C2_New, delete, Biến tham chiếu, hàm overloading…

Ngày học	@September 6, 2022
Property	Bai giang LTHDT - Chuong 2.pdf
Property 1	bai tap chuong 2.pdf

1. Toán tử new để cấp phát bộ nhớ:

```
biến con trỏ = new Tên kiểu;
```

- Tên kiểu là kiểu dữ liệu của biến con trỏ

VD:

```
int *p;
p = new int; //cấp phát vùng nhớ cho biến con trỏ p
```

Hoặc:

```
int *p = new int;
```

- Để cấp phát bộ nhớ cho mảng một chiều:

```
Biến con trỏ = new kiểu [n];
```

Trong đó ***n là số nguyên dương xác định số phần tử của mảng***.

VD:

```
int *a = new int [100];
```

- Khi sử dụng toán tử new để cấp phát bộ nhớ, **nếu không đủ bộ nhớ để cấp phát**, new **sẽ trả lại giá trị NULL cho con trỏ**.

  (Mảng được chuyển đổi ngầm định thành con trỏ khi được chuyển đến 1 hàm, đó là lí do tại sao ***thay đổi mảng trong hàm sẽ thay đổi mảng thực tế được truyền vào***.)

Mảng động: (đề yêu cầu mảng động chứa các phần tử)

Khi khai báo thuộc tính vùng private:

```
private:
  int size;
  int *A;
```

Ở hàm nhập:

```
cout << "\tNhap kich thuoc : ";
  cin >> size;
  A = new int[size];
```

Size của mảng là số nguyên cố định

Tạo hằng toàn cục (tùy chọn giá trị(

```
#define n 5
```

Khi báo thuộc tính ở vùng Private:

```
private:
  int A[n];
```

2. Toán tử delete: giải phóng bộ nhớ đã cấp phát

```
delete con trỏ;
(con trỏ = nullptr;)

// Vd:
int *a = new int [n];
int *b = new int [n];
delete a, b;
```

---

3. Biến tham chiếu

Biến tham chiếu là một tên khác (***bí danh) của biến đã định nghĩa trước đó***

```
Kiểu &Biến tham chiếu = Biến;
```

Biến tham chiếu có đặc điểm là nó được dùng ***làm bí danh cho một biến (kiểu giá trị)*** nào đó và ***sử dụng vùng nhớ của biến này***

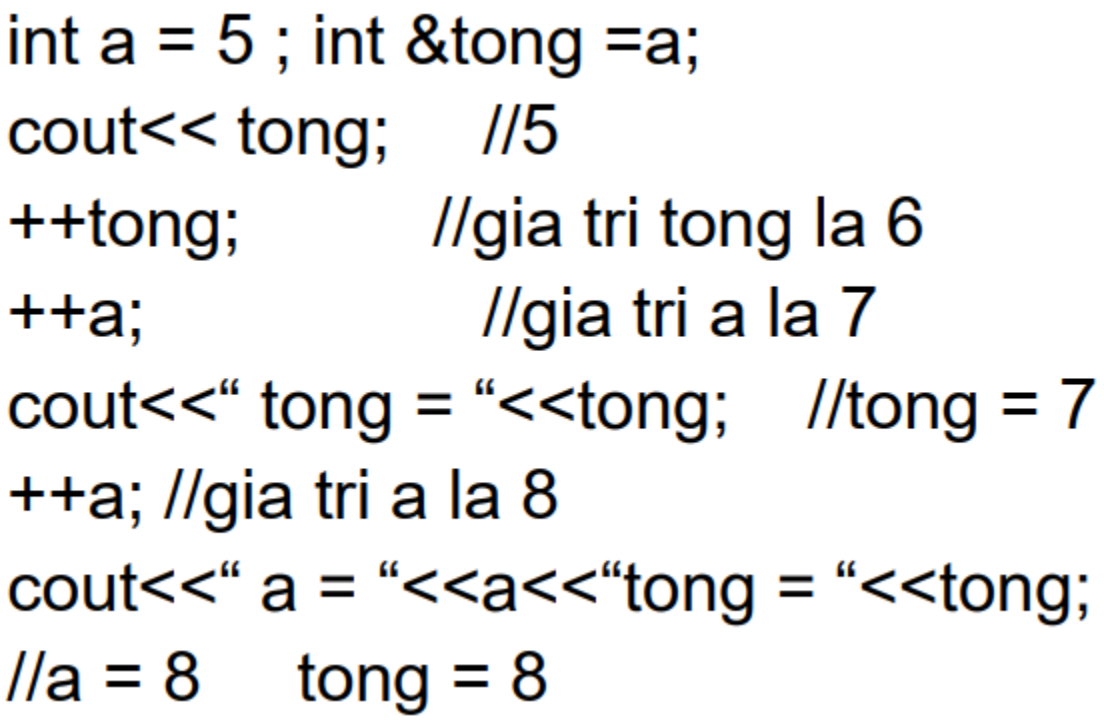
Biến tham chiếu **không có vùng nhớ riêng** mà **dùng chung vùng nhớ với biến mà nó tham chiếu**

