一、数学问题

1. 精度计算——大数阶乘

```
#include<iostream>
using namespace std;
#include<cstring>
int main(){
int num[100000]={0};
num[0]=1,num[1]=1;
int n;cin>>n;
int book=0;
for(int i=1;i<=n;i++){
    for(int j=1;j<=num[0];j++){
        num[j]*=i;
    }
    for(int j=1;j<=num[0];j++){
            if(num[j]<10){
            continue;}
            num[j+1]+=num[j]/10;
            num[j]%=10;
            num[0]+=(j==num[0]);
        }
     }
   for(int i=num[0];i>0;i--){
    cout<<num[i];
   }
   return 0;
}
```

2. 精度计算——乘法（大数乘小数）

语法：mult(char c[],char t[],int m);

```
参数:
c[]：被乘数，用字符串表示，位数不限
t[]：结果，用字符串表示
m：乘数，限定10以内
返回值：null
注意:
 需要 string.h
源程序:
```

```
     void mult(char c[],char t[],int m)
    {
        int i,l,k,flag,add=0;
        char s[100];
```

```
        l=strlen(c);
        for (i=0;i<l;i++)
            s[l-i-1]=c[i]-'0';
        for (i=0;i<l;i++)
               {
               k=s[i]*m+add;
               if (k>=10) {s[i]=k%10;add=k/10;flag=1;} else
    {s[i]=k;flag=0;add=0;}
               }
        if (flag) {l=i+1;s[i]=add;} else l=i;
        for (i=0;i<l;i++)
            t[l-1-i]=s[i]+'0';
        t[l]='\0';
    }
```

3. 精度计算——乘法（大数乘大数）

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
const int L=110;
string mul(string a,string b)//高精度乘法a,b,均为非负整数
{
    string s;
    int na[L],nb[L],nc[L],La=a.size(),Lb=b.size();//na存储被乘数，nb存储乘数，nc存储积
    fill(na,na+L,0);fill(nb,nb+L,0);fill(nc,nc+L,0);//将na,nb,nc都置为0
    for(int i=La-1;i>=0;i--) na[La-i]=a[i]-'0';//将字符串表示的大整形数转成i整形数组表示的大整
形数
    for(int i=Lb-1;i>=0;i--) nb[Lb-i]=b[i]-'0';
    for(int i=1;i<=La;i++)
        for(int j=1;j<=Lb;j++)
        nc[i+j-1]+=na[i]*nb[j];//a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）
    for(int i=1;i<=La+Lb;i++)
        nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
    if(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
    for(int i=La+Lb-1;i>=1;i--)
        s+=nc[i]+'0';//将整形数组转成字符串
    return s;
}
int main()
{
    string a,b;
    while(cin>>a>>b) cout<<mul(a,b)<<endl;
    return 0;
}
```

4. 精度计算——除法 （大数除法包含取模）

```
#include<iostream>
```

```
#include<cstring>
#include<algorithm>
using namespace std;
const int L=110;
int sub(int *a,int *b,int La,int Lb)
{
    if(La<Lb) return -1;//如果a小于b，则返回-1
    if(La==Lb)
    {
        for(int i=La-1;i>=0;i--)
            if(a[i]>b[i]) break;
            else if(a[i]<b[i]) return -1;//如果a小于b，则返回-1

    }
    for(int i=0;i<La;i++)//高精度减法
    {
        a[i]-=b[i];
        if(a[i]<0) a[i]+=10,a[i+1]--;
    }
    for(int i=La-1;i>=0;i--)
        if(a[i]) return i+1;//返回差的位数
    return 0;//返回差的位数

}
string div(string n1,string n2,int nn)//n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数

{
    string s,v;//s存商,v存余数
     int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;//a，b是整形数组表示被除数，除数，
tp保存被除数的长度
     fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为0
     for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
     for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
     if(La<Lb || (La==Lb && n1<n2)) {
            //cout<<0<<endl;
     return n1;}//如果a<b,则商为0，余数为被除数
     int t=La-Lb;//除被数和除数的位数之差
     for(int i=La-1;i>=0;i--)//将除数扩大10^t倍
        if(i>=t) b[i]=b[i-t];
        else b[i]=0;
     Lb=La;
     for(int j=0;j<=t;j++)
     {
         int temp;
         while((temp=sub(a,b+j,La,Lb-j))>=0)//如果被除数比除数大继续减
         {
             La=temp;
             r[t-j]++;
         }
     }
     for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%=10;//统一处理进位
     while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
     while(i>=0) s+=r[i--]+'0';
```

