UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

ĐA NĂNG HÓA TOÁN TỬ (TT)

C++

Microsoft Visual Studio

Nội dung

- Gán và khởi động
- Phép toán << và >>
- Phép toán lấy phần tử mảng: [ ]
- Phép toán gọi hàm: ()
- Phép toán tăng và giảm: ++ và --

Gán và khởi động

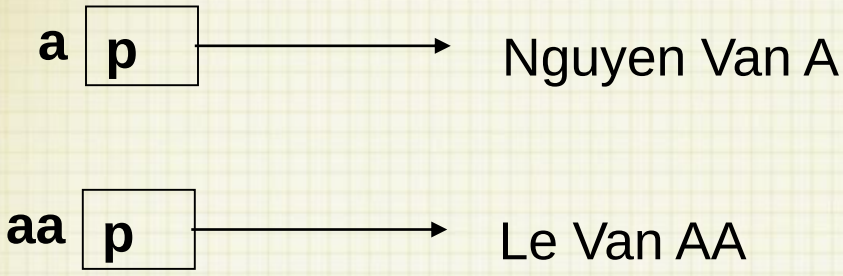
- Khi lớp đối tượng có nhu cầu cấp phát tài nguyên:
  - Việc khởi động đối tượng đòi hỏi phải có phương thức thiết lập sao chép để tránh hiện tượng các đối tượng chia sẻ tài nguyên dẫn đến một vùng tài nguyên bị giải phóng nhiều lần khi các đối tượng bị hủy bỏ.
- Khi thực hiện phép gán trên các đối tượng cùng kiểu, cơ chế gán mặc nhiên là gán từng thành phần →làm cho đối tượng bên trái của phép gán "bỏ rơi" tài nguyên cũ và chia sẻ tài nguyên với đối tượng ở vế phải.

Gán và khởi động

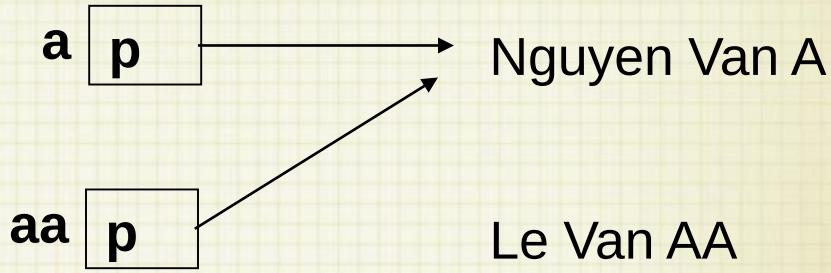
```
class String{
    char *p;
public:
    String(char *s = "") { p = strdup(s); }
    String(const String &s) { p = strdup(s.p); }
    ~String() { cout <<"delete"<<(void*)p<<"\n"; delete [] p; }
    void Output() const { cout << p; }
};
void main(){
    String a("Nguyen Van A");
    String b = a;           //Khoi dong
    String aa = "Le van AA";
    cout << "aa = "; aa.Output(); cout << "\n";
    aa = a;                 //Gan
    cout << "aa = "; aa.Output(); cout << "\n";
}
```

Gán và khởi động

Trước khi gán



Sau khi gán



❖ Thực hiện chương trình trên ta được kết xuất như sau:

