**引用**

1. **初始引用**
2. 引用原则
   * 1. 只能引用一次，不能多次引用，且必须与原类型保持一直（除非加上const关键字）
     2. 这就很有趣了，那引用就是地址的引用，我们可以通过引用来修改原来元素的值
     3. **变量的引用**

int a = 10;

int &ra = a; //这就是引用

print(“%d”, ra); //输出10

ra = 20;

Print(“%d”, a); // 输出20

* + 1. **指针的引用**

int a = 3;

int \*p = &a;

int\* &rp = p; //对指针进行引用

\*rp = 4; // 对值进行操作

std::cout << \*p;

1. **引用的加深**
2. 引用不需要浪费内存、
   * 1. 只比如现在我们要写一个交换a，b中的函数

void swap(int a, int b){

}

* + 1. 这就很有趣了，那引用就是地址的引用，我们可以通过引用来修改原来元素的值，我们用引用来做就不需要浪费内存，并且是a， b自身来进行操作的，所以会提高效率

void swapAB(int &a, int &b){

a ^= b;

b ^= a;

a ^= b;

}

* + 1. 那现在还有指针来进行交换呢 char \*p = ‘china’ char \*q = ‘ameria’,那我们怎么用指针引用来进行操作呢
       1. 那我们可以直接对其地址进行交换是不是

###法一

void swapPQ(char\*\* p, char\*\* q){

char \*t;

t = \*p; // 我们来交换地址不久得了？？我们\*p是 p是地址，那我们\*\*p时是不是\*p就代表地址了

\*p = \*q;

\*q = t;

}

int main(){

char \*p = "china";

char \*q = "ameria";

swapPQ(&p, &q);

std::cout << p << std::endl << q;

}

###法二

void swapPQ(char\* &p, char\* &q){

char \*t;

t = p; // 我们来交换地址不久得了？？

p = q;

q = t;

}

int main(){

char \*p = "china";

char \*q = "ameria";

swapPQ(p, q);

std::cout << p << std::endl << q;

}

1. **引用的高级部分**
2. 我们可以建立指针的指针，但是我们不能建立引用的引用

int \*p = &a; int \*\*pp = &p; //是可以的

int &ra = a; int && rara = ra; // 是不可以的

1. 可以定义指针的指针，但是不可以定义引用的指针（因为引用本身就是对指针的一次包装）

int \*p = &a; int \*\*pp = &p; //是可以的

int &ra = a; int& \*rara = &ra; //是不行的

1. 可以定义指针数组，但是不能定义引用数组，可以定义数组引用

int a, b, c;

int \*p[] = {&a, &b, &c}; //可以，这是指针数组

int (\*p)[] = {&a, &b, &c}; //这是数组指针

int &rp[] = {q, b, b}; //不可以，这是引用数组是不允许的

int arr[] = {1, 2, 3};

int (&rarr)[5] = arr; //可以，这就是数组引用

1. 常引用
   * 1. const 的引用也必须时const的

const int a;

const int &ra = a; //这样才行

* + 1. const 也可以定义不同的引用类型

int a = 3;

const double &ra = a;

* + 1. 本来我们不能 int &ra = a + 3;

但是加了const int &ra = a + 3; 时成立的

总结：const 是流氓