Chủ đề: Con trỏ trong C++

I. Khái niệm về con trỏ

- Trong C++, con trỏ là 1 biến đặc biệt, có vai trò lưu trữ địa chỉ của 1 biến khác trong bộ nhớ, cùng kiểu với nó
- Con trỏ và biến mà con trỏ đó lưu trữ địa chỉ phải có cùng kiểu dữ liệu
- Cú pháp khai báo:
  <kiểu dữ liệu> *<tên con trỏ>
  VD: int *a,;
  Double *b;
  SINHVIEN *x; // đây là con trỏ đối tượng, sẽ học ở phần môn lập trình hướng đối tượng
- Mỗi con trỏ chỉ trỏ được tới 1 biến, mỗi biến có thể được trỏ bằng nhiều con trỏ
- Con trỏ không chỉ vào biến hay con trỏ không lưu trữ đến địa chỉ nào, sẽ nhận giá trị NULL

II. Một số thao tác cơ bản trên con trỏ

1. Lấy địa chỉ của biến đặt vào con trỏ

Giả sử có biến con trỏ x lưu địa chỉ của biến a (2 biến cùng mang kiểu dữ liệu int). Cú pháp lấy địa chỉ của biến a lưu vào con trỏ x:

$$x = \&a$$

Khi đó ta nói: Con trỏ x trỏ tới biến a
Toán tử & có vai trò lấy địa chỉ của 1 biến trong bộ nhớ

2. Gán con trỏ cho con trỏ

Khi gán con trỏ này và con trỏ kia, 2 con trỏ sẽ lưu trữ cùng 1 địa chỉ, tức là cùng trỏ vào 1 biến
VD: int *p, *q;
p = &a;
q = p;

➔ p và q là 2 con trỏ cùng trỏ vào biến a

3. Thao tác gián tiếp giữa con trỏ và biến

a) Giải tham chiếu

- Khi khai báo con trỏ, dấu * sẽ có vai trò xác định xem đó là con trỏ hay biến thông thường. Tuy nhiên, trong quá trình thao tác, dấu * còn được gọi là giải tham chiếu. Khi sử dụng biến con trỏ có giải tham chiếu, nó sẽ trả về giá trị của biến mà con trỏ đó chỉ tới

VD: int *p;
Int a = 10;
p = &a;
cout << *p;
return 0;

➔ Khi đó ở dòng cout << *p, dấu * có vai trò là giải tham chiếu, và sẽ in ra màn hình giá trị của biến mà con trỏ p trỏ tới, là biến a. Mà biến a có giá trị là 10 -> In ra màn hình là 10

b) Thao tác với dữ liệu thông qua con trỏ:

Qua việc sử dụng giải tham chiếu, chúng ta có thể thao tác với các dữ liệu thông qua con trỏ

VD: Cho chương trình sau :

```
#include <iostream>

using namespace std;


int main(){
        int *p;
        int a;
```

```
    p = &a;
    cin >> *p;
    cout << *p;
    return 0;
}
```

Ở chương trình này, chúng ta sử dụng con trỏ p để gián tiếp nhập dữ liệu cho biến a, thông qua sử dụng giải tham chiếu

4. Kiểu dữ liệu con trỏ

- Kiểu dữ liệu con trỏ là kiểu dữ liệu đặc biệt. Khi khai báo một biến mang kiểu dữ liệu con trỏ, nó sẽ mặc định là 1 con trỏ mà không cần sử dụng toán tử *
- Cú pháp khai báo:

<tên kiểu dữ liệu>* < tên biến>

(Dấu * được viết cạnh tên của kiểu dữ liệu)
VD: int* a;
double* t;

➔ Khi đó 2 biến a và t sẽ là con trỏ, và khi thao tác chúng ta thao tác tương tự như những con trỏ thường khác

Bài tập áp dụng:

Bài 1: Nhập 2 số nguyên dương a và b từ bàn phím, sử dụng 2 con trỏ p,q lưu địa chỉ lần lượt của a và b để thao tác. Tính tổng 2 số đó và in ra màn hình (có thể sử dụng kiểu dữ liệu con trỏ hoặc không)

Bài làm:

```
#include <iostream>
using namespace std;
```

```
int main(){
	int *p;
	int *q;
	int a,b;
	p = &a;
	q = &b;
	cin >> *p >> *q;
	cout << *p + *q;
	return 0;
}
```

Bài 2: Nhập 3 số nguyên dương a, b và c từ bàn phím, sử dụng 3 con trỏ x,y,z lần lượt lưu địa chỉ của a,b,c để thao tác. Tính trung bình cộng của 3 số đó, in ra màn hình số lớn nhất và nhỏ nhất trong 3 số đó ra màn hình

Bài 3: Nhập 3 số nguyên dương a,b và c từ bàn phím, sử dụng 3 con trỏ x,y,z lần lượt lưu địa chỉ của a,b,c để thao tác. Cho biết a,b,c có phải 3 cạnh của 1 tam giác vuông hay không ?. In ra màn hình YES nếu đúng, NO nếu sai

Bài 4: Nhập 3 số nguyên dương a,b,c từ bàn phím là 3 cạnh của 1 tam giác, sử dụng 3 con trỏ x,y,z lần lượt lưu địa chỉ của a,b,c để thao tác. Tính diện tích của tam giác đó theo công thức Hê-rông (kết quả làm tròn đến chữ số thập phân thứ 2).

