C++ | GVC. Võ Xuân Thể

**4) Hằng (Constant):**

4.1 Khái niệm: Hằng

[1] **Hằng** là đại lượng có giá trị thuộc một kiểu dữ liệu nào đó, được chấp nhận bởi ngôn ngữ, giá trị của hằng ***không thay đổi*** trong thời gian tồn tại của nó (trong vòng đời của hằng).  Thay vì sử dụng giá trị hằng trực tiếp (trực hằng) trong chương trình, thì ta dùng tên (danh hiệu)  thay thế cho trực hằng đó

[2] chẳng hạn: PI = 3.14, g = 9.81……

[3] Hằng là một danh hiệu (Identifier), nên phải tuân thủ các qui tắc đặt tên danh hiệu *(xem [2] của 4 trong Bài 1)* [4] Một hằng thường cũng được xác định bởi các đặc trưng sau:

– Tên hằng.

– Kiểu dữ liệu: kiểu của hằng.

– Giá trị của hằng.

4.2.Cú pháp khai báo & khởi tạo hằng:

Có 2 cách khai báo & khởi tạo hằng:

**Cách 1:** dùng **#define** (hằng : không phải cấp phát vùng nhớ, chỉ dùng tên thay thế giá trị hằng)

Chỉ thị **#define** không phải là một lệnh thực thi, nó là chỉ thị tiền xử lý (preprocessor)  [dùng “bí danh”/alias cho giá trị hằng; không cần kết thúc bằng dấu chấm phẩy]

**#define <tên\_hằng> <giá\_trị>** // không có dấu ;

 trường hợp này chỉ có thể khai báo các hằng đơn giản, kiểu dữ liệu cơ bản . . .

**Cách 2:** dùng từ khóa **const** trước khai báo biến

 [xem “biến” là một “hằng”: phải cấp phát vùng nhớ; là một câu lênh khai báo nên phải kết thúc bằng ;]  **const <kiểu\_dữ\_liệu> <tên\_hằng> = <giá\_trị>;** // có dấu ;  trường hợp này có thể khai báo các trường hợp hằng; đặc biệt hằng theo kiểu phức tạp, có cấu trúc, . . . **Ví dụ:**

#define MAX 100 // không có dấu ;

#define PI 3.14 // khống dó dấu ;

const int MAX = 100;

const int x = 5; // có dấu ;

const int a = 5; // định nghĩa hằng a kiểu nguyên, có giá trị là 5

const float x = 4; // hằng x kiểu thực, có giá trị là 4.0

const int d = 7; // hằng d kiểu int, giá trị là 7

1

C++ | GVC. Võ Xuân Thể

|  |
| --- |
| **Bài 4: Chương trình con (SubProgram)** |

**I GIỚI THIỆU VỀ CHƯƠNG TRÌNH CON**

**1. Khái niệm CHƯƠNG TRÌNH CON**

\* Chương trình con = là một phần mã lệnh trong một chương trình lớn hơn, phần mã này: + thực hiện một tác vụ [hoặc một khối công việc] tương đối trọn vẹn và  + tương đối độc lập với phần mã lệnh còn lại.

\* Chương trình con thường được dùng trong 2 trường hợp (quen thuộc):

[1] Một tác vụ [một khối công việc] được lặp đi lặp lại nhiều lần (tại những vị trí khác nhau  trong chương lớn)

VD1: tính Tổ hợp không lặp có ! C(k,n) = n! / (k! (n-k)!)

 Thủ tục (hàm) tính giai thừa ! được sử dụng 3 lần trong chương trinh => viết thành chương  trình con

#include <conio.h>

#include <iostream.h>

|  |
| --- |
| **long giaithua(long n)**  **{ long kq = 1;**  **for(int i = 1; i<=n; i++) kq \*= i;** // kq = kq\*i;  **return kq;**  **}** |

void main()

{

 int n,k,kq;

 do{ cout<<"nhap n = "; cin>>n;

 cout<<"nhap k = "; cin>>k;

 }while(n<=0 || k<=0 || k>=n);

|  |
| --- |
| **kq = giaithua(n)/(giaithua(k)\*giaithua(n-k));** |

cout<<"To hop C cua “<<n<<” chap “<<k<<” phan tu = "<<kq<<endl;  getch();

}

 Chú ý: có thể nó được lặp lại [gọi bởi] trong chính nó = gọi là “đệ qui”.

2 

VD2: tính cấp số cộng . . .

**int cong(int n)**

{ int kq ;

 if(n = 0) kq = 1;

 else kq = n + **cong(n-1)**;  return kq;

}

. . .

C++ | GVC. Võ Xuân Thể 

[2] Dùng tách tác vụ [một khối công việc] thành các tác vụ [một khối công việc] nhỏ hơn để chương trình tác vụ [một khối công việc] lớn đỡ phức tạp, dễ hiểu và dễ sửa chữa hơn.