```
    //cout<<s<<endl;
    i=tp;
    while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的</span>
    while(i>=0) v+=a[i--]+'0';
    if(v.empty()) v="0";
    //cout<<v<<endl;
    if(nn==1) return s;
    if(nn==2) return v;
}
int main()
{
    string a,b;
    while(cin>>a>>b) cout<<div(a,b,1)<<endl;
    return 0;
}
```

5. 精度计算——加法

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
const int L=110;
string add(string a,string b)//只限两个非负整数相加
{
    string ans;
    int na[L]={0},nb[L]={0};
    int la=a.size(),lb=b.size();
    for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
    if(na[lmax]) lmax++;
    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
    return ans;
}
int main()
{
    string a,b;
    while(cin>>a>>b) cout<<add(a,b)<<endl;
    return 0;
}
```

6. 精度计算——减法

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <algorithm>
#include <cmath>
```

```
#define N 1001
using namespace std ;
int main ( )
{
    int a [ N ] , b [ N ] , c [ N ] , i ;
    char n [ N ] , n1 [ N ] , n2 [ N ] ;
    memset ( a , 0 , sizeof ( a ) ) ;
    memset ( b , 0 , sizeof ( b ) ) ;
    memset ( c , 0 , sizeof ( c ) ) ;
    gets ( n1 ) ;
    gets ( n2 ) ;
    int lena = strlen ( n1 ) , lenb = strlen ( n2 ) ;
    if ( lena < lenb || ( lena == lenb && strcmp ( n1 , n2 ) < 0 ) )
    //strcmp()为字符串比较函数，当n1=n2时，返回0，
    //n1>n2时，返回正整数；n1<n2时返回负整数
    //比完大小后，发现被减数小于减数，就交换。
    {
        strcpy ( n , n1 ) ;  //将n1数组的值完全赋值给n数组
        strcpy ( n1 , n2 ) ;
        strcpy ( n2 , n ) ;
        swap ( lena , lenb ) ;  //这步不能忘
        printf ( "-" ) ;  //别忘了输出负号
    }
    for ( i = 0 ; i < lena ; i ++ ) a [ lena - i ] = int ( n1 [ i ] - '0' ) ;
    for ( i = 0 ; i < lenb ; i ++ ) b [ lenb - i ] = int ( n2 [ i ] - '0' ) ;
    i = 1 ;
    while ( i <= lena || i<= lenb )
    {
        if ( a [ i ] < b [ i ] )   //借位
        {
            a [ i ] += 10 ;
            a [ i + 1 ] -- ;
        }
        c [ i ] = a [ i ] - b [ i ] ;
        i ++ ;
    }
    int lenc = i ;
    while ( c [ lenc ] == 0 && lenc > 1 ) lenc -- ;    //最高位为0，则不输出
    for ( i = lenc ; i >= 1 ; i -- ) printf ( "%d" , c [ i ] ) ;
    return 0 ;
}
```

7. 高精度比较大小

```
#include<iostream>
#include<algorithm>
using namespace std;
main()
{
    string a,b;
    while(cin>>a>>b&&a!="0"&&b!="0")
    {
        bool check=true,same=true;
```

```
        int xa[1000]={},xb[1000]={};
        for(int i=0;i<a.length();i++)
            xa[i]=a[a.length()-i-1]-'0';
        for(int i=0;i<b.length();i++)
            xb[i]=b[b.length()-i-1]-'0';
        for(int i=max(a.length(),b.length());i>=0;i--)
        {
            if(xa[i]!=xb[i]) same=false;
            if(xa[i]<xb[i]) {check=false;break;}
            if(xa[i]>xb[i]) break;
        }
        if(same)
            cout<<"a==b"<<endl;
        else
            check?cout<<"a>b"<<endl:cout<<"a<b"<<endl;
    }
}
```

8. 高精度乘单精度

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
const int L=100005;
int na[L];
string mul(string a,int b)//高精度a乘单精度b
{
    string ans;
    int La=a.size();
    fill(na,na+L,0);
    for(int i=La-1;i>=0;i--) na[La-i-1]=a[i]-'0';
    int w=0;
    for(int i=0;i<La;i++) na[i]=na[i]*b+w,w=na[i]/10,na[i]=na[i]%10;
    while(w) na[La++]=w%10,w/=10;
    La--;
    while(La>=0) ans+=na[La--]+'0';
    return ans;
}
int main()
{
    string a;
    int b;
    while(cin>>a>>b) cout<<mul(a,b)<<endl;
    return 0;
}
```

