaticLocal();     // useStaticLocal has static local x
32     useGlobal();          // useGlobal uses global x
33     useLocal();           // useLocal reinitializes its local x
34     useStaticLocal();     // static local x retains its prior value
35     useGlobal();          // global x also retains its value
36
37     cout << "\nlocal x in main is " << x << endl;
38
39     return 0;             // indicates successful termination
40
41 } // end main
42
```

Phạm vi biến

Phạm vi biến

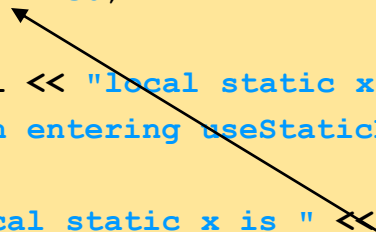
```
43 // useLocal reinitializes local variable x during each call
44 void useLocal( void )
45 {
46     int x = 25; // initialized each time useLocal is called
47
48     cout << endl << "local x is " << x << endl << " on entering useLocal"
49         << " on entering useLocal"
50     ++x;
51     cout << "local x is " << x << endl << " on exiting useLocal"
52         << " on exiting useLocal"
53
54 } // end function useLocal
55
```



Automatic variable (local variable of function). This is destroyed when the function exits, and reinitialized when the function begins.

Phạm vi biến

```
56 // useStaticLocal initializes static local variable x only the
57 // first time the function is called; value of x is saved
58 // between calls to this function
59 void useStaticLocal( void )
60 {
61     // initialized only first time useStaticLocal is called
62     static int x = 50;
63
64     cout << endl << "local static x is " << x
65         << " on entering useStaticLocal" << endl;
66     ++x;
67     cout << "local static x is " <<
68         << " on exiting useStaticL
69
70 } // end function useStaticLocal
71
```



Static local variable of function; it is initialized only once, and retains its value between function calls.

```

72 // useGlobal modifies global variable x during each call
73 void useGlobal( void )
74 {
75     cout << endl << "global x is " << x
76         << " on entering useGlobal" << endl;
77     x *= 10;
78     cout << "global x is " << x
79         << " on exiting useGlobal" << endl;
80
81 } // end function useGlobal

```

This function does not declare any variables. It uses the global x declared in the beginning of the program.

Phạm vi biến

```

local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

```

```

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

```

```

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

```

```

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

```

```
local x is 25 on entering useLocal
```

```
local x is 26 on exiting useLocal
```

```
local static x is 51 on entering useStaticLocal
```

```
local static x is 52 on exiting useStaticLocal
```

```
global x is 10 on entering useGlobal
```

```
global x is 100 on exiting useGlobal
```

```
local x in main is 5
```

Phạm vi biến

Tham chiếu



- ❖ Tham chiếu là địa chỉ vùng nhớ được cấp phát cho một biến.
- ❖ Ký hiệu & đặt trước biến hoặc hàm để xác định tham chiếu của chúng
- ❖ Ví dụ 1:
 - `int x = 10, *px = &x, &y = x;`
 - `*px = 20; // *px = x = y = 20`
 - `y = 30; // y = x = *px = 30`
- ❖ Ví dụ 2:
 - `int arrget(int *a, int i) { return a[i]; }`
 - `int arrget(int a[], int i) { return a[i]; }`
- ❖ Ví dụ 3:
 - `void swap1(int x, int y) { int t = x; x = y; y = t; }`
 - `void swap2(int *x, int *y) { int *t = *x; *x = *y; *y = *t; }`
 - `void swap3(int &x, int &y) { int t = x; x = y; y = t; }`

Ví dụ

```
1 // Fig. 3.20: fig03_20.cpp
2 // Comparing pass-by-value and pass-by-reference
3 // with references.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 int squareByValue( int );           // function definition
10 void squareByReference( int & );    // function prototype
11
12 int main()
13 {
14     int x = 2;
15     int z = 4;
16
17     // demonstrate squareByValue
18     cout << "x = " << x << " before squareByValue\n";
19     cout << "Value returned by squareByValue: "
20         << squareByValue( x ) << endl;
21     cout << "x = " << x << " after squareByValue\n" << endl;
22
```

Notice the & operator,
indicating pass-by-
reference.

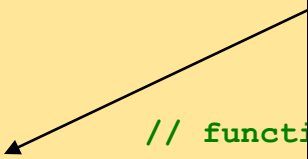


fig03_20.cpp
(2 of 2)

```
23 // demonstrate squareByReference
24 cout << "z = " << z << " before squareByReference" << endl;
25 squareByReference( z );
26 cout << "z = " << z << " after squareByReference" << endl;
27
28 return 0; // indicates successful termination
29 } // end main
30
31 // squareByValue multiplies number by itself
32 // result in number and returns the new value
33 int squareByValue( int number )
34 {
35     return number *= number; // caller's argument not modified
36
37 } // end function squareByValue
38
39 // squareByReference multiplies numberRef by itself
40 // stores the result in the variable to which numberRef
41 // refers in function main
42 void squareByReference( int &numberRef )
43 {
44     numberRef *= numberRef; // caller's argument modified
45
46 } // end function squareByReference
```

Changes number, but original parameter (x) is not modified.

Changes numberRef, an alias for the original parameter. Thus, z is changed.

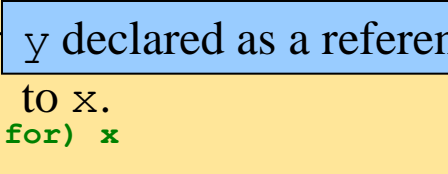
```
x = 2 before squareByValue  
Value returned by squareByValue: 4  
x = 2 after squareByValue
```

