THUỘC TÍNH & PHƯƠNG THỨC

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

GVHD: Trương Toàn Thịnh

NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

GIỚI THIỆU

- Lớp trong C++ bao gồm: thuộc tính và phương thức
- Thuộc tính và phương thức có tầm vực
  - public: phạm vi truy xuất từ ngoài và trong lớp
  - private: phạm vi truy xuất từ trong lớp

```
class phanso{
  private:
    int tu;
    int mau;
  public:
    void nhap(int, int);
    void xuat();
};
```

```
void main(){
  phanso p;
  p.nhap(2, 3);
  p.tu = 4;
}
```

Hợp lệ vì nhap có tầm vực public

Không hợp lệ vì tu có tầm vực private

NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

PHƯƠNG THỨC TẠO LẬP

- Khởi tạo dữ liệu đối tượng có thể dùng
  - Hàm (ví dụ hàm ‘nhap(int, int)’): Nếu người lập trình quên gọi hàm này thì đối tượng sẽ chứa dữ liệu ‘rác’
  - Phương thức tạo lập (constructor)
- Đặc điểm constructor:
  - Tên trùng với tên lớp & không có kiểu trả về
  - Tự động chạy khi tạo đối tượng để chuẩn bị dữ liệu cho đối tượng
  - Có nhiều phương thức tạo lập chồng nhau

PHƯƠNG THỨC TẠO LẬP

- Các loại phương thức tạo lập

```
class PhanSo{
  private:
    int tu, mau;
  public:
    PhanSo();
    PhanSo(int, int);
    PhanSo(const PhanSo&)
};
PhanSo::PhanSo(){
  tu = 0; mau = 1;
}
PhanSo::PhanSo(int t, int m){
  tu = t; mau = m;
}
PhanSo::PhanSo(const PhanSo& p){
  tu = p.tu; mau = p.mau;
}
```

Constructor mặc định

Constructor chứa đầy đủ tham số

Constructor từ một đối tượng khác

```
void main(){
  PhanSo p(2, 3);
  PhanSo q(p);
  PhanSo t();
  cout << p.tu << "/" << p.mau << endl;
  cout << q.tu << "/" << q.mau << endl;
  cout << t.tu << "/" << t.mau << endl;
}
```

PHƯƠNG THỨC TẠO LẬP

- Một số chú ý:
  - Nếu không khai báo/định nghĩa bất kì một constructor nào thì trình biên dịch sẽ tự cung cấp constructor mặc định không tham số

```
class PhanSo{
  private:
    int tu, mau;
  public:
    void xuat();
    void nhap(int, int);
    //…
};
void main(){
  PhanSo t();
}
```

PHƯƠNG THỨC TẠO LẬP

- Một số chú ý:
  - Nếu trường dữ liệu không chứa con trỏ thì ta có thể bỏ constructor sao chép từ một đối tượng khác.

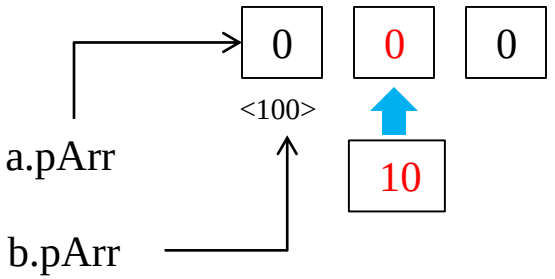
```
class PhanSo{
  int tu, mau;
  public:
    PhanSo(int, int);
};
PhanSo::PhanSo(int t, int m){
  tu = t; mau = m;
}
void main(){
  PhanSo p(2, 3);
  PhanSo q(p);
  PhanSo t;
}
```