VD3: Tinh tam giac

…

float a, b, c;

|  |
| --- |
| void nhap()  { cout<<"Nhap 3 canh tam giac:"<<endl;   cout<<"Nhap a = "; cin>>a;   cout<<"Nhap b = "; cin>>b;   cout<<"Nhap c = "; cin>>c;  } |

|  |
| --- |
| int kiemtra(float a, float b, float c)  { if(a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a) return 1;  else return 0;  } |

|  |
| --- |
| int tamgiaccan(float a, float b, float c)  { if(kiemtra(a,b,c))   if(a == b || a == c || b == c) return 1;   else return 0;   else return 0;  } |

3

C++ | GVC. Võ Xuân Thể

|  |
| --- |
| int tamgiacdeu(float a, float b, float c)  { if(kiemtra(a,b,c))   if(a == b && a == c) return 1;   else return 0;   else return 0;  } |

void main()

{ clrscr();

**nhap();**

**if(kiemtra(a,b,c))**

{ cout<<"Day la 3 canh tam giac"<<endl;

**if(tamgiacdeu(a,b,c)) cout<<"Tam giac deu "<<endl;**

**if(tamgiaccan(a,b,c)) cout<<"Tam giac can "<<endl;**

}

else cout<<"Khong phai 3 canh tam giac"<<endl;

getch();

}

Chú ý: Các chương trình con thường được tập trung thành các thư viện   =>cơ chế chia sẻ và tái sử dụng mã lệnh trong các kỹ thuật lập trình hiệu quả hiện nay.

VD4 Thư viện CV = Computer Vision = thư việc các thủ tục xử lý nhận diện qua hình  ảnh / âm thanh (có sẵn => lấy ra sử dụng, ko viết lại)

4

C++ | GVC. Võ Xuân Thể

**2. Các loại chương trình con**

\* Chương trình con có 2 loại: Thủ tục (Procedure) và hàm (Function): *[trong C/C++ gọi chung là “Hàm”: Procedure là Fuction đặc biệt trả về void]*

[1] Thủ tục (Procedure): Dùng để thực hiện một hay nhiều công việc/nhiệm vụ nào đó.  “Thủ tục” có vai trò như một lệnh / phát biểu (VD: if, switch, while, cin>>, cout<<, . . .)  Vì vậy, “thủ tục” Không dùng được trong biểu thức

 VD: void nhap() => thủ tục

 VD: x \* nhap() : SAI (không dùng trong biểu thức)

*Trong C/C++:*

“Thủ tục” = “Hàm” có kết quả trả về kiểu “bỏ qua” = void = không có kết quả trả về

[2] Hàm (Function): Thực hiện một hay nhiều công việc/nhiệm vụ nào đó và luôn có kết  quả trả về một giá trị cụ thể *(có kiểu không định trước = object, hoặc một kiểu cụ thể nào đó: VD int, float, . . .và thậm chí là kiểu void = như nêu trên)*.

 “Hàm” có vai trò như một hàm thư viện (VD: sqrt, toupper, clrscr, getch, scanf, printf,...)  Vì vậy, “hàm”: Có thể sử dụng trong các biểu thức.

 Không [nên:C.C++/ ko được:nn khác] dùng độc lập như câu lệnh / phát biểu  VD: int kiemtra(float a, float b, float c) => hàm

 long giaithua(int n) => hàm

 VD: kq = giaithua(n)/(giathua(k)\*giaithua(n-k)) : dùng được trong biểu thức  Không nên : giaithua(n); ==> trong các ngôn ngữ lập trình khác : LỖI

5

C++ | GVC. Võ Xuân Thể

**II. CÁCH XÂY DỰNG CHƯƠNG TRÌNH CON TRONG C/C++**

**1. Khai báo và định nghĩa “hàm” = chương trình con**

**1.1. Khái báo “hàm”** = đặc tả hàm (function specification = prototype): là khai báo tất  cả các yếu tố cần thiết về “hàm” đó để có thể “hiểu” và sử dụng trong các “hàm”/chương  trình (khác hoặc chính nó = đệ qui)

=> Các yếu tố về một “hàm” gồm:

+ tên hàm (function name): int **kiemtra**(float a, float b, float c) + kiểu dữ liệu kết quả trả về (return data type); **int** kiemtra(float a, float b, float c) + danh sách các tham/đối số của hàm đó (parameters) int kiemtra(**float a, float b, float c**)  VD: **int kiemtra(float a, float b, float c)**

=> Luôn phải khai báo “hàm” trước khi dùng

**1.2. Định nghĩa “hàm”** = thân hàm (function body): là tất cả các phát biểu / lệnh trong  “hàm” đó được lập trình để thực thi các tác vụ / khối công việc/nhiệm vụ của “hàm” đó.

 VD:

|  |
| --- |
| int kiemtra(float a, float b, float c)  **{ if(a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a) return 1;  else return 0;**  **}** |

=> Có thể định nghĩa “hàm” sau khi dùng/gọi hàm; nhưng phải khai báo “hàm” trước. **1.3. Việc Khai báo & định nghĩa “hàm” có thể**

**[1] thực hiện đồng thời (khai báo & định nghĩa) trước khi dùng**

DẠNG KHAI BÁO & ĐỊNH NGHĨA “HÀM”:

|  |
| --- |
| Kiểu\_dữ\_liệu\_kết\_quả Tên\_hàm(danh\_sách\_tham\_số) //phần khai báo “hàm”  { //Phần định nghĩa “hàm”   Danh\_sách\_các\_lệnh;   Có\_thể\_có\_lệnh\_return…(tùy Kiểu\_dữ\_liệu\_kết\_quả ≠ void);  } |

 Sau đó dùng “hàm”

 Chú ý: không có dấu ; sau //phần khai báo “hàm”

 VD: xem VD trên

6

C++ | GVC. Võ Xuân Thể

**[2] hoặc khai báo trước khi dùng (gọi là: khai báo Prototype) sau đó định nghĩa sau.** DẠNG KHAI BÁO (Prototype) để dùng, rồi ĐỊNH NGHĨA sau:

|  |
| --- |
| Kiểu\_dữ\_liệu\_kết\_quả Tên\_hàm(danh\_sách\_tham\_số); //khai báo Prototype |

