一、常用库函数

字符或字符串处理函数

- 判断字符是否为阿拉伯数字，返回类型为bool

```
char c;
isdigit(c);
```

- 数值转字符数组

```
char s[10];
double a=1.2345;
sprintf(s,"%0.2lf",a); //double对应lf，float对应f，int对应d
```

- 字符数组与数字的转换

```
int atoi(const char *s) 将s所指向的字符串转换成整数
double atof(const char *s)  将s所指向的字符串转换成实数
long atol(const char *s)    将s所指的字符串转换成长整数
double strtod(const char *s)将字符串转换成浮点数
```

- 字符数组和字符串的转换

```
char s[10] = "abcd";
string a(s, strlen(s));
```

- 字符串和字符数组转换

```
string a = "abc";
 const char * s = a.c_str(); //s 的存储的内容为 abc\0
```

- 字符数组比较函数

```
字符数组指定长度进行比较
```

```
char * a = "ffabcd";
char * b = "abcd";
cout << memcmp(a, b, 4) << endl; //比较a，b所指内存空间的前4个字节，完全相同返回0，小于返回负数，大于返回正数

//比较字符串
int stcmp( const char *s1, const char* s2);  //相同返回0，小于返回负数，大于返回正数

```

- 字符串查找函数

```
char * strstr(const char*  S, const char * s);//在S中搜寻字符串s，并将第一次出现的地址返回，无子串时返回0。
```

- 字符数组相关函数

```
字符数组拷贝
memcpy(dest, src, 30);  //src指向的内容 复制到 dest，长度为30字节

连接两字符串
char * strncat(char * dest, const char * src, size_t n); //src的前n个字符拷贝到dest的后面，且dest空间够大。

字符查找,从头开始搜寻s所指的前n个字节，知道发现第一个值为c的字节，则返回指向该字节的指针，否则返回0
void *memchr(const void* s, int c, size_t n);
```

cmath

对数、幂函数、绝对值

```
//求平方根
sqrt(x);

//对数
int n = log(x) / log(base);  //base 的 n 次方 等于 x，求n
int n = log(8) / log(2);  //2 的 3 次方等于 8
```

```

//幂函数
double x = pow(base, n); // 求base 的 n 次方

指数函数
double exp(double x);   底为e，x次方的计算结果

//绝对值
abs(a - b); // 整数
fabs(a - b); //浮点数
labs(a - b); //长整形数 long

//浮点数的余数
double fmod(double x,double y);   //x除以y的余数

//浮点数的整数和小数拆分
double modf(double x,double *integer);  //传入值x；返回x的小数部分，符号与x相同，integer是指向一个对象的指针。
double n,m;  m = modf(10.1, &n); //则m为0.1，  n为 10

```

数的最大值最小值

```
#include <limits.h>
int n1  =  INT_MIN;  （通过F12查看其它类型的最大值最小值）
int n2  =  INT_MAX;
float  f1  =  FLT_MIN;
float  f2  =  FLT_MAX;
double  d1  =  DBL_MIN;
double  d2  =  DBL_MAX;
long ln1 = LONG_MAX;
long ln2 = LONG_MIN;
long long lln1 = LONG_LONG_MAX;
long long lln1 = LONG_LONG_MIN;
```

algorithm

```
#incllude <algorithm>
```

反转

```
reverse(res.begin(), res.end());

```

排序

快排

```
sort(res.begin(), res.end()); //快速排序，容器和数组都可以
```

堆排序

```
make_heap(res.begin(), res.end(), greater<int>()); //小顶堆，注意第三个参数 greater<int>() （需要头文件functional）
make_heap(res.begin(), res.end()); //对res排成大顶堆

res.push_back(8);
push_heap(res.begin(), res.end()); //添加元素，并调整堆

pop_heap(res.begin(), res.end());
res.pop_back(); 删除元素，并调整堆
```

优先队列

```
priority_queue <int>res(res.begin(), res.end()); //默认的大顶堆
priority_queue <int,vector<int>,less<int> >q; //大顶堆，出队列是权值最大的（默认情况下就是大顶堆），降序队列
priority_queue <int,vector<int>,greater<int> > q; 小顶堆，出队列的是权值最小的，升序队列
```

查找函数

二分查找

对于已排序的数组，才可用二分查找

```
//容器的起始地址为a.begin()，数组的起始地址为 a
binary_search(nums.begin(), nums.end(), 6); // 返回类型为bool，表示是否存在数值6

lower_bound(起始地址，结束地址，要查找的数值)  返回大于或等于val的 第一个元素位置（准确的来说是地址）
int pos = lower_bound(nums.begin(), nums.end(), 6) - nums.begin();//pos为6在容器的下标

upper_bound(起始地址，结束地址，要查找的数值)  返回大于val的 第一个元素位置
int pos = upper_bound(nums.begin(), nums.end(), 4) - nums.begin();
```

二、简单功能接口函数

编程题的输入输出问题

数据精度问题

32系统中 short与short int型数据占2个字节（16位）；int、long int、long型数据占4个字节（32位）；long long 型数据占8个字节（64位）。

