阶乘

```
#include<iostream>
#include<cstdio>
using namespace std;

int num[1000000], len;

void init() {
    len = 1;
    num[0] = 1;
}
int mult(int num[], int len, int n) {
    long long tmp = 0;
    for(long long i = 0;i < len;++i) {
        tmp = tmp + num[i] * n;
        num[i] = tmp % 10;
        tmp /= 10;
    }
    while(tmp) {
        num[len++] = tmp % 10;
        tmp /= 10;
    }
    return len;
}

int main() {
    int n;
    cin>>n;
    init();
    for(int i = 2;i <= n;++i) {
        len = mult(num,len,i);
    }
    for(int i = len - 1;i >= 0;i--) {
        printf("%d",num[i]);
    }
    return 0;
}
```

大数相加

```
string add(string s1, string s2) {
    if(s1 == "" && s2 == "") return "0";
    if(s1 == "") return s2;
    if(s2 == "") return s1;
    string maxx = s1, minn = s2;
    if(s1.length() < s2.length() ) {
        maxx = s2;

        minn = s1;
    }
    int a = maxx.length() - 1, b = minn.length() - 1;
    for(int i = b;i >= 0;i--) {
        maxx[a--] += minn[i] - '0';
    }
    for(int i = maxx.length() - 1;i > 0;i--) {
        if(maxx[i] > '9') {
            maxx[i] -= 10;
            maxx[i-1] ++;
        }
    }
    if(maxx[0] > '9') {
        maxx[0] -= 10;
        maxx = '1' + maxx;
    }
    return maxx;
}
```

快速幂

> sum = 2 ^ n sum的位数 n * log10(2) + 1

```
int qpow(long long  a, long long  b) {        //qpowmod(ll a, ll b, ll m)
    long long ans = 1, base = a;              //base = a % m
    while(b > 0) {
        if(b & 1) {
            ans *= base;                      //ans = ans * base % m
        }
        base *= base;                         //base = base * base % m
        b >>= 1;
    }
    return ans;                               //ans % m
}
```

快排

```
void quicksort(int *a, int left, int right) {
    int i = left, j = right;
    int mid = a[(i + j) / 2];
    while(i <= j) {
        while(a[i] < mid) i++;
        while(a[j] > mid) j--;
        if(i <= j) {
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
            i++;
            j--;
        }
    }
    if(i < right) quicksort(a,i,right);
    if(j > left) quicksort(a,left,j);
}
//quicksort(a,0,n-1)
```

归并排序

```
int temp[100000];
void mergesort(int *a, int left, int right){
    if(left == right) {
        return;
    }
    int mid = (left + right) / 2;
    mergesort(a,left,mid);
    mergesort(a,mid+1,right);
    int i = left, j = mid + 1,k = left;
    while(i <= mid && j <= right) {
        if(a[i] <= a[j]) {
            temp[k++] = a[i++];
        } else {
            temp[k++] = a[j++];
        }
    }
    while(i <= mid) {
        temp[k++] = a[i++];
    }
    while(j <= right) {
        temp[k++] = a[j++];
    }
    for(int i = left;i <= right; i++) {
        a[i] = temp[i];
    }
}
//mergesort(a,0,n-1)
```

冒泡排序

```
void bubblesort(int *a, int n) {
    int i, j, flag;
    for(i = 0;i < n - 1;++i) {
        flag = 0;
        for(j = 0;j < n - i - 1;++j) {
            if(a[j] > a[j+1]) {
                int t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                flag = 1;
            }
        }
        if(!flag) {
            break;
        }
    }
}
//bubblesort(a, n)
```

树状数组

```
int tree[500100], n;
int lowbit(int x){
    return x & -x;
}
void add(int x,int k){
    while(x<=n){
        tree[x]+=k;
        x+=lowbit(x);
    }
}
int sum(int x){
    int ans=0;
    while(x!=0){
        ans+=tree[x];
        x-=lowbit(x);
    }
    return ans;
}
```

> 1 单点更新与区间求和

```
int main() {
    cin>>n>>m;
    for(int i=1;i<=n;i++) {
        int a;
        scanf("%d",&a);
        add(i,a);
    }
    for(int i = 1;i <= m;i++) {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        if(a==1)
            add(b,c);//将某一个数加上c
        if(a==2)
            cout<<sum(c)-sum(b-1)<<endl; //求出某区间每一个数的和
    }
    return 0;
}
```