 Sau đó dùng “hàm”

 Chú ý: Có dấu ; sau //khai báo Prototype

|  |
| --- |
| Kiểu\_dữ\_liệu\_kết\_quả Tên\_hàm(danh\_sách\_tham\_số) //rồi: khai báo & định nghĩa  { //Phần định nghĩa “hàm”   Danh\_sách\_các\_lệnh;   Có\_thể\_có\_lệnh\_return…(tùy Kiểu\_dữ\_liệu\_kết\_quả ≠ void);  } |

 Chú ý: không có dấu ; sau //phần khai báo “hàm”

 VD . . .

|  |
| --- |
| **long giaithua(long n);** // khai báo prototype (có ; ) |

void main()

{

 int n,k,kq;

 do{ cout<<"nhap n = "; cin>>n;

 cout<<"nhap k = "; cin>>k;

 }while(n<=0 || k<=0 || k>=n);

|  |
| --- |
| **kq = giaithua(n)/(giaithua(k)\*giaithua(n-k));** // dùng = gọi hàm |

 cout<<"To hop C cua “<<n<<” chap “<<k<<” phan tu = "<<kq<<endl;  getch();

}

|  |
| --- |
| **long giaithua(long n)** // đĩnh nghĩa sau (không ; )  **{ long kq = 1;**  **for(int i = 1; i<=n; i++) kq \*= i; // kq = kq\*i;**  **return kq;**  **}** |

7 

**1.4. CÁC GIẢI THÍCH**

C++ | GVC. Võ Xuân Thể 

**[1] Kiểu\_dữ\_liệu\_kết\_quả** : (=kiểu của “hàm”) : chính là kiểu dữ liệu của giá trị được  trả về trong câu lệnh ở cuối cùng nhất của thân “hàm” đó là lệnh: return

 Kiểu\_dữ\_liệu\_kết\_quả Tên\_hàm(danh\_sách\_tham\_số)

 { Danh\_sách\_các\_lệnh;

**return giá\_trị**; // giá\_trị thuộc kiểu Kiểu\_dữ\_liệu\_kết\_quả  }

 VD: xem VD trên

|  |
| --- |
| **long** giaithua(long n) // Kiểu\_dữ\_liệu\_kết\_quả là **long**  { long kq = 1;   for(int i = 1; i<=n; i++) kq \*= i; // kq = kq\*i;  **return kq;** // kq kiểu **long** đúng với Kiểu\_dữ\_liệu\_kết\_quả  } |

=> Có 2 trường hợp:

+ nếu Kiểu\_dữ\_liệu\_kết\_quả là void thì không có lệnh return ở cuối thân của “hàm”   *[Gọi là “thủ tục” = Procedure]*

void Tên\_hàm(danh\_sách\_tham\_số)

 { Danh\_sách\_các\_lệnh;

 }

 VD

|  |
| --- |
| **void** nhap() // Kiểu\_dữ\_liệu\_kết\_quả là void  { cout<<"Nhap 3 canh tam giac:"<<endl;   cout<<"Nhap a = "; cin>>a;   cout<<"Nhap b = "; cin>>b;   cout<<"Nhap c = "; cin>>c;  } // KHÔNG có lệnh return …. ở cuối thân “hàm” |

+ nếu Kiểu\_dữ\_liệu\_kết\_quả khác void thì bắt buộc có lệnh return ở cuối thân của “hàm”   *[Gọi là “hàm” = Function]*

VD

|  |
| --- |
| **long** giaithua(long n) // Kiểu\_dữ\_liệu\_kết\_quả ≠ void  { long kq = 1;   for(int i = 1; i<=n; i++) kq \*= i;  **return kq;** // bắt buộc có lệnh return …. ở cuối thân “hàm”  } |

8

C++ | GVC. Võ Xuân Thể

=> Nếu không khai báo (bỏ qua) : Kiểu\_dữ\_liệu\_kết\_quả tự hiểu là kiểu int (bắt buộc có  return <giá\_trị\_int>; cuối thân “hàm”

 VD

|  |
| --- |
| giaithua(long n) //Không khai báo Kiểu\_dữ\_liệu\_kết\_quả : tự hiểu là **int** giaithua(long n) { int kq = 1;   for(int i = 1; i<=n; i++) kq \*= i;  **return kq;** // bắt buộc có lệnh return …. ở cuối thân “hàm” thuộc kiểu **int** } |

=> trong nội dung (thân) của “hàm” có thể có nhiều return, nhưng chỉ có return cuối cùng  được chạy trả kết quả về và chỉ trả về 1 giá trị đúng kiểu với Kiểu\_dữ\_liệu\_kết\_quả  VD1

|  |
| --- |
| **long** tich(long n)  { long kq = 1;   for(int i = 1; i<=n; i++) { if(i < 10) kq \*= i; else **return;** // chỉ là return thoát for  **return kq;** // return kết quả tính toán của hàm  } |

 VD2

|  |
| --- |
| **int** diemthi()  { int diem, tinchi;   cout<<”Nhap diem thi: “; cin>>diem;   cout<<”Nhap so tin chi: “; cin>>tinchi;  **return diem;** // CHỈ ĐƯỢC return MỘT GIÁ TRỊ  **return tinchi;** // SAI [Muốn return 2 giá trị trở lên, dùng tham biến (bên dưới sẽ rõ)] } |

**[2] Tên\_hàm**: đặt tên gợi nhớ, ngắn gọn và theo qui tắc “danh hiệu”: không khoảng trống,  không bắt đầu số, không ký tự đặc biệt, không trùng từ khóa, . . .