Với câu lệnh:

```
int a;
int &tong=a;
//tong bay gio tro thanh bi danh cua a
```

> thì ***tong*** là ***bí danh của biến a*** và ***biến tong dùng chung vùng nhớ của biến a***. Lúc này, trong mọi câu lệnh, viết ***a hay viết tong đều có ý nghĩa như nhau***, vì **đều truy nhập đến cùng một vùng nhớ**. Mọi sự **thay đổi đối với biến tong đều ảnh hưởng đối với biến a và ngược lại**

VD:



---

4. Hằng tham chiếu:

```
const Kiểu dữ liệu &Biến = Biến/Hằng;
```

VD:

`int n = 10;`

`const int &m = n;`

`const int &p = 123;`

Biến tham chiếu và ***hằng tham chiếu*** khác nhau ở chỗ: **không cho phép *dùng hằng tham chiếu để làm thay đổi giá trị của vùng nhớ mà nó tham chiếu.***

Thường được dùng để sử dụng làm tham số của hàm để **cho phép sử dụng giá trị** của các tham số trong lời gọi hàm, nhưng **tránh làm thay đổi** giá trị tham số

---

5. Truyền tham số cho hàm theo tham chiếu:

Hàm này sẽ **thao tác trực tiếp trên vùng nhớ của các tham số**, do đó dễ dàng **thay đổi giá trị các tham số** khi cần.

Truyền tham chiếu (&biến tham chiếu): thao tác (tham chiếu) trực tiếp trên vùng nhớ của biến (do đó khi ra khỏi hàm, biến tồn tại và có thể được thay đổi)

```
int f(int &a, int b){
++a;
++b;
return a+b;
}
int main(){
int m = 3, n = 5;
cout << "\n f = " << f(m,n); // f =10
cout << "\n m = " << m; // m = 4
cout << "\n n = " << n; // n = 5
return 0;
}
```

```
int f(int &a, int &b){
  ++a;
  ++b;
  return a+b;
}

int main(){
  int m = 3, n = 5;
  cout << "\n f = " << f(m,n); // f = 10
  cout << "\n m = " << m; // m = 4
  cout << "\n n = " << n; // n = 6
  return 0;
}
```

```
int f(int a, int b){
  ++a;
  ++b;
  return a+b;
}
int main(){
  int m = 3, n = 5;
  cout << "\n f = " << f(m,n); // f = 10
  cout << "\n m = " << m; // m = 3
  cout << "\n n = " << n; // n = 5
  return 0;
}
```

```
float Gpt (float a, float b, float c, float &x1, float &x2){
    float delta = b*b-(4*a*c);
  if(delta > 0){
    x1 = (-b + sqrt(delta))/(2*a);
    x2 = (-b - sqrt(delta))/(2*a);
    return 2;
  }else if(delta == 0){
    x1 = x2 = -b/2*a;
        return 1;
    }
  else{
    x1 = x2 = 0;
      return 0;
    }
}

int main(){
    float a, b, c, x1, x2;
    cout << "Nhap he so a, b, c: ";
    cin >> a >> b >> c;
    cout << "\nKET QUA GIAI PHUONG TRINH BAC NHAT ax + b = 0: x = ";
    if(a == 0 && b == 0) cout << "Phuong trinh co vo so nghiem\n";
    else if(a == 0 && b != 0) cout << "Phuong trinh co vo nghiem\n";
    else if(a != 0) cout << "x = " << Gpt(a, b);
}
```

6. Hàm trả về giá trị tham chiếu:

```
Kiểu &Tên hàm(...){
//thân hàm
return <biến phạm vi toàn cục>;
}
```

Biểu thức được trả lại ***trong câu lệnh return*** phải là tên của một ***biến xác định từ bên ngoài hàm***, bởi vì khi đó mới có thể sử dụng được giá trị của hàm.