- short与short int可以表示的的最大范围是：-32768<---->32767
- int、long int、long 可以表示的最大范围是：-2147483648<---->2147483647 $2 * 10^9$
- long long 可以表示的最大范围是：-9223372036854775808<--->9223372036854775807 $9 * 10^{18}$

```
short int nCount1 = 32767; printf("%hd\n",nCount1);
 long nCount2 = 2147483647; printf("%ld\n",nCount2);
 long long nCount3 = 9223372036854775807; printf("%lld\n",nCount3);
 long nCount4 = 2147483649; printf("%ld\n",nCount4);
```

使用文件作为输入源

- input_data.txt的数据如下

```
9
1 3 4 7 2 6 5 12 32
```

- 主代码

```
int main(){
 freopen("input_data.txt","r",stdin);
 cin >> n;
 vector<LL> num(n);
 for (int i = 0; i < n; ++i)
  cin >> num[i];
//逻辑业务代码
 freopen("CON", "r", stdin);
 system("pause");
 return 1;
}
```

```
freopen("input_data.txt", "r", stdin);
 freopen("out.txt", "w", stdout);
 fclose(stdin);
 fclose(stdout);
```

申请动态二维数组

- 申请vector

```
// 行数N，列数M
vector< vector<int> > a(N, vector<int>(M));
```

- 动态申请arr[M][N]

```
int ** arr = new int *[M];
 for (int i = 0; i < M; ++i)
  arr[i] = new int[N];
```

按行读数据

```
char str[10000];
cin.getline(str, 10000);

#include <sstream>
//读取每行个数不确定的情况（可能会遇到）
int num;
string str;
getline(cin, str);
istringstream ist(str);
 while (ist >> num)
  nums.push_back(num);

//读取不确定行数，每行数据个数不确定（几乎遇不到）
while (getline(cin, str)){//读取输入的一行数据
  if (str.size() == 0)
   break;//如果读取的是空，则读取结束
  istringstream ist(str);//把读取的str送入流中
  while (ist >> num)
   nums.push_back(num);//逐个读取流中的int
 }
```

按指定符号分割字符串

```
void split(string str, vector<string> &result, char split_c){
 string temp("");
 str += split_c;//字符串结束标记，方便将最后一个单词入vector
 for (int i = 0; i < str.size(); i++){
  if (str[i] == split_c){
   result.push_back(temp);
   temp = "";
  }
  else
   temp += str[i];
 }
}
```

```
char * strtok（char * s, const char * delim); //strtok()用来将字符串割成一个个片断。s指向源字符串，delim为分割字符串，分割是把将该字符改为\0字符。 第一次调用strtok的时候需给予参数s，往后的调用则将参数s设成NULL。

 char a[10] = "abcdcecf";
 const char * b = "c";
 char *p;
 cout << strtok(a, b) << endl;
 while (p = strtok(NULL, b))
  cout << p << " ";
```

实现输入多组测试数据

> 输入描述：
> 包含多组测试用例。
> 对于每组测试用例：
> 第一行包括2个整数，N（1 <= N <= 1000），M(0 <= M <= N*(N-1)/2)，代表现有N个人（用1~N编号）和M组关系；
> 在接下来的M行里，每行包括3个整数，a，b, c，如果c为1，则代表a跟b是同乡；如果c为0，则代表a跟b不是同乡；
> 输入样例：
> 3 1
> 2 3 1
> 5 4 //第二组测试数据
> 1 2 1
> 3 4 0
> 2 5 1
> 3 2 1

```
#include <iostream>
using namespace std;
int main(){
    int N, M;
    // 每组第一行是2个整数，N和M，用while实现输入多组数据。
    while(cin>> N >> M) {
      cout << N << " " << M << endl;
      // 循环读取“接下来的M行”
```

```
      for (int i=0; i<M; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        cout << a << " " << b << " " << c << endl;
      }
    }
    return 0;
}
```

输出

- 输出遍历

```
string s = "addf";
 for (auto c : s) //遍历s的每个字符
  cout  << c;

 for (auto &c : s) //&表示c为引用，可以修改s的值
  c = toupper(c);
```

- 输出格式

```
整数：
%d 有符号十进制  %u 无符号十进制
%o 无符号八进制
%x 无符号十六进制，小写
%X 无符号十六进制，大写

浮点型数：
%f double型参数转十进制数字，小数点以下六位，四舍五入
%e double行参数以指数（科学记号）形式打印
printf("%.4f ", 12.3333333);   //输出 12.3333，显示四位小数
printf("%#o  %#x\n",1234,1234);//分别输出8进制和16进制的值，其中#的左右是自动添加 0或 0X

控制结果打印
 cout << res[0];
    for(int i = 1; i < N; i++)
        cout << " " << res[i];
```