VD: long **gia thua**(long n); SAI (Không dùng khoảng trống)

 long **giai\_thua**(long n); ĐÚNG

 long **giai-thua**(long n); SAI (không sử dụng ký tự đặc biệt: + - \* / \ ?….)  long **1tong**(int x) ; SAI (không bắt đầu bằng số)

 long **tong1**(int x); ĐÚNG

 int **if**(int x); SAI (trùng từ khóa if )

 int **neu**(int x); ĐÚNG

 . . .

9

C++ | GVC. Võ Xuân Thể

**[3] danh\_sách\_tham\_số** : mỗi tham số (đối số) gồm

 kiểu\_dữ\_liệu tên\_tham\_số

+ Khai báo các tham số tương tự khai báo biến và được sử dụng trong hàm như 1 biến

|  |
| --- |
| VD1: long giaithua(**long n**) // tham số n có vai trò như 1 biến  { long kq = 1;   for(int i = 1; i<=**n**; i++) kq \*= i; // n : được sử dụng như 1 biến (không cần khai báo lại)  return kq;  } |

|  |
| --- |
| VD2: long giaithua()  { **long n**, kq = 1; // n : là 1 biến được khai báo trong “hàm”  cout<<”Nhap n =”; cin>>**n**;   for(int i = 1; i<=**n**; i++) kq \*= i; // n : là 1 biến được khai báo trong “hàm”  return kq;  } |

+ Có thể có nhiều tham số, giữa các tham số cách nhau bởi dấu , (từ 2 tham số trở lên) *Số lượng tham số tối đa trong 1 “hàm” là tùy thuộc ngôn ngữ lập trình, thông thường <=254??* VD: int kiemtra(**float a**, **float b**, **float c**);

+ “hàm” có thể không có tham số, nhưng vẫn phải có dấu () kế sau Tên\_hàm  VD void **nhap()**;

 long **giai\_thua()**;

**[4] Danh\_sách\_các\_lệnh;** : luôn đặt trong {…..}

+ Các lệnh trong chương trình con (“hàm”) tương tự như trong main(): cũng có khai báo  biến, hằng, . . .: và trường hợp này gọi là “biến cục bộ” (phần sau sẽ rõ)

+ tùy vào Kiểu\_dữ\_liệu\_kết\_quả là void hay kiểu khác mà cuối cùng không hoặc có return…

10

C++ | GVC. Võ Xuân Thể

**1.5. CÁC CHÚ Ý**

[1] Các “hàm” được khai báo và định nghĩa không nhất thiết theo thứ tự (miễn là khai báo  trước khi dùng)

 VD: Tinh tam giac

…

float a, b, c;

…// khai báo cần theo thứ tự dùng

**int tamgiacdeu(float a, float b, float c);**

**int kiemtra(float a, float b, float c);**

**int tamgiaccan(float a, float b, float c);**

**void nhap();**

….//dùng

void main()

{ clrscr();

**nhap();**

**if(kiemtra(a,b,c))**

{ cout<<"Day la 3 canh tam giac"<<endl;

**if(tamgiacdeu(a,b,c)) cout<<"Tam giac deu "<<endl;**

**if(tamgiaccan(a,b,c)) cout<<"Tam giac can "<<endl;**

}

else cout<<"Khong phai 3 canh tam giac"<<endl;

getch();

}

….

11

C++ | GVC. Võ Xuân Thể

[2] Không được định nghĩa hàm này trong nội dung định nghĩa hàm khác [kể cả main() ] VD1:

**long giai\_thua(long n)**

{ long n, kq = 1;

**void nhap()** // SAI : không định nghĩa “hàm” nhap() trong “hàm” giai\_thua()  **{**

**. . .**

**}**

for(int i = 1; i<=n; i++) kq \*= i;

 return kq;

}

**SỬA LẠI:**

**long giai\_thua(long n)**

{ long n, kq = 1;

 for(int i = 1; i<=n; i++) kq \*= i;

 return kq;

}

**void nhap()** // định nghĩa “hàm” nhap() RA NGOÀI “hàm” giai\_thua() **{ . . .**

**}**

**VD2:**

. . .

**void main()**

{ long n, kq = 1;

**void nhap()** // SAI : không định nghĩa “hàm” nhap() trong “hàm” main()  **{**

**. . .**

**}**

for(int i = 1; i<=n; i++) kq \*= i;

}

Chú ý: một số ngôn ngữ lập trình khác cho phép định nghĩa “hàm” bên trong “hàm” khác

12

C++ | GVC. Võ Xuân Thể

[3] hàm main() cũng có vai trò và các vấn đề nêu trên như các hàm bình thương khác, tuy  nhiên đây là một hàm đặc biệt trong một chương trình C/C++, bởi:

+ Luôn luôn ít nhất là có nó (có hàm main)

+ Luôn được chạy đầu tiên (khi nhấn F9 / Ctrl\_F9); từ main() sẽ gọi các “hàm” khác.  Vì vậy,

-> với một “hàm” đã khai báo & định nghĩa nhưng không liên quan main() thì “hàm” đó  không có tác dụng

-> gọi dùng trực tiếp / gián tiếp trong main() đều được: main gọi hàm A, hàm A gọi hàm  B => hàm B gián tiếp được dùng trong main() OK;

