Ngôn ngữ lập trình C++

Chương 2. Lập trình hướng cấu trúc

2.1 Một số kỹ thuật lập trình

1) Lập trình tuyến tính
2) Lập trình hướng cấu trúc/thủ tục
3) Lập trình hướng đối tượng

   $\Rightarrow$Lập trình hướng sự kiện

   $\Rightarrow$Lập trình hướng thành phần

Lập trình tuyến tính

- Tư duy theo lối tuần tự, đơn giản và đơn luồng $\Rightarrow$ không thể giải quyết các ứng dụng phức tạp $\Rightarrow$ sử dụng trong phạm vi các mô-đun nhỏ nhất của chương trình.

Lập trình hướng cấu trúc/thủ tục

- Dựa trên ba cấu trúc: tuần tự, phân nhánh và vòng lặp.
- Chương trình được chia thành các hàm (chương trình con) thực hiện một nhiệm vụ/phương thức nhất định.
- **Chương trình = Cấu trúc dữ liệu + Giải thuật**

$\Rightarrow$Quan tâm đến nhiệm vụ, ít quan tâm đến dữ liệu.

$\Rightarrow$Khi thay đổi cấu trúc dữ liệu thì giải thuật phải thay đổi theo $\rightarrow$ viết lại hàm.

$\Rightarrow$Không phù hợp với các dự án lớn.

Lập trình hướng đối tượng

- Quy ước các thực thể trong chương trình thành các đối tượng, trừu tượng hóa các đối tượng thành các lớp đối tượng.
- Đối tượng mô tả sự liên kết giữa dữ liệu và các thủ tục/phương thức thao tác trên dữ liệu đó.

Đối tượng = Dữ liệu + phương thức

- Dữ liệu được tổ chức thành các lớp đối tượng.

$\Rightarrow$ Tập trung vào dữ liệu.

$\Rightarrow$ Chương trình được chia thành các đối tượng hoàn toàn độc lập.

$\Rightarrow$ Dữ liệu được đóng gói, truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp $\rightarrow$ bảo mật dữ liệu.

$\Rightarrow$ Có tính kế thừa $\rightarrow$ Sử dụng lại mã nguồn.

$\Rightarrow$ Có tính tiến hóa: Tính đa hình cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau.

2.2. Lập trình hướng cấu trúc/thủ tục

- Hàm/Chương trình con là một mô-đun chương trình độc lập thực hiện một nhiệm vụ nhất định.
- Lý do dùng hàm/chương trình con:
  - Chia chương trình lớn thành nhiều chương trình con (chia để trị) $\Rightarrow$ dễ tìm lỗi, bảo trì, nâng cấp.
  - Tái sử dụng.
- Cách viết hàm trong C++
- Lời gọi hàm trong C++
- Biến cục bộ và biến toàn cục

A. Cách viết hàm trong C++

- Mỗi chương trình C/C++ có ít nhất một hàm **main().**
- Hầu hết các chương trình đều định nghĩa thêm các hàm.
- Phân loại hàm:
  - Hàm có sẵn trong các thư viện của C++
  - Hàm do người lập trình tự định nghĩa

A. Cách viết hàm trong C++

- Định nghĩa hàm:

<kiểu dữ liệu trả về > tên_hàm(danh sách tham số)

{

Thân hàm

}

- Kiểu dữ liệu trả về: kiểu dữ liệu của giá trị mà hàm trả về, nếu hàm **không** trả về giá trị nào $\Rightarrow$ khai báo kiểu **void**.
- Danh sách tham số: Khai báo kiểu dữ liệu và thứ tự các tham số, số lượng tham số do người lập trình quyết định $\Rightarrow$ Mới chỉ là tham số hình thức, chỉ trở thành tham số thực sự khi được truyền tham số.
- Giá trị hàm trả về viết trong thân hàm bằng lệnh

return biểu_thức;

- Kiểu dữ liệu trả về là **void** lệnh **return;** chỉ để kết thúc hàm.
- Hàm được viết **bên ngoài** hàm **main()**

A. Cách viết hàm trong C++

- Khai báo hàm nguyên mẫu:

  <kiểu dữ liệu trả về> tên_hàm(danh sách kiểu dữ liệu của các tham số);

- Hàm nguyên mẫu: Một dạng khai báo hàm, được sử dụng để thông báo cho chương trình biên dịch biết kiểu dữ liệu trả về, kiểu, số lượng và thứ tự của các tham số được truyền cho hàm.
- Chỉ quan tâm đến kiểu dữ liệu của tham số, không quan tâm đến tên tham số (trừ tham số là mảng)
- Ví dụ : **float tong(float, float);**
- Khai báo hàm nguyên mẫu **trước** hàm **main()**.