Lưu ý: Phải xét điều kiện 3 cạnh đó có tạo nên 1 tam giác hay không. Nếu không tạo thành 1 tam giác, yêu cầu người dùng nhập lại.

Bài 5: Nhập 4 số nguyên dương a,b,c từ bàn phím, sử dụng 3 con trỏ x,y,z lần lượt lưu địa chỉ của a,b,c để thao tác.. Sắp xếp 3 biến đó theo thứ tự tăng dần và in ra màn hình kết quả, mỗi số cách nhau bởi 1 dấu cách

III. Con trỏ và mảng

1. Đặt vấn đề

- Trước đây ở bài mảng, chúng ta cấp phát 1 mảng (hay còn gọi là cấp phát tĩnh), với 1 khối lượng bộ nhớ nhất định trong mảng đó và thực hiện các thao tác và công việc với các phần tử
  VD: int a[100];
  Việc cấp phát này sẽ dễ dàng hơn, tuy nhiên sẽ gặp phải 1 vấn đề: Trong quá trình thao tác, khối lượng cấp phát cho mảng đó sẽ cố định và không thể thay đổi(ví dụ cấp phát 100 ô nhớ thì trong quá trình thao tác, chúng ta không thể xin cấp phát thêm ô nhớ, mà phải tạo hẳn 1 mảng khác, có kích cỡ lớn hơn để phục vụ lợi ích của mình). Điều này sẽ gây bất tiện và thiếu linh hoạt trong quản lý bộ nhớ và chương trình

- Việc sử dụng mảng và con trỏ(hay còn gọi là cấp phát động) sẽ khiến người lập trình chủ động trong việc cấp phát bộ nhớ trong chương trình và việc quản lý bộ nhớ sẽ tối ưu hơn. Chúng ta có thể cấp phát lại, thêm, bớt "thủ công", dựa trên nhu cầu của chúng ta

2. Cấp phát

- Để cấp phát mảng bằng con trỏ, chúng ta sử dụng từ khóa new
  + Cấp phát 1 ô nhớ (cấp phát biến thông thường, cách này sử dụng tương đối ít)

<Tên con trỏ> = new <Kiểu của con trỏ>;

  + Cấp phát một khối nhớ (nhiều ô nhớ, memory block)

<Tên con trỏ> = new <Kiểu của con trỏ>[n];

  + Khi cấp phát một khối nhớ (hay gọi đơn giản là mảng), địa chỉ của con trỏ sẽ là địa chỉ của phần tử đầu tiên trong mảng đó

  Ngoài việc cấp phát bộ nhớ bằng từ khóa new, chúng ta còn có thể cấp phát cho các vùng nhớ thông qua từ khóa calloc(), malloc(), realloc()

- Hàm calloc(): thuộc thư viện alloc.h
  Cú pháp:

<con trỏ> = (<Kiểu dữ liệu con trỏ>*) calloc (<n>, <size>);

+ n là số ô nhớ cần cấp phát
+ size: kích thước của 1 ô nhớ (tính bằng byte)

- Hàm malloc: Có chức năng tương tự thằng calloc. Cú pháp
  <con trỏ> = (<Kiểu dữ liệu con trỏ>*) malloc (<size>);
  <size> là kích thước của 1 ô nhớ, sau khi câu lệnh thực thi người dùng sẽ được cấp phát <size> x sizeof (Kiểu dữ liệu) (byte) ô nhớ

- Hàm realloc: Dùng để tái cấp phát cho bộ nhớ
  Cú pháp:

  <con trỏ> = (<Kiểu dữ liệu con trỏ>*) realloc (<con trỏ>,<new size>);

- Sau khi cấp phát động, bộ nhớ sẽ cung cấp cho người dùng 1 dãy các ô nhớ liên tiếp, chúng ta vẫn có thể truy cập các phần tử trong mảng theo cách truy cập ở mảng tĩnh hoặc cách truy cập ở mảng động
- Cách truy cập ở mảng tĩnh

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]

- Cách truy cập ở mảng động

*a	*(a + 1)	*(a + 2)	*(a + 3)	*(a + 4)	*(a + 5)	*(a + 6)	*(a + 7)

VD: Cấp phát động 1 mảng a gồm có n phần tử. Sao chép các giá trị chẵn của mảng đặt vào cuối mảng.