aa = Le van AA

aa = Nguyen Van A

delete 0x0d36

delete 0x0d48

delete 0x0d36

Null pointer assignment

Gán và khởi động

- Lỗi sai trên được khắc phục bằng cách định nghĩa phép gán cho lớp String

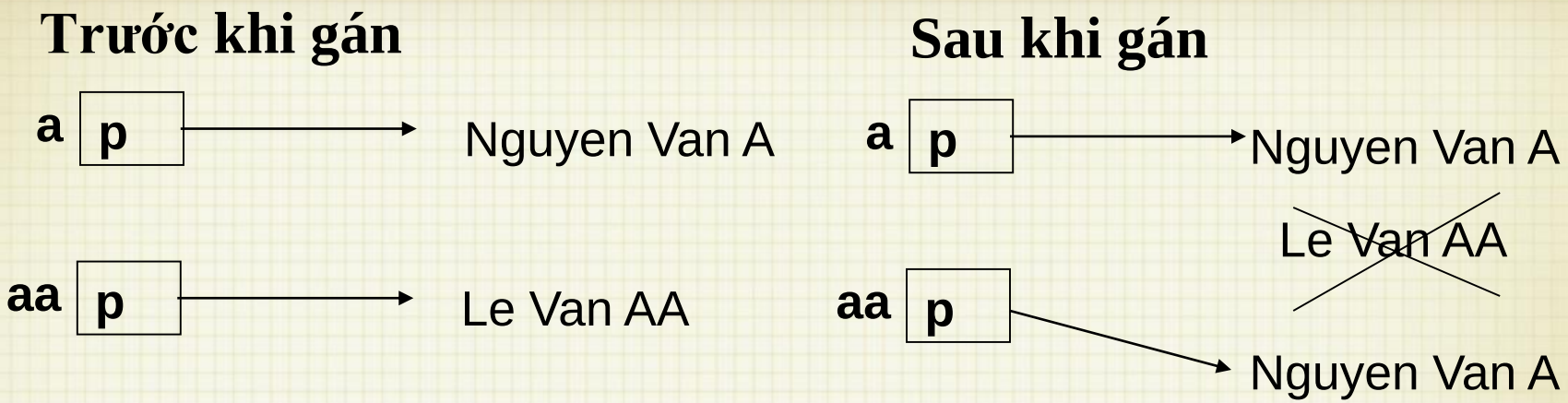
```
class String {
    char *p;
public:
    String(char *s = "") {p = strdup(s);}
    String(const String &s) {p = strdup(s.p);}
    ~String() {cout << "delete "<< (void *)p << "\n"; delete [] p;}
    String & operator = (const String &s);
    void Output() const {cout << p;}
};
```

Gán và khởi động

- Phép gán thực hiện hai thao tác chính là dọn dẹp tài nguyên cũ và sao chép mới.

```
String & String::operator = (const String &s) {
    if (this != &s)
    {
        delete [] p;
        p = strdup(s.p);
    }
    return *this;
}
```

Gán và khởi động



- Thực hiện chương trình trên ta được kết xuất như sau:

aa = La van AA

aa = Nguyen Van A

delete 0x0d5a

delete 0x0d48

delete 0x0d36

Phép toán << và >>

- << và >> là hai phép toán thao tác trên từng bit khi các toán hạng là số nguyên.
- C++ định nghĩa lại hai phép toán để dùng với các đối tượng thuộc lớp ostream và istream để thực hiện các thao tác xuất, nhập.
- Lớp ostream (dòng dữ liệu xuất) định nghĩa phép toán << áp dụng cho các kiểu dữ liệu cơ bản (số nguyên, số thực, char*,…)

Phép toán << và >>

- Khi định nghĩa hai phép toán trên, cần thể hiện ý nghĩa sau:

```
a >> b;            //bỏ a vào b
a << b;            //bỏ b vào a
cout << a << "\n";     // bỏ a và "\n" vào cout
cin >> a >> b; // bỏ cin vào a và b
```

Phép toán << và >>

- cout, cerr là các biến thuộc lớp ostream đại diện cho thiết bị xuất chuẩn (mặc nhiên là màn hình) và thiết bị báo lỗi chuẩn (luôn luôn là màn hình).

- cin là một đối tượng thuộc lớp istream đại diện cho thiết bị nhập chuẩn, mặc nhiên là bàn phím.

Phép toán << và >>

- Với khai báo của lớp ostream như trên ta có thể thực hiện phép toán << với toán hạng thứ nhất là một dòng dữ liệu xuất (cout, cerr, tập tin…), toán hạng thứ hai thuộc các kiểu cơ bản (nguyên, thực, char *, con trỏ…).
- Tương tự, ta có thể áp dụng phép toán >> với toán hạng thứ nhất thuộc lớp istream (ví dụ cin), toán hạng thứ hai là tham chiếu đến kiểu cơ bản hoặc con trỏ (nguyên, thực, char *).