PHƯƠNG THỨC TẠO LẬP

- Ví dụ: Xét bài tập MyIntArray

```
class MyIntArray{
  private:
    int *pArr; int size;
  public:
    MyIntArray(int);
    MyIntArray();
    void update(int k, int val){pArr[k] = val;}
    int get(int k){return pArr[k]}
};
MyIntArray::MyIntArray(){
  size = 0; pArr = NULL;
}
MyIntArray::MyIntArray(int n){
  size = n;
  pArr = new int[size];
  for(int i = 0; i < size; i++)
    pArr[i] = 0;
}
```

```
void main(){
  MyIntArray a(3);
  MyIntArray b(a);
  a.update(1, 10); //a[1] = 10
  cout << b.get(1) << endl;
}
```

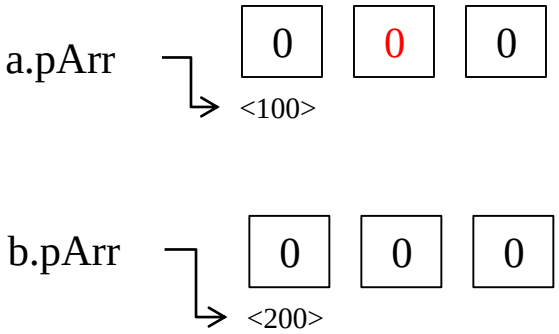


PHƯƠNG THỨC TẠO LẬP

- Ví dụ: ta cần xây dựng hàm tạo lập sao chép từ một đối tượng trong trường hợp dữ liệu chứa con trỏ

```
class MyIntArray{
  private:
    int *pArr; int size;
  public:
    //…
    MyIntArray(const MyIntArr&);
};
MyIntArray::MyIntArray(const MyIntArr& src){
  size = src.size;
  pArr = new int[size];
  for(int i = 0; i < size; i++)
    pArr[i] = src.pArr[i];
}
```

```
void main(){
  MyIntArray a(3);
  MyIntArray b(a);
  a.update(1, 10); //a[1] = 10
  cout << b.get(1) << endl;
}
```



NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

PHƯƠNG THỨC HỦY

- Được gọi mặc định ngay khi đối tượng không còn được dùng nữa và bị hủy
- Thu hồi các tài nguyên đã cấp cho đối tượng trong quá trình sống
- Việc dọn dẹp giúp tránh rò rỉ bộ nhớ:
  - Chương trình dừng do thiếu bộ nhớ
  - Các phần mềm khác không chạy được do chương trình ta đã chiếm hết bộ nhớ
  - Sau khi chương trình ta chạy xong thì treo máy

PHƯƠNG THỨC HỦY

- Đặc điểm phương thức hủy:
  - Có tên trùng với tên lớp kết hợp với dấu '~'
  - Không có tham số & giá trị trả về
  - Mỗi lớp có duy nhất MỘT phương thức hủy
  - Tự động chạy khi đối tượng hết phạm vi sử dụng
  - Được gọi MỘT lần duy nhất

PHƯƠNG THỨC HỦY

- Ví dụ: lấy tiếp ví dụ về MyIntArray

```
class MyIntArray{
  private:
    int *pArr; int size;
  public:
    MyIntArray(int);
    ~MyIntArray();
    //…
};
MyIntArray::~MyIntArray(){
  if(size > 0){
    size = 0; delete []pArr; pArr = NULL;
  }
}
MyIntArray::MyIntArray(int n){
  size = n;
  pArr = new int[size];
  for(int i = 0; i < size; i++)
    pArr[i] = 0;
}
```

```
void main(){
  MyIntArray a(10);
  MyIntArray* b = new MyIntArray(20);
  //…
  delete b;
}
```

1. Phương thức hủy của đối tượng a sẽ được tự động gọi
2. Phương thức hủy của đối tượng b sẽ được tự động gọi khi thực hiện câu lệnh ‘delete b’

NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

TOÁN TỬ GÁN

- Toán tử giúp câu lệnh trông tự nhiên hơn
  - Ví dụ:

```
PhanSo a(1, 2), b(2, 3);
PhanSo kq = a + b;
```

