指针引用

6、指针

指针的作用： 可以通过指针间接访问内存

6.1、指针变量

指针变量定义语法： `数据类型 * 变量名;`

```
int main() {

    //1、指针的定义
    int a = 10; //定义整型变量a

    //指针定义语法:  数据类型 * 变量名 ;
    int * p;

    //指针变量赋值
    p = &a; //指针指向变量a的地址
    cout << &a << endl; //打印数据a的地址
    cout << p << endl;  //打印指针变量p
    //0073F8BC
    //0073F8BC

    //2、指针的使用
    //通过*操作指针变量指向的内存
    cout << "*p = " << *p << endl;
    // *p = 10

    system("pause");

    return 0;
}
```

指针变量和普通变量的区别

- 普通变量存放的是数据,指针变量存放的是地址
- 指针变量可以通过" * "操作符，操作指针变量指向的内存空间，这个过程称为解引用
  -> 总结1： 我们可以通过 & 符号 获取变量的地址

> 总结2：利用指针可以记录地址

> 总结3：对指针变量解引用，可以操作指针指向的内存

> 总结4：所有指针类型在32位操作系统下是4个字节（了解）

6.2、const修饰指针

const修饰指针有三种情况

1. const修饰指针 — 常量指针
2. const修饰常量 — 指针常量
3. const既修饰指针，又修饰常量

```
int main() {

    int a = 10;
    int b = 10;

    //const修饰的是指针，指针指向可以改，指针指向的值不可以更改
    const int * p1 = &a;
    p1 = &b; //正确
    //*p1 = 100;  报错


    //const修饰的是常量，指针指向不可以改，指针指向的值可以更改
    int * const p2 = &a;
    //p2 = &b; //错误
    *p2 = 100; //正确

    //const既修饰指针又修饰常量
    const int * const p3 = &a;
    //p3 = &b; //错误
    //*p3 = 100; //错误

    system("pause");

    return 0;
```

搜索 评论 笔记

4.

5.技巧：看const右侧紧跟着的是指针还是常量，是指针就是常量指针，是常量就是指针常量

6.3、指针和数组

作用：利用指针访问数组中元素

- C++规定，数组名就是数组的起始地址
- 数组的指针就是数组的起始地址

- 数组名可以作函数的实参和形参，传递的是数组的地址

```cpp
int main() {

    int arr[] = { 1,2,3,4,5,6,7,8,9,10 };

    int * p = arr;  //指向数组的指针

    cout << "第一个元素：  " << arr[0] << endl;        //1
    cout << "指针访问第一个元素：  " << *p << endl;      //1

    for (int i = 0; i < 10; i++)
    {
        //利用指针遍历数组
        cout << *p << endl;
        p++;
    }

    system("pause");

    return 0;
}
```

-

6.4、指针和函数

作用：利用指针作函数参数，可以修改实参的值（**地址传递**

```cpp
//值传递
void swap1(int a ,int b)
{
    int temp = a;
    a = b;
    b = temp;
}
//地址传递
void swap2(int * p1, int *p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main() {

    int a = 10;
    int b = 20;
    swap1(a, b); // 值传递不会改变实参

    swap2(&a, &b); //地址传递会改变实参

    cout << "a = " << a << endl;

    cout << "b = " << b << endl;

    system("pause");

    return 0;
}
```

```
int a[10];

int *p = &a[0];  // 等价于 int *p = a;
*p = 1;          // 等价于 a[0] = 1;
*(p+1) = 2;      // 等价于 a[1] = 2;
// 所以 *(p+1) = a[1]; *(p+2) = a[2];
```

- C++规定， p+1 指向数组的 下一个元素

```
void main()
{
    int array[10];
    // 用数组名作形参，因为接收的是地址，所以可以不指定具体的元素个数
    f(array,10);
}
```

```
void main()
{
    int array[10];
    // 用数组名作形参，因为接收的是地址，所以可以不指定具体的元素个数
    f(array,10);
}

// 形参数组
f(int arr[],int n)
{
    ....
}
```

```
void main()
{
    int a[10];
    // 实参数组
    f(a,10);
}
// 形参指针
f(int *x,int n)
{
    ...
}
```

总结：如果不想改变实参，就用值传递。如过想改变实参，就用地址传递

6.4、返回指针值的函数

- 返回指针值的函数简称指针函数。
- 定义指针函数的一般形式为：
- ```
  // 类型名 * 函数名（参数列表）
  int  * a(int x,int y);
  ```

-## 7、引用

作用： 给变量起别名
  ```

语法： `数据类型 &别名 = 原名`

```cpp
int main() {

    int a = 10;
    int &b = a;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    // 10
    // 10

    b = 100;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    // 100
    // 100

    system("pause");

    return 0;
}
```


7.1、引用注意事项

- **引用必须初始化**
- int &c; // 错误，引用必须初始化
- - **在声明一个引用后，不能再使之作为另一变量的引用**
  -### 7.2、引用做函数参数

作用：函数传参时，可以利用引用的技术让形参修饰实参

优点：可以简化指针修改实参

- **通过引用参数产生的效果同按地址传递是一样的。引用的语法更清楚简单**
  -

```cpp
//1. 值传递
void mySwap01(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

//2. 地址传递
void mySwap02(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
//参数：把地址传进去，用指针接收

//3. 引用传递
void mySwap03(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
//参数：别名，下面的a是上面的a的别名，用别名操作修改可原名操作修改是一样的
int main() {

    int a = 10;
    int b = 20;
    // 值传递，形参不会修饰实参
    mySwap01(a, b);
    cout << "a:" << a << " b:" << b << endl;
    // a:10 b:20
```

```
/参数：别名，下面的a是上面的a的别名，用别名操作修改可原名操作修改是一样的
nt main() {

    int a = 10;
    int b = 20;
    // 值传递，形参不会修饰实参
    mySwap01(a, b);
    cout << "a:" << a << " b:" << b << endl;
    // a:10 b:20

    // 地址传递，形参会修饰实参
    mySwap02(&a, &b);
    cout << "a:" << a << " b:" << b << endl;
    // a:20 b:10

    // 引用传递，形参会修饰实参
    mySwap03(a, b);
    cout << "a:" << a << " b:" << b << endl;
    // a:20 b:10

    system("pause");

    return 0;
```

7.3、引用做函数返回值

作用：引用是可以作为函数的返回值存在的

```cpp
//数据类型后加&，相当于用引用的方式返回
int& test02() {
    // 必须使用静态变量，需加 static 关键字
    static int a = 20;
    return a;
}


int main(){
    int& ref2 = test02();
    system("pause");
    return 0;
}
```