Phương pháp Lập trình Hướng đối tượng

Nhìn lại lập trình cơ bản Qua lăng kính của Lập trình Hướng đối tượng

GV: Lê Xuân Định

Cấu trúc Điều khiển

– *Khung xương của Chương trình* –

cuu duong than cong . com

Cấu trúc Điều khiển

- Lựa chọn (Rẽ nhánh có Điều kiện)
  - Rẽ đôi: **`if; if else;`**
  - Rẽ nhiều nhánh: **`switch case break; if else if...;`**
- Vòng lặp
  - Lặp xác định: **`for;`**
  - Lặp không xác định: **`while; do while; for;`**
- Lệnh nhảy (Rẽ nhánh không điều kiện)

  (*nếu không nắm vững thì đừng dùng!*)
  - Điều khiển vòng lặp: **`break; continue;`**
  - Kết thúc hàm: **`return; return`** ***`giá_trị`*****`;`**
  - Kết thúc chương trình: **`exit(-1);`**

Lựa chọn (Rẽ nhánh có ĐK)

- Rẽ đôi: **`if; if else;`**
- Rẽ nhiều nhánh
  - **`switch(`*biến_nguyên*`){`** `// kiểu char, int, long`
    `case` ***giá_trị_nguyên_1*****`:`** công việc 1; ***break***;
    `case` ***giá_trị_nguyên_2*****`:`** công việc 2; ***break***;
    `...`
    `default:` công việc mặc định; ***break***;
    `}`
  - **`if(`*điều_kiện_1*`){`** công việc 1; **`}`**
    `else if(`*điều_kiện_2*`){` công việc 2; **`}`**
    `...`
    `else{` công việc mặc định; **`}`**

Vòng lặp

- Lặp xác định:
  - `for(int i=`*giá_trị_đầu*`; i <` *giá_trị_cuối* `+1; i++){...}`
  - `for(int i=`*giá_trị_đầu*`; i <=` *giá_trị_cuối*`; i++){...}`
- Lặp không xác định:
  - **`while(`*`điều_kiện_lặp`*`){`** thực hiện... *nếu* **thoả *điều_kiện_lặp***; **}**
  - **`do{`** thực hiện **lần đầu**, và những lần sau *nếu* **thoả *điều_kiện_lặp***; **}**
    `while(`*`điều_kiện_lặp`*`);` `//` cho đến khi *điều_kiện_lặp* **không thoả**.
  - **`for(`**khởi tạo; ***điều_kiện_lặp;*** tăng biến chạy**)**
    { thực hiện ... *nếu* **thoả *điều_kiện_lặp***; **}**
  - Sau vòng lặp, *điều_kiện_lặp* **không thoả**. VD:

```
int i=1; for(; i<=5; i+=3){ cout<<i<<" "; } // 1 4
cout<<i<<endl; // 7
```

Lệnh nhảy (Rẽ nhánh không ĐK)

- Điều khiển vòng lặp:
  - **Kết thúc** vòng lặp trong cùng: **`break;`**
  - **Quay lại đầu** vòng lặp trong cùng: **`continue;`**
- Kết thúc hàm: **`return; return`** ***giá_trị*****`;`**
  - Không thực hiện ***các lệnh sau return***.

    VD: "*Tính điểmTK = (điểmLT + điểmTH)/2, cắt xuống 10 nếu vượt quá 10.*"

```
float tinhDiemTK(float diemLT, float diemTH){
  float diemTK = (diemLT + diemTH)/2;
  return diemTK;
  if(diemTK > 10){ return 10; }
}
```

- Kết thúc chương trình: **`exit(`*****mã_lỗi*****`);`**
  - Thực ra đây là một ***hàm***: *mã_lỗi* = **0** nghĩa là **không có lỗi**!

