情人节：

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

int main()
{
    vector<string> v;//每次读入的字符串
    string s;
    while(cin>>s)//不断读入字符串
    {
        if(s==".") break;//当读入的字符串是"."
        v.push_back(s);
    }
    if(v.size()>=14)
        cout << v[1] << " and " << v[13] << " are inviting you to dinner...";
    else if(v.size()>=2)
        cout << v[1] << " is the only one for you...";
    else
        cout << "Momo... No one is for you ...";
    return 0;
}
```

吃火锅：

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

int main()
{
    int num = 0, cnt = 0;
    int start = 0;
    string s;
    while(1)
    {
        getline(cin, s);
        if(s==".")
            break;
        num++;
        if(s.find("chi1 huo3 guo1")!=string::npos)
        {
            if(start==0) start = num;
            cnt++;
        }
    }

    cout << num << endl;
    if(cnt==0) cout << "-_-#" << endl;
    else cout << start << " " << cnt;
}
```

点赞：

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

int main()
{
    int n, label; //用户点赞博文的数量
    cin >> n;
    int a[1001] = {0}; //a[i]:标签为i的出现次数
    for(int i=0; i<n; i++)  //有n行
    {
        int k;
        cin >> k;
        for(int j=0; j<k; j++)  //每行有k个数
        {
            cin >> label;
            a[label]++;  //数组下标由1开始
        }
    }
    label = 1; //label:出现最多的标签,假设1号标签出现最多
    for(int i=2; i<=1000; i++)//从2号开始比较
        if(a[i] >= a[label])
            label = i;

    cout << label << " " << a[label];

    return 0;
}
```

N个数求和：

测试点5是答案为0的情况
可使用 algorithm 库的 __gcd()函数
读一个求和一个，结果是suma/sumb
如何求和？分母通分后分子求和，求和后再约分
注意不同情况下的输出格式

```
#include<iostream>
#include<algorithm>
//#include<cmath>
using namespace std;

void sum(int a, int b, int &suma, int &sumb);
int lcd(int x, int y);
void output(int a, int b);

int main()
{
    int a, b, suma=0, sumb=1;
    int n;
    cin >> n;
    int i;
    char c;
```

```
    for(i=0; i<n; i++)
    {
        cin >> a >> c >> b;
        sum(a, b, suma, sumb);
    }
    output(suma, sumb);
}

void sum(int a, int b, int &suma, int &sumb)
{
    int l = lcd(b, sumb);//求两个分母的最小公倍数
    suma = suma*(l/sumb) + a*(l/b);
    sumb = l;
    //cout << suma << " " << sumb << endl;

    int g = __gcd(abs(suma), sumb);//求suma与sumb的最大公约数
    suma = suma/g;
    sumb = sumb/g;
}

int lcd(int x, int y)
{//求x与y的最小公倍数
    return x*y/__gcd(x, y);
}

void output(int a, int b)
{//按格式要求输出分子为a，分母为b 的数
    if(a<0)
    {
        cout << "-";
        a = -a;
    }
    if(a==0)
        cout << "0" << endl;
    if(a/b>0)
    {
        cout << a/b;
        if(a%b!=0)
            cout << " ";
    }
    if(a%b!=0)
        cout << a%b << "/" << b << endl;
}
```

阅览室：

注意干扰因素：借还没有匹配，或者多次借/还同一本书

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    while(n--)
    {
```

```
        int a[1001];
        fill(a, a+1001, -1);
        int cnt = 0, sum = 0;
        while(1)
        {
            int x, h, m;
            char c;
            scanf("%d %c %d:%d", &x, &c, &h, &m);
            if(x==0)
            {
                printf("%d %d\n", cnt, cnt==0?0:int(ceil(1.0*sum/cnt)));
                //printf("%d %d\n", cnt, cnt==0?0:int(round(1.0*sum/cnt)));
                break;
            }
            else if(c=='S') a[x] = h*60+m;
            else if(c=='E' && a[x]!=-1) cnt++, sum+=h*60+m-a[x], a[x]=-1;
        }
    }
    return 0;
}
```

