1

$\color{red} {手写 memmove} $

// 但是 库函数中 memmove(dst, src, int n); // 从后往前移动

```
void* Mymemmove(const void* src, void* dst,size_t n) {
    char* d = (char*)dst;
    const char* s = (const char*)src;

    if (s > d) {
        while (n--) {
            *(d++) = *(s++);
        }
    }else if (s < d) {
        s -= n;
        d -= n;
        while (n--) {
            *(--d) = *(--s);
        }
    }

    return dst;
}

int main(){
    const char* src = "1234567";
    char dst[] = "567890";

    Mymemmove(src, dst, 3);
    printf("%s\n", dst);

    char str1[] = "1234567890";
    Mymemmove(str1 + 2, str1, 3);

    printf("str1=%s\n", str1);
}
```

2 $\color{red} {反转str中单词，但注意 只在 , 间隔开的区域内各自反转：} $

例子： Hello Wrold , Ni Hao ----> Wrold Hello , Hao Ni

$\color{green} {getline(istream,str1,'char')}$ 将 istream用 char分割开，然后填给 str1
所以可以 while(getline(cin,str1,',')) 来先把 str 划分，形成一个个 vector<string> 集合,然后对 集合进行 reverse

操作，接着 最后把集合 拼接！！

```
#include <iostream>
#include <vector>
#include <sstream>                //** 流操作 得包含此文件 而不是 istream! !
#include <string>
#include <algorithm>
using namespace std;

int main() {
    string str1 = "Hello World,Ni hao";
    char splitChar = ',';

    stringstream ss(str1);
    string str_temp;
    string ret;
    while (getline(ss, str_temp, splitChar)) {
        stringstream was(str_temp);

        vector<string> words;
        string temp;
        while (was >> temp) {
            words.push_back(temp);
        }

        reverse(words.begin(), words.end());

        for (int i = 0; i < words.size(); i++) {
            ret += words[i];
            if (i != words.size() - 1) {
                ret += ' ';
            } else {
                ret += ',';
            }
        }
    }

    if (!ret.empty() && ret.back() == ',') {
        ret.pop_back();
    }

    cout << ret << endl;

    return 0;
}
```

变成流之后，可以使用流文件操作

$\color{green} { stringstream ss(str1);}$ $\color{green} { while (getline(ss, str_temp, splitChar))}$

$\color{green} { stringstream was(str_temp);}$ $\color{green} { while (was >> temp)}$

3 $\color{red} {反转32位有符号整数，且若反转后，结果大于INT_MAX 或 小于 INT_MIN 则返回0} $

$\color{red} {ret设为int，然后它的越界是在 res更新的前一步进行判断的很秀} $

```
class Solution {
public:
    int reverse(int x) {
        int res = 0;
        while (x != 0) {
            int rem = x % 10;
            x /= 10;
            if (res > INT_MAX / 10 || (res == INT_MAX / 10 && rem > INT_MAX % 10))
{
                return 0;
            }
            if (res < INT_MIN / 10 || (res == INT_MIN / 10 && rem < INT_MIN % 10))
{
                return 0;
            }
            res = res * 10 + rem;
        }
        return res;
    }
};
```

$\color{blue} {ret设为long long，越界可以直接判断，但是 这里的话 long long 每次设计到数据转换，所以会性能上慢很多} $

```
class Solution {
public:
    int reverse(int x) {
        if(x==0)return 0;
        long long ret=0;

        while(x!=0){
            int temp=x%10;
            x/=10;
            if(ret==0&&temp==0)
                continue;
            ret=(ret*10+temp);
            if(ret>INT_MAX||ret<INT_MIN)return 0;
        }

        return ret;
```

```
    }
};
```

快排 QuickSort

/// 快排几个注意点 ///
/// 1、对方动：每次交换后，“己方”这边已经有序了，所以不要动，让对方动 index_L 或者 index_R /// 2、while循环中 要考虑有重复元素的情况 所以 两个while，有一方要考虑 =的情况，来打破循环，不然会一直swap交换 如 2 1 1 2 L=0 R=3 会一直循环，因为 nums[R]>nums[L] 和 nums[L]<nums[R]的判断 永远不成立，所以 L和R 永远不会改变 /// 3、循环中 R和L也要保证 外部循环条件成立，不然 在内部的while中，可能某次循环 R会走到L的左边或者L走到R的右边，然后swap，这会导致结果出错 (其实 只要 while (L < R && nums[L] <= nums[R]) 循环加上即可，因为只有这里 是nums[L] <= nums[R]，L++，这里会导致 跳出最外部循环前，L往前再额外+1，所以出错) void QuickSort(vector& nums,int L,int R) { if (L >= R) { return; } int start = L; int end = R; while (L < R) { // 这里不加 L<R也行，因为L==R 时候，必然 nums[R]是等于nums[L]的会自动跳出此内部循环 while (L<R&&nums[R] > nums[L]) { R--; } swap(nums[L], nums[R]); while (L < R && nums[L] <= nums[R]) { L++; } swap(nums[L], nums[R]); }

```
  QuickSort(nums, start, L-1);
  QuickSort(nums, L+1, end);

  return;
```

}