1 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 ô nhớ khác  5  nhau nên khi tăng a, không làm thay đổi ca  b *dc100*  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) | *dc100* (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 (truyền tham số)* 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 (truyền tham số)* 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.

Chú ý: 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. F(5,d) :Đúng

**vi dụ 1:** 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  } |

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

**ví dụ 2:**

#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 NÊN không thể ++ - Nếu sửa g: int &g(int n){…} lệnh (\*) sẽ báo lỗi trả về tham chiếu (&) biến cục bộ n **Ví dụ 3:**

#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

}

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

|  |
| --- |
| **BÀI 2: LỚP VÀ ĐỐI TƯỢNG: CLASS & OBJECT** |

**I. Dữ liệu tĩnh, phương thức tĩnh**

**1. Dữ liệu tĩnh (static data)**

- Nếu **biến** trong một **hàm** được khai báo là static thì gọi là biến tĩnh. Gía trị của biến tĩnh sẽ đựơc giữ lại  trong những lần gọi lại hàm.

**Vi dụ:**

void f()

{ int x=0; //bien thuong = mỗi lần gọi hàm (kể cả các lần sau), luôn được cấp phát lại (và sẽ gán lại) **static int y=0;** //bien tĩnh = khi gọi hàm các lần sau, không cấp phát lại (và cũng ko gán lại)

x++; y++;

cout<<”x=”<<x<<”, y=”<<y<<endl;

}

void main()

{ f();

 f();

/\* kq in la:

x= 1, y= 1

x= 1, y= 2

\*/

}

- Nếu **biến** trong **lớp** được khai báo là static thì gọi là biến lớp (class variable) và phải có lệnh gán trị ban  đầu (để ở ngoài lớp) cho biến lớp, khi đó biến lớp sẽ được dùng chung cho tất cả các đối tượng thuộc  lớp. Để phân biệt, biến *không có static* còn gọi là *biến thể hiện (instance variable)*, mỗi đối tượng thuộc  cùng lớp sẽ có biến thể hiện riêng nhưng dùng chung biến lớp.

**Ví dụ:**

#include <iostream>

class C

{ public:

int x; //bien the hien, moi doi tuong se co bien the hien rieng

static int y;//bien lop, tat ca cac doi tuong thuoc lop se dung chung bien nay

};

int C::y=0; // lenh gan tri ban dau cho bien tinh, lenh nay bat buoc phai co

int main()

{ C c1,c2;

c1.x=1; c1.y=2;

c2.x=3; c2.y=4;

cout<<"c1.x="<<c1.x<<" ,c1.y="<<c1.y<<endl; 1 4

 c1.y++;

cout<<"c2.x="<<c2.x<<", c2.y="<<c2.y; 3 5

}

 y = 0 = 2 = 4 = 5 

https://lh5.googleusercontent.com/PIEkXSjrT3AYm_xw3vacd1xIrmWt3gpFdHlgxJ0wVqpGgYkDIG4hAHe0URxXpQdJ2_WeS-kaNTPVbRIc15rZN0ghlDYvGZtBK8Fde5e1QJHgahjn2WktEVi7aVhWyv5Y0UNq9WGQ=s01 https://lh3.googleusercontent.com/ih1-LnllLnF4uMf55o3iCBs73fMObrZceU0i29UzcmgM42N-1dM82aEuRWbeVke7YTEbNjiL7W8V2en4OJm68i318JG4xkJTrZywNdNwecNT2RmJmbjuG1OTqs7RSX2fps_U64T2=s0x

https://lh4.googleusercontent.com/29CPxBBCMCspw3uCToYoFOPv1DXfnxg9oig0UVqdZ_bp5RWVwOJsP4OYgtB4G21ytEQQESB5mv440isK_468mTvIQ5YeU9Or4yFO3exjdLUDmfH8ZZT87qFVK81S_8dggrK91G_Q=s04 https://lh3.googleusercontent.com/fXUPWQCONfp4KKaj4JXEB7xex1XlBe9pY5UyD23_2lur1c2FNx31W7kAPXRkyvwS80M4hJpuIaZFRPqW3jG1HMT89XJeHXdLFXtZSBWQ3DXIq_OopRDW3KgdOFP3HGpWXsOSIObV=s0y

