|  |
| --- |
| **BÀI 1: TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG** |

***I. Giới thiệu về lập trình hđt***

***1. Các phương pháp lập trình truyền thống***

**a) Lập trình tuyến tính**

Toàn bộ chương trình chỉ là một đơn thể duy nhất, các lệnh được thực hiện tuần tự theo thứ tự xuất hiện  trong chương trình. Trong ngôn ngữ C, lập trình theo kiểu tuyến tính sẽ chỉ có một hàm main.

**Ví dụ** : viết ct nhập họ tên sv, đlt, đth và tính đtb của sv.

#include <stdio.h>

int main()

{

char ht[30];

float dlt, dth, tb;

printf("\nNhap ho ten:"); gets(ht);

printf("\nNhap diem LT:"); scanf("%f",&dlt);

printf("\nNhap diem TH:"); scanf("%f",&dth);

tb = (dlt + dth)/2;

printf("\nHo ten : %s", ht);

printf("\n Diem TB = %.2f",tb);

}

**\* Nhận xét:**

- Ưu điểm: đơn giản

- Khuyết điểm: khó sửa lỗi, khó mở rộng

**b) Lập trình hướng thủ tục**

Là lập trình dựa vào các thủ tục (hàm). Mỗi hàm sẽ thực hiện một chức năng của chương trình.  Khi chương trình thực thi thì hàm main sẽ được thực hiện đầu tiên, hàm main sẽ gọi các hàm khác  khi cần và các hàm khác có thể gọi lẫn nhau.

**Ví dụ:**

int x; //x la du lieu co the truy xuat boi bat cu ham nao

void B()

{

int x;

}

void A(int x)

{ …

B();

}

int main()

{ int x=1;

A(x);

…

}

2 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**Ví dụ** : viết lại ct tính đtb bằng cách tách ct thành 3 hàm: hàm nhập, hàm tính dtb, hàm xuất. #include <stdio.h>

void nhap(char ht[], float &dlt,float &dth)

{

printf("\nNhap Ho ten:"); gets(ht);

printf("\nNhap diem LT: "); scanf("%f",&dlt);

printf("\nNhap diem TH: "); scanf("%f",&dth);

}

float tinh(float dlt, float dth)

{

return (dlt+dth)/2;

}

void xuat(char ht[], float dtb)

{

printf("\nHo ten: %s", ht);

printf("\n Diem TB = %.2f", dtb);

}

int main()

{

char ht[30];

float dlt, dth, dtb;

nhap(ht, dlt, dth);

dtb = tinh(dlt, dth);

xuat(ht,dtb);

}

**\* Nhận xét:**

**-** Ưu điểm: dễ sửa lỗi, dễ mở rộng, phù hợp khi viết chương trình nhỏ

**-** Khuyết điểm: vì dữ liệu và hàm tách biệt nên có các khuyết điểm sau:

+ *Khó bảo vệ dữ liệu và hàm để không bị truy xuất bởi các hàm không mong đợi*, khi sửa đổi dữ liệu các  hàm truy xuất phải thay đổi theo (do không có tính đóng gói).

+ Khó sử dụng lại các hàm đã viết sẵn (do không có tính thừa kế)

+ Không phù hợp với suy nghĩ của con người (do không có tính trừu tượng)

Để khắc phục các khuyết điểm của lập trình tuyến tính cũng như lập trình hướng thủ tục, người  ta đưa ra một phương pháp lập trình mới là lập trình hướng đối tượng.

