Chương 3- Con trỏ và tệp tin (phần 1)

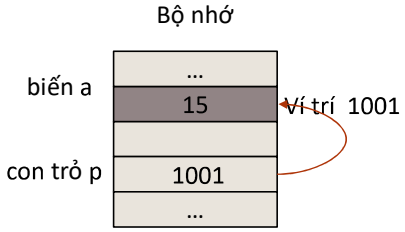
Bộ môn KTHT và mạng máy tính

Nội dung

- 3.1 Giới thiệu về con trỏ (Pointer)
- 3.2 Các trường hợp sử dụng con trỏ

3.1 Con trỏ (Pointer)

- Con trỏ là gì?



Biến a có giá trị là 15, và đặt tại vị trí 1001 trong bộ nhớ

Con trỏ p có giá trị là 1001 và chỉ đến vị trí của biến a

Tạo con trỏ trong chương trình qua 2 bước như

- B1. Khai báo biến trỏ
  - Cú pháp: `<kiểu dữ liệu> *<biến trỏ>;`
- B2. Gán địa chỉ của biến dữ liệu cho biến trỏ
  - Cú pháp: `<biến trỏ> = &<biến dữ liệu>`

Chú ý

- Trước khi sử dụng con trỏ, PHẢI
- Trong trường hợp chưa xác định địa chỉ, PHẢI
- Con trỏ thuộc kiểu dữ liệu nào PHẢI

3.2) Các trường hợp sử dụng con trỏ

- A. Truy nhập dữ liệu đơn qua con trỏ
- B. Truy nhập dữ liệu mảng 1 chiều qua con trỏ
- C. Truy nhập đối tượng qua con trỏ
- D. Cấp phát bộ nhớ động
- E. Con trỏ làm tham số của hàm
- F. Hàm trả về con trỏ
- G. Mảng với phần tử là con trỏ

A) Truy nhập dữ liệu đơn qua con trỏ

- Cú pháp: ***<con trỏ>**

```
float x = 2.0;
float* p = &x; // p trỏ đến x
```

- Như vậy, nếu p trỏ đến x thì

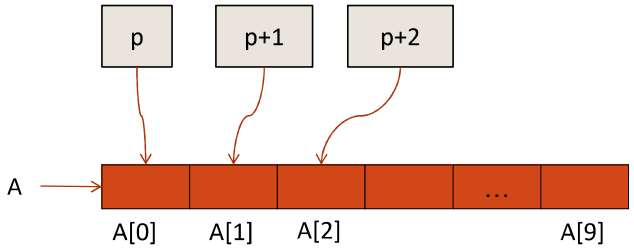
B) Truy cập dữ liệu mảng qua con trỏ

- Đầu tiên, cần gán mảng cho biến trỏ với cú pháp:
  <biến trỏ> = <tên mảng>;
- Sau đó, sử dụng con trỏ

```
p[0] =  0; //   A[0] = 0
p[1] =  1; //   A[1] = 1
cout << p[0] << " " << p[1]; // in A[0] A[1]
```

Tăng giảm biến con trỏ

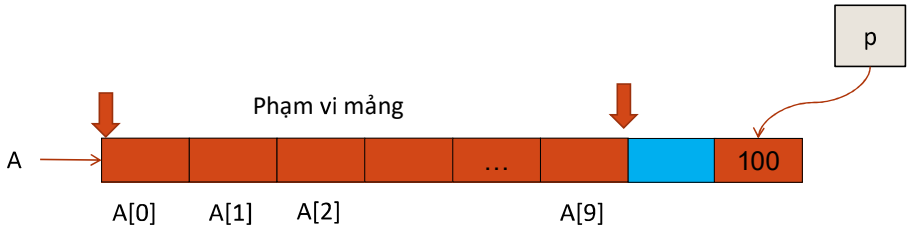
- Có thể thực hiện phép toán +, ++, -, -- với biến trỏ trong khi truy nhập mảng để
- Ví dụ



Chú ý

- Tránh để con trỏ vượt ra ngoài

```
int A[10];  int *p = A;
p += 11; / / con trỏ đã vượt ra khỏi mảng A
*p = 100 ; // có thể gây lỗi
```



C) Truy nhập đối tượng qua con trỏ

- Đầu tiên, cần gán địa chỉ đối tượng cho biến trỏ theo cú pháp:

  <class> *<biến trỏ> = &<đối tượng>;

- Sau đó, sử dụng con trỏ và phép toán **->** để

```
cout << p->getElement(0,0);
```

- Mặt khác cũng có thể sử dụng *<biến trỏ> như đối tượng tương ứng

```
*p.setElement(1,1,2);
cout << *p.setElement(1,1);
Matrix m2 = *p;
```