- Toán tử có thể hai ngôi hoặc một ngôi
- Một số toán tử không thể nạp chồng
  - Toán tử ‘.’, ‘?:’, ‘sizeof’ (thuộc C)
  - Toán tử ‘::’, ‘.*’, ‘typeid’ (thuộc C++)

TOÁN TỬ GÁN

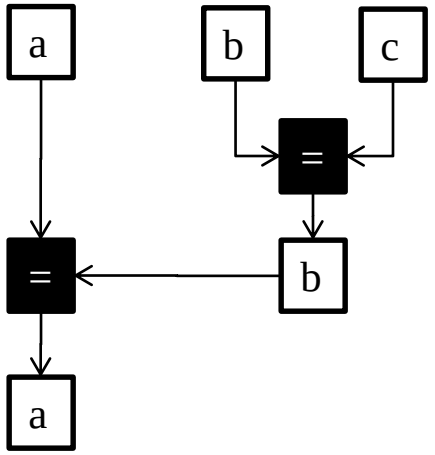
- Toán tử gán dùng để gán giá trị một đối tượng sang đối tượng khác
- C++ tự cung cấp toán tử gán mặc định, sao chép từng thành phần dữ liệu từ đối tượng này sang đối tượng kia
- Lớp có chứa trường dữ liệu con trỏ thì cần định nghĩa lại toán tử gán
- Lớp không chứa trường dữ liệu con trỏ thì không cần định nghĩa toán tử gán

TOÁN TỬ GÁN

- Ví dụ: thêm toán tử gán trong MyIntArray

```
class MyIntArray{
  private:
    int *pArr; int size;
  public:
    MyIntArray& operator=(const MyIntArray&);
    //…
};
MyIntArray& MyIntArray::operator=(const MyIntArray& src){
  if(this != &src){
    delete []pArr;
    size = src.size;
    pArr = new int[size];
    for(int i = 0; i < size; i++)
      pArr[i] = src.pArr[i];
  }
  return *this;
}
```

```
void main(){
  MyIntArray a, b, c(3);
  a = b = c;
}
```



Bên trong một phương thức, this là con trỏ chứa địa chỉ của biến đối tượng đang gọi thực hiện phương thức

TOÁN TỬ GÁN

- Nếu lớp có kế thừa, nên gọi toán tử gán lớp cha trước

```
class MyIntArray : public Base{
  private:
    int *pArr; int size;
  public:
    MyIntArray& operator=(const MyIntArray&);
    //…
};
MyIntArray& MyIntArray::operator=(const MyIntArray& src){
  if(this != &src){
    Base::operator=(src);
    delete []pArr;
    size = src.size;
    pArr = new int[size];
    for(int i = 0; i < size; i++) pArr[i] = src.pArr[i];
  }
  return *this;
}
```

TOÁN TỬ GÁN

- Có thể định nghĩa toán tử '>>' hay '<<' trong tập tin '.h'/'cpp' kết hợp dùng từ khóa friend (hàm ý toán tử này là bạn của lớp)
- Cú pháp toán tử một ngôi (tiền tố hay hậu tố): Kiểu_trả_về Tên_lớp::operator@()
  - @: là kí tự đại diện toán tử, ví dụ ++ hay –
  - Toán tử này không có tham số đầu vào
- Quyết định dùng toán tử một hay hai ngôi tùy thuộc vào nhu cầu bài toán

TOÁN TỬ GÁN

- Ví dụ: xét lớp PhanSo

```
class PhanSo{
  int tu, mau;
  public:
    PhanSo();
    PhanSo(int, int);
    PhanSo& operator+=(const PhanSo&);
    PhanSo operator+(const PhanSo&);
    bool operator==(const PhanSo&);
    PhanSo& operator++();//++ tiền tố
    PhanSo operator++(int);//++ hậu tố
    friend ostream& operator<<(ostream&, const PhanSo&);
    //…
};
```