**2. Lập trình hướng đối tượng (Object Oriented Programming)**

Là lập trình dựa vào các đối tượng (object), đối tượng được tạo ra từ lớp, lớp gồm có dữ liệu và phương thức  (hàm) xử lý dữ liệu của lớp.

private : Attributes = Properties = Data || Operation = Behavior function = Method : public 

https://lh6.googleusercontent.com/PEHl91-JBM07JIaL1kih88YrHgs5dR4f_y9pKnQoa7gHjBilgAap-JTnAQKwSq3yIH7oRFZH7M98bFFTs-41uVplaii5iHVol_C8CaouyARIBr0QQ0e3xj0updCeKKkkNan_K6TW=s0Object A https://lh4.googleusercontent.com/2U6k4beSkpXKkOBmhV6KypvNCqYaUPoBDpUx2ol5S9BOYE8hS6J2iIEAH8gpo-Sz8qQe0awKS46egAz4cXeeeA9jtj8P6RTx4BKqP-dYr1ys7ArabmeG7sP8Um7KAS1hwtPZzghz=s0DATAS

METHOD S https://lh3.googleusercontent.com/KhJDN-zFL6bYkeiV6mOZ-WUrdGdCFkY5G4aerTJLgV1RwtnDp8iaWgL4DMsIfsirU-r6gHuHWcUDLLf6rssQxHIfTmTBN_yRgwnQYlLePPBuwGx9KN2pRxfNNGzkhgHGSJYUk_Aq=s0

https://lh4.googleusercontent.com/qYWsDODb0z6lnuwftKYa4IzkSZa7znO1o9lWlyJpeo0n4N3pPN0ziI_J1iNYpWP7Q_HjNCkM9L5P7rFUPm0dgRGf1a4ulsVMJMVZpfpfJplipvGhk0BZzRH5uwrMypuWV0mMcdNX=s0Object B

https://lh5.googleusercontent.com/i95qGAaeqj8FwnHIyOkA75CSFv79VK_v2KeNHySCz-QpHUl5i6zUHXY785MGDUr_5uKNwrATXlFFxIzmrD7nLDYVZBmSMQGxnZs0VwMctF5-Rhyn8zr1OilVMrCiFRAVzQbqqZVI=s0DATAS

METHOD https://lh5.googleusercontent.com/48GajeYXz50O5xFUmTM_rDRQOueSYVogv-hnHV5Yjcz8ObkDuiMOQ842tYaVCaUMdjZ3QWeo8BMkomtWIfSd_g7sN_vb30AEtVK92C-ruzzNHOTis_HaIw63zTq7tq3nh_n7BjAR=s0

S

https://lh3.googleusercontent.com/4NpOXtYhpcvw546x8kdbYhNdvSXFX_foPwLtD_wSiqcXBWRT6UOnmuRcJQtN-79oaSxEUaUdpBO6Q_7_x8muE1KYrYlHxlpdBJ37segPOBWF5Rl7lc6UsgvyezewosCVBYzCamsK=s0MÔ HÌNH CỦA LTHĐT

3 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**Ví dụ**: viết lại ct tính đtb bằng cách thiết kế lớp sinh viên có các thuộc tính là hoten, dlt, dth và các phương  thức là nhập, tính dtb, xuất.

-------------------

#include <iostream>

using namespace std;

class SV

{private:

char ht[30];

float dlt, dth;

public:

void nhap()

{

cout<<"\nNhap Ho ten: "; fflush(stdin); cin.getline(ht,30);

cout<<"\nNhap diem LT: "; cin>>dlt;

cout<<"\nNhap diem TH: "; cin>>dth;

}

float tinh()

{

return (dlt + dth)/2;

}

void xuat()

{

cout<<"\nHo ten SV: "<<ht<<endl;

cout<<"\n Diem TB = "<<tinh()<<endl;

}

};

int main()

{//Tr.h 1: khai báo và su dung Lop doi tuong theo thong thuong

 cout<<"TRUONG HOP SU DUNG BIEN OBJECT"<<endl;

SV s; // SV = class s = object

s.nhap(); s.xuat();

//Tr.h 2: khai báo và su dung CON TRO Lop doi tuong

 cout<<"TRUONG HOP SU DUNG CON TRO DEN OBJECT"<<endl;

SV \*t ; // SV = class t = con tro den object

t = new SV;

t->nhap(); t->xuat();