**2. Sử dụng = gọi : “hàm” = chương trình con**

[1] “Hàm” chỉ chạy / thực thi khi được “gọi” sử dụng trong “hàm” khác; xuất phát từ hàm main

[2] Và khi đó các giá trị tham số thực tế được thế chỗ tương ứng vào vị trí các tham số đã  khai báo khi định nghĩa “hàm” (gọi là “truyền tham số”):

+ Các tham số khi khai báo và định nghĩa hàm : gọi là “tham số hình thức” + Các giá trị thực tế thế chỗ tương ứng vào vị trí các tham số hình thức: gọi là “tham số thực”  VD: ….

|  |
| --- |
| **long giaithua(long n)** // n : “tham số hình thức” : hàm giaithua() chưa chay **{**  **…..**  **}** |

void main()

{ long n,k,kq;

 …..

|  |
| --- |
| **kq = giaithua(n)/(giaithua(k)\*giaithua(n-k));** //gọi dùng hàm giaithua(…) : chạy // n, k, n-k:tương ứng là các “tham số thực” thế chỗ vào “tham số hình thức” n nêu  trên |

…

}

=> Kiêu dữ liệu của “tham số thực” phải cùng kiểu dữ liệu đã khai báo cho “tham số hình thức”

=> Các giá trị “tham số thực” khi truyền thay thế “tham số hình thức” khi gọi hàm: phải  dúng thứ tự vị trí (và đúng kiểu dữ liệu: như nêu trên).

=> Với mỗi “tham số hình thức” khi “gọi” hàm có thể truyền các giá trị “tham số thực”  khác nhau, miễn là dúng kiểu dữ liệu.

13

C++ | GVC. Võ Xuân Thể

VD: **float tinh\_diem(float diem, int tin\_chi); // “tham số hình thức”** …

 void main()

{…

 int tc = 3;

 float d =5;

 ….

 // float kq = tinh\_diem(tc, d); // SAI THỨ TỰ

 …

**float kq = tinh\_diem(d, tc); // ĐÚNG: d và tc : “tham số thực”** …

 }

14

C++ | GVC. Võ Xuân Thể

**III. TRUYỀN THAM SỐ CHO CHƯƠNG TRÌNH CON**

Có 2 dạng truyền tham số trong các chương trình con (hàm): tức là khi thế chỗ các “tham số thực”  vào vị trí các “tham số hình thức” khi gọi “hàm”; có 2 dạng:

**1. Tham trị = By Value** (truyền tham số dạng tham trị)

- Mọi thay đổi giá trị “tham số hình thức” trong “hàm” KHÔNG LÀM THAY ĐỔI giá trị “tham số thực” tương ứng khi truyền tham số sử dụng (gọi) hàm.

- Mặc định khai báo “tham số hình thức” của các hàm là THAM TRỊ

 kiểu\_dữ\_liệu tên\_tham\_số

- VD: …

**void swap(int a, int b)** // hoán đổi nội dung 2 biến a và b -> “tham số hình thức”  **{ int t = a;**

**a = b;**

**b = t;**

**}**

. . .

 void main()

 { int x = 5, y =7;

**swap(x, y);** // vì kiểu dl kết quả là void, nên có thể dùng như 1 lệnh  // x và y “tham số thực”

 // VÌ By Value, nên mọi thay đổi của a, b trong “hàm” trên

 sẽ KHÔNG ảnh hương đến x, y

 cout<<”x = “<<x<<” : y = “<<y<<endl;

 getch();

 }

⇨ KẾT QUẢ: x = 5 : y = 7 (không thay đổi)

15

C++ | GVC. Võ Xuân Thể

**2 Tham biến / tham chiếu = By Reference** (truyền tham số dạng tham biến / chiếu)

- Là trường hợp: Mọi thay đổi giá trị “tham số hình thức” trong “hàm” LÀM THAY ĐỔI giá trị “tham số thực” tương ứng khi truyền tham số sử dụng (gọi) hàm.

- Khai báo thêm dấu **&** vào kế trước tên\_tham\_số (giữa: kiểu\_dữ\_liệu và tên\_tham\_số) trong “tham  số hình thức” của các hàm khi định nghĩa “hàm”

 kiểu\_dữ\_liệu **&**tên\_tham\_số

- Bản chất của việc này là: dùng địa chỉ các ô nhớ đã được cấp phát cho tham số khi sử dụng trong  các lệnh trong thân của “hàm” (thay vì là giá trị trong ô nhớ như trường hợp tham trị)

 VD: tham trị ; void swap(int a, int b) : thế các giá trị 5 và 7 vào để tính  đ/c.a[5] đ/c.b[7]

 tham chiếu ; void swap(int &a, int &b): thế các đ/c.a và đ/c.b vào để tính  đ/c.a[5] đ/c.b[7]

- Thường dùng “tham biến / chiếu” để trả các giá trị kết quả về khi gọi hàm (như return ở cuối thân  “hàm” theo Kiểu\_dữ\_liệu\_kết\_quả), để khắc phục trường hợp cần trả về nhiều kết quả một lúc (trong khi return chỉ trà về 1 kết quả).