> 2 区间更新与单点求值

```
int main(){
    cin>>n>>m;
    int now = 0;
    for(int i=1;i<=n;i++) {
        int a;
        cin>>a;
        add(i,a-now);
        now = a;
    }
    for(int i=1;i<=m;i++){
        int a;
        scanf("%d",&a);
        if(a==1){
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            add(x,z);     //将某区间
            add(y+1,-z);  //每一个数数加上x
        }
        if(a==2){
            int x;
            scanf("%d",&x);
            printf("%d\n",sum(x)); //求出某一个数的值
        }
    }
    return 0;
}
```

快速读入

```
inline int read() {
    register int x = 0, f = 1;
    char c = getchar();
    while(c < '0' || c > '9') {
        if(c == '-') f = -1;
        c = getchar();
    }
    while(c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * f;
}
```

快速输出

```
inline void write(register int x) {
    if(x < 0) {
        putchar('-');
        x = -x;
    }
    if(x > 9) {
        write(x / 10);
    }
    putchar(x % 10 + '0');
}
```

优先队列

```
#include<queue>  //p.top()
priority_queue <int, vector<int>, greater<int> > p;//从小到大
priority_queue <int> p; //从大到小
```

> 结构体_priority

```
#include<queue>
struct Node{
    int value;
    int key;
}p[10];
struct cmp{
    bool operator()(Node a,Node b){
        if(a.key == b.key){
            return a.value < b.value;
        }
        return a.key < b.key;      //注意与sort分开
    }
};
priority_queue<Node,vector<Node>,cmp> heap; //按第一关键字 从大到小排序
```

> 实现优先队列

```
#include <vector>
template <class type>
class priority_queue {
    private:
        vector<type> data;
    public:
        void push(type t){
            data.push_back(t);
            push_heap( data.begin(), data.end());
        }

        void pop(){
            pop_heap( data.begin(), data.end() );
            data.pop_back();
        }
        type top() { return data.front(); }
        int size() { return data.size(); }
        bool empty() { return data.empty(); }
};
```

逆序数

> 给定一个数组A[0...N-1]，若对于某两个元素a[i]、a[j]，若i < j且a[i] > a[j]，则称(a[i],a[j])为逆序对。一个数组中包含的逆序对的数目称为该数组的逆序数。

```
while(i <= mid && j <= right) {
        if(a[i] < a[j]) {
            temp[n++] = a[i++];
        } else {
            count += (mid - i + 1);   //修改归并排序的merge函数
            temp[n++] = a[j++];
        }
    }
//特别注意当输入多组数的时候count要初始化为0
```

最大公因数

```
int gcd(int a, int b) {
    return a == 0 ? b : gcd(b % a, a);
}
```

全排列

```
#include<iostream>
#include<algorithm>
using namespace std;
int main() {
    int ans[4]={1,2,3,4};
    sort(ans,ans+4);    /* 这个sort可以不用，因为{1，2，3，4}已经排好序*/
    do                  /*注意这步，如果是while循环，则需要提前输出*/
    {
        for(int i=0;i<4;++i)
            cout<<ans[i]<<" ";
        cout<<endl;
    }while(next_permutation(ans,ans+4));
    return 0;
}
```

```
void perm(int *a, int low, int high) {
    if(low == high) {
        for(int i = 0;i <= low;++i) {
            printf("%d ",a[i]);
        }
        printf("\n");
    } else {
        for(int i = low;i <= high;++i) {
            swap(a[i],a[low]);
            perm(a,low+1,high);
            swap(a[i],a[low]);
        }
    }
}
//perm(a,0,n-1)
```

二分搜索

```
int binary_search(int* a, int len, int goal) {
    int low = 0;
    int high = len;
    while (low < high) {
        int middle = (high - low) / 2 + low; // 直接使用(high + low) / 2 可能导致溢出
        if (a[middle] == goal) {
            return middle;
        } else if (a[middle] > goal) {
            high = middle - 1;
        } else {
            low = middle + 1;
        }
    }
    return -1;
}
```