9. 高精度除单精度

```
#include<iostream>
#include<algorithm>
```

```
using namespace std;
string div(string a,int b)//高精度a除以单精度b
{
    string r,ans;
    int d=0;
    if(a=="0") return a;//特判
    for(int i=0;i<a.size();i++)
    {
            r+=(d*10+a[i]-'0')/b+'0';//求出商
            d=(d*10+(a[i]-'0'))%b;//求出余数
    }
    int p=0;
    for(int i=0;i<r.size();i++)
    if(r[i]!='0') {p=i;break;}
    return r.substr(p);
}
int main()
{
    string a;
    int b;
    while(cin>>a>>b)
    {
        cout<<div(a,b)<<endl;
    }
    return 0;
}
```

10. 高精度幂

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
#include <map>
#include <queue>
#include <set>
#include <vector>
using namespace std;
#define L(x) (1 << (x))
const double PI = acos(-1.0);
const int Maxn = 133015;
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
char sa[Maxn/2],sb[Maxn/2];
int sum[Maxn];
int x1[Maxn],x2[Maxn];
int revv(int x, int bits)
{
    int ret = 0;
    for (int i = 0; i < bits; i++)
    {
        ret <<= 1;
```

```
        ret |= x & 1;
        x >>= 1;
    }
    return ret;
}
void fft(double * a, double * b, int n, bool rev)
{
    int bits = 0;
    while (1 << bits < n) ++bits;
    for (int i = 0; i < n; i++)
    {
        int j = revv(i, bits);
        if (i < j)
            swap(a[i], a[j]), swap(b[i], b[j]);
    }
    for (int len = 2; len <= n; len <<= 1)
    {
        int half = len >> 1;
        double wmx = cos(2 * PI / len), wmy = sin(2 * PI / len);
        if (rev) wmy = -wmy;
        for (int i = 0; i < n; i += len)
        {
            double wx = 1, wy = 0;
            for (int j = 0; j < half; j++)
            {
                double cx = a[i + j], cy = b[i + j];
                double dx = a[i + j + half], dy = b[i + j + half];
                double ex = dx * wx - dy * wy, ey = dx * wy + dy * wx;
                a[i + j] = cx + ex, b[i + j] = cy + ey;
                a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;
                double wnx = wx * wmx - wy * wmy, wny = wx * wmy + wy * wmx;
                wx = wnx, wy = wny;
            }
        }
    }
    if (rev)
    {
        for (int i = 0; i < n; i++)
            a[i] /= n, b[i] /= n;
    }
}
int solve(int a[],int na,int b[],int nb,int ans[])
{
    int len = max(na, nb), ln;
    for(ln=0; L(ln)<len; ++ln);
    len=L(++ln);
    for (int i = 0; i < len ; ++i)
    {
        if (i >= na) ax[i] = 0, ay[i] =0;
        else ax[i] = a[i], ay[i] = 0;
    }
    fft(ax, ay, len, 0);
    for (int i = 0; i < len; ++i)
```

```
    {
        if (i >= nb) bx[i] = 0, by[i] = 0;
        else bx[i] = b[i], by[i] = 0;
    }
    fft(bx, by, len, 0);
    for (int i = 0; i < len; ++i)
    {
        double cx = ax[i] * bx[i] - ay[i] * by[i];
        double cy = ax[i] * by[i] + ay[i] * bx[i];
        ax[i] = cx, ay[i] = cy;
    }
    fft(ax, ay, len, 1);
    for (int i = 0; i < len; ++i)
        ans[i] = (int)(ax[i] + 0.5);
    return len;
}
string mul(string sa,string sb)
{
    int l1,l2,l;
    int i;
    string ans;
    memset(sum, 0, sizeof(sum));
    l1 = sa.size();
    l2 = sb.size();
    for(i = 0; i < l1; i++)
        x1[i] = sa[l1 - i - 1]-'0';
    for(i = 0; i < l2; i++)
        x2[i] = sb[l2-i-1]-'0';
    l = solve(x1, l1, x2, l2, sum);
    for(i = 0; i<l || sum[i] >= 10; i++) // 进位
    {
        sum[i + 1] += sum[i] / 10;
        sum[i] %= 10;
    }
    l = i;
    while(sum[l] <= 0 && l>0)    l--; // 检索最高位
    for(i = l; i >= 0; i--)    ans+=sum[i] + '0'; // 倒序输出
    return ans;
}
string Pow(string a,int n)
{
    if(n==1) return a;
    if(n&1) return mul(Pow(a,n-1),a);
    string ans=Pow(a,n/2);
    return mul(ans,ans);
}
int main()
{
    cin.sync_with_stdio(false);
    string a;
    int b;
    while(cin>>a>>b) cout<<Pow(a,b)<<endl;
    return 0;
```