Biến

– Đơn vị lưu trữ Dữ liệu –

Đơn vị lưu trữ dữ liệu

- Mỗi dữ liệu trong chương trình đều phải lưu trong một **biến** nào đó.
  - Tương đương với 1 **danh từ** [(1)] trong ngôn ngữ tự nhiên.
- Ví dụ:
  - Tính tổng tất cả các ước số của một số nguyên cho trước.
  - Cho một mảng các số thập phân, tìm số lớn nhất trong những phần tử mảng nhỏ hơn một số nguyên cho trước.
  - Kiểm tra xem tổng các số trong một mảng các số nguyên có phải là một số nguyên tố hay không.
  - Hãy viết chương trình cho phép nhập điểm (lý thuyết, thực hành) của một SV từ bàn phím, và xuất ra màn hình điểm tổng kết của SV đó.

---

1) Với các biến *cờ hiệu (đúng/sai)*, ta thường đặt tên là *tính từ* tương ứng.

Đơn vị lưu trữ dữ liệu

- Mỗi dữ liệu trong chương trình đều phải lưu trong một **biến** nào đó.
  - Tương đương với 1 **danh từ** [1] trong ngôn ngữ tự nhiên.
- Ví dụ:
  - Tính ***tổng*** tất cả **các ước số** của một **số nguyên** cho trước.
  - Cho một **mảng các số thập phân**, tìm ***số lớn nhất*** trong những phần tử mảng nhỏ hơn một **số nguyên** cho trước.
  - Kiểm tra xem **tổng các số** trong một **mảng các số nguyên** có phải là một số nguyên tố hay không.
  - Hãy viết chương trình cho phép nhập **điểm (lý thuyết, thực hành, điểm cộng)** của một SV từ bàn phím, và xuất ra màn hình điểm tổng kết của SV đó.

---

1) Với các biến *cờ hiệu (đúng/sai)*, ta thường đặt tên là *tính từ* tương ứng.

Biến – Kiểu, Giá trị

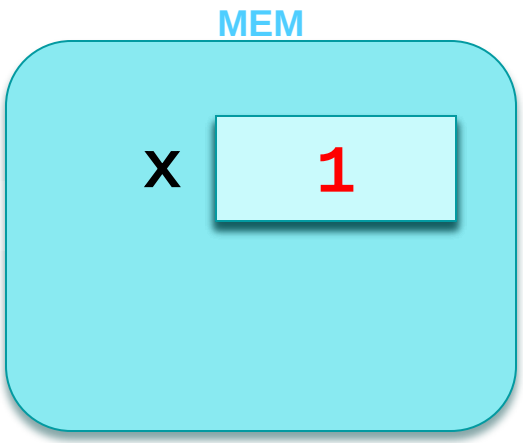
- Mỗi biến phải gắn liền với 1 **kiểu** dữ liệu nào đó.
  - **Khai báo**, VD: `int x;`
- Các cách **sử dụng** biến:
  - **Đọc**: Lấy ra giá trị của biến để đưa vào...
    - Công thức, VD: `diemTK = (6*diemLT + 4*diemTH)/10 + diemCong;`
    - Hàm, VD: `printf("Diem tong ket: %f\n", diemTK);`
    - Gán vào biến khác, VD: `tam = x;`

  ✶ Trước khi đọc thì biến phải có *dữ liệu xác định* (được ***ghi vào từ trước***.)

  - **Ghi**: Gán giá trị nào đó vào biến thông qua...
    - Phép gán, VD: `diemTK = (6*diemLT + 4*diemTH)/10;`
    - Tham biến trong hàm, VD: `scanf("Diem ly thuyet: %f", &diemLT);`

Biến – Kiểu, Giá trị, Ô nhớ

- Mỗi biến phải gắn liền với 1 **kiểu** dữ liệu nào đó.
  - **Khai báo**, VD: `int x;`
  - Nên gộp với đặt giá trị mặc định thành "***khởi tạo***":
    VD: `int x = 1;`
    //→ Đọc máy móc: khai báo biến ***tên*** x có ***kiểu*** int và được khởi tạo bằng ***giá trị*** 1.
    //→ Đọc tự nhiên: *khai báo biến số nguyên x được khởi tạo bằng 1.*
  - ✶ Trước khi đọc thì biến phải có *dữ liệu xác định* (được ***ghi vào từ trước***.)