并查集

```
int f[10010];
int find(int k){
    if(f[k] == k) {
        return k;
    }
    return f[k] = find(f[k]);
}
void join(int x,int y) {
    int fx = find(x);
    int fy = find(y);
    f[fx] = fy;
}
for(int i = 1;i <= 100;++i) {
    f[i] = i;
}
//切记  f数组一定要初始化
```

前式链向星

```
#include<iostream>
#include<string.h>
using namespace std;
#define MAXN 100501
struct NODE{
    int w;
    int to;
    int next; //next[i]表示与第i条边同起点的上一条边的储存位置
}edge[MAXN];
int cnt = 1;
int head[MAXN];
void add(int u,int v,int w){
    edge[cnt].w=w;
    edge[cnt].to=v;    //edge[i]表示第i条边的终点
    edge[cnt].next=head[u]; //head[i]表示以i为起点的最后一条边的储存位置
    head[u]=cnt++;
}
int main(){
    memset(head,0,sizeof(head));  //重点
    cnt=1;
    int n, m;
    cin>>n>>m;
    int a,b,c;
    for(int i = 1;i <= m;++i) {
        cin>>a>>b>>c;
        add(a,b,c);
    }
    for(int i = 1;i <= n;++i) {
        for(int j = head[i];j;j = edge[j].next ) {
            cout<<i<<"->"<<edge[j].to <<" "<<edge[j].w ;
            cout<<endl;
        }
    }
    return 0;
}
```

最小生成树

> kruskal

```
int kruskal(int m, int n) { //m： 边的个数  n： 顶点数
    int num = 0, ans = 0;
    sort(p,p+m,cmp);
    for(int i = 0;i < m;i++) {
        int fu = find(p[i].u);
        int fv = find(p[i].v);
        if(fu == fv) {
            continue;
        }
        ans += p[i].w;
        f[fu] = fv;
        if(++num == n - 1) { //已连边个数是点个数-1时，停止循环，最小生成树完成
            break;
        }
    }
    return ans;
}
```

大根堆

```
int size = 0, heap[1000000];
void push(int e){
    heap[++size] = e;
    int son = size, father = son / 2;
    while(heap[son] > heap[father] && father >= 1){
        swap(heap[son],heap[father]);
        son = father,father = son / 2;
    }
}
void pop(){
    swap(heap[1],heap[size]);
    heap[size--]=0;
    int father = 1,son = 2;
    while(son <= size){
        if(son < size && heap[son] < heap[son+1]) son++;
        if(heap[father] < heap[son]){
            swap(heap[father],heap[son]);
            father = son, son = father * 2;
        }else break;
    }
}
int top(){
    return heap[1];
}
```

线性筛

```
bool is_prime[10000001];
int prime[10000001], cnt = 0;
void getprime(int n) {      // cnt  质数个数
                            // prime 存的是质数 2 3 5 7
                            // is_prime 存的是这个数是不是素数 真就是素数
    memset(is_prime, 1, sizeof(is_prime));
    is_prime[1] = 0;
    is_prime[0] = 0;
    for(int i = 2; i <= n; i++) {
        if(is_prime[i]) {
             prime[cnt++] = i;
        }
        for(int j = 0; j < cnt && i * prime[j] <= n; j++)  {
            is_prime[ i * prime[j]] = 0;
            if(i % prime[j] == 0) {
                break;
            }
        }
    }
}
```

栈

```
class _stack {
    private:
        int *top;
        int *base;
        int stacksize;
    public:
        int init();
        int _push(int e);
        int _pop();
        int _top();
        int _empty();
        int getlen();
        int destory();
};
int _stack::init() {
    base = (int*)malloc(sizeof(int));
    if(!base) {
        exit(0);
    }
    top = base;
    stacksize = 1;
    return 1;
}
int _stack::_push(int e) {
    int *p;
    if(top - base >= stacksize) {
        p = (int*)realloc(base,(stacksize + 1) * sizeof(int));
        if(!p) {
            exit(0);
        }
        base = p;
        top = base + stacksize;
        stacksize++;
    }
    *top = e;
    top++;
    return 1;

}
```

```
int _stack::_pop() {
    if(top == base) {
        return 0;
    }
    --top;
}
int _stack::_top() {
    if(base == top) {
        return 0;
    }
    return *(top - 1);
}
int _stack::_empty() {
    return top == base;
}
int _stack::getlen() {
    return top - base;
}
int _stack::destory() {
    if(!stacksize) {
        exit(0);
    }
    free(base);
    stacksize = 1;
    return 1;
}
```