```
z = 4 before squareByReference  
z = 16 after squareByReference
```

fig03_20.cpp
output (1 of 1)

Ví dụ

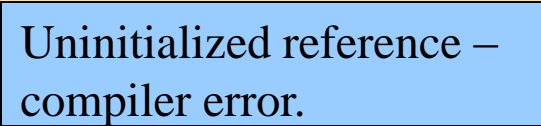
```
1 // Fig. 3.21: fig03_21.cpp
2 // References must be initialized.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     int x = 3;
11
12     // y refers to (is an alias for) x
13     int &y = x;
14
15     cout << "x = " << x << endl << "y = " << y << endl;
16     y = 7;
17     cout << "x = " << x << endl << "y = " << y << endl;
18
19     return 0; // indicates successful termination
20
21 } // end main
```



```
x = 3
y = 3
x = 7
y = 7
```

Ví dụ

```
1 // Fig. 3.22: fig03_22.cpp
2 // References must be initialized.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     int x = 3;
11     int &y; // Error: y must be initialized
12
13     cout << "x = " << x << endl << "y = " << y << endl;
14     y = 7;
15     cout << "x = " << x << endl << "y = " << y << endl;
16
17     return 0; // indicates successful termination
18
19 } // end main
```



Borland C++ command-line compiler error message:

```
Error E2304 Fig03_22.cpp 11: Reference variable 'y' must be
  initialized in function main()
```

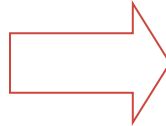
Microsoft Visual C++ compiler error message:

```
D:\cpphttp4_examples\ch03\Fig03_22.cpp(11) : error C2530: 'y' :
  references must be initialized
```

Đặt vấn đề - Nạp chồng hàm (*Functions overloading*)



```
int abs(int i);  
long labs(long l);  
double fabs(double d);
```



```
int abs(int i);  
long abs(long l);  
double abs(double d);
```

```
int abs(int i) { return abs(i); }  
long abs(long l) { return labs(l); }  
double abs(double d) { return fabs(d); }  
  
void test_abs() {  
    int i = abs(10);    // abs(int )  
    long l = abs(-10l); // abs(long )  
    double = abs(0.11); // abs(double )  
}
```

Nạp chồng hàm

(*Function overloading*)



❖ Một số lưu ý:

- Các hàm nạp chồng sẽ có tên trùng nhau nhưng các tham số của hàm khác nhau
- Các hàm nạp chồng nên có các tác vụ giống nhau.

- Ví dụ chức năng của hàm ***square(int)*** và hàm ***square(float)*** là như nhau:

```
int square( int x) {return x * x;}  
float square(float x) { return x * x; }
```

Ví dụ

Overloaded functions have the same name, but the different parameters distinguish them.

```
1 // Fig. 3.25: fig03_25.cpp
2 // Using overloaded functions.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // function square for int values
9 int square( int x )
10 {
11     cout << "Called square with int argument: " << x << endl;
12     return x * x;
13 }
14 // end int version of function square
15
16 // function square for double values
17 double square( double y )
18 {
19     cout << "Called square with double argument: " << y << endl;
20     return y * y;
21 }
22 // end double version of function square
23
```

```

24 int main()
25 {
26     int intResult = square( 7 );           // calls int version
27     double doubleResult = square( 7.5 ); // calls double version
28
29     cout << "\nThe square of integer 7 is 49\n";
30     cout << "\nThe square of double 7.5 is 56.25\n";
31     cout << endl;
32
33     return 0; // indicates success
34
35 } // end main

```

The proper function is called based upon the argument (int or double).

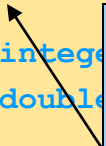


fig03_25.cpp
(2 of 2)

fig03_25.cpp
output (1 of 1)

```

Called square with int argument: 7
Called square with double argument: 7.5

The square of integer 7 is 49
The square of double 7.5 is 56.25

```

Tham số ngầm định trong lời gọi hàm

```
void inc(int &a, int b = 1);  
void inc(int &a, int b) {  
    a = a + b;  
}  
  
int x = 5, y = 10;  
inc(x, 10);           // x = x + 10  
inc(y);               // y = y + 1
```

Lưu ý:

- Các tham số có giá trị ngầm định phải đặt cuối danh sách tham số, để tránh nhầm lẫn các giá trị.
- Các giá trị ngầm định của tham số chỉ được khai báo trong khuôn mẫu hàm