```
}
```

11. 任意进制转换

```
   进制的转换，用数组做了一个。 // 输入一个数 输入他的进制 转换为的进制
 语法: conversion(char a[],char b[],int n,int m);
       参数:
       a[]: 转换前的数字
       b[]: 转换后的数字
       n: 原进制数
       m: 需要转换到的进制数
       返回值: null
       注意:
        高于9的位数用大写'A'~'Z'表示，2~16位进制通过验证
       源程序:
#include<iostream>
#include<cstring>
#include<cmath>
using namespace std;
void conversation(char a[],char b[],int n,int m)
{
    int num=0,i,j,t;
    char c;
    int p=strlen(a)-1;
    for(i=0;a[i]!='\0';i++)
    {
        if(a[i]>='0'&&a[i]<='9')
        {
           t=a[i]-'0';
        }
        else
        {
            t=a[i]-'A'+10;
        }
    //  num=num*n+t;//这个和num+=t*pow(n,p);的效果是一样的，开始我用的是这个。
        num+=t*pow(n,p);//为了方便你们理解，我还是换成了这个
        --p;
    }
//  cout<<num<<endl;
    i=0;
    while(1)
    {
     if(num==0)break;
      t=num%m;
      if(t<=9)
      b[i]=t+'0';
      else
      b[i]=t+'A'-10;
      num/=m;
      i++;
    }
    for(j=i-1;j>=0;j--)
```

```
        cout<<b[j];
    for(j=0;j<=i/2;j++)
    {
       c=b[j];
       b[j]=b[i-j];
       b[i-j]=c;
    }
    b[i+1]='\0';
        cout<<endl;

}
int main()
{
    char a[100],b[100];
    int n,m;
   while(cin>>a>>n>>m)
   {
    conversation(a,b,n,m);
    for(int i=1;b[i]!='\0';i++)
        cout<<b[i];
    cout<<endl;
   }
    return 0;
}
```

12. 最大公约数，最小公倍数

```
最大公约数:
int hcf(int a, int b){
return b == 0 ? a : hcf(b, a % b);
}
最小公倍数:
lcd = a / hcf (a, b) * b;
```

13. 求排列组合数

```
 语法: result=P(long n,long m); / result=long C(long n,long m);
      参数:
      m: 排列组合的上系数
      n: 排列组合的下系数
      返回值: 排列组合数
      注意:
       符合数学规则: m<=n
      源程序:
       long P(long n,long m)
      {
          long p=1;
          while(m!=0)
              {p*=n;n--;m--;}
          return p;
      }
```

```
long C(long n,long m)
{
    long i,c=1;
    i=m;
    while(i!=0)
        {c*=n;n--;i--;}
    while(m!=0)
        {c/=m;m--;}
    return c;
}
```

14. 全排列

语法: void qp(int Array[],int begin,int end);
注意:当参与全排列的数字稍大的时候将会有很大的计算量

```
#include<iostream>
using namespace std;
const int MaxNum=20;
static int a[MaxNum];
void qp(int Array[],int begin,int end);
int main()
{
    int i;
    for(i=0;i<MaxNum;i++)
        a[i]=i+1;
    //初始化数组为:1,2,3..
    qp(a,0,10);
    return 0;
}
void qp(int Array[],int begin,int end)
{
    int i;
    if(begin>=end)
    {
        for(i=0;i<end;i++)
            cout<<Array[i]<<"\t";
        cout<<endl;
    }
    else for(i=begin;i<end;i++)
    {
        swap(a[begin],a[i]);
        qp(a,begin+1,end);
        swap(a[begin],a[i]);
    }
}
//stl里面的全排列生成函数next_permutation
#include<iostream>
#include <algorithm>
using namespace std;

int main()
```

```
{
    int i;
    int A[] = {0,1,2,3};
    while(next_permutation(A, A+4)==true)   //prev_permutation(A, A+4);

    {
        for(i=0;i<4;i++)
            cout<<A[i]<<"\t";
        cout<<endl;