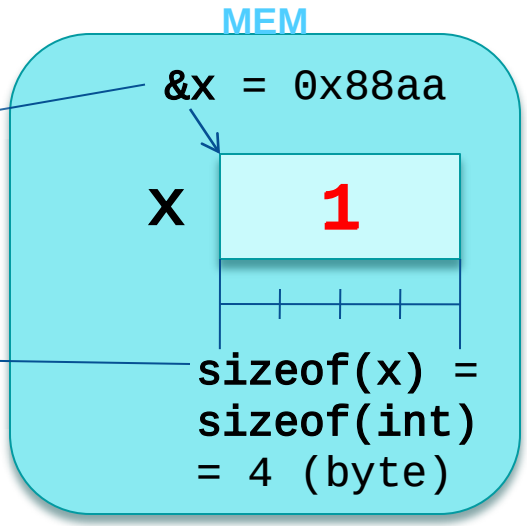


- Khi **sử dụng (đọc)** biến, phải xác định được giá trị (dữ liệu) của nó.

Các thuộc tính của Biến

```
int x = 1;
```

- Biến:
  - Thuộc tính *logic*: kiểu, tên, giá trị
  - Thuộc tính *vật lý*: vùng nhớ (ô nhớ)
    - Địa chỉ
    - Kích thước



Vòng đời của Biến

- Biến:
  - Thuộc tính *logic*: kiểu, tên, giá trị
  - Thuộc tính *vật lý*: vùng nhớ (ô nhớ)
- Sử dụng biến, VD:

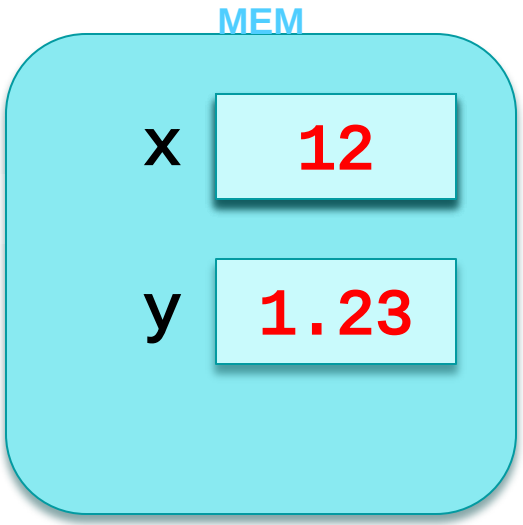
```
{
  int x = 1;
  {
    x = 10;
    float y = 1.23;
    x = floor(y*x);
  }
}
```

Tạo biến (*cấp phát* bộ nhớ)