队列

```
struct node {
    int data;
    struct node* next;
};
class linkqueue {
    private:
        node *front;
        node *rear;
    public:
        int init();
        int _push(int e);
        int _pop();
        int _top();
        int _empty();
        int destory();
};
int linkqueue::init() {
    front = (node*)malloc(sizeof(node));
    if(!front) {
        exit(0);
    }
    front -> next = NULL;

    rear = front;
    return 1;
}
int linkqueue::_push(int e) {
    node *p;
    p = (node*)malloc(sizeof(node));
    if(!p) {
        exit(0);
    }
    p -> data = e;
    p -> next = NULL;
    rear -> next = p;
    rear = p;
    return 1;
}
int linkqueue::_pop() {
    node *p;
    if(front == rear) {
        return 0;
    }
    p = front -> next;
    front -> next = p -> next;
    if(p == rear) {
        rear = front;
    }
    free(p);
    return 1;
}
int linkqueue::_top() {
    if(front == rear) {
        return 0;
    }
    return front -> next -> data;
}
int linkqueue::_empty() {
    return front == rear;
}
int linkqueue::destory() {
    while(front) {
        rear = front -> next;
        free(front);
        front = rear;
    }
}
```

π acos(-1.0)

不用加减乘除做加法

```
int p(int a,int b) {
    if(b == 0) { //如果b(进位)是0(没有进位了)，返回a的值
        return a;
    } else{
        int x, y;
        x = a ^ b; //x是a和b不进位加法的值
        y = (a & b) << 1;//y是a和b进位的值(左移一位是进位加在左面一位)
        return p(x, y);//把不进位加法和进位的值的和就是结果
    }
}
```

文件操作

> **创建并打开一个文本文档 -> 左上角文件 另存为 -> 下面选择所有文件 -> 输入文件名**
>
> **data.in 里要提前保存样例 输出结果在data.out中**

```
#include<cstdio>
#define begin
int main() {
    #ifdef begin
    freopen("data.in", "r", stdin);     //重定向版本
    freopen("data.out", "w", stdout);  // 只有定义了begin才执行这两条语句
    #endif
    int x;
    int s = 0;
    while(scanf("%d", &x) == 1) {
        s += x;
    }
    printf("%d",s);
    return 0;
}
```

比赛中要求用文件输入输出 但禁止用重定向版本

```
#include<stdio.h>
int main() {
    FILE *fin, *fout;
    fin = fopen("data.in", "rb"); //fopen版本
    fout = fopen("data.out", "wb");
    int x;
    int s = 0;
    while(fscanf(fin, "%d", &x) == 1) {
        s += x;
    }
    fprintf(fout, "%d", s);
    fclose(fin);
    fclose(fout);
    return 0;
}
```

#include

> 对于这两个算法，它们所查找的序列都必须是有序的 (返回值是下标的地址)
>
> upper_bound(a, a + n, num) : 在前两个参数定义的范围内查找大于第三个参数的第一个元素
>
> lower_bound(a, a + n, num) : 在前两个参数指定的范围内查找不小于第三个参数的第一个元素
>
> (upper_bound(a,a+n, num) - a) 下标

二分查找求上界的函数：

当 v 存在时返回它最后出现的下一个位置。如果不存在，返回这样一个下标i ：在此处插入 v后数列仍然有序

```
int up_bound(int *A, int x, int y, int v)
{
    int m;
    while(x < y)
    {
        m = x + (y-x)/2;
        if(A[m] <= v)
            x = m+1;
        else
            y = m;
    }
    return x;
}
```

二分查找求下界的函数：

当 v 存在时返回它出现的第一个位置。如果不存在，返回这样一个下标i：*在此处插入 v后数列仍然有序。*

```
int low_bound(int *A, int x, int y, int v)
{
    int m;
    while(x < y)
    {
        m = x + (y-x)/2;
        if(A[m] >= v)
            y = m;
        else
            x = m+1;
    }
    return x;
}
```