    }
}
```

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
string str="abc";
while(next_permutation(str.begin(),str.end()))
cout<<str<<endl;
return 0;
}
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
vector<int> dp;
dp.push_back(1);
dp.push_back(2);
dp.push_back(3);
while(next_permutation(dp.begin(),dp.end())){
cout<<dp[0]<<dp[1]<<dp[2]<<endl;
}
return 0;
}
```

二、字符串处理

1.字符串标准库

C标准库（针对字符数组）

`int strlen(const char *str)`：返回从 `str[0]` 开始直到 `'\0'` 的字符数。

`sscanf(const char *__source, const char *__format, ...)`：从字符串 `__source` 里读取变量，比如 `sscanf(str,"%d",&a)`。

`sprintf(char *__stream, const char *__format, ...)`：将 `__format` 字符串里的内容输出到 `__stream` 中，比如 `sprintf(str,"%d",i)`。

`int strcmp(const char *str1, const char *str2)`：按照字典序比较 `str1 str2` 若 `str1` 字典序小返回负值，一样返回 0，大返回正值 请注意，不要简单的认为只有 `0，1，-1` 三种，在不同平台下的返回值都遵循正负，但并非都是 `0，1，-1`

`char *strcpy(char *str, const char *src)`：把 `src` 中的字符复制到 `str` 中， `str` `src` 均为字符数组头指针，返回值为 `str` 包含空终止符号 `'\0'`。

`char *strcat(char *str1, const char *str2)`：将 `str2` 接到 `str1` 的结尾，用 `*str2` 替换 `str1` 末尾的 `'\0'` 返回 `str1`。

C++标准库（针对字符串，兼容字符数组）

std::string 头文件string

- 访问运算符 `[cur]` 返回 `cur` 位置的字符（引用）。
- 访问函数 `data()/c_str()` 返回一个 `const char*` 指针，内容与该 `string` 相同。
- 容量函数 `size()` 返回字符串字符个数。
- `getline(cin, st1)` 读取字符直到换行，单纯的cin遇到空格停止
- `getline(cin, st1, 'a')` 一个直到'a'结束，其中任何字符包括'\n'都能够读入
- `st1.length()` 返回字符个数
- `st1 + st2` `st1 = st2` 可直接连接、替换
- `tolower(c)` `toupper(c)` 大小写转换（字符）
- `s.substr(i, j)` 截取s串中从i到j的子串
- `s.insert(it, p)` 把字符串p插入到it的位置
- `s.erase(3)` `s.earse(0, 4)` 删除对应下标或下标范围的元素
- `s.compare("good")` s与"good"比较相等返回0，比"good"大返回1，小则返回-1（字典序
- `reverse(s.begin(), s.end())` 字符串反转

2.字符串替换

使用STL的replace函数实现，结果是母串中所有相同的都被替换

用法：`string_replace(要替换的母串，要替换掉的子串，用来替换的字符串)`

如：`string_replace(st1, "Winter", "AutoWA")` 把st1中的winter换成AutoWA

```
void string_replace(string & strBig, const string & strsrc, const string &strdst) {
        string::size_type pos=0;
        string::size_type srclen=strsrc.size();
        string::size_type dstlen=strdst.size();
        while( (pos=strBig.find(strsrc, pos)) != string::npos){
                strBig.replace(pos, srclen, strdst);
                pos += dstlen;
        }
}
```

3.字符串查找

查找函数`find()`找到并返回某字符的位置

- `st1.find('a')`输出第一个'a'的下标（0开始），查找对象可以也是一个字符串
- `st1.find('a', 3)`从下标3开始查询a的下标，返回的下标同上
- `st1.rfind('a', 5)`从0到5从后向前查找a所在该串的位置
- `find()`函数返回的是一个size_t类型（或string::size_type类型）

  当没有匹配到目标的时候，返回的是str.npos

  例：定义一个size_t类型变量，可判断是否匹配到了

```
size_t position;
string s = "AutoWA";
position = s.find("WA");
cout << position << endl;  //输出4
position = s.find("ACM");
if(position == s.npos) cout << "None" << endl;  //未匹配到，输出None
```

- `st1.find_first_of(st2, 0)`在str1中从0开始向后查找，只要在st1中遇到一个字符与st2中任意一个字符相同，就返回该字符在st1中的位置，匹配失败返回npos
- `st1.find_last_of(st2, 4)`同上，从指定位置向前查找
- `st1.find_first_not_of(st2, 0)`同上，遇到一个字符与st2中任何一个不同，返回位置，last同理。