Ghi giá trị

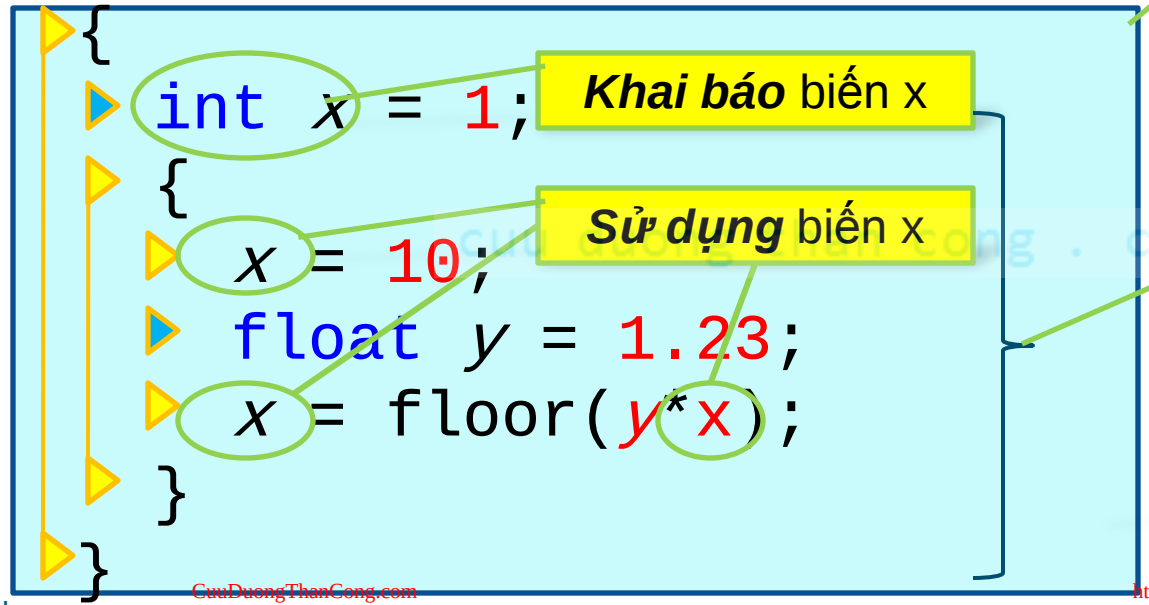
Đọc giá trị

Huỷ biến (*dọn dẹp* bộ nhớ)



Tầm vực của Biến

- Biến:
  - Vòng đời: tạo, đọc/ghi, huỷ
  - Tầm vực: chỗ khai báo → cuối khối lệnh KB
- Sử dụng biến, VD:



Khối lệnh KB của biến x := *Khối lệnh* trong cùng chứa câu khai báo biến x

Tầm vực của biến x := ***Phạm vi sử dụng*** biến x := Từ chỗ *khai báo* biến x đến *cuối khối lệnh KB* của nó

Tầm vực & Vòng đời của Biến

- Thảo luận: Tầm vực & thời gian sống của...
  - Tham số (tham trị / tham biến)
    - Tham biến ≠ Tham trị!
  - Biến trong vòng lặp
    - Biến điều khiển lặp (biến chạy) ≠ Biến khai báo trong vòng lặp!
  - Biến toàn cục & static
    - Biến static ≠ Biến toàn cục!
  - Biến cấp phát động
    - Biến con trỏ (p) ≠ Biến được trỏ tới (*p)
  - Tầm vực có trùng với thời gian sống?
    - Thường thì trùng, nhưng đó là 2 khái niệm độc lập

Sơ kết về Biến

- Biến là đơn vị lưu trữ dữ liệu để xử lý.
  - "Đơn vị": Được sử dụng như ***một khối liền***, không chia nhỏ.
  - Thuộc tính *logic*: kiểu, tên, giá trị
  - Thuộc tính *vật lý*: vùng nhớ (ô nhớ)
    - Địa chỉ, kích thước, cấu trúc (với các biến kiểu phức)
  - Vòng đời: tạo, đọc/ghi, huỷ
    - Có nhiều cách sử dụng biến (*gán, tính toán, truyền tham số,...*) nhưng đều quy được về *đọc/ghi*.
  - Tầm vực: chỗ khai báo → cuối khối lệnh KB
    - "Khối lệnh KB": *khối lệnh trong cùng* chứa câu khai báo biến
      - Riêng biến *toàn cục* thì có tầm vực đến hết chương trình (không nên dùng).