}

**\* Nhận xét:** LTHĐT có 4 đặc tính sau:

- **Tính trừu tượng (Abstraction):** đối tượng trong LTHĐT là sự trừu tượng của các đối tượng trong tự  nhiên. Tính trừu tượng giúp việc lập trình trở nên tự nhiên hơn, gần với suy nghĩ của con người hơn. - **Tính đóng gói (Encapsulation):** Việc tổ chức dữ liệu và hàm trong một lớp gọi là tính đóng gói, tính

đóng gói cho phép che dấu dữ liệu và phương thức trong lớp, bảo vệ dữ liệu không bị truy xuất bởi những  hàm không hợp lệ.

- ***Tính thừa kế (Inheritance):*** *Sử dụng lớp có trước (lớp cha) để xây dựng lớp mới (lớp con) gọi là tính  thừa kế. Lớp con được thừa hưởng những thuộc tính, phương thức của lớp cha và có thể có thêm những  thuộc tính, phương thức riêng. Tính thừa kế giúp chương trình dễ mở rộng.*

- **Tính đa hình (Polymorphism):** Một phương thức có thể thực hiện theo nhiều cách khác nhau trên các  lớp khác nhau gọi là tính đa hình. Tính đa hình giúp cho việc viết chương trình trở nên đơn giản hơn.

Ngoài ra, khi LTHĐT không còn phải tìm cách chia chương trình thành các hàm mà chỉ cần xem xét  chương trình cần sử dụng những đối tượng nào, mỗi đối tượng cần có những thuộc tính (dữ liệu, biến) và  phương thức (hàm, thủ tục) nào, từ đó xây dựng các lớp tương ứng.

4 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**Ví dụ 1:**

Viết chương trình quản lý sinh viên gồm có các chức năng sau: quản lý hồ sơ sinh viên, quản lý lớp  mà sinh viên đang học.

Phân tích: Cần hai đối tượng sau

- Đối tượng sinh viên:

Thuộc tính: mã sv, họ tên sv, năm sinh, mã lớp mà sv đang học

Phương thức: Nhập sv, tìm sv, xem, xoá, sửa sv, xem ds sv

- Đối tượng lớp:

Thuộc tính: mã lớp, tên lớp, gvcn, sỉ số, …

Phương thức: Tạo lớp mới, xem thông tin về lớp, xem ds lop

=========================================

**YÊU CẦU 2 VIỆC: SÁNG 24/6/2020**

1. LÀM LẠI BÀI TẬP MẪU TRÊN : nhập điểm LT, TH của SV => Diem TB

Data trong Class : <tensv> -> ht

 <ho> , <lot> -> dlt, dth

 <tensv> -> s

 <hosv> -> t

2. LÀM BÀI VÍ DỤ 1 NỂU TRÊN THEO OOP:

. Nhập lớp => xuất ra lớp đó [dùng object và con trỏ object]

, Nhập SV => xuất ra sv đó [dùng object và con trỏ object]

NỘP LÊN CLASSROOM & MAIL THEO HƯỚNG DẪN TRONG FILE “..INTRODUCTION..” ============================

5 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

II. Những khái niệm cơ bản

**1. Đối tượng (object):**

Đối tượng dùng để biểu diễn một thực thể của thế giới thực. Mỗi đối tượng được xác định bởi  thuộc tính (dữ liệu, biến) và hành vi (phương thức). Thuộc tính để xác định tính chất riêng của  đối tượng, hành vi là hành động tác động lên đối tượng.