TOÁN TỬ GÁN

- Ví dụ: xét lớp PhanSo

```
PhanSo& operator+=(const PhanSo& src){
  tu = tu*src.mau + mau*src.tu;
  mau = mau*src.mau;
  return *this;
}
PhanSo operator+(const PhanSo& src){
  PhanSo tmp = (*this);
  tmp+=src;
  return tmp;
}
bool operator==(const PhanSo& src){
  return ((tu == src.tu) && (mau == src.mau));
}
```

```
PhanSo& operator++(){//++ tiền tố
  tu+=mau;
  return (*this);
}
PhanSo operator++(int){//++ hậu tố
  PhanSo tmp = *this;
  ++(*this);
  return tmp;
}
```

```
void main(){
  PhanSo a(1, 3), b(2, 5), c = a + b;
  cout << c << endl; // 11/15
  cout << ++c << endl; // (11+15)/15
  a += b;
  cout << a << endl; // 11/15
  cout << b++ << endl; // 2/5
  cout << b << endl; // (2+5)/5
}
```

TOÁN TỬ GÁN

- Một số qui ước/lời khuyên:
  - Chỉ nạp chồng toán tử khi giúp chương trình tự nhiên hơn
  - Đảm bảo hợp lý cho toán tử, ví dụ không thể nạp chồng toán tử + cho phân số, nhưng bên trong lại cài đặt phép trừ
  - Chỉ nạp chồng toán tử khi hợp ngữ nghĩa, ví dụ không thể NHÂN hai NGÀY
  - Đảm bảo tính nhất quán cho các cặp toán tử, ví dụ ==/!=, >/<…

NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

CÁC VẤN ĐỀ THAM SỐ

- Tương tự như biến cấu trúc, biến đối tượng có cùng cơ chế truyền tham số:
  - Truyền tham trị
  - Truyền tham chiếu
  - Truyền con trỏ
- Truyền tham trị: trình biên dịch tạo một bảng sao của biến đối tượng truyền vào
- Truyền tham chiếu: trình biên dịch tạo một ‘alias’ cho biến đối tượng truyền vào
- Truyền tham con trỏ: tùy vào cách truyền tham trị hay tham chiếu trình biên dịch sẽ xử lý như trong tham trị và tham chiếu

CÁC VẤN ĐỀ THAM SỐ

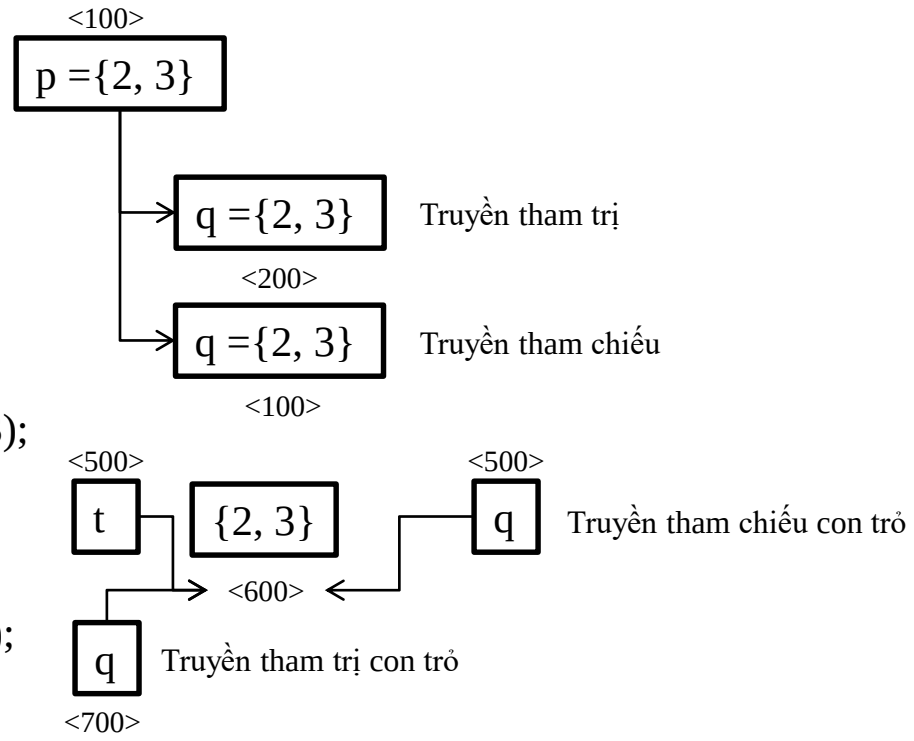
- Ví dụ: xét ví dụ PhanSo

```
class PhanSo{
  int tu, mau;
  public:
    PhanSo(int, int);
    void Update(int, int);
    friend ostream& operator<<(ostream& os, const PhanSo&);
};
PhanSo::PhanSo(int t, int m) {
  tu = t; mau = m;
}
void PhanSo::Update(int t, int m) {
  tu = t; mau = m;
}
ostream& operator<<(ostream& os, const PhanSo& ps) {
  os << ps.tu << "/" << ps.mau << endl;
  return os;
}
```