Hoạt cảnh Sử dụng Biến

- Demo vòng đời của biến & con trỏ

Hoạt cảnh Sử dụng Biến

- Tính thương (nguyên) của 2 số nguyên dương

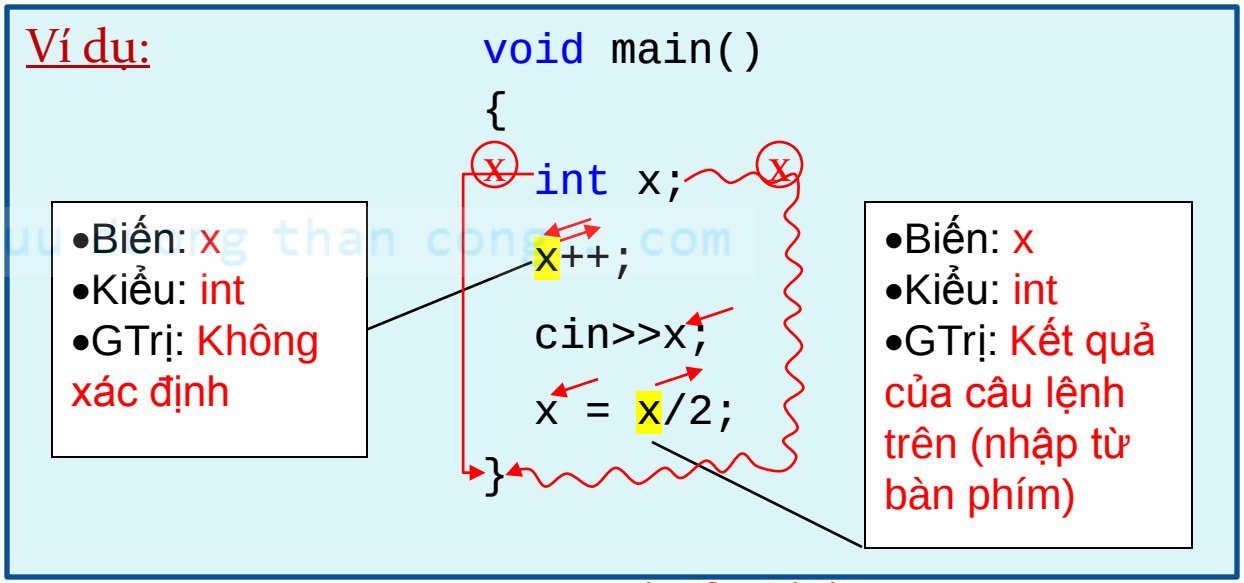
```
int Thuong(int a, int b)
{ int* p = new int(0);
  for(int i=0; a >= b; i++){
    int t = a-b;
    *p = i+1;
    a = t;
  }
  return *p;
}
```

Hoạt cảnh Sử dụng Biến

BT Ứng dụng 1 (về nhà)

- Hãy chú thích vòng đời & tầm vực của các biến được tô vàng trong chương trình "Tính điểm SV"
  - Ở những chỗ tô vàng, vẽ ô chú thích: Tên biến, kiểu, giá trị
  - Vẽ vòng đời xuyên qua tất cả những hàm có thể gọi tới.
  - Coi câu lệnh "T * p;" là khai báo cả con trỏ **p** lẫn biến ***p**

- ↳ : Tầm vực
- ↝ : Vòng đời
- ↗ : Đọc biến
- ↙ : Ghi biến



Biến Đối tượng

– Biến biết tự xử lý dữ liệu –

"Object = Var + Func"

Biến Đối tượng trong C++

- Thiết bị nhập/xuất chuẩn
  - C++ định nghĩa sẵn *2 biến* **`cin, cout`** trong thư viện <iostream>
  - Dùng 2 toán tử tương ứng để nhập/xuất
    `cin >>` biến`;` **`cout <<`** dữ_liệu`;`
- Chuỗi ký tự
  - C++ định nghĩa sẵn *kiểu* **`string`** trong thư viện <string> thay cho kiểu `char*`.
  - Khai báo biến: `string s;`
    - Khởi tạo: `string s =` ***`string`***`(`**`“<giá trị khởi tạo>”`**`);`
  - Nhập xuất: `cin >> s; cout << s;`

Như một biến bình thường

Biến string	Biến int
Khai báo biến & Khởi tạo:	
`string s = string("abc"), s1, s2;`	`int i = 2, i1, i2;`
Gán:	
`s2 = string("def"); s1 = s2;`	`i2 = 5; i1 = i2;`
Truyền Tham số cho Hàm:	

`fs(s); fs(string("ghi"));`
với hàm *fs*( ) được khai báo như sau:
`void fs(string st);`
	`f(i); f(6);`
	với hàm *f*( ) được khai báo như sau:
	`void f(int n);`

- Với ***string*(char* ist)** là *hàm tạo* đối tượng string từ chuỗi cổ điển (char*).
  - *Hàm tạo* đối tượng luôn có tên trùng với tên kiểu.