https://lh6.googleusercontent.com/doZHGxgMi7iPGUv9FUMdvFRpocSpzecEJzIxM-h05h4ZysjvJzRZiokhqKGMJHCzdCRZ8wYx7cpGaB2cWp2ParyQeRF6wM5evhK06ShmqgonWnX5nFU_FiFmr-eIY5JIAJH6uNG7=s03 https://lh3.googleusercontent.com/tfZhGLXCySyMB2yC183lv_0CRBiP6EZF4VkY6rCNGBRvzMCTr-CdWr7HO60d8cwZbfAXP5JJVPglX7v07u_7kXJxQ4Ot1zWC0c3wh8RKIhgsHrDdukmqlg9abQVBEZtFWipscaZm=s0x

https://lh3.googleusercontent.com/bvp9tJ_whO7q3Cz9CP2RxSjC1GL6ccZ8kSiPkdnBKFBsBLC7QD3oRq_diBqP36-zNnzJ-ZOcDD6Dh0u3Z1KSDB-Ajm_29Oigb99M20LV8zV2to-2NIrrDLqpYZWFCk06oNDIUJxe=s0c1https://lh5.googleusercontent.com/S4Er9g65UMLooIja0BblC5HdBZ1P8lzASmkp3y_Jc7EG21JKzgn6RBZkkxRufIWNmhvME7lGGZHLysuqbCgbQ7Ip0L41-QfzcEY3SBuOs5RBIs5QUGySmT6RN7p09fJFNC7U0PHE=s0c2

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

**2. Phương thức tĩnh (static method):**

Nếu **phương thức** được khai báo là static thì gọi là phương thức tĩnh. Phương thức tĩnh có thể truy xuất  bằng tên lớp hoặc bằng tên đối tượng. Phương thức *không có static* gọi là *phương thức thể hiện*, chỉ được  *truy xuất thông qua tên đối tượng*. Thông thường những phương thức *không có liên quan đến dữ liệu của  đối tượng thì khai báo là phương thức tĩnh*, ngược lại phải khai báo là phương thức thể hiện.

**Ví dụ:**

#include <iostream>

class calculate

{

int a,b;

public:

void input() //phương thức thể hiện, khi gọi phải thông qua tên đt

{

cout<<"Input two numbers:";

cin>>a>>b;

}

int add() //phương thức thể hiện, khi gọi phải thông qua tên đt

{

return a+b;

}

static void Bye() //phương thức lớp, có thể gọi bằng tên lớp

{

cout<<endl<<"Good-bye, see you again!";

}

};

int main()

{ calculate::Bye();

calculate c;

c.input();

cout<<"Sum="<<c.add(); //input() la phuong thuc the hien, khi goi phai dung ten dt  calculate::Bye(); //Bye() la phuong thuc lop, khi goi dung ten lop hoac ten dt

//c.Bye() cung duoc

}

**CHÚ Ý:** NẾU đinh nghĩa phương thức static ngoài class thì KHÔNG kèm static

void calculate::Bye() //không thêm static phía trước.

{

cout<<endl<<"Good-bye, see you again!";

}

**\* Nhận Xét:**

Các phương thức input(), add() có liên quan đến dữ liệu của đối tượng nên không k/báo là pt tĩnh, phương  thức Bye() không phụ thuộc vào dl nên khai báo là phương thức tĩnh.

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

**II. Các phương thức đặc biệt trong lớp**

**1. Phương thức khởi tạo (constructor method)**

Là phương thức public có dạng **Tenlop (các tham số)**

Phương thức khởi tạo sẽ cùng tên với tên lớp, không có kiểu trả về và không có void. Mỗi lớp có thể có  nhiều phương thức khởi tạo (chồng hàm), các phương thức này sẽ khác nhau ở số tham số, hoặc kiểu tham  số. Phương thức khởi tạo được tự động thực hiện khi đối tuợng được tạo ra. Mục đích của phương thức khởi  tạo là gán giá trị ban đầu cho đối tượng trước khi sử dụng, mà công việc này người lập trình hay quên.