4.最长公共子序列（LCS）

给定两个字符串 str1 和 str2 ，如果将 str1 去掉一些字符后得到字符串str，将str2去掉一些字符后也能得到字符串 str ，则 str 是 str1 和 str2 的一个公共子串。在得到str的时候不能改变字符的相对顺序。求字符串 str 长度的最大值。

如 ABCEF BMCGUAFB，则最长公共子序列为3: BCF

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int dp[5005][5005];
    char a[5005],b[5005];
int main()
{

    cin>>a+1>>b+1;
    int n=strlen(a+1);
    int m=strlen(b+1);
    for(int i=1;i<=n;++i)
    for(int j=1;j<=m;++j)
    {
        if(a[i]==b[j])dp[i][j]=dp[i-1][j-1]+1;
        else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
```

```
    }
    cout<<dp[n][m];
}
```

三、排序/查找

1.快速排序 一般较快-取决于元素排列顺序

说明：quickSort(数组名，起始元素下标， 终止元素下标)，默认升序排列

```
void quickSort(int *arr, int left, int right){
    int i = left, j = right;
    int mid = arr[(i+j)/2];
    while(i <= j){
        while(arr[i] < mid) i ++;
        while(arr[j] > mid) j --;
        if(i <= j){
            int tmp;
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i ++;
            j --;
        }
    }
    if(i < right) quickSort(arr,i, right);
    if(left < j) quickSort(arr,left, j);
}
```

2.归并排序 稳定-时间复杂度固定

用法同上

```
int temp[max];
void merge(int *a, int left, int mid, int right) {
    int i = left, j = mid + 1, n = 0, length = right - left;
    while(i <= mid && j <= right) {
        if(a[i] < a[j]) {
            temp[n++] = a[i++];
        } else {
            temp[n++] = a[j++];
        }
    }
    if(i > mid) {
        while(j <= right) {
            temp[n++] = a[j++];
        }
    } else {
```

```
        while(i <= mid) {
            temp[n++] = a[i++];
        }
    }
    for(int k = 0;k <= length;++k) {
        a[left+k] = temp[k];
    }
}
void mergesort(int *a, int left, int right) {
    if(left < right) {
        int mid = (left + right) / 2;
        mergesort(a,left,mid);
        mergesort(a,mid+1,right);
        merge(a,left,mid,right);
    }
}
```

3.二分查找

说明：*arr为数组名， begin为起始下标，end为终止下标， e为要查找的内容，最终返回查找的内容的下标，如果没有找到则返回比它小的第一个数的下标

数组必须已经按照升序排列好

```
int bSearch(int *arr, int begin, int end, int e) {
    int mid, left = begin, right = end;
    while(left <= right) {
        mid = (left + right) >> 1;
        if(arr[mid] > e) right = mid - 1;
        else left = mid + 1;
    }
    return right;
}
```

4.STL排序

头文件依赖：`<algorithm>`

`sort(begin, end, cmp)` 范围是左闭右开

begin数组首地址，end数组尾地址，cmp自定义比较规则，默认为升序排序。

降序排列`sort(begin, end, greater<int>())`

或自定义比较方法

```
bool cmp(int a, int b) {
    return a > b;
}
```

结构体排序自定义cmp：

```
bool cmp(T a, T b) {
    return a.time > b.time;
}
```

四、常用STL整理

容器和配接器

list (链表)

list可以认为是一个线性的双向链表，具有链表的特性，不使用连续的内存空间，可以快速的插入和删除，不支持随机的内部访问。使用需包含`include<list>`头文件，`std`命名空间。

常用使用方法:
1.创建实例，迭代器

```
int a[] = { 1,2,3,4,5 };
list<int> lt;
list<int>::iterator it;//创建迭代器
list<int> lt(a, a + 5);
list<int> lt(2, 100);
```

2.输入输出，插入删除，迭代器遍历

```
//push_back,push_front(插入尾，插入头)
int a;
cin >> a;
lt.push_back(a);
lt.push_front(a);

//pop_back,pop_front(删除尾，删除头)
lt.pop_back();
lt.pop_front();

//assign(插入)
list<int> first;
list<int> second;
first.assgin(2,100);//添加2个100的元素
second.assgin(first.begin(),first.end())//将first拷贝给second

//insert(指定位置插入)
/*
iterator insert (iterator position, const value_type& val);
//position是要插入的这个list的迭代器，val是要插入的值
void insert (iterator position, size_type n, const value_type& val);
//从该list容器中的position位置处开始，插入n个值为val的元素
```