**ví dụ** : Đối tượng sinh viên

|  |  |
| --- | --- |
| \* Đối tượng sinh viên thứ 1 | \* Đối tượng sinh viên thứ 2 |
| - Thuộc tính:  họ tên: Vương Ngọc Yến  đlt= 1  đth= 2  - Hành vi:  tính đtb của sv: dtb=(dlt+dth)/2=1.5 | - Thuộc tính:  họ tên: Đoàn Dự  đlt= 2  đth= 1  - Hành vi:  tính đtb của sv: dtb=(dlt+dth)/2=1.5 |

**Ví dụ**: Đối tượng hcn

|  |  |
| --- | --- |
| Đối tượng hcn thứ 1 | Đối tượng hcn thứ 2 |
| - Thuộc tính:  chiều dài=4  chiều rộng=3  - Hành vi:  Tính dt: dt=cd\*cr=12 | - Thuộc tính:  chiều dài=6  chiều rộng=5  - Hành vi:  Tính dt: dt=cd\*cr=30 |

6 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**2. Lớp (class)**

Là khái niệm dùng để mô tả các đối tượng có cùng thuộc tính và hành vi. Mỗi lớp sẽ khai báo các thuộc tính,  hành vi của các đối tượng thuộc lớp. Các đối tượng thuộc cùng một lớp sẽ có cùng tên các thuộc tính nhưng  có các giá trị thuộc tính khác nhau. Thuộc tính còn gọi là dữ liệu hay là biến, hành vi còn gọi là hàm hay  phương thức. Ta có thể xem class là kiểu dữ liệu

**ví dụ :**

Đối tượng hcn thứ 1 và đối tượng hcn thứ 2 có cùng tên các thuộc tính đó là chiều dài và chiều rộng. Nhưng  giá trị chiều dài và chiều rộng của đối tượng hcn thứ 1 là 3 và 4, trong khi đó giá trị chiều dài và chiều rộng  của đối tượng hcn thứ 2 là 5 và 6.

Lớp hcn dùng để mô tả tất cả các đối tượng hcn, và lớp hcn có thể khai báo như sau:

#include <iostream>

using namespace std;

class hcn

{

private: //các thuộc tính của đối tượng hcn (còn gọi là biến, dữ liệu)

float cd,cr; // **mặc định** thuộc tính truy cập là **private**

//các phương thức của đối tượng hcn (còn gọi là hàm, thủ tục)

 public:

void nhap();

float tinhdt();

void xuat();

};

void hcn::nhap() //phuong thuc nhap , :: là toán tử phân định phạm vi

{

cout<<"Nhap cd, cr:"; cin>>cd>>cr;

}

float hcn::tinhdt()

{

return (cd\*cr);

}

void hcn::xuat()

{

cout<<"\nCD:"<<cd<<"\nCR:"<<cr<<"\nDT:"<< tinhdt()**;**

}

//hàm main để thử sử dụng lớp hcn

int main()

{

**hcn h; //khai báo và tao dt hcn**

**h.nhap(); h.xuat();**

}

7 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**Ghi chú:**

- Tất cả các đối tượng hcn sẽ dùng chung các phương thức nhap(), xuat(), tinhdt(). Nhưng mỗi đối tượng  hcn sẽ có biến cd,cr riêng để có thể chứa các giá trị khác nhau.

cd,cr nhap();xuat();

tinhdt(); cd,cr

h1h2

Mô hình lớp hcn

**3. So sánh struct và class**

Trong struct chỉ có dữ liệu, trong class có dữ liệu và phương thức xử lý dữ liệu và trong struct tất cả dữ  liệu mặc định là public (do đó tất cả các hàm đều có thể truy xuất), trong lớp mặc định là private (chỉ có hàm  trong lớp được truy xuất)

**Ví dụ:**

struct hcn

{

float cd,cr;

};

void nhap(hcn h) //hàm nhap

{

cout<<"Nhap cd, cr:"; cin>>h.cd>>h.cr;

}

float tinhdt(hcn h)

{

return (h.cd\*h.cr);

}

void xuat(float dt)

{

cout<<"\nCD:"<<cd<<"\nCR:"<<cr<<"\nDT:"<<dt;

}

void main()

{

hcn h; float dt;

nhap(h); dt=h.tinhdt(); xuat(dt);

