函数对象

实际表现像个函数，实际是个对象，只需重载小括号运算符即可。

`Function *q = &func;` 这样写的问题是：没有办法匹配所有的函数对象。与函数指针的原意相违背了。

作为一个函数对象，它原本的类型重要吗？——不重要，但是外在表现重要。
函数指针本质上是在指向拥有相同外在表现的一类函数

原本C语言的写法：`int (*p)(int, int) = func1;` 就是一种外在表现的描述；传入两个整型，返回一个int的这样一类函数，p指针都能指向。

那么在C++中的函数指针长什么样呢？

```
function<int(int, int)> q = func
q(3, 4)
```

q 是函数指针对象。

指针对象

指针的外在表现形式很多，所以需要重载的东西也很多。

要比纯指针的功能更强大

```
->
<<
*
+ -
[ ]
```

二倍扩容法的均摊时间复杂度是O(1)的：数组中要是有n个元素，对于这n个元素，我曾经所有用在扩容上的时间复杂度，我可以平均地分配给n个元素。

证明：
当前数组有n位空间的时候，上一次扩容一定产生了n/2的额外拷贝动作。

$$\frac{n}{2}+\frac{n}{4}+\cdots+\frac{n}{2m} \overset{m\to\infty}{\to} n$$

所以总的时间是 $n/n=1$（次）

12.friend.cpp

```cpp
#include<iostream>
#include<cstdlib>
#include<functional>
using namespace std;
#define BEGINS(x) namespace x {
#define ENDS(x) }

BEGINS(haizei)
class Point;
ENDS(haizei)

BEGINS(array_object)
class Array {
public:
    Array(int n = 100) : n(n), data(new int[n]) {}
    int &operator[](int ind) {
        return data[ind];
    }
    const int &operator[](int ind) const {
        return data[ind];
    }//返回值是const类型的保证不会被修改。
    int operator[](const char *s) {
        int ind = stoi(s);
        return data[ind];
    }
    int &operator[](const haizei::Point &a);
private:
    int n;
    int *data;
};
ENDS(array_object)

BEGINS(haizei)
class Point {
public:
    Point(int x, int y) : x(x), y(y) {
        z1 = rand();
        z2 = z1 + 1;
        z3 = z2 + 1;
        cout << this << "rand value : " << z1 << endl;
    }
    void output() {
        cout << "inner : " << x << " " << y << endl;
    }
    friend void output(Point &); // 类外的output函数是友元
    friend ostream &operator<<(ostream &, const Point &);
```

```
    friend Point operator+(const Point &, const Point &);
    friend int &array_object::Array::operator[](const Point &);
    Point operator+(int n) const {
        return Point(x + n, y + n);
    }
    Point &operator+=(int x) {
        this->x += x;
        this->y += x;
        return *this;
    }
    Point operator++(int) {//返回加一之前的那个值,所以不传引用
        Point ret(*this);
        x++;
        y++;
        return ret;
    }
    Point &operator++() {
        x++;
        y++;
        return *this;
    }
    int z1, z2, z3;

private:
    int x, y;
};
void output(Point &a) {
    cout << "outer : " << a.x << " " << a.y <<endl;
}
ostream &operator<<(ostream &out, const Point &p) {
    out << "Point(" << p.x << ", " << p.y << ")";
    return out;
}
Point operator+(const Point &a, const Point &b) {
    return Point(a.x + b.x, a.y + b.y);
}
ENDS(haizei)

BEGINS(function_object)
class Function {
public:
    int operator()(int a,int b) {
        cout << "inner class : ";
        return a + b;
    }
private:
```

```
};

int func1(int a, int b) {
    cout << "inner func1 : ";
    return a + b;
}

int main() {
    Function func;
    cout << func(3, 4) << endl; // 本质是一个对象，看起来像函数
    int (*p)(int, int) = func1; // 函数指针
    cout << p(3, 4) << endl;
    function<int(int, int)> q;
    q = func1;
    cout << "q pointer : " << q(3, 4) << endl;
    q = func;
    cout << "q pointer : " << q(3, 4) << endl;

    cout << greater<int>()(4, 3) << endl; //greater是用来比较第一个数是否大于第二个数的
    cout << greater<int>()(3, 4) << endl;
    //greater<int>是类型，小括号()一写，就是在构造一个临时对象，后面再接一个小括号，里面还有参数，那这个对
    return 0;
}
ENDS(function_object)

BEGINS(pointer_object)
struct A {
    A() : x(0), y(0) {}
    int x, y;
};
ostream &operator<<(ostream &out, const A &a) {
    cout << a.x << " " << a.y;
    return out;
}
class Pointer {
public:
    Pointer(A *p = nullptr) : p(p) {}
    A *operator->() { return p; }
    A &operator*() { return *p; }
    A *operator+(int n) { return p + n; }
    A *operator-(int n) { return p - n; }
    A &operator[](int n) { return *(p + n); }
    int operator-(const Pointer a) { return p - a.p; }

    A &operator*() const { return *p; }
```

```
    A *operator+(int n) const { return p + n; }
    A *operator-(int n) const { return p - n; }
    A &operator[](int n) const { return *(p + n); }
    friend ostream &operator<<(ostream &,const Pointer &);
private:
    A *p;
};
ostream &operator<<(ostream &out, const Pointer &p) {
    cout << p.p;
    return out;
}
int main() {
    A a, b;
    Pointer p = &a, q = &b;//外在表现就像一个指针一样
    cout << a << endl;
    p->x = 3;
    p->y = 4;
    cout << a << endl;
    cout << *p << endl;
    cout << p << endl;
    cout << p + 1 << endl;
    cout << p - 1 << endl;
    cout << p - q << endl; //TODO
    cout << p[0] << endl; //TODO

    const  Pointer cp = &a; //const Pointer指针对象中存储的值不能变

    cout << *cp << endl; // TODO
    cout << cp + 1 << endl; //TODO
    cout << cp - 1 << endl; //TODO
    cout << cp[0] << endl;

    return 0;
}
ENDS(pointer_object)

int main(){
    // test1::main(); 第一个测试用例
    //test2::main();
    //array_object::main();
    //function_object::main();
    pointer_object::main();
    return 0;
}
```