Bài làm:

```
#include <iostream>
using namespace std;

int main(){
        int *a;
        int n = 5;
```

```
        a = new int[n];
        for (int i = 0; i < 5; i++)
                cin >> *(a + i);
        for (int i = 0; i < 5; i++){
                if (*(a + i) % 2 == 0){
                        a = (int*) realloc(a, (n + 1)*sizeof(int));
                        *(a + n) = *(a + i);
                        n++;
                }
        }
        for (int i = 0; i < n; i++)
                cout << *(a + i) << " ";
        return 0;
}
```

3. Giải phóng con trỏ

- Nếu bộ nhớ được cấp phát bằng từ khóa new, ta sử dụng từ khóa delete
  Cú pháp

delete[] <tên con trỏ>

- Nếu bộ nhớ được cấp phát bởi malloc, calloc, ta dùng free
  Cú pháp

free(<Tên con trỏ>)

Bài tập: Sử dụng con trỏ và mảng cấp phát động để giải quyết các bài toán sau, mỗi gạch đầu dòng dùng tối thiểu 1 hàm, hàm main thực hiện bằng cách gọi các hàm đã cài đặt ở các câu trước

Bài 1:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Tính tổng các phần tử trong mảng a
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và tính tổng các phần tử trong mảng y đó. In mảng kết quả ra màn hình

Bài 2:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Kiểm tra xem mảng có phải mảng đối xứng không. Biết mảng đối xứng là mảng có các phần tử đối xứng nhau qua 2 bên bằng nhau.
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và kiểm tra mảng y đó có phải mảng đối xứng không ? . In kết quả ra màn hình (In “DUNG” nếu là mảng đối xứng, in “SAI” nếu không phải)

Bài 3:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Thay các phần tử có giá trị âm trong mảng bằng số đối của nó
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và thực hiện thay các phần tử có giá trị âm trong mảng bằng số đối của nó. In mảng kết quả ra màn hình

Bài 4:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Tính tổng các phần tử cực tiểu trong mảng a
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và tính tổng các phần tử cực tiểu trong mảng y đó. In mảng kết quả ra màn hình

Bài 5:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- In ra các số nguyên tố trong mảng a.
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và in ra màn hình các phần tử là số nguyên tố trong y.

Bài 6:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- In ra màn hình các phần tử trong mảng a, sao cho mỗi phần tử chỉ xuất hiện
1 lần
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và in ra màn hình các phần tử của y, sao cho mỗi phần tử chỉ xuất hiện 1 lần.

Bài 7:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Tính tổng các số chính phương trong mảng a
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử và in ra màn hình tổng các số chính phương trong mảng y

Bài 8:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Xóa các phần tử âm trong mảng
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử, sau đó xóa các phần tử có giá trị âm trong mảng. In mảng kết quả ra màn hình

Bài 9:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Sắp xếp các phần tử trong mảng theo thứ tự giảm dần
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử,

sau đó sắp xếp các phần tử trong mảng theo thứ tự giảm dần. In mảng kết quả ra màn hình

Bài 10:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Xóa các phần tử có giá trị dương trong mảng
- Sắp xếp các phần tử có giá trị không quá -5 theo thứ tự tăng dần (giữ nguyên vị trí của các phần tử có giá trị lớn hơn -5)
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử, sau đó xóa hết các phần tử có giá trị dương trong mảng, và xếp các phần tử có giá trị không quá -5 theo thứ tự tăng dần (giữ nguyên vị trí của các phần tử có giá trị lớn hơn -5). In mảng kết quả ra màn hình

Bài 11:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử số nguyên
- Tính trung bình cộng của các phần tử trong mảng
- Nhập 1 số nguyên dương n. Chèn n trước các phần tử có giá trị âm trong mảng
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có x phần tử,sau đó tính trung bình cộng các phần tử trong mảng, và sau đó nhập 1 số nguyên n, chèn n trước các phần tử có giá trị âm trong mảng. In mảng kết quả ra màn hình

Bài 12:

- Nhập vào từ bàn phím một số nguyên n và 1 mảng a có n phần tử là các chuỗi ký tự (string)
- Sắp xếp các chuỗi ký tự theo thứ tự từ điểm
- Hàm main() thực hiện nhập 1 số nguyên dương x và 1 mảng y có n phần tử là các chuỗi ký tự, sau đó sắp xếp các phần tử theo thứ tự từ điển. In kết quả ra màn hình

Bài 13:

- Nhập vào từ bàn phím một chuỗi ký tự string, có dấu cách để ngăn cách giữa các từ
- Sắp xếp các từ trong chuỗi theo thứ tự từ điển
- Hàm main() thực hiện nhập 1 chuỗi ký tự string, có dấu cách để ngăn cách giữa các từ. Thực hiện sắp xếp các từ trong chuỗi theo thứ tự từ điển. In chuỗi kết quả ra màn hình