Nhưng "thông minh" hơn

- Không chỉ chứa dữ liệu
  - `string s=string("abc");` // s chứa 3 ký tự 'a', 'b', 'c'
- Mà còn biết tự xử lý dữ liệu của mình
  - Tính độ dài: `s.length()` // trả về số nguyên 3 với s trên
  - So sánh theo thứ tự từ điển: `s1.compare(s2)` // trả về 0, số âm, số dương tương ứng với s1==s2, s1 < s2, s1 > s2
  - Xoá một phần (chuỗi con): `s.erase(1, 2);` // xoá 2 ký tự bắt đầu từ vị trí 1 (xoá "bc" với s bên trên, còn lại chuỗi "a")
  - Cho truy cập đến từng phần tử: `s.at(0)` // trả về ký tự ở vị trí 0 (tức 'a' với s bên trên)
  - ...

Nhưng "thông minh" hơn

- Và quan trọng nhất là *luôn biết* ***đảm bảo an toàn dữ liệu!***
  - Quên khởi tạo cũng không sao: `string s;` // Giá trị mặc định là chuỗi rỗng ("")
  - Xoá quá tay?: `s.erase(1, 20);` // không thực hiện (xoá mù quáng) mà chỉ báo lỗi "tham số không hợp lệ"
  - Truy cập đến vị trí không hợp lệ?: `s.at(-1)` // không thực hiện (mù quáng) mà chỉ báo lỗi "tham số không hợp lệ"
  - `...`

Ví dụ Sử dụng Đối tượng string

Chương trình cho nhập 2 chuỗi và so sánh chúng theo thứ tự từ điển.

- "`string`" là một kiểu → khai báo biến: `string s, t;`
- `s` là một đối tượng → biết tự tính độ dài của mình.
- `t` là một biến → truyền tham số cho hàm so sánh của s.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
  string s, t;
  cout<<"do dai s: "<<s.length()<<endl;
  cin>>s>>t;
  int sgt = s.compare(t);
  if(sgt==0){ cout<<"s == t"<<endl; }
  else if(sgt<0){ cout<<"s < t"<<endl; }
  else { cout<<"s > t"<<endl; }
}
```

BT Ứng dụng 2 (về nhà)

- Mỗi SV trong lớp học có các trường dữ liệu:
  - Điểm LT (0..10), điểm TH (0..10), điểm cộng (-1..1),
  - Mã số SV, tên (đều là chuỗi không có khoảng cách)
- Hãy viết chương trình cho nhập thông tin của các SV trong một lớp học, sắp xếp *tăng dần theo* ***tên*** rồi xuất ra màn hình danh sách *điểm tổng kết* các SV theo định dạng:

  STT <tab> **MSSV** <tab> **Tên** <tab> **ĐiểmTK**
  - ***điểm TK*** = (6 * ***điểm LT*** + 4 * ***điểm TH***)/10 + ***điểm Cộng***
  - *Yêu cầu: dùng* `cin`, `cout` *để nhập xuất và dùng* `string` *cho chuỗi.*

Bảng đối chiếu thuật ngữ

Tiếng Việt	Tiếng Anh	Chú thích
Biến	Variable	
Kiểu dữ liệu (tắt “Kiểu”)	Data type (abr. “Type”)	
Tầm vực	Scope	Mang ý nghĩa cú pháp, thể hiện qua các luật tầm vực.
Luật tầm vực	Scoping rule	
Vòng đời	Life cycle	Là ngữ nghĩa đằng sau cú pháp khai báo, tạo, sử dụng, huỷ biến
Thời gian sống	Lifetime	Là thời gian diễn ra 1 vòng đời
Hàm tạo, sự tạo đ.tượng	Constructor, construction	Tạo cả phần vật lý (vùng nhớ) lẫn phần logic (giá trị) cho biến.
Khởi tạo, sự kh.tạo g.trị	Initialize, initialization	Nhấn mạnh phần logic (giá trị)