CÁC VẤN ĐỀ THAM SỐ

- Ví dụ: xét ví dụ PhanSo

```
void ThayDoiThamTri(PhanSo q) {q.Update(10, 10);}
void ThayDoiThamChieu(PhanSo& q){q.Update(10, 10);}
void ThayDoiThamTriContro(PhanSo* q) {q->Update(10, 10);}
void ThayDoiThamChieuContro(PhanSo*& q) {q->Update(10, 10);}
void main() {
  PhanSo p(2, 3);
  cout << p;
  ThayDoiThamTri(p);
  cout << p;
  ThayDoiThamChieu(p);
  cout << p;
////////////////////////////////////
  PhanSo* t = new PhanSo(2, 3);
  cout << *t;
  ThayDoiThamTriContro(t);
  cout << *t;
  ThayDoiThamChieuContro(t);
  cout << *t;
}
```



NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

MỘT SỐ CÁCH ÉP KIỂU

- C++ hỗ trợ hai loại ép kiểu:
  - Ep kiểu ngầm định
  - Ép kiểu tường minh
    - Chuẩn C (C-style)
    - Hằng (const)
    - Dịch lại (re-interpret)
- Người lập trình có thể :
  - Dùng phương thức tạo lập để ép kiểu
  - Dùng toán tử để ép kiểu

MỘT SỐ CÁCH ÉP KIỂU

- Ép kiểu ngầm định
  - Trình biên dịch tự ép kiểu sau đó sử dụng kết quả để thực hiện các thao tác kế tiếp

```
void main(){
  short a = 2;
  int b = a;
  cout << b;
}
```

- Ép kiểu chuẩn C: dùng lại kĩ thuật trong C

```
void main(){
  short a = 2;
  int b = (int)a;
  cout << b;
}
```

MỘT SỐ CÁCH ÉP KIỂU

- Ép kiểu hằng: ép một biến đối tượng hằng sang một biến đối tượng

```
class A{
  public:
    void doSomething(){};
};

void test(const A& a){
  a.doSomething(); // ERROR
  A* b = const_cast<A*>(&a);
  b->doSomething();
}
```

MỘT SỐ CÁCH ÉP KIỂU

- Ép kiểu dịch lại: ép một biến đối tượng thuộc lớp này sang một biến đối tượng thuộc lớp kia (Hai lớp có thể hoàn toàn khác nhau)

```
class A{};
class B{};

void main(){
  A* a = new A();
  B* b = reinterpret_cast<B*>(a);
}
```

ÉP KIỂU BẰNG PHƯƠNG THỨC TẠO LẬP

- Trong một số tình huống ta có thể tận dụng constructor để hỗ trợ việc ép kiểu
- Xét ví dụ PhanSo

```
class PhanSo{
  int tu, mau;
  public:
    PhanSo(int);
};
PhanSo::PhanSo(int t){
  tu = t; mau = 1;
}
void main(){
  int t = 5;
  PhanSo p = t;
}
```