Khi **giá trị trả về của hàm là tham chiếu**, ta có thể gặp câu lệnh gán hơi khác thường, trong đó **vế trái là một lời gọi hàm chứ không phải là tên của một biến.** Nói cách khác, **vế trái của lệnh gán** có thể là **lời gọi đến một hàm có giá trị trả về là một tham chiếu**.

VD:

```
#include <iostream>
using namespace std;
int z; //z la bien toan cuc
int &f()//ham tra ve mot bi danh cua bien toan cuc z
{ return z;
}
int main(){
z = 10;
cout<<"\n f() = "<< f(); // f = 10
f() = 20; //luc nay z = ? = 20
return 0;
}
```

```
#include <iostream>
using namespace std;
int &max(int &a, int &b) //hàm tra ve max cua a và b
{
return a>b ? a:b;
}
int main(){
  int a = 7, b = 10, c = 20;
  cout << "Max a,b : " << max(b,a) << endl; // Max a,b : 10
  max(b,a)++; //tang b lên 1
  cout << " a = "<< a <<" b = "<<b <<endl; //a = 7 b = 11
  max(b,c) = 5; //gan lai gia tri 5 cho bien c
  cout << " a = " << a << " b = " << b << " c = " << c << endl;//a=7 b=11 c=5
  return 0;
}
```

```
#include <iostream>
using namespace std;
int &max(int &a, int &b){
return a>b ? a:b;
}
int main(){
int a = 10, b = 4, c = 2;
cout << "Max a,b : "<<max(b,a) << endl; // Max a,b: 10
max(b,a)++; //tang a lên 1
cout << " a = "<< a <<" b = "<<b <<endl; //a = 11 b = 4
max(b,c) = 5; // gan lai gia tri 5 cho b
cout << " a = "<< a <<" b = "<<b <<" c = "<<c<< endl;//a= 11 b=5 c=2
return 0;
}
```

7. Hàm với tham số có giá trị mặc định: (gán từ phải sang trái, giá trị mặc định phải nằm bên phải)

Xây dựng hàm với các tham số được khởi gán ***giá trị mặc định***. Quy tắc xây dựng hàm với tham số mặc định như sau:

- Các ***tham số có giá trị mặc định*** cần là các ***tham số cuối cùng*** tính ***từ trái qua phải***.
- Nếu chương trình **sử dụng khai báo nguyên mẫu hàm** thì các ***tham số mặc định cần được khởi gán trong nguyên mẫu hàm***, không được khởi gán lại cho các tham số mặc định trong dòng đầu của định nghĩa hàm.

VD:

```
void f(int a, float x, char *st=”TRUNG TAM”, int b=1); //nguyên mẫu hàm f

void f(int a, float x, char *st, int b){
//Code
}
```

Đối với các hàm có đối số mặc định thì lời gọi hàm cần viết theo quy định: ***Các tham số vắng mặt trong lời gọi hàm tương ứng với các đối số mặc định cuối cùng*** (tính từ trái sang phải), ví dụ:

```
void f(int a, float x, char *st=”TRUNG TAM”, int b=1);
thì các lời gọi hàm đúng:
f(3,3.4,"TIN HOC",10);//Đầy đủ tham số
f(3,3.4,"ABC"); //Thiếu 1 tham số cuối
f(3,3.4); //Thiếu 2 tham số cuối
```

```
void f1(int a=1, int b, int c); //S
void f1(int a=1, int b=2, int c); //S
void f1(int a=1, int b, int c=2); //S
void f1(int a=1, int b=2, int c=3); //Đ
void f1(int a, int b=2, int c=3); //Đ
```

```
#include <iostream>
using namespace std;
void ht(char *dc="TRUNG TAM", int n=5){
```

```
for(int i=0;i<n;++i)
  cout<<"\n" <<dc;
}
int main(){
ht();//
ht("ABC",3); // 3 lan ABC
ht("DEF"); // 5 lan DEF
return 0;
}
```

---

8. Hàm tải bội (overloading function)

Các hàm tải bội (còn gọi là hàm quá tải) là ***các hàm có cùng một tên*** và ***có tập đối khác nhau (về số lượng các đối hoặc kiểu).***
Khi gặp lời gọi các hàm tải bội thì trình biên dịch sẽ ***căn cứ vào số lượng và kiểu các tham số để gọi hàm*** có đúng tên và đúng các đối số tương ứng