Lớp ostream

```
class ostream : virtual public ios {
public:
    // Formatted insertion operations
    ostream & operator<< (signed char);
    ostream & operator<< (unsigned char);
    ostream & operator<< (int);
    ostream & operator<< (unsigned int);
    ostream & operator<< (long);
    ostream & operator<< (unsigned long);
    ostream & operator<< (float);
    ostream & operator<< (double);
    ostream & operator<< (const signed char *);
    ostream & operator<< (const unsigned char *);
    ostream & operator<< (void *);
    // ...
private:
    //data ...
};
```

Lớp istream

```
class istream : virtual public ios {
public:
    istream & getline(char  *, int, char = '\n');
    istream & operator>> (signed char  *);
    istream & operator>> (unsigned char  *);
    istream & operator>> (unsigned char  &);
    istream & operator>> (signed char  &);
    istream & operator>> (short  &);
    istream & operator>> (int  &);
    istream & operator>> (long  &);
    istream & operator>> (unsigned short  &);
    istream & operator>> (unsigned int  &);
    istream & operator>> (unsigned long  &);
    istream & operator>> (float  &);
    istream & operator>> (double  &);
private:
    // data...
};
```

Phép toán << và >>

- Để định nghĩa phép toán << theo nghĩa xuất ra dòng dữ liệu xuất cho kiểu dữ liệu đang định nghĩa:
  - Ta định nghĩa phép toán như hàm toàn cục với tham số thứ nhất là tham chiếu đến đối tượng thuộc lớp ostream
  - Kết quả trả về là tham chiếu đến chính ostream đó.
  - Toán hạng thứ hai thuộc lớp đang định nghĩa.

Phép toán << và >>

- Để định nghĩa phép toán >> theo nghĩa nhập từ dòng dữ liệu nhập cho kiểu dữ liệu đang định nghĩa:
  - Ta định nghĩa phép toán >> như hàm toàn cục với tham số thứ nhất là tham chiếu đến một đối tượng thuộc lớp istream
  - Kết quả trả về là tham chiếu đến chính istream đó.
  - Toán hạng thứ hai là tham chiếu đến đối tượng thuộc lớp đang định nghĩa.

Ví dụ phép toán << và >>

```
class PhanSo {
    long tu, mau;
    void UocLuoc();
public:
    PhanSo ( long t = 0, long m = 1) {Set(t,m);}
    void Set ( long t, long m);
    long LayTu() const { return tu;}
    long LayMau() const { return mau;}
    friend PhanSo operator + (PhanSo a, PhanSo b);
    friend PhanSo operator - (PhanSo a, PhanSo b);
    friend PhanSo operator * (PhanSo a, PhanSo b);
    friend PhanSo operator / (PhanSo a, PhanSo b);
    PhanSo operator -() const {return PhanSo(-tu,mau);}
    friend istream& operator >> (istream &is, PhanSo &p);
    friend ostream& operator << (ostream &os, PhanSo p);
};
```

Ví dụ phép toán << và >>

```
istream& operator >> (istream &is, PhanSo &p){
    is >> p.tu >> p.mau;
    while (!p.mau){
        cout << “Nhap lai mau so: ”;
        is >> p.mau;
    }
    p.UocLuoc();
    return is;
}
ostream& operator << (ostream &os, PhanSo p){
    os << p.tu;
    if (p.tu != 0 && p.mau != 1)
        os << "/" << p.mau;
    return os;
}
```

Ví dụ phép toán << và >>

```
void main(){
    PhanSo a, b;
    cout << “Nhap phan so a: ”; cin >> a;
    cout << “Nhap phan so b: ”; cin >> b;
    cout << a << " + " << b << " = " << a + b << "\n";
    cout << a << " - " << b << " = " << a - b << "\n";
    cout << a << " * " << b << " = " << a * b << "\n";
    cout << a << " / " << b << " = " << a / b << "\n";
}
```