古风排版：

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

int main()
{
    int k;
    string s;
    cin >> k;
    getchar();
    getline(cin, s);
    int i, j;
    for(int i=0; i<k; i++)//输出k行
    {
        for(int j=(s.size()-1)/k; j>=0; j--)//每行
            if(j*k+i < s.size()) cout << s[j*k+i];
            else cout << " ";
        cout << endl;
    }
}
```

这是二叉搜索树吗：

先判断本身是否二叉搜索树，不是再判断其镜像是否
使用函数ju()判断本身或其镜像，参数mirror用于标记判断本身还是判断镜像。参数s用于返回本次调用后产生的后序序列。
一棵先序序列的树是二叉搜索树的条件：左子树都小于根&&右子树都大于等于根&&左子树是二叉搜索树 && 右子树是二叉搜索树。后两步可使用递归完成。
后序遍历如何产生？递归过程中产生：左子树先序遍历序列+右子树先序遍历序列+根。前面两个序列通过递归调用产生。

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

bool ju(int a[], int l, int r, bool mirror, string &s)
{//判断a[l:r]这部分数据构成的子树或其镜像是否二叉搜索树。s用于返回本次遍历产生的先序序列。
    if(r-l<=0)
    {
        if(r==l) s = to_string(a[l]);//只剩一个结点
        else s = "";
        return true;
    }
    int i, j;
    if(mirror)//镜像
    {
        for(i=l+1; i<=r && a[i]>=a[l]; i++);
        for(j=i; j<=r && a[j]<a[l]; j++);
    }
    else
    {
        for(i=l+1; i<=r && a[i]<a[l]; i++);
        for(j=i; j<=r && a[j]>=a[l]; j++);
    }
    if(j<=r) return false;
    string s1="", s2="";
    if(ju(a, l+1, i-1, mirror, s1) && ju(a, i, r, mirror, s2))
    {
        if(s1!="") s1 += " ";
        if(s2!="") s2 += " ";
        s = s1 + s2 + to_string(a[l]);
        return true;
    }
    else return false;
}

int main()
{
    int n;
    cin >> n;
    int a[1000];
    for(int i=0; i<n; i++) cin >> a[i];
    string res1 = "", res2 = "";
    //false：本身  true：镜像
    if(ju(a, 0, n-1, false, res1)) cout << "YES\n" << res1;
    else if(ju(a, 0, n-1, true, res2)) cout << "YES\n" << res2;
    else cout << "NO\n";
    return 0;
}
```

红色警报：

这个没啥，就是每被攻占一个城市就重新计算一次连通分量数。如果分量数比上一次多了不止一个，就说明Red Alert
为什么本次可以比上次多一个？因为某城市被占领后，它会独立为一个连通分量
使用dfs(t)深搜从t开始的每个连接点。使用 fun()多次调用dfs(), 调用次数就是连通分量数。

```
//不断计算连通分量
```

```cpp
#include<bits/stdc++.h>
using namespace std;

bool vis[500] = {0};
int n;
int a[500][500] = {0};

void dfs(int t)
{
    vis[t] = 0;
    for(int i=0; i<n; i++)
        if(a[t][i]==1 && vis[i]==1)
            dfs(i);
}

int fun()
{
    for(int i=0; i<n; i++) vis[i] = 1;
    int cnt = 0;
    for(int i=0; i<n; i++)
    {
        if(vis[i]==1)
        {
            cnt++;
            dfs(i);
        }
    }
    return cnt;
}

int main()
{
    int m;
    cin >> n >> m;
    for(int i=0; i<m; i++)
    {
        int x, y;
        cin >> x >> y;
        a[x][y] = a[y][x] = 1;
    }
    int cnt = fun();
    int k;
    cin >> k;
    for(int i=0; i<k; i++)
    {
        int x;
        cin >> x;
        for(int j=0; j<n; j++) a[x][j] = a[j][x] = 0;
        int tmp = fun();
        //被删的城市本身成了一个连通分量
        if(tmp>cnt+1) printf("Red Alert: City %d is lost!\n", x);
        else printf("City %d is lost.\n", x);
        cnt = tmp;
        if(i==n-1) cout << "Game Over.";
    }
    return 0;
}
```