- VD: hàm hoán vị

 …

**void swap(int &a, int &b)** // hoán đổi nội dung 2 biến a và b -> “tham số hình thức”  **{ int t = a;**

**a = b;**

**b = t;**

**}**

. . .

 void main()

 { int x = 5, y =7;

**swap(x, y);** // vì kiểu dl kết quả là void, nên có thể dùng như 1 lệnh  // x và y “tham số thực”

 // VÌ By Reference, nên mọi thay đổi của a, b trong “hàm” trên

 sẽ CÓ ảnh hương đến x, y

 cout<<”x = “<<x<<” : y = “<<y<<endl;

 getch();

 }

KẾT QUẢ: x = 7 : y = 5 (CÓ thay đổi)

16

C++ | GVC. Võ Xuân Thể

**IV. TẦM VỰC (scope)**

**[1] Có 3 vị trí khai báo các biến:**

+ trong 1 hàm (kể cả hàm main): biến cục bộ

VD void nhap()

 { **float a, b, c; // biến được khai báo bên trong “hàm” = biến “cục bộ”** cout<<”Nhap a = “; cin>>a;

 cout<<”Nhap b = “; cin>>b;

 cout<<”Nhap c = “; cin>>c;

 . . .

 }

+ trong định nghĩa danh sách các “tham số hình thức” của hàm: biến cục bộ VD long giaithua(**int n**) **// biến dạng “tham số hình thức” trong “hàm” = biến “cục bộ”** {

 . . .

 }

+ bên ngoài tất cả các hàm: biến toàn cục

VD: …

**int chon; // biến được khai báo bên ngoài tất cả các “hàm” = biến “toàn cục”** …

 long giaithua(int n)

 {

 …

 }

 . . .

 void main()

 {

 …

 }

17

C++ | GVC. Võ Xuân Thể

**[2] tương ứng có 2 trường hợp**

**2.1 Biến cục bộ (địa phương) : Local Variable**

- Được khai báo (định nghĩa) bên trong 1 hàm (kể cả “tham số hình thức”)

- Được khai báo bất kỳ vị trí hợp lý trong hàm: và chỉ các lệnh / phát biểu trong  phạm vi “khối lệnh” (hoặc cấp dưới của “khối lệnh”) có chứa khai báo “biến” đó  mới hiểu và sử dụng được biến đó *(các “lệnh” / “khối lệnh” khác bên ngoài  “không hiểu” và không truy xuất được)*.

- Biến cục bộ: chỉ tồn tại (có “chu kỳ sống”) trong “khối lệnh” khai báo nó. - VD

long giaithua(**long n**)

{ **long kq** = 1;

 for(int i = 1; i<=**n**; i++) **kq** \*= i;

 return **kq**;

}

 …

void main()

{

..

 kq = 5; //SAI, trong hàm main() không hiểu biến kq là cục bộ trong hàm giaithua(..) …

}

\* Biến n và kq : là các biến cục bộ

 => nên chỉ có thể “hiểu” và sử dụng bên trong hàm giaithua(…) \* Biến **int i** bên trong lệnh for: for(**int i** = 1; i<=n; i++) kq \*= i;

 Là biến cục bộ bên trong lệnh for

18

C++ | GVC. Võ Xuân Thể

**2.2. Biến toàn cục : Global Variable**

- Được khai báo (định nghĩa) bên NGOÀI tất cả các hàm (kể cả main)

- Có thể “hiểu” và truy xuất sử dụng được trong bất kỳ phát biểu / lệnh nào trong toàn  bộ chương trình kể từ khi nó được khai báo / định nghĩa (“hiểu” và dùng được trong  tất cả các “hàm” của chương trình, kể cả main)

Chú ý: khai báo / định nghĩa trước khi dùng.

- Biến cục bộ: tồn tại (“sống”) trong toàn thời gian thực thi chương trình. VD:

**int chon; // biến được khai báo bên ngoài tất cả các “hàm” = biến “toàn cục”** …

 long giaithua(int n)

 { …

**chon = 5;** // Đúng, vì chon là biến toàn cục

 ….

 }

 . . .

 void main()

 { …

**chon = 9;** // Đúng, vì chon là biến toàn cục

 ….

 }

**[3] Tấm vực (scope) / phạm vi hiệu lực của một biến / hàm:** là phạm vi các “lệnh” /  “khối lệnh” có thể “hiểu” và truy xuất sử dụng được “biến” đó / “hàm” đó = phạm vi tồn  tại / “sống” của “biến” / “hàm” đó.

**[4] Các chú ý:**

- Không được trùng tên biến / hàm cùng dạng (toàn cục / cục bộ): trong cùng 1 tầm vực

VD long giaithua(long n)

{ **long** kq = 1, **i** = 5;

 for(**int i** = 1; i<=n; i++) kq \*= i; //LỖI: 2 biến cục bộ trung tên i trong hàm giaithua() return kq;

}

19

C++ | GVC. Võ Xuân Thể

- Nếu biến / hàm cục bộ =trùng= tên biến / hàm toàn cục: biến / hàm **cục bộ được ưu tiên** dùng; nếu muốn dùng biến / hàm toàn cục thì dùng phép toán phạm vi :: kế trước tên biến /  hàm

VD

**int chon;** // biến được khai báo bên ngoài tất cả các “hàm” = **biến “toàn cục”** …

 void main()

 { **int chon;** // biến được khai báo bên trong “hàm” main() = **biến “cục bộ”** …

 cout>>”Nhap lua chon : “;

**cin>>chon;** // sử dụng biến chon “cục bộ”

 ….