**ví dụ:**

#include <iostream>

class phanso

{ private:

int ts,ms;

 public:

phanso(int x=0, int y=1);//phuong thuc khoi tao co hai tham so

void nhap();

void xuat();

};

phanso::phanso(int x , int y)

{ ts=x;ms=y;

}

void phanso::nhap() //phuong thuc nhap ps

{ cout<<"\nNhap ts:"; cin>>ts;

do

{ cout<<"Nhap ms:"; cin>>ms;

if (ms==0)

 cout<<"Mau so phai khac khong"<<endl;

 }while (ms==0);

}

void phanso::xuat()

{ cout<<endl<<ts<<”/”<<ms;

}

int main()

{

phanso ps1(5,6); //tao doi tuong ps1 = 5/6

cout<<"Phan so 1:";

ps1.xuat();

phanso ps2(5);//tao dt ps2= 5/1

cout<<"\nPhan so 2 truoc khi nhap:";

ps2.xuat();

cout<<"\nNhap phan so 2:";

 ps2.nhap();

cout<<"Phan so 2 sau khi nhap:";

ps2.xuat();

 phanso \*ps3=new phanso(7,2);

cout<<"Phan so 3:";

ps3->xuat();

 phanso ps4;

cout<<"Phan so 4:";

ps4.xuat(); //tao dt ps4= 0/1

}

\* Ghi chú: [KHÔNG NHẤT THIẾT]

*Nếu đã định nghiã phương thức khởi tạo thì phải định nghĩa phương thức khởi tạo không tham số mặc dù  có thể không sử dụng.*

6 TS. GVC. VÕ XUÂN THỂ: voxuanthephd@gmail.com (công việc) tranhailua@yahoo.com (cá nhân) **2. Phương thức hủy bỏ (destructor method)**

Là phương thức public có dạng :

**~tenlop()**

Pt hủy bỏ không có kiểu trả về, không có void và không có tham số. Mỗi lớp nếu có thì chỉ có một pt hủy  bỏ. Pt hủy bỏ được gọi tự động khi đối tượng "chết đi" (đt bị thu hồi bộ nhớ), ví dụ như đối tượng là biến  cục bộ của một hàm, khi hàm thực hiện xong, đt sẽ bị thu hồi bộ nhớ.

Mục đích của pt hủy bỏ là thực hiện các công việc "dọn dẹp" liên quan đến đt như là: thu hồi bộ nhớ được  cấp phát động, đóng tập tin,...

**ví dụ:**

#include <iostream>

class mang

{ private:

int size;

 int\* array;

 public:

mang(int n); //phương thức khởi tạo (construtor)

~mang(); //phương thức hủy bỏ (destructor)

void nhap();

void xuat();

};

mang::mang(int n)

{ size=n;

array=new int[n];//cap phat day n o nho kieu int (cap phat bo nho dong)

}

mang::~mang()

{ delete []array;

 cout<<"\nmang "<<size<<" pt da duoc thu hoi bo nho";

}

void mang::nhap() //phuong thuc nhap mang

{

for(int i=0; i<size; i++) cin>>array[i];

}

void mang::xuat()

{

cout<<endl;

for(int i=0; i<size; i++) cout<<array[i]<<” “;

}

void main()

{

mang a(3), b(4);

cout<<"\nNhap mang a co 3 phan tu: "; a.nhap();

cout<<"\nXuat mang a: "; a.xuat();

 //mang 4 duoc thu hoi truoc mang 3

}

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

**3. Phương thức khởi tao sao chép (copy constructor method)**

Phương thức khởi tạo sao chép là phương thức có dạng

**tenlop(const tenlop& tendt)**

Phương thức khởi tạo sao chép dùng để tạo ra đối tượng mới từ đối tượng có sẵn. Nếu không định nghĩa pt  khởi tạo sao chép thì dùng pt khởi tạo sao chép mặc định, khi đó đt mới sẽ hoàn toàn giống đt có sẵn. THƯỜNG TỰ ĐỘNG CHẠY KHI: + GÁN 2 ĐỐI TƯỢNG CHO NHAU

 + TRUYỀN THAM SỐ THỰC dạng ĐỐI TƯỢNG=> THAM SỐ HÌNH THỨC   + kết quả Hàm là Đối tượng [XEM DƯỚI]

**Ví dụ:**

#include <iostream>

class C

{

int x;

public:

C(int k)

{

x=k;//phuong thuc khoi tao

}

void xuat()

{

cout<<"\nx="<<x;

}

C(const C& c)// phuong thuc khoi tao sao chep (\*)

{

x=c.x+5;

}

};

void main()

{

C a(1); //tao dt a co a.x=1

C b(a); //tao dt b co b.x=a.x+5;dung phuong thuc khoi tao sao chep (\*)

 HOẶC C b = a; //cũng được

a.xuat(); //xuat x=1 (x cua a)

b.xuat(); //xuat x=6 (x cua b)