```
template <class InputIterator>
void insert (iterator position, InputIterator first, InputIterator last);
//first, last是我们选择的把值插入到这个list中的值所在的容器的迭代器
*/
list<int> lt;
list<int>::iterator it;
it = lt.begin();
lt.insert(it, 2);
lt.insert(it, 2, 100);
lt.insert(it, sth.begin(), sth.end());
//在指定位置插入某容器的一个区段

//遍历
list<int> lt;
list<int>::iterator it;
for (it = lt.begin(); it != lt.end(); it++) cout << *it;
```

3.常用成员函数

```
//erase(删除元素)erase函数是可以有返回值的，注意当删除元素的同时，迭代器也被销毁了。
lt.erase(iterator it);//删除it位置的元素
lt.erase(iterator begin,iterator end);//删除一定区间的元素

//swap(交换)
list<int> first;
list<int> second;
first.swap(second);

//clear(清空)
lt.clear();

//splice(转移元素)
/*
void splice (iterator position, list& x);
//将列表x中的所有元素移到当前list中，从当前列表的position指向的位置开始，此时列表x为空
void splice (iterator position, list& x, iterator i);
//将列表x中迭代器 i 指向的元素移到当前list的position指向的位置处，由于i指向的元素从列表x中被移，所以迭代器 i 此时是invalid的；position是当前列表的迭代器，i是列表x的迭代器
void splice (iterator position, list& x, iterator first, iterator last);
//将列表x中[first, last)的元素移到当前list中，从position指向的位置开始；first,last是列表x的迭代器
*/

//remove(移除指定元素)
/*
void remove (const value_type& val);
//从list中删除所有值为val的元素
*/
lt.remove(100);

//unique(删除重复值)
/*
void unique();
//只能删除相邻的重复元素，然后保留第一个值，因此这个函数只对排好序的list有用
```

```
*/

//sort(排序) 默认升序，可自写cmp函数
lt.sort(cmp);

//reverse(逆序)
lt.reserve();

//merge(合并有序的list)
list<int> first;
list<int> second;
first.merge(second);

//remove_if(按条件移除元素)
bool single_digit (const int& value) { return (value < 10); }
lt.remove_if (single_digit);
```

stack（栈）

stack就是一个标准的栈，后进先出，不能遍历。使用需包含`include<stack>`头文件，`std`命名空间。

常用使用方法：

```
int a;
stack<int> st;//创建实例
cin >> a;
st.push(a);//ru'zhan
a = st.top();//返回入栈
st.pop();出栈
bool b = st.empty () ;//判断栈是否为空
a = st.size();//返回栈长度
```

queue（队列）

queue先进先出，不能遍历。使用需包含`include<queue>`头文件，`std`命名空间。

```
int a;
queue<int> que;//创建实例
cin >> a;
que.push(a);//进队
a = que.front();//返回队头
a = que.back();//返回队尾
que.pop();//出队
bool b = que.empty();//队列是否为空
a = que.size();//返回队列长度
```

priority_queue（优先队列）

可以认为是队列的一种，但是会按照一种优先规律，将优先级最高的元素始终置于队头。底层通过heap（堆）来实现，所以默认为一个大根堆。

常用使用方法：

```
struct node
{
    int x,y;
    bool operator < (const node & a) const
    {
        return x<a.x;
    }
};


int a;
priority_queue <int> que;//创建实例，默认降序
priority_queue <int, vector<int>, greater<int> > que2;//升序
priority_queue <node>;//重载小于，可以利用重载小于来自定义优先级
priority_queue
cin >> a;
que.push(a);//进队
a = que.top();//返回队头
que.pop();//出队
bool b = que.empty();//队列是否为空
a = que.size();//返回队列长度
```

set（集合）

set内部通过红黑树实现，实现了一个自动排序，元素值唯一的容器。查找的复杂度为（logn），set中的元素值不能直接被修改，在其中的查找属于二分查找。使用需包含`include<set>`头文件，`std`命名空间。

常用使用方法：

1.创建实例，迭代器

```
set <int> se;
set <int>::iterator it;
```

2.插入删除(insert依然可以插入一段元素)

```
//insert(插入)
int a;
set <int> se;
cin >> a;
se.insert(a);
//erase(删除)
se.erase(iterator it);
```

3.常用成员函数

```
//find(查找某个值)
se.find(2);//返回2所在的迭代器，否则返回end()

//lower_bound(查找第一个大于等于key的值)upper_bound(查找第一个大于key的值)
se.lower_bound(2);
se.upper.bound(2);
```

4.mutiset（可重复插入的set)