**cin>>::chon;** // sử dụng biến chon “toàn cục” nhờ toán tử **::** kế trước (C++)  . . .

 }

- Có thể dùng #define để định nghĩa hàm đơn giản (không ;) [chỉ có trong C/C++] VD:

#define sum(x, y) (x+y)

#define ERR(s) cout<<s<<endl

20

C++ | GVC. Võ Xuân Thể

|  |
| --- |
| **Bài 6: Giới thiệu Hướng đối tượng trong lập trình** |

**I. GIỚI THIỆU LỚP ĐỐI TƯỢNG (CLASS)**

**1 Khái niệm Lớp các đối tượng (Class)**

[1] Kiểu lớp đối tượng (**class**) là là kiểu dữ liệu tự định nghĩa được mở rộng từ kiểu struct:  + nó biểu diễn các đối tượng trong thực tiễn liên quan đến chương trình;

+ trong mỗi lớp đối tượng gồm có 2 loại thành phần được lưu trữ & truy xuất thông qua một tên chung: - có một số (hữu hạn) các thành phần thông tin

 mà có thể thuộc các kiểu dữ liệu khác nhau & được phân biệt bởi tên của nó = tên thuộc tính

- có một số hữu hạn các hàm phương thức (hàm hành vi = behavior) mà mỗi hàm phương thức cho phép thựhiện một thao tác hành vi nào đó trên lớp đối tượng với các thành phần thông tin của chính nó.

Chú ý:

+ Các hàm phương thức (hàm hành vi) chỉ thực hiện các thao tác hành vi trên chính lớp đối tượng  đó nên còn gọi là **tính đóng gói (encapsulation)** trong lớp đối tượng đó và độc lập tương đối so  với các lớp đối tượng khác

+ Giữa các lớp đối tượng tương tác thông tin với nhau thông qua việc tuyền thông báo và nhận kết  quả (không can thiệp vào nhau)

[2] VD: Lớp đối tượng “sinh viên” [ký hiệu **tên lớp đối tượng là sv**]

 gồm có 2 loại thành phần:

 + **3 thành phần thông tin**:

 . Mã số sinh viên [ký hiệu **mssv**]: kiểu số nguyên **int**

. Họ tên sinh viên [ký hiệu **hoten**]: kiểu chuỗi ký tự tối đa 50 ký tự (mảng kiểu **char[50]**)  . Điểm thi [ký hiệu **diem**]: kiểu **float**

**+ 3 hàm hành vi** (hàm phương thức)

 void nhapthongtinsv();

 void xuatthongtinsv();

 int xetketquadiem();

21

C++ | GVC. Võ Xuân Thể

**2 Khai báo và định nghỉa**

|  |
| --- |
| Khai báo và định nghĩa kiểu class (mẫu)  **class** tên\_kiểu\_clas  **{ private:**  kiểu\_thành\_phần\_tt\_1 tên\_thành\_phần\_tt1;   . . .   kiểu\_thành\_phần\_tt\_n tên\_thành\_phần\_ttn;  **public:**  kieu\_dl\_ket\_qua\_1 tên\_hàm\_hành\_vi\_1(ds\_tham\_số\_1);   . . .   kieu\_dl\_ket\_qua\_m tên\_hàm\_hành\_vi\_m(ds\_tham\_số\_m);  **};**  Khai báo biến thuộc kiểu class (đã định nghĩa)  **class** tên\_kiểu\_class tên\_biến\_class ; HOẶC [C++] tên\_kiểu\_class tên\_biến\_class ; |

VD:

class **sv**

{ **private:**

int **mssv**;

 char **hoten**[50];

 float **diem**;

**public:**

void **nhapthongtinsv**();

 void **xuatthongtinsv**();

 int **xetketquadiem**();

 }; class có tên sv gồm 3 thành phần thông tin: mssv kiểu int, hoten kiểu char[50] và diem kiểu float  Và 3 hàm hành vi: void nhapthongtinsv(); void xuatthongtinsv(); int xetketquadiem();

class **sv s1**, **s2**; khai báo 2 biến s1 và s2 thuộc kiểu class có tên sv

HOẶC:

**sv s1**, **s2**; khai báo 2 biến s1 và s2 thuộc kiểu class có tên sv

22

C++ | GVC. Võ Xuân Thể

CŨNG CÓ THỂ KHAI BÁO BIẾN NGAY SAU KHU KHAI BÁO & ĐỊNH NGHĨA KIỂU class class **sv**

{ **private:**

int **mssv**;

 char **hoten**[50];

 float **diem**;

**public:**

void **nhapthongtinsv**();

 void **xuatthongtinsv**();

 int **xetketquadiem**();

 } **s1**, **s2**;

**CHÚ Ý:**

+ Lớp đối tượng (class) : VD “Sinh viên” = có vai trò như kiểu dữ liệu (khai báo & định nghĩa như trên)

+ Đối tượng thuộc lớp đối tượng (object): VD: **s1** = sinh viên mã số 0123 = có vai trò như một  biến thuộc kiểu lớp đối tượng “Sinh viên”

**2. Định nghĩa các hàm phương thức (hàm hành vi)**

|  |
| --- |
| kieu\_dl\_ket\_qua\_i tên\_kiểu\_clas**::**tên\_hàm\_hành\_vi\_i(ds\_tham\_số\_i)  { . . .  } |

**[1] với 1 lớp đối tượng (class) luôn có 3 hàm đặc biệt:**

1.1. Hàm thiết lập (Constructor) = trung tên lớp đối tượng (không bắt buộc định nghĩa) VD: sv::sv() { mssv = 0; hoten = “”; diem = 0; }

1.2. Hàm thiết lập sao chép (Contructor copy) = trung tên lớp đối tượng (không bắt buộc định nghĩa)  sv sv::sv() { sv t;

 t.mssv = mssv;

 t.hoten = hoten;

 t.diem = diem;

 return t;