}

**4. Phép toán phân giải phạm vi :: (scope resolution operator):**

Khi lớp có nhiều phương thức ta chỉ nên khai báo tên phương thức trong lớp, định nghĩa phương thức ghi ở  ngoài lớp và dùng phép toán phân giải phạm vi để xác định phương thức thuộc lớp nào.  **ví dụ**: khai báo

void hcn::nhap() //nghĩa là phương thức nhập thuộc lớp hcn

**5. Từ khoá public, private, protected : (thuộc tính truy cập)**

Được đặt ở trứơc các thành phần của lớp (dữ liệu hoặc phương thức), nếu không có thì mặc định là private. + private: thành phần chỉ được sử dụng bên trong lớp, bên ngoài lớp không thể truy xuất.  + public: có thể truy xuất bên ngoài lớp.

+ protected : thành phần có thuộc tính này sẽ được truy cập ở bên trong lớp, và trong lớp dẫn xuất trực tiếp.

8 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**ví dụ:** lớp A có một thành phần private và một thành phần là public. Đối với thành phần private thì chỉ có  những thành phần trong cùng lớp A mới được truy xuất. Đối với thành phần public thì có thể được truy xuất  bởi các phương thức của lớp B hoặc hàm main, hoặc bất kỳ pt trong lớp nào khác (với điều kiện là ta phải  khai báo được đối tượng thuộc lớp A ở bên trong lớp B hoặc trong hàm ngoài).

https://lh6.googleusercontent.com/lGymW4fAvv8mKbqvPynyBiV3Nl41bsB19PniCq2itbugc-KUGj-u9UfuQNqTH9-zuxSs8BKrUkcNVkBIigkK61BsQlhpKM0ITzPWU0OFONpaYS4Z8H0AP1LhIwzMhUP1QRkXkNYK=s0private

https://lh4.googleusercontent.com/lAbzcrxnjHv-rpWkXQFmVaf0N7RFb1niGC3fHuhWWEqF8LCrUeIZDX8ZpkpExXh6Oqr884sclqqXGXx8bHNMtBUGZUsxL3SSjV0cOYRsNAJcNQegjpejmtFf2v428BAeE46CZJLM=s0public 

class B class A **ví dụ:**https://lh5.googleusercontent.com/GDh8zPKmilEjxtivO7KE1HIiuVZWwHvdM6GKxlbazITGu8ZFw2_UmNIDPAGIoeacWjfGEtp3ozR6JuQ72xJ81My7sIiK12fY5Z1qC4uUGATm7HkPf39kaxhSNito4Z1kQf-EQETb=s0https://lh3.googleusercontent.com/8Js-NUNoq_K4YODHTXYKnj9yf1LL1lWLw43Hb99hoHWq8hwjbh7_yAV-flsbIt9iS1HlBylMi8K4IOrRmBBkmAyiIQMc7OwjLg_URiWRG7m8STVxnimZmME5h6FY1JGnh--3s5uI=s0

**https://lh5.googleusercontent.com/TAXf2vdx-w_RkXZulggBZ1ycG2Lmr---w5NEojV1TUvN6FrYJO3X8jzZOPQjHG3hu_G3gvIcC4uSQAjso1VQ-3rM5VT4LlUvNoPfG6HUnliUx3OW7nI7XuE2LQ1631oNqOiY9rjB=s0**main

#include <iostream>

class A

{

*//mac dinh la private*

int x;

void g()

{

x++;

}

public:

int f()

{

 x=1;

g();

return x;

}

};

**3. Con trỏ this (con trỏ đối tượng):**

class B

{ void h()

{

A a;

*a.x++;*

}

};

int main()

{

A a;

*a.g(); a.x++;*

cout<<a.f();

B b ; *b.h() ;*

} 

Là con trỏ chứa địa chỉ của đối tượng hiện hành (đối tượng đang truy xuất phương thức). Thông thường các  phương thức trong lớp khi truy xuất các thành phần của đối tượng hiện tại thì có thể bỏ this nếu không gây  ra nhầm lẫn.