ÉP KIỂU BẰNG TOÁN TỬ

- Xây dựng toán tử ép kiểu cho lớp PhanSo

```
class PhanSo{
  int tu, mau;
  public:
    PhanSo(int, int);
    operator int();
};
PhanSo::operator int(){
  return tu/mau;
}
void main(){
  PhanSo p(6, 2);
  int t1 = p;
  int t2 = int(p);
  int t3 = (int)p;
}
```

NỘI DUNG

- Giới thiệu
- Phương thức tạo lập
- Phương thức hủy
- Toán tử gán (‘=’)
- Các vấn đề tham số
- Một số cách ép kiểu
- Một số vấn đề khác

MỘT SỐ VẤN ĐỀ KHÁC

- Một số vấn đề về thành phần dữ liệu
  - Thành phần tĩnh
  - Thành phần có kiểu dữ liệu thuộc lớp khác
  - Phương thức const
- Mẫu singleton
- Lớp với các phương thức tĩnh

THÀNH PHẦN TĨNH

- Tình huống: nhu cầu sử dụng một biến là thành phần dữ liệu chung cho tất cả các đối tượng
- C++ hỗ trợ biến/hàm tĩnh (class-level member)
  - Không phụ thuộc vào bất kì đối tượng nào
  - Thành phần chung cho tất cả các đối tượng
- Cú pháp:
  - Khai báo:

    static <Kiểu_dữ_liệu> Tên_hàm(danh_sách_tham_số);

    static <Kiểu dữ liệu> Tên_biến;
  - Định nghĩa:

    <Kiểu_dữ_liệu> Tên_lớp::Tên_biến = …;

THÀNH PHẦN TĨNH

- Xét ví dụ

```
class Test{
  static int count;
  public:
    static int Show();
    Test(){count++;}
    ~Test(){count--;};
};
int Test::Show(){
  return count;
}
int Test::count = 0;
void main(){
  Test a, b;
  cout<<Test::Show();
}
```

THÀNH PHẦN KIỂU LỚP

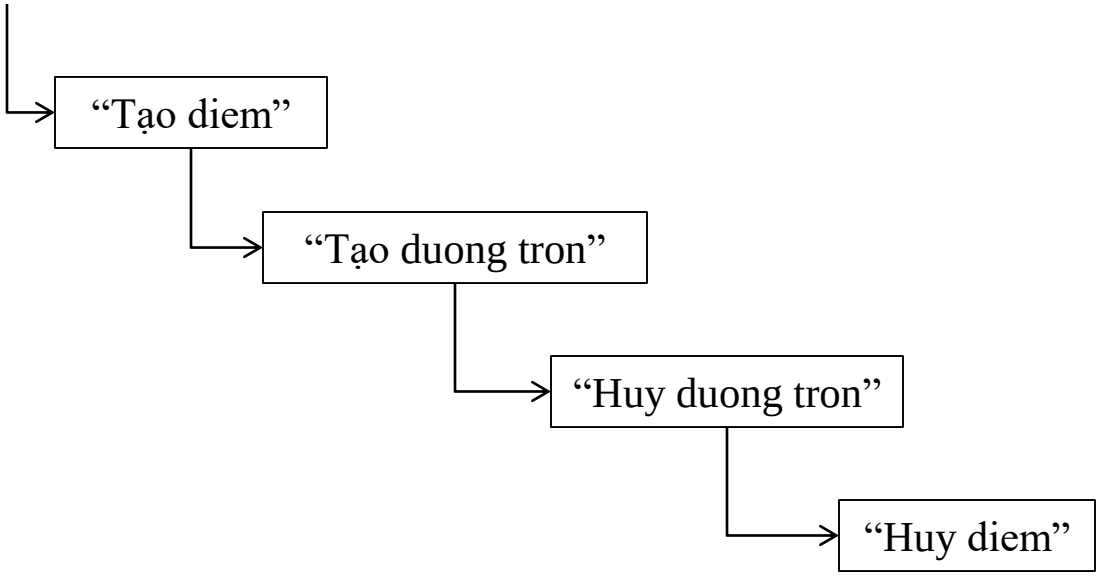
- Xét hai lớp DUONGTRON & DIEM

```
class DUONGTRON{
  DIEM I;
  float R;
  public:
    DUONGTRON(){cout<<"Tao duong tron";};
    ~DUONGTRON{cout<<"Huy duong tron";};
};
class DIEM{
  float X, Y;
  public:
    DIEM(){cout<<"Tao diem";};
    ~DIEM(){cout<<"Huy diem";};
};
```