 }

1.3. Hàm hủy bỏ (Destructor) = Giải phóng vùng nhớ cho biến đối tượng (không bắt buộc định nghĩa) VD: sv::~sv() . . .

23

C++ | GVC. Võ Xuân Thể

**[2] VD định nghĩa hàm hành vi**

void **sv::nhapthongtinsv**()

{ cout<<”Nhap mssv: “; cin>>mssv;

 cout<<”Nhap ho ten sv: “; cin>>hoten;

 cout<<”Nhap diem: “; cin>>diem;

}

void **sv::xuatthongtinsv**()

{ cout<<”Mã sinh: “<<mssv<<” – Ho ten : “<<hoten<<” – diem :”<<diem<<endl; }

int **sv::xetketquadiem**()

{

 return diem>=5;

}

**3. Truy xuất sử dụng**

**[1]** Truy xuất sử dụng từng thành phần thông tin trong class qua biến chung và tên của thành phần

|  |
| --- |
| tên\_biến\_class**.**tên\_thành\_phần\_tt\_i (dấu chấm phân cách) Và chỉ thành phần thuộc Public: HOẶC:  tên\_biến\_class**.**tên\_hàm\_hành\_vi\_i(ds\_tham\_số\_i) |

\* Lúc này mỗi thành phần thông tin / hàm hành vi được truy xuất

 có vai trò như một biến độc lập thuộc kiểu kiểu\_dl\_thành\_phần\_i

 hàm độc lập trà về kieu\_dl\_ket\_qua\_i

 VD

…..

**class sv s1;** //biến s1 thuộc kiểu class như định nghĩa ở trên

….

….

**sv.nhapthongtinsv();**

if(**sv.xetketquadiem**()) cout<<”Có điểm đạt yêu cầu “<<endl;

…

**[2]** Thường rất ít khi truy xuất sử dụng biến struct: tên\_biến\_class

24

C++ | GVC. Võ Xuân Thể

**II. GIỚI THIỆU LẬP TRÌNH LỚP ĐỐI TƯỢNG (Object-Oriented Programming)**

**1. Khái niệm:** Lập trình hướng đối tượng (Object-Oriented Programming: OOP) là phương pháp lập trình lấy đối tượng làm nền tảng xuất phát để xây dựng chương trình

**2. Tính Thừa kế (Inheritance)**

[1] Dùng trong trường hợp có nhiều lớp đối tượng là chí tiết (hoặc cụ thể hóa) của một lớp đối  tượng khác thì ta khai báo: Các lớp đối tượng chi tiết (cụ thể hóa) là lớp đối tượng con (child class = còn gọi là: lớp đối tượng dẫn xuất = derived class) của lớp đối tượng tổng quát, gọi là lớp đối  tượng cha/mẹ (parent class = còn gọi là lớp đối tượng cơ sở = base class).

VD: SV lớp đối tượng cha/mẹ => 2 lớp đối tượng con : SVCQ, SVTC thừa kế lớp đối tượng SV

[2] Lớp đối tượng con : thừa hưởng toàn bộ các thành phần thông tin & hàm hành vi của lớp tượng  cha/mẹ đồng thời có thể bổ sung hoặc cải tiến (chồng hàm) các thuộc tính hoặc hàm hành vi của  lớp đối tượng cha / mẹ đã có.

**[3] Khai báo và định nghĩa:**

|  |
| --- |
| **class** tên\_kiểu\_class: **public** tên\_kiếu\_class\_cha\_1, **public** tên\_kiếu\_class\_cha\_2, . .   **{ privatec:**  kiểu\_thành\_phần\_tt\_1 tên\_thành\_phần\_tt1;   . . .   kiểu\_thành\_phần\_tt\_n tên\_thành\_phần\_ttn;  **public:**  kieu\_dl\_ket\_qua\_1 tên\_hàm\_hành\_vi\_1(ds\_tham\_số\_1);   . . .   kieu\_dl\_ket\_qua\_m tên\_hàm\_hành\_vi\_m(ds\_tham\_số\_m);  **};** |

**2. Chồng hàm (Overload function)**

[1] là phương pháp định nghĩa một hàm / phép toán (cùng tên hàm/phép toán) theo nhiều  cách khác nhau trong cùng một phạm vi hiệu lực (scope). Trình biên dịch sẽ lựa chọn hàm  /toán tử nào thích hợp để thực hiện dựa trên các tham số mà nó được gọi.

[2] Nhận diện hàm / toán tử thích hợp dựa trên:

+ số lượng tham số

+ kiểu dữ liệu của các tham số.

25

C++ | GVC. Võ Xuân Thể

[3] VD1

float tinh\_diem(float diem) {return diem; } // hàm 1

float tinh\_diem(float diem, int sotinchi) {return diem\*sotinchi; } // hàm 2 float tinh\_diem(float diem, int sotinchi, float diemthuong)   {return (diem\*sotinchi + diemthuong); } // hàm 3 ….GỌI HÀM….

float d = tinh\_diem(7.5); // gọi hàm 1

float d = tinh\_diem(7.5, 2 ); // gọi hàm 2

float d = tinh\_diem(7.5, 2 , 0.5); // gọi hàm 3

…

VD2

int mod10(int so) {return so % 10; } // hàm 1

int mod10(float so) {return floor(so)%10; } // hàm 2

….GỌI HÀM….

int d = mod10(18); // gọi hàm 1

int d = mod10(18.5); // gọi hàm 2

. . .

**3 Chương trình mẫu**

26