**ví dụ:** tránh nhầm lần x là thuộc tính của lớp C <=> x là tham số của hàm thiết lập C(int x)

|  |  |
| --- | --- |
| class C  {  int x;  public:  C (int x) *//phương thức constructor*  {  **this**->x=x; // ;  }  void xuat()  {  cout<<x;  }  }; | int main()  {  C c1(1), c2(2);  c1.xuat();  c2.xuat();  }  Nhận xét:  c1.xuat(); thì this là địa chỉ của đối tượng c1,  do đó kết quả xuất là 1.  c2.xuat(); thì this là địa chỉ của đối tượng c2,  do đó kết quả xuất là 2. |

9 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**ví dụ** KHÔNG nhầm lần x là thuộc tính của lớp C <=> k là tham số của hàm thiết lập C(int k) KHÔNG dùng **this**

|  |  |
| --- | --- |
| class C  {  int x;  public:  C(int k) {x=k;} *//không cần this->*  void xuat()  {  int x=5; *// x cục bộ của phương thức xuat*  this->x += x; *// trg hợp này this*  *// không bỏ được*  *// [this->x = this->x + x;]* cout<< this->x <<endl<<x;  }  }; | int main()  {  C c1(1);  c1.xuat(); //xuat ra ????  { |

**4. Hàm, phương thức có tham số mặc định**

Có thể gọi hàm, pt mà không cần gởi đủ tham số. Khi đó khai báo hàm phải cung cấp những giá trị mặc định  cho những tham số có thể không được gởi này. Những tham số bắt buộc phải có thì phải khai báo ở đầu danh  sách tham số. Khi một tham số nào đó được gán trị mặc định thì tất cả các tham số theo sau tham số này  cũng phải gán gía trị mặc định. Những giá trị mặc định có thể ghi trong phần khai báo phương thức hoặc ghi  ở phần định nghĩa nhưng không được ghi ở cả hai (nên ghi ở phần khai báo phương thức) .

**ví dụ**

#include <iostream.h>

class calculate

{

public:

int sum(int m=1, int n=10);

};

int calculate::sum(int *m~~=1~~*, int *n~~=10~~*)

{

int s=0;

for (int i=m;i<=n;i++) s+=i;

 return s;

}

void main()

{

calculate c;

 cout<<c.sum(9); //m=9, không có n nên sum dùng giá trị n mặc định là n=10 }

**5. Định nghĩa chồng phương thức (method overloading)**

Trong cùng lớp có thể định nghĩa nhiều phương thức cùng tên nhưng khác số lượng tham số hoặc khác kiểu  của các tham số, và gọi là định nghĩa chồng phương thức.

Định nghĩa chồng phương thức không phân biệt kiểu trả về, do đó không thể định nghĩa hai phương thức  cùng tên và *chỉ khác nhau ở kiểu trả về.*

Ta sẽ định nghĩa các phương thức cùng tên khi cách thực hiện của các phương thức là giống nhau, chỉ khác  nhau ở số tham số hoặc kiểu của các tham số. Việc định nghĩa các phương thức cùng tên giúp việc lập trình  đơn giản hơn, dễ hiểu hơn.

10 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**Ví dụ:**

Để tính bình phương của một số, đối với C phải định nghĩa hai phương thức có tên khác nhau sau:

int SqrInt(int x) //tính bình phương số nguyên int

{

return x\*x;

}

float SqrFloat(float x) //tính bình phương số thực float

{

return x\*x;

}

C++ cho phép ta định nghĩa hai phương thức có cùng tên như sau:

int Sqr(int x,) //tính bình phương số nguyên int

{

return x\*x;

}

float Sqr(float x) //tính bình phương số thực float

{

return x\*x;

}

void main()

{

int a=2; float b=3;

cout<<”\na\*a=”<<Sqr(a); //C++ sẽ gọi hàm int Sqr(int x)

cout<<” \nb\*b= “<<Sqr(b);//C++ sẽ gọi hàm float Sqr(float x)

}

**Ví dụ:**

int f(int x)

{

//các lệnh

}

float f(int y)

{

//các lệnh

}

sẽ báo lỗi vì C++ không cho phép định nghĩa hai phương thức *chỉ khác nhau kiểu trả về*

**6. Từ khoá const**

Dùng để khai báo dữ liệu hằng. Hằng phải được gán trị ban đầu và không thể thay đổi giá trị. Mục đích của  việc sử dụng hằng là tránh việc vô ý thay đổi giá trị hằng.