> 1、set迭代器与map的不同：(1)set使用接引用运算符*取值，而map使用first和second取值。 (2)set的迭代器都是常量迭代器，不能用来修改所指向的元素， 而map的迭代器是可以修改所指向元素的。 2、set没有重载[]运算符，而map中重载了，因为直接使用[]改变元素值会打乱原本正确的顺序， 要改变元素值必须先删除旧元素，则插入新元素

#include

```
#include<iostream>
#include<map>
#include<cstring>
#include<cstdio>
using namespace std;
typedef map <int, string> mydata;
mydata stu;
int main() {
    int n;
    cin>>n;
    //map中的元素是自动按Key升序排序，所以不能对map用sort函数
    /*
    用insert函数插入数据,当map中有这个关键字时,insert操作是插入数据不了的，
    但是用数组方式就不同了,它可以覆盖以前该关键字对应的值，
    for(int i = 0;i < n;++i) {
        string s;
        cin>>s;
        stu.insert(pair<int, string>(i,s));
    }
    */
    for(int i = 0;i < n;++i) {
        char s[100];
        int a;
        scanf("%s",s);
        cin>>a;
        stu[a] = s;
    }

    //mydata::iterator it = stu.find(1); //删除
    //stu.erase(it);
    //stu.clear() ;
    //stu.size();
    //stu.empty();

    mydata::reverse_iterator rit; //定义反向迭代器
    for(rit = stu.rbegin(); rit != stu.rend(); rit++)
        cout<<rit->first<<" "<<rit->second<<endl;

    cout<<endl;

    mydata::iterator it1 = stu.begin() ;////定义前向迭代器
    for(it1; it1 != stu.end() ;++it1) {
        cout<<it1->first<<" "<<it1->second<<endl;
    }
    int findid = 2;
    mydata::iterator it = stu.find(findid);//查找键值为2的元素
    if(it != stu.end() )
        cout<<it->first<<" "<<it->second<<endl;

    return 0;
}
```

#include

（1）正常

```
#include<iostream>
#include<set>
using namespace std;
int main() {
    //不允许重复
    //键和值是同一个元素
    set<int> s;
    s.insert(1);
    s.insert(3);
    s.insert(6);
    //s.erase(6);  //删除
    //s.size();
    //s.clear();
    //s.empty();
    // STL中区间是左闭右开，end()函数返回迭代器即指向集合中最大元素下一位置迭代器
    //--s.end()最大元素迭代器
    //最大值 : maxn = *(--s.end());
    set<int>::iterator it; //定义前向迭代器
    for(it = s.begin(); it != s.end(); it++)  cout << *it << " ";

    cout << endl;

    set<int>::reverse_iterator rit; //定义反向迭代器
    for(rit = s.rbegin(); rit != s.rend(); rit++) cout << *rit << " ";

    set<int>::iterator it;
    it = s.find(6); //查找键值为6的元素
    if(it != s.end())  cout << *it << endl;
    return 0;
}
```

（2）自定义比较函数

```
struct cmp { //自定义比较函数
    bool operator() (const int &a, const int &b) { //从大到小
        if(a != b)
            return a > b;
        else
            return a > b;
    }
};
set<int, cmp> num;
```

（3）结构体

```
#include<iostream>
#include<set>
#include<string>
using namespace std;
struct stu {
    int math;
    double score;
};
bool operator < (const stu &a, const stu &b) {
        if(a.math == b.math) {    //按第一关键字：数学 由大到小
            return a.score > b.score;
        }
        return a.math > b.math;
}
int main() {
    set<stu> s;
    stu a;

    a.math = 1;
    a.score = 2;
    s.insert(a);

    a.math = 3;
    a.score = 3;
    s.insert(a);

    a.math = 1;
    a.score = 4;
    s.insert(a);

    set<stu>::iterator it;
    for(it = s.begin(); it != s.end(); it++)
        cout << (*it).math << " " << (*it).score << endl;
    return 0;
}
```

#include

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

int main() {
    /* 二维数组
    vector<vector<int> > a;
    a.resize(m);//行数为m
    for(int i=0;i<m;i++)//每行列数为n
        a[i].resize(n);