Phép toán lấy phần tử mảng: [ ]

- Ta có thể định nghĩa phép toán [ ] để truy xuất phần tử của một đối tượng có ý nghĩa mảng.

```
class String {
    char *p;
public:
    String( char *s = "") { p = strdup(s); }
    String( const String &s) { p = strdup(s.p); }
    ~String() { delete [ ] p; }
    String & operator = ( const String &s);
    char & operator[ ] (int i) { return p[i]; }
    friend ostream& operator << (ostream &o, const String& s);
};
```

Phép toán lấy phần tử mảng: [ ]

- Sau khi định nghĩa như trên, ta có thể sử dụng đối tượng trả về ở cả hai vế của phép toán gán.

```
void main() {
    String a("Nguyen van A");
    cout << a[7] << "\n";       // a.operator[ ](7)
    a[7] = 'V';
    cout << a[7] << "\n";       // a.operator[ ](7)
    cout << a << "\n";
}
```

Phép toán [ ] cho đối tượng hằng

- Phép toán [ ] không hợp lệ với đối tượng hằng

```
void main() {
    String a("Nguyen van A");
    const String aa("Dai Hoc Tu Nhien");
    cout << a[7] << "\n";
    a[7] = 'V';
    cout << a[7] << "\n";
    cout << aa[4] << "\n";
    aa[4] = 'L';
    cout << aa[4] << "\n";
    cout << aa << "\n";
}
```

Phép toán [ ] cho đối tượng hằng

❖ Cách khắc phục?

```
class String {
    char *p;
    static char c;
public:
    String(char *s = "") {p = strdup(s);}
    String(const String &s) {p = strdup(s.p);}
    ~String() {delete [] p;}
    String & operator = (const String &s);
    char & operator[](int i) {return (i>=0 && i<strlen(p))?p[i]:c;}
    char operator[](int i) const {return p[i];}
};
char String::c = 'A';
```

Phép toán [ ] cho đối tượng hằng

```
void main() {
    String a("Nguyen van A");
    const String aa("Dai Hoc Tu Nhien");
    cout << a[7] << "\n";
    a[7] = 'V';
    cout << a[7] << "\n";
    cout << aa[4] << "\n";   // String::operator[](int) const : Ok
    aa[4] = 'L';             // Bao Loi: Khong the la lvalue
    cout << aa[4] << "\n";   // String::operator[](int) const : Ok
    cout << aa << "\n";
}
```

Phép toán gọi hàm: ()

- Phép toán [ ] chỉ có thể có một tham số, vì vậy dùng phép toán trên không thuận tiện khi ta muốn lấy phần tử của một ma trận hai chiều.
- Lớp ma trận sau đây định nghĩa phép toán () với hai tham số, nhờ vậy ta có thể truy xuất phần tử của ma trận thông qua số dòng và số cột.

Phép toán gọi hàm: ()

```
class MATRIX{
    float **M;
    int row, col;
public:
    MATRIX (int, int);
    ~MATRIX();
    float& operator() (int, int);
};
float MATRIX::operator() (int i, int j){
    return M[i][j];
}
```

Phép toán gọi hàm: ()

```
MATRIX::MATRIX ( int r, int c){
    M = new float* [r];
    for ( int i=0; i<r; i++)
        M[i] = new float[c];
    row = r;
    col = c;
}
~MATRIX::MATRIX(){
    for ( int i=0; i<col; i++)
        delete [ ] M[i];
    delete [ ] M;
}
```

Phép toán gọi hàm: ()

```
void main(){
    cout<<“Cho ma tran 2x3\n”;
    MATRIX a(2, 3);
    int i, j;
    for (i = 0; i<2; i++)
          for (j = 0; j<3; j++)
                cin>>a(i,j);
     for (i = 0; i<2; i++){
          for (j = 0; j<3; j++)
                cout<<a(i,j)<<“ ”;
          cout<<endl;
    }
}
```