```
mutiset <int> se;
//count(返回某一键值出现次数,set中使用此函数只会返回1或0)
int a = se.count(2);
```

vector（向量)

可以将其近似的认为是一个动态的数组。使用需包含`include<vector>`头文件，`std`命名空间。

常用使用方法： 1.创建实例，迭代器

```
vector <int> vec;
vector <int> vec(10,1);//创建了有10个元素的向量，并赋初值为1
int b[7]={1,2,3,4,5,9,8};vector<int> a(b,b+7);
vector <int>::iterator it;
```

2.插入删除，访问

```
//push_back(尾部插入一个元素)
vec.push_back(a);

//insert(插入一个元素)
vec.insert(position,elem)   //在pos位置插入一个elem拷贝，传回新数据位置。
vec.insert(position,n,elem) //在pos位置插入n个elem数据，无返回值。
vec.insert(position,begin,end)  //在pos位置插入在[beg,end)区间的数据，无返回值。

//erase(删除指定位置的元素)
vec.erase(iterator it);

//[]访问，由于vector重载了[]，所以可以利用[]直接访问已有元素
cout << vec[1];

//at(返回指定位置的元素)
vec.at(1);//at()函数具有检测是否越界的功能，如果越界会抛出错误，所以安全性高于[]

//迭代器访问
vector <int> vec;
vector <int>::iterator it;
for (it = vec.begin(); it != vec.end(); it++) cout << *it;
```

3.常用成员函数

```
//vec.clear();//清空

//vec.back(),vec.front(),vec.empty()//返回末尾，返回头，判断是否为空

//vec.begin(),vec.end()//传回对应位置的迭代器

//vector作为容器，可以使用较多algorithm中的函数，例如sort，reverse，swap。
```

map&&pair（关联）

map内部也是通过红黑树实现的，map的形式为一个键值对，和set一样查找的复杂度为（logn）可以修改value值，但不能修改key值。使用需包含`include<map>`头文件，`std`命名空间。

常用使用方法： 1.创建实例，迭代器

```
map <int,string> mp;//创建了一个以int为key，string为value的键值对。
map <int,string>::iterator it;
```

2.插入删除

```
//insert(插入)//注意前两种只能在map内无此元素的时候插入，而最后一种可以实现覆盖再赋值
mp.insert(make_pair(1, "one"));//利用make_pair函数构造出一对关联量插入
mp.insert(map<int, string>::value_type(1, "one"));//插入map的value_type数据
mp[1] = "one";//利用重载[]输入赋值

//earse(删除)
mp.erase (iterator it);//通过一个条目对象删除
mp.erase (iterator first, iterator last) //删除一个范围
int n = erase(con容器st Key&key);//通过关键字删除，删除成功n==1.否则==0
```

3.常用成员函数

```
//find(查找某个值)
mp.find(1);//返回key所在的迭代器，否则返回end()

//lower_bound(查找第一个大于等于key的值)upper_bound(查找第一个大于key的值)
mp.lower_bound(1);
mp.upper.bound(1);

//count(返回某一key值出现次数,map中使用此函数只会返回1或0)
int a = mp.count(1);
```

常用算法

#include<algorthm>（算法）

```
//sort(快速排序)stable_sort(稳定排序)
sort(start, end, 排序方法);
```

```
//reserve(反转容器)
reserve(vec.begin(),vec.end());

//lower_bound,upper_bound(二分查找)//返回的是位置,前提要有序
int num[6]={1,2,4,7,15,34};
sort(num,num+6);
int pos1=lower_bound(num,num+6,7)-num;     //返回数组中第一个大于或等于被查数的值
int pos2=upper_bound(num,num+6,7)-num;     //返回数组中第一个大于被查数的值

//集合操作(前提容器序列有序)
includes(s1.begin(), s1.end(), s2.begin(), s2.end());
//s1是否包含s2，递增序列使用less<int>(),递减序列使用greater<int>()。
set_union(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(s3));
//求并集，并输入到支持insert操作的s3中，也可以使用back_inserter(s3)输入到支持push_back操作的s3
set_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(s3));
//求交集
set_difference(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(s3));
//求差集

//堆操作
make_heap(begin(),end());//对一个序列建立一个堆，默认大根堆，greater<int>()则是小根堆
pop_heap(begin(),end());//将堆顶元素移到序列末尾，一般搭配pop_back();使用
push_heap(begin(), end());//有新元素插入序列末尾后的加入操作。
```