    */
    vector<int> v1;//   创建一个空vector容器
    vector<int> v2(10);  //创建一个有10个空元素的vector容器
    vector<int> v3(10,0); //创建一个有10个元素的vector容器，并赋值为0；
    //使用时加上  *   迭代器法遍历
    //下标遍历

    vector<int> v4(10,0);




    /*
    v4.clear()
    v4.empty()
    v4.erase(iter)
    v4.erase(beg,end)//删除beg end区间的数据
    v4.front()//传回第一个数据
    v4.insert(iter,elem)
    v4.pop_back()//删除最后一个数据
    v4.push_back(elem)//在尾部加入一个数据
    v4.resize(num)//改容器的大小
    v4.size()
    v4.begin()
    v4.end()
    */

    return 0;
}
```

Huffman 编码

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

```
//Huffman树的节点类
typedef struct Node
{
    char value;               //结点的字符值
    int weight;               //结点字符出现的频度
    Node *lchild,*rchild;     //结点的左右孩子
}Node;

//自定义排序规则，即以vector中node结点weight值升序排序
bool ComNode(Node *p,Node *q)
{
    return p->weight < q->weight;
}

//构造Huffman树，返回根结点指针
Node* BuildHuffmanTree(vector<Node*> vctNode)
{
    while(vctNode.size()>1) //vctNode森林中树个数大于1时循环进行合并
    {
        sort(vctNode.begin(),vctNode.end(),ComNode);  //依频度高低对森林中的树进行升序排序

        Node *first=vctNode[0];    //取排完序后vctNode森林中频度最小的树根
        Node *second=vctNode[1];   //取排完序后vctNode森林中频度第二小的树根
        Node *merge=new Node;      //合并上面两个树
        merge->weight = first->weight + second->weight;
        merge->lchild = first;
        merge->rchild = second;
        vector<Node*>::iterator iter;
        iter=vctNode.erase(vctNode.begin(),vctNode.begin()+2);    //从vctNode森林中删除上诉频度最
小的两个节点first和second
        vctNode.push_back(merge);                                 //向vctNode森林中添加合并后的
merge树
    }
    return vctNode[0];            //返回构造好的根节点
}

//用回溯法来打印编码
void PrintHuffman(Node *node,vector<int> vctchar)
{
    if(node->lchild==NULL && node->rchild==NULL)
    {//若走到叶子节点，则迭代打印vctchar中存的编码
        cout<<node->value<<": ";
        for(vector<int>::iterator iter=vctchar.begin();iter!=vctchar.end();iter++)
            cout<<*iter;
        cout<<endl;
        return;
    }
    else
    {
        vctchar.push_back(1);     //遇到左子树时给vctchar中加一个1
        PrintHuffman(node->lchild,vctchar);

        vctchar.pop_back();       //回溯，删除刚刚加进去的1
        vctchar.push_back(0);     //遇到左子树时给vctchar中加一个0
        PrintHuffman(node->rchild,vctchar);
        vctchar.pop_back();       //回溯，删除刚刚加进去的0

    }
}

int main()
{
    //a b c d e
    //12 34 56 6 73
    cout<<"************ Huffman编码问题 ***************"<<endl;
    cout<<"请输入要编码的字符,并以空格隔开（个数任意）："<<endl;
    vector<Node*> vctNode;         //存放Node结点的vector容器vctNode
    char ch;                       //临时存放控制台输入的字符
    while((ch=getchar())!='\n')
    {
        if(ch==' ')continue;      //遇到空格时跳过，即没每入一个字符空一格空格
        Node *temp=new Node;
        temp->value=ch;
        temp->lchild=temp->rchild = NULL;
        vctNode.push_back(temp);  //将新的节点插入到容器vctNode中
    }

    cout<<endl<<"请输入每个字符对应的频度，并以空格隔开："<<endl;
    for(int i=0;i<vctNode.size();i++)
        cin>>vctNode[i]->weight;

    Node *root = BuildHuffmanTree(vctNode);  //构造Huffman树，将返回的树根赋给root
    vector<int> vctchar;
    cout<<endl<<"对应的Huffman编码如下："<<endl;
    PrintHuffman(root,vctchar);

    system("pause");
}
```