Con trỏ **this** trong lập trình hướng đối tượng

- **this** là con trỏ đến
- Con trỏ **this** chỉ
- Ví dụ về con trỏ **this**

```
Matrix::setElement(int i, int j, int e){
    this->a[i][j] = e;
}

Matrix Matrix::operator++(){
   for(int i=0;i<row;i++)
      for(int j=0;j<col;++j)
          ++a[i][j];
   return *this;  // trả lại chính đối tượng gọi hàm ++
}
```

Chương trình minh họa

D) Cấp phát bộ nhớ động

- Cấp phát bộ nhớ là khởi tạo vùng bộ nhớ lưu trữ dữ liệu trong chương trình
- Có 2 dạng cấp phát bộ nhớ
- Cấp phát tĩnh được thực hiện
- Cấp phát động được thực hiện

Cấp bộ nhớ tĩnh

- Cấp phát cho các biến dữ liệu được khai báo
- Thực hiện **trước khi** chương trình bắt đầu chạy
- Bộ nhớ được cấp thuộc vùng GLOBAL hoặc STACK
- Tự động thu hồi bộ nhớ sau khi biến dữ liệu ra khỏi phạm vi sử dụng

Cấp bộ nhớ động

- Cấp phát cho **biến trỏ** thông qua câu lệnh
- Thực hiện **trong khi** chương trình đang chạy
- Bộ nhớ được cấp thuộc vùng HEAP
- Bộ nhớ đã cấp tồn tại cho đến khi chương trình kết thúc
- Phải thu hồi thông qua câu lệnh

Cách cấp phát bộ nhớ động cho 1 biến đơn

- Đầu tiên, cần khai báo biến trỏ
- Tiếp theo, sử dụng phép toán **new** theo cú pháp sau

```
<biến trỏ> = new  <Kiểu dữ liệu>;
```

- Sau khi cấp phát động, có thể sử dụng con trỏ để truy nhập dữ liệu vừa được cấp phát

Sử dụng sau khi cấp phát động

- Truy nhập biến dữ liệu thông qua biến trỏ với phép toán *****
- Truy nhập đối tượng thông qua biến trỏ với phép toán **->**

Cấp phát động cho 1 mảng (1 chiều)

- Sử dung phép toán **new** theo cú pháp sau

```
<biến trỏ> = new  <Kiểu dữ liệu>[size];
```

```
int m,n;
cin >> m >> n; // nhập m, n
```

- Khi cấp phát 1 mảng các đối tượng,

Truy nhập mảng động

- Sử dụng biến trỏ và chỉ số mảng

```
for(int i=0;i<m;i++)
{
    cin >> p[i];
}
```

Thu hồi bộ nhớ cấp phát động

- Thu hồi biến đơn với cú pháp

```
delete  <biến trỏ>;
```

- Thu hồi biến mảng với cú pháp

```
delete [] <biến trỏ>;
```

- Sau khi thu hồi, các biến được cấp phát động
- Sử dụng mảng động linh hoạt hơn,

E) Con trỏ làm tham số của hàm

- Thông qua con trỏ, có thể truy nhập đến tham số thực sự của hàm
- Ví dụ

```
void printMatrixPointer(const Matrix* pm){
    int m = pm->getRow();
    int n = pm->getCol();
    for(int i=0;i<m;++i)
    {
        for(int j=0;j<n;j++)
        {
            cout << pm->getElement(i,j) << " ";
        }
        cout << endl;
    }
}
```

F) Hàm trả về kết quả là con trỏ

- Sử dụng con trỏ làm kết quả của hàm

```
Matrix* initMatrixPointer() { // n x n
    int m,n;
    cin >> m >> n;
    // Cap phat bo nho dong
    Matrix* pm = new Matrix(m,n);
    for(int i=0;i<m;i++) // nhap du lieu cho ma tran
        for(int j=0;j<n;j++)
        {
            int t;
            cin >> t;
            pm->setElement(i,j,t);
        }
    return pm; // tra lai con tro Matrix
}
```

G) Mảng với phần tử là con trỏ

- Khi cần cấp phát động 1 mảng nhiều đối tượng, mỗi đối tượng lại **sử dụng 1 hàm tạo có tham số nhất định**, ta áp dụng **mảng với phần tử là con trỏ**

- Đầu tiên, khai báo 1 biến trỏ kép

```
Matrix** mp;
```

- Tiếp theo, cấp phát động mảng 1 chiều, mỗi phần tử là 1 con trỏ

VD:

```
mp = new Matrix* [n]; // n phần tử
```

- Với mỗi phần tử, cấp phát động 1 đối tượng và khởi tạo bằng 1 hàm tạo nhất định

- Thu hồi bộ nhớ khi không còn sử dụng

```
for(int i=0;i<n;i++)
```

Chương trình minh họa