B. Lời gọi hàm trong C++

- Lời gọi hàm bao gồm:
  - Tên hàm
  - Các tham số **truyền cho hàm**
- Truyền tham số cho hàm $\Rightarrow$ tham số hình thức trở thành tham số thực sự.
- Cần đảm bảo chất và lượng:
  - Chất: Kiểu dữ liệu và thứ tự các tham số như đã khai báo.
  - Lượng: Số lượng các tham số như đã khai báo
- Giá trị mặc định cho các tham số: Tham số có một giá trị khởi tạo tại thời điểm khai báo.
  - Khi người dùng không cung cấp đối số cho tham số mặc định, giá trị mặc định sẽ được sử dụng.
  - Khi người dùng cung cấp đối số cho tham số mặc định, tham số sẽ được gán lại bằng giá trị của đối số được truyền vào.
  - Cách khai báo tham số mặc định: Sử dụng toán tử gán như khởi tạo biến thông thường.
  - **Chú ý:** Mọi tham số mặc định khi khai báo phải đặt phía sau tham số không có giá trị mặc định.
  - **Ưu điểm:** Giúp người dùng có nhiều lựa chọn hơn trong việc truyền đối số.

B. Lời gọi hàm trong C++

- Các kiểu truyền tham số cho hàm:
  - Truyền theo trị: Hàm không làm thay đổi được giá trị của tham số thực sự.
  - Truyền theo biến: Hàm làm thay đổi được giá trị của tham số thực sự.
- Các kiểu tham số trong C++:
  - Tham trị
  - Tham chiếu
  - Con trỏ
- Mảng có thể làm tham số cho hàm trong C++

C. Chồng hàm trong C++

- Cho phép sử dụng cùng một tên gọi cho các hàm giống nhau (có cùng mục đích) nhưng khác nhau về kiểu dữ liệu tham số hoặc số lượng tham số.
- Chỉ có trong C++.
- Có thể nạp chồng hàm nếu như kiểu của tham số trong hàm phải khác nhau hoặc có số lượng các tham số khác nhau.
- Không thể nạp chồng hàm nếu như chỉ khác biệt về kiểu trả về.

D. Biến toàn cục và biến cục bộ

- Phân loại biến dựa trên:
  - **Phạm vi của biến**: Xác định nơi có thể truy cập vào biến.
  - **Thời gian tồn tại của biến**: Xác định khi nào biến được tạo ra và bị hủy.

D. Biến toàn cục và biến cục bộ

- **Biến toàn cục:** Biến được khai báo ở đầu tệp tin (sau #include…), bên ngoài tất cả các hàm.
- Biến toàn cục có **thời gian tĩnh:** Được tạo ra khi chương trình bắt đầu và chỉ bị hủy khi chương trình kết thúc.
- Lý do cần hạn chế dùng biến toàn cục:
  - Giá trị của nó có thể bị **thay đổi bởi bất cứ hàm nào** mỗi khi hàm đó được gọi $\Rightarrow$ **khó kiểm soát**
  - Mỗi hàm thực hiện **một chức năng khác nhau**, **hoạt động độc lập** $\Rightarrow$ Một hàm sử dụng biến toàn cục sẽ phụ thuộc hoàn toàn vào biến toàn cục nên không thể tái sử dụng và không hoạt động độc lập được.
- Khi nào nên dùng biến toàn cục: Với những loại dữ liệu muốn sử dụng cho toàn bộ chương trình.

D. Biến toàn cục và biến cục bộ

- **Biến cục bộ:** Biến được định nghĩa bên trong một khối lệnh.
- Các biến cục bộ có **thời gian động**: Được tạo tại thời điểm định nghĩa, và bị hủy khi ra khỏi khối lệnh mà biến đó được định nghĩa.
- Các biến cục bộ chỉ có nghĩa ở phạm vi bên trong khối lệnh (cục bộ) $\Rightarrow$ sẽ không truy cập được biến khi ở bên ngoài khối lệnh.
- Biến cục bộ trùng tên với biến toàn cục sẽ ẩn các biến toàn cục trong khối lệnh của biến cục bộ được định nghĩa $\Rightarrow$ có thể sử dụng toán tử phạm vi (scope operator ::) để thông báo cho trình biên dịch biết đó là biến cục bộ hay biến toàn cục $\Rightarrow$ **tránh** việc đặt tên biến toàn cục và biến cục bộ trùng nhau.
- Nên định nghĩa các biến trong phạm vi nhỏ nhất có thể.