//neu khong co pt khoi tao sao chep, se dung pt mac dinh, khi do a.x=b.x=1

}

|  |  |
| --- | --- |
| Dùng pt sao chép mặc định | Dùng pt sao chép tự tạo |
| 1  1  x  x  b  a | 6  1  x  x  b  a |

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

**\* Chú ý:**

**a) Nếu lớp có thành phần dữ liệu là con trỏ, việc dùng phương thức sao chép mặc định sẽ dẫn tới hai  thành phần con trỏ chỉ tới cùng một địa chỉ.**

**ví dụ:**

Nếu thêm vào lớp mảng phương thức khởi tạo sao chép như sau:

mang::mang(const mang& x)

{

size=x.size;

 array=new int [size];

for(int i=0; i<x.size; i++) array[i]=x.array[i];

}

void main()

{

mang a(3);

mang b(a); // mang b = a;

}

Khi đó b sẽ có mảng array được cấp phát bộ nhớ riêng.

|  |  |
| --- | --- |
| Dùng pt sao chép mặc định | Dùng pt sao chép tự tạo |
| 1 2 3  array3  array3  size  size  b  a  a và b dùng chung mảng array | 1 2 3 1 2 3  array3  array3  size  size  a  b  a và b có mảng array riêng |

**b) Phương thức khởi tạo sao chép được gọi tự động trong các trường hợp sau:** - **Khi một đối tượng của lớp được khởi tạo bằng một đối tượng khác nhưng cùng lớp** - **Khi một đối tượng được gởi như một tham số tới một hàm/ phương thức**

- **Khi hàm/phương thức trả về đối tượng**

**Ví Dụ:**

A a1;

A a2(a1);//tao dt a2 va goi pt sao chep a1 vao a2

A a3=a1;//tao dt a3 va goi pt sao chep a1 vao a3

**Ví Dụ:** chương trình sau gọi pt sao chép 7 lần (mỗi lần gọi pt f, sẽ cần gọi pt sao chép 3 lần) A f(A a) //goi pt sao chep

{ A a1(a); // tao dt a1, goi pt sao chep, chep a vao a1

A a2=a1; // lenh nay tuong tu lenh A a1(a)

return a2;

}

main()

{

A x;

A y=f(f(x)); //goi pt sao chep

}

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

**Ví Dụ:**

#include <iostream>

class A

{ static int n;

public:

A(){ n++;}

~A(){n--;}

static void xuat()

{ cout<<n<<endl; }

};

int A::n=0;

void main()

{

A::xuat();

A a1; a1.xuat();

A\* pa; a1.xuat();

 pa= new A; a1.xuat();

delete pa; a1.xuat();

A a2=a1; //goi pt sao chep mac dinh

a2.xuat();

}

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

**III.Hàm bạn và lớp bạn**

**1. Hàm bạn**

Hàm bạn là hàm không phải là hàm thành viên của lớp nhưng truy xuất được dữ liệu private của lớp. Khai  báo hàm bạn bằng từ khoá friend và có thể khai báo ở phần public hoặc private của lớp. 

private  datas

methods

normal function

friend  function

|  |  |
| --- | --- |
| class C  {  int x;  public:  void f() {x++;}  friend void g(int, int); }; | void g(int x, int y)  {  C c;  c.x=5;//đúng  }  Hàm g là bạn của lớp C nên  có thể truy xuất được dữ liệu  private của C |

Hàm bạn có thể là bạn của nhiều lớp nên có thể truy xuất được nhiều thành phần private của các lớp khác  nhau và đây chính là lý do để sử dụng hàm bạn.

**Ví dụ:**

class A

{

int x;

public: friend void xuat();

};

class B

{

int y;

public: friend void xuat();

};

void xuat(A a, B b)

{

cout<<a.x<<b.y;

}

**2. Lớp bạn**

Để tất cả các phương thức của lớp A là bạn của lớp B, ta chỉ cần khai báo là lớp A là bạn của lớp B, thay vì  phải khai báo từng phương thức là bạn.

Ví dụ:

|  |  |
| --- | --- |
| class A;  class B  {  int x;  public:  B(int k){ x =k;}  friend class A;  }; | class A  {   int x;   public:   A(int k){ x =k; }   void xuat(B b)   {   cout<<"xA ="<<x<<" xB ="<<b.x<<endl;  }  };  Do lớp A là bạn của lớp B nên pt xuat() sẽ là bạn  của B, do đó có thể truy xuất đến thành phần x  private của B. |

int main()

{ A a(5);

B b(7);

a.xuat(b);

}