对多个键值按指定规则进行排序

通过结构体来实现

```
struct Node{
 int a;
 int b;
 int *next;
 Node(int x,int y) :a(x),b(y), next(NULL){ }
};

bool cmp(Node node1, Node node2){
 return node1.b < node2.b;
}

vector<Node> nums;
 for (int i = 0; i < 10; ++i)
  nums.push_back(Node(i, rand() % 20));
 sort(nums.begin(), nums.end(), cmp);
 for (auto p : nums){
  cout << p.a << " " << p.b << endl;
 }
```

map按key或value进行排序

```
typedef pair<int, int> pii;
bool cmp(pii a, pii b) {
 return a.first < b.first; //按key进行升排序
//  return a.second < b.second; 按value升进行排序
}

 map<int, int> mp;
 vector<pii> vc;
// map<int, int>::iterator mit;  vector<pii>::iterator vit;
 mp[7] = 1;mp[6] = 5; mp[4] = 7;

 for (auto mit = mp.begin(); mit != mp.end(); mit++)
```

```
  vc.push_back(pii(mit->first, mit->second));
 sort(vc.begin(), vc.end(), cmp);

 for (auto vit = vc.begin(); vit != vc.end(); vit++)
  cout << vit->first << " " << vit->second << endl;
```

树的三种非递归方法

- 前序遍历

```
void PreOrder(TreeNode * root){
 TreeNode * p = root;
 stack<TreeNode*> s;
 while (p || !s.empty()){
  while (p){
   s.push(p);
   cout << p->val << " ";
   p = p->left;
  }
  if (!s.empty()){
   p = s.top(), s.pop();
   p = p->right;
  }
 }
}
```

- 中序遍历

```
void MiddleOrder(){
    Node p = root;
   stack <Node *> s;
    while (p || !s.empty()){
        while (p){
            s.push(p);
            p = p->l;
        }
        if (!s.empty()){ //到这一步说明s.top()的左子树访问完了(或者左子树为空)
```

```
            p = s.top(), s.pop();
            cout<<p->idx<<" ";
            p = p->r;
        }
    }
}
```

- 后续遍历

```
void LaterOrder(TreeNode * root){
 TreeNode * p = root, * last_p = NULL;
 stack<TreeNode*> s;
 while (p || !s.empty()){
  while (p){
   s.push(p);
   p = p->left;
  }
  if (!s.empty()){
   p = s.top();
   s.pop();

   if ( p->right && p->right != last_p)//右子树存在且未被访问
    p = p->right;
   else{
    cout << p->val << " ";
    last_p = p;
    p = NULL; //方便往上遍历
   }

  }
 }
}
```

求最大公约数（以及多个数）

```
int  gcd(int a, int b){
 return b ? gcd(b, a%b) : a;
}
```

```cpp
int FindMin(vector<int> & nums){
 int len = nums.size();
 if (len <= 1)
  return len;
 int min = nums[0];
 for (int i = 1; i < nums.size(); ++i){
  if (nums[i] < min)
   min = nums[i];
 }
 bool find_min = false;
 while (min){
  int j = 0;
  while (j < len){
   if (nums[j] % min != 0)
    break;
   ++j;
  }
  if (j == len){
   find_min = true;
   break;
  }
  --min;
 }
 int sum = 0;
 for (int i = 0; i < len; ++i){
  sum += nums[i] / min;
 }
 return sum;
}
```

随机数

输出随机的0.1到1之间的值

```cpp
cout << 0.1*(rand()%10 + 1) << endl;
```

翻转函数

```
void reverse(string &str, int low, int high){
        while(low < high){
            char temp = str[low];
            str[low++] = str[high];
            str[high--] = temp;
        }
    }
```

获取长度

```
//容器
vector <int> nums;
int len = nums.size();
//字符串
string str = “aaaaa”;
int len = str.length();
//数组
char * ptr = ”abcdef“;
int len = strlen(ptr); //len为7
```

交换两个值

- 算术运算
  坐标系思想

```
a = 10; b = 12;
a = b - a;
b = b - a;
a = b + a;
```

- 位运算

```
a = 10; b = 12;
a=a^b; //a=0110^b=1100;
b=a^b; //a=0110^b=1010;
a=a^b; //a=1100=12;b=1010;
```

基础程序语句

- 逻辑与和或的优先级

```
按位与 （&）优先级 8
逻辑与 （&&）优先级 11
逻辑或 （ || ）优先级 12
条件运算符 （?:） 优先级 13
```

- ?表达式

```
    判断语句 ？ 表达式1 :表达式2;
    //判断为真，则运行表达式1;
    //判断是为假，则运行表达式2;
 sum = (tempsum > sum) ? tempsum : sum;
```

- switch()

```
switch( n){
    case 1: 函数（）;break;.
    case 2: 函数（）;break;
    default:函数（）;break;
}
```

- new一个结构体空间

```
struct RandomListNode {
    int label;
    struct RandomListNode *next, *random;
    RandomListNode(int x) :
            label(x), next(NULL), random(NULL) {
    }
};

RandomListNode * p_curren
RandomListNode * node = new RandomListNode(p_current->label);
```