**ví dụ:**

const int n=10; *//hằng n phải được gán trị ban đầu*

void main()

{

n++; *//sai vì hằng không được thay đổi giá trị*

}

11 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**ví dụ:**

int m=1, n=2;

const int \*p = &n; *//p là con trỏ trỏ tới một hằng nguyên (nội dung của \*p là hằng, p không phải là hằng)* \*p=3; *//sai, vì p trỏ (\*p) tới hằng nguyên*

p=&m; *//đúng, vì p không phải là hằng*

int \* const q = &n; *//q là con trỏ : hằng, trỏ đến vùng nhớ của hằng n (nội dung vùng nhớ \*q không phải là hằng)* \*q=3; *//đúng, n=3*

q=&m; *//sai, vì q la hằng, không mang đi trỏ chỗ khác*

**7. Cấp phát bộ nhớ động**

Mô hình bộ nhớ: 4 phần

|  |  |  |  |
| --- | --- | --- | --- |
| CODE | DATA | STACK | HEAP |

|  |  |
| --- | --- |
| Cấp phát tĩnh | Cấp phát động |
| - Cấp phát trong vùng Data của ct  DATA<64K  - Cấp phát lúc biên dịch  - Không thể tự cấp phát thêm hoặc thu hồi. | - Cấp phát trong vùng Heap của ct  HEAP=*KTCT*-CODE-DATA-STACK - Cấp phát lúc thực thi  - Có thể tự cấp phát thêm hoặc thu hồi. |

**Ví dụ:**

- Cấp phát bộ nhớ động cho **biến** kiểu float

|  |  |  |
| --- | --- | --- |
|  | Trong C thường | Trong C++ |
| Cấp phát | float \*a= (float \*) malloc(sizeof(float));  //a kiểu con trỏ float, chứa đc của ô nhớ kiểu float | float \*a=new float; |
| Thu hồi | free(a); | delete a; |
| Sử dụng | \*a=5; //cất 5 vào biến a | |
| Giả sử ô nhớ  float được  cấp phát ở  địa chỉ bắt  đầu là 100 | 100  5  ô nhớ 4 bytes để chứa số  100  thực kiểu float  a (2 bytes) | |

- Cấp phát **mảng** một chiều động 4 phần tử kiểu int.

|  |  |  |
| --- | --- | --- |
|  | Trong C thường | Trong C++ |
| Cấp phát | int \*a= (int \*) malloc(4\*sizeof(int)); | int \*a=new int [4]; |
| Thu hồi | free(a); | delete []a; |
| Sử dụng | a[0]=5; //cất 5 vào ô nhớ đầu tiên | |
| Giả sử dãy ô  nhớ int được  cấp phát ở  địa chỉ bắt  đầu là 100 | 100  5  dãy 4 ô nhớ, mỗi ô nhớ 2 bytes để chứa số  100  nguyên kiểu int.  a (2 bytes) | |

12 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**8. Tham chiếu (reference)**

Những “tham số hình thức” trong hàm có thể được khai báo là sẽ nhận “tham số thực” theo giá trị hoặc theo  tham chiếu.