```

1  // Fig. 3.23: fig03_23.cpp
2  // Using default arguments.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function prototype that specifies default arguments
9  int boxVolume( int length = 1, int width = 1, int height = 1 );
10
11 int main()
12 {
13     // no arguments--use default values for all dimensions
14     cout << "The default box volume is: " << boxVolume();
15
16     // specify length; default width and height
17     cout << "\n\nThe volume of a box with length 10,\n"
18         << "width 1 and height 1 is: " << boxVolume( 10 );
19
20     // specify length and width; default height
21     cout << "\n\nThe volume of a box with length 10,\n"
22         << "width 5 and height 1 is: " << boxVolume( 10, 5 );
23

```

Set defaults in function prototype.

Function calls with some parameters missing – the rightmost parameters get their defaults.

Ví dụ

Ví dụ

```
24 // specify all arguments
25 cout << "\n\nThe volume of a box with length 10,\n"
26     << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
27     << endl;
28
29 return 0; // indicates successful termination
30
31 } // end main
32
33 // function boxVolume calculates the volume of a box
34 int boxVolume( int length, int width, int height )
35 {
36     return length * width * height;
37
38 } // end function boxVolume
```

The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100

Toán tử quản lý bộ nhớ động



❖ Toán tử cấp phát bộ nhớ động new

```
int *x;
```

```
x = new int;           // x = (int*)malloc(sizeof(int));
```

```
char *y;
```

```
y = new char[100];     // y = (char*)malloc(100);
```

❖ Toán tử giải phóng vùng nhớ động delete

```
delete x;              // free(x);
```

```
delete y;              // free(y);
```

Const



❖ Nên khai báo hằng đối với:

- Các đối tượng mà ta không định sửa đổi
 - **const** double PI = 3.14;
 - **const** Date openDate(18,8,2018);
- Các tham số của hàm mà ta không định cho hàm đó sửa đổi
 - void printHeight(**const** LargeObj &LO)
 { cout << LO.height; }
- Các hàm thành viên không thay đổi đối tượng chủ
 - int Date::getDay() **const** { return day; }

Function Templates



- ❖ Compact way to make overloaded functions
 - Generate separate function for different data types
- ❖ Format
 - Begin with keyword **template**
 - Formal type parameters in brackets <>

Function Templates



❖ Example

```
template < class T > // or template < typename T >
T square( T value1 )
{
    return value1 * value1;
}
```

- **T** is a formal type, used as parameter type
 - Above function returns variable of same type as parameter
- In function call, **T** replaced by real type
 - If **int**, all **T**'s become **int**

```
int x;
int y = square(x);
```

fig03_27.cpp
(1 of 3)

```
1  // Fig. 3.27: fig03_27.cpp
2  // Using a function template.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // definition of function template
10 template < class T > // or templ
11 T maximum( T value1, T value2, T value3,
12 {
13     T max = value1;
14
15     if ( value2 > max )
16         max = value2;
17
18     if ( value3 > max )
19         max = value3;
20
21     return max;
22
23 } // end function template maximum
24
```

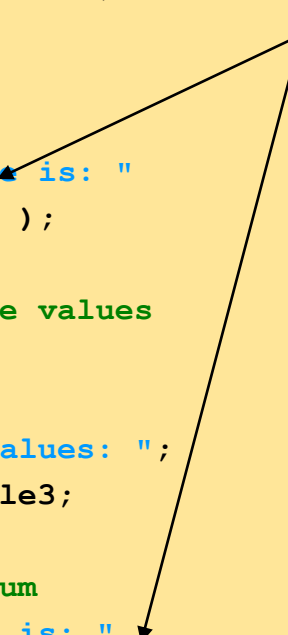
Formal type parameter T
placeholder for type of
data to be tested by
maximum.

maximum expects all
parameters to be of the
same type.

fig03_27.cpp
(2 of 3)

```
25 int main()
26 {
27     // demonstrate maximum with int values
28     int int1, int2, int3;
29
30     cout << "Input three integer values: ";
31     cin >> int1 >> int2 >> int3;
32
33     // invoke int version of maximum
34     cout << "The maximum integer value is: "
35           << maximum( int1, int2, int3 );
36
37     // demonstrate maximum with double values
38     double double1, double2, double3;
39
40     cout << "\n\nInput three double values: ";
41     cin >> double1 >> double2 >> double3;
42
43     // invoke double version of maximum
44     cout << "The maximum double value is: "
45           << maximum( double1, double2, double3 );
46
```

maximum called with
various data types.



```

47 // demonstrate maximum with char values
48 char char1, char2, char3;
49
50 cout << "\n\nInput three characters: ";
51 cin >> char1 >> char2 >> char3;
52
53 // invoke char version of maximum
54 cout << "The maximum character value is: "
55      << maximum( char1, char2, char3 )
56      << endl;
57
58 return 0; // indicates successful termination
59
60 } // end main

```

fig03_27.cpp
(3 of 3)

fig03_27.cpp
output (1 of 1)

```

Input three integer values: 1 2 3
The maximum integer value is: 3

```

```

Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3

```

```

Input three characters: A C B
The maximum character value is: C

```