THÀNH PHẦN KIỂU LỚP

- Xét hai lớp DUONGTRON & DIEM

```
void main(){
  DUONGTRON dt;
}
```



PHƯƠNG THỨC CONST

- Là các phương thức trong quá trình hoạt động không làm ảnh hưởng tới giá trị biến thành viên của các đối tượng
- Qui tắc:
  - Đối tượng non-const/const có thể gọi phương thức const
  - Đối tượng const KHÔNG thể gọi phương thức non-const
- Cú pháp

  <Kiểu_trả_về> Tên_phương_thức(danh sách tham số) const;

PHƯƠNG THỨC CONST

- Ví dụ

```
class A{
  public:
    void test(){}
     void testConst() const{}
};
void doSomething(A a, const A& b){
  a.test();//OK
  a.testConst();//OK
  b.test();//NOT OK
  b.testConst();//OK
}
```

MẪU SINGLETON

- Khi áp dụng mẫu singleton lên một lớp, sẽ đảm bảo chỉ có duy nhất MỘT đối tượng thuộc lớp đó được sinh ra
- Đặc điểm:
  - Hàm tạo lập có tầm vực là private
  - Có hàm tĩnh lấy đối tượng tĩnh bên trong lớp

MẪU SINGLETON

- Mã nguồn mẫu singleton

```
class A{
  static A* obj;
  A();
  public:
    static A* getInstance();
};
A* A::obj = NULL;
A::A(){//…}
A* A::getInstance(){
  if(!obj) obj = new A();
  return obj;
}
void main(){
  A* a = A::getInstance();
}
```

MẪU SINGLETON

- Ví dụ áp dụng singleton

```
class A{
  static A* obj;
  A();

  public:
    static A* getInstance();



};
A* A::obj = NULL;
A::A(){//…}
A* A::getInstance(){
  if(!obj) obj = new A();

  return obj;
}
```

```
class SortAlg{
  static SortAlg* obj;
 SortAlg();
 void (*currentAlg)(float[], int);
  public:
    static SortAlg* getInstance(void (*pAlg)(float[], int) = NULL);
    static void InterchangeSort(float[], int);
    //…các thuật toán sắp xếp
    void Sort(float[], int);
};
SortAlg* SortAlg::obj = NULL;
SortAlg::SortAlg(){currentAlg = InterchangeSort;}
SortAlg* SortAlg::getInstance(void (*pAlg)(float[], int) = NULL){
  if(!obj) obj = new SortAlg();
  if(pAlg != NULL) obj->currentAlg = pAlg;
  return obj;
}
void SortAlg::InterchangeSort(float a[], int n){//…}
void SortAlg::Sort(float a[], int n){
  if(currentAlg != NULL) currentAlg(a, n);
}
```

MẪU SINGLETON

- Ví dụ áp dụng singleton

```
void main(){
  //Chuẩn bị dữ liệu
  A* a = A::getInstance();
  //Sử dụng a để xử lý bài toán
}
```

```
void main(){
  float a[] = {1.4F, -5.2F, 3.3F, 0}; int n = sizeof(a)/sizeof(a[0]);
  SortAlg* a = SortAlg::getInstance();
  a->Sort(a, n);
}
```

  - Hàm getInstance không truyền tham số sẽ dùng InterchangeSort đã truyền trong hàm tạo lập
  - Dùng toán tử '->' khi muốn gọi hàm từ con trỏ cấu trúc