Phép toán tăng và giảm: ++ và --

- ++ là phép toán một ngôi có vai trò tăng giá trị một đối tượng lên giá trị kế tiếp. Tương tự -- giảm giá trị một đối tượng xuống giá trị trước đó.
- ++ và -- chỉ áp dụng cho các kiểu dữ liệu đếm được, nghĩa là mỗi giá trị của đối tượng đều có giá trị kế tiếp hoặc giá trị trước đó.
- ++ và -- có thể được dùng theo hai cách, tiếp đầu ngữ hoặc tiếp vị ngữ.

Phép toán tăng và giảm: ++ và --

- Khi dùng như tiếp đầu ngữ, ++a có hai vai trò:
  - Tăng a lên giá trị kế tiếp.
  - Trả về tham chiếu đến chính a.
- Khi dùng như tiếp vị ngữ, a++ có hai vai trò:
  - Tăng a lên giá trị kế tiếp.
  - Trả về giá trị bằng với a trước khi tăng.

Phép toán tăng và giảm: ++ và --

```
class ThoiDiem{
    long tsgiay;
    static bool HopLe(int g, int p, int gy);
public:
    ThoiDiem(int g = 0, int p = 0, int gy = 0);
    void Set(int g, int p, int gy);
    int LayGio() const {return tsgiay / 3600;}
    int LayPhut() const {return (tsgiay%3600)/60;}
    int LayGiay() const {return tsgiay % 60;}
    void Tang();
    void Giam();
    ThoiDiem &operator ++();
};
```

Phép toán tăng và giảm: ++ và --

```
void ThoiDiem::Tang(){
    tsgiay = ++tsgiay%SOGIAY_NGAY;
}
void ThoiDiem::Giam()
{
    if (--tsgiay < 0) tsgiay = SOGIAY_NGAY-1;
}
ThoiDiem &ThoiDiem::operator ++() {
    Tang();
    return *this;
}
```

Phép toán tăng và giảm: ++ và --

```
void main()
{
    ThoiDiem t(23,59,59),t1,t2;
    cout << "t = " << t << "\n";
    t1 = ++t; // t.operator ++();
          // t = 0:00:00, t1 = 0:00:00
    cout << "t = " << t << "\tt1 = " << t1 << "\n";
    t1 = t++; // t.operator ++();
          // t = 0:00:01, t1 = 0:00:00
    cout << "t = " << t << "\tt1 = " << t1 << "\n";
}
```

Phép toán tăng và giảm: ++ và --

- Để có thể có phép toán ++ và -- hoạt động khác nhau cho hai cách dùng (++a và a++) ta cần định nghĩa hai phiên bản ứng với hai cách dùng kể trên.
- Khi đó, phiên bản tiếp vị ngữ có thêm một tham số giả để phân biệt.

```
ThoiDiem &operator ++();
ThoiDiem operator ++(int);
```

Phép toán tăng và giảm: ++ và --

```
void ThoiDiem::Tang() {
    tsgiay = ++tsgiay%SOGIAY_NGAY;
}
void ThoiDiem::Giam() {
    if (--tsgiay < 0) tsgiay = SOGIAY_NGAY-1;
}
ThoiDiem &ThoiDiem::operator ++()  {
    Tang();
    return *this;
}
ThoiDiem ThoiDiem::operator ++(int) {
    ThoiDiem t = *this;
    Tang();
    return t;
}
```

Phép toán tăng và giảm: ++ và --

```
void main()
{
    ThoiDiem t(23,59,59),t1,t2;
    cout << "t = " << t << "\n";
    t1 = ++t; // t.operator ++();
          // t = 0:00:00, t1 = 0:00:00
    cout << "t = " << t << "\tt1 = " << t1 << "\n";
    t1 = t++; // t.operator ++(int);
          // t = 0:00:01, t1 = 0:00:00
    cout << "t = " << t << "\tt1 = " << t1 << "\n";
}
```

The end