**-** Nhận “tham số thực” theo giá trị: hàm gọi sẽ gởi giá trị của “tham số thực” cho “tham số hình thức”  tương ứng của hàm được gọi (“tham số hình thức” sẽ chứa giá trị của “tham số thực”) **-** Nhận “tham số thực“ theo tham chiếu: hàm gọi sẽ gởi địa chỉ của “tham số thực” cho “tham số hình  thức” tương ứng của hàm được gọi (“tham số hình thức” sẽ chứa địa chỉ của “tham số thực” và C++  xem “tham số thực” và “tham số hình thức” như là một)

|  |  |
| --- | --- |
| a, b gọi là “tham số hình thức”. Hàm F được khai  báo là sẽ nhận tham số a theo giá trị, tham số b theo  tham chiếu. | c, d gọi là “tham số thực”. |
| void F(int a, int &b)  {  a++; b++;  cout<<a<<’,’<<b;  }  Kết quả in ra là: 6, 9 | void main()  {  int c=5; d=8;  F(c,d);  cout<<c<<’,’<<d;  }  Kết quả in ra là: 5, 9 |
| a chứa giá trị của c, a và c là khác nhau  5  nên khi tăng a, không làm thay đổi c  a  b *100*  b chứa đ/c của d (C++ xem b và d là một), nên  khi tăng b chính là tăng d.  Các “tham số hình thức” của hàm F(a,b) | *100* (đ/c của d)  5   8 9  c  d  c không đổi, d thay đổi  Các “tham số thực” c,d của hàm main() sẽ được  truyền cho hàm F(a,b) |

**Nhận xét:**

Khi “tham số thực” được gởi theo giá trị, thì những thay đổi đối với “tham số hình thức” tương ứng sẽ **không** ảnh hưởng tới “tham số thực”.

Khi “tham số thực” được gởi theo dia chi thì nếu “tham số hình thức” thay đổi thì “tham số thực” tương  ứng sẽ **thay đổi** theo (vì khi đó hai tham số xem như là một) .Tham chiếu cung cấp một bí danh hay một tên  thay thế cho 1 đối tượng.

Gọi F(a,**8**) sẽ báo lỗi vì F đang chờ nhận tham số thứ 2 như là một địa chỉ của một biến chứ không phải là  một hằng số.

**vi dụ:** viết ct hoán chuyển giá trị hai biến nguyên

|  |  |
| --- | --- |
| Dùng con trỏ (C thường) | Dùng tham chiếu (C++) |
| void hoanvi(int\* a, int\* b)//a,b giu dc cua x,y {  int c=\*a; \*a=\*b; \*b=c;  }  void main()  {  int x=1,y=2;  hoanvi(&x,&y);  printf("%d,%d",x,y);//xuat 2,1  } | void hoanvi(int& a, int& b) *//a,b la ten khac cua x,y* {  int c=a; a=b; b=c;  }  void main()  {  int x=1,y=2;  hoanvi(x,y);  cout<<x<<','<<y; //xuat 2,1  } |

13 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân)

**ví dụ:**

#include <iostream>

int &f(int &n)

{

n++; return n;

}

int &g(int &n)

{

n+=2; return n; //(\*)

}

void main()

{

int n=1;

g(f(n))++; //(\*\*)

 cout<<n; //xuat 5

}

- Nếu sửa g: int g(int& n){…} lệnh (\*\*) sẽ báo lỗi không phải là biến

- Nếu sửa g: int& g(int n){…} lệnh (\*) sẽ báo lỗi trả về tham chiếu biến cục bộ n **Ví dụ:**

#include <iostream>

int &f(int &a, int b)

{

b+=a++; return (a);

}

int main()

{

int i=2, j=4; int k=f(i,j); k++; // k++ ko liên quan a/i // a/i = 3 b/j = 4 k = 4 cout<<i<<" "<<j<<" "<<k<<endl; //xuat 3 4 4

i=2, j=4;

 int **&**l=f(i,j); l++; // *l chinh là i/a nên l++ chính là a/i++* = vì “tham chiếu” a/i // a/i = 4 b/j = 4 l = 4 cout<<i<<" "<<j<<" "<<l<<endl; //xuat 4 4 4

}