单身狗：

使用a数组存储伴侣关系，下标表示ID号。
因为要按照ID 递增顺序列出落单的客人，所以使用b数组标记要判断的客人，下标表示ID号。回头顺序遍历b数组并判断就行。
测试点2是全0的问题

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

int main()
{
    int n;
    cin >> n;
    int a[100000];
    for(int i=0; i<100000; i++) a[i] = -1;
    for(int i=0; i<n; i++)
    {
        int x, y;
        cin >> x >> y;
        a[x] = y, a[y] = x;
    }
    int b[100000] = {0};
    cin >> n;
    for(int i=0; i<n; i++)
    {
        int x;
        cin >> x;
        b[x] = 1;
    }
    int cnt = 0;
    for(int i=0; i<=99999; i++)
        if(b[i] && (a[i]==-1 || b[a[i]]==0)) cnt++, b[i]=-1;

    if(cnt==0) {cout << 0; return 0;}
    cout << cnt << endl;
    int flag = 0;
    for(int i=0; i<=99999; i++)
        if(b[i]==-1)
        {
            if(flag) printf(" %05d", i);
            else printf("%05d", i);
            flag = 1;
        }
    return 0;
}
```

插入排序还是归并排序：

先判断是否插入排序，不是再判断是否归并排序
如果是插入排序产生的中间序列：前一截一定是非降序，剩下那截一定跟原来未排序的序列一致。
归并的判断我也没有好办法，只能归并一次对比一次，有点傻

```
#include<iostream>
```

```cpp
#include<algorithm>
using namespace std;

bool equal(int a[], int b[], int n)
{
    for(int i=0; i<n; i++)
        if(a[i]!=b[i]) return false;
    return true;
}

void output(int a[], int n)
{
    cout << a[0];
    for(int i=1; i<n; i++)
        cout << " " << a[i];
}

bool insert(int a[], int b[], int n)
{
    int i, j;
    for(i=1; i<n && b[i]>=b[i-1]; i++);
    for(j=i; j<n && b[j]==a[j]; j++);
    if(j<n) return false;
    int x = b[i];
    for(j=i-1; j>=0 && b[j]>x; j--) b[j+1] = b[j];
    b[j+1] = x;
    cout << "Insertion Sort\n";
    output(b, n);
    return true;
}

void merge(int a[], int b[], int n)
{
    int i, k, end;
    int flag = 0;
    for(k=1; k<n; k*=2)
    {
        if(equal(a, b, n)) flag = 1;
        for(i=0; i+k<n; i+=2*k)
        {
            end = (i+k+k<=n ? i+k+k : n);
            sort(a+i, a+end);
        }
        if(flag) break;
    }
    cout << "Merge Sort\n";
    output(a, n);
}

int main()
{
    int n, i;
    cin >> n;
    int a[100], b[100];
    for(i=0; i<n; i++) cin >> a[i];
    for(i=0; i<n; i++) cin >> b[i];
    if(insert(a, b, n)) return 0;
    merge(a, b, n);
```

```
    return 0;
}
```

关于堆的判断：

技巧：注意数据有可能是负数。来一个调整一个（往上调整），比全部存入数组再调整方便（从n/2开始往下调整），因为往上调只有一个父结点，往下有两个孩子结点。而且全部存入数组再调整，答案未必正确。因为得到的堆跟来一个调整一个可能不相同。

```
#include<bits/stdc++.h>
using namespace std;

void adjust(int a[], int s)
{//假设a[1..s-1]已经是堆，将a[1..s]调整为以a[1]为根的小根堆
    int j = s/2;
    while(j>=1)
    {
        if(a[j]<a[s]) break;
        swap(a[s], a[j]);
        s = j;
        j = s/2;
    }
}

int main()
{
    int n, m;
    cin >> n >> m;
    int a[1001];
    for(int i=1; i<=n; i++)
    {
        cin >> a[i];
        adjust(a, i);
    }
    for(int i=0; i<m; i++)
    {
        string s;
        int x;
        cin >> x >> s;
        int xpos;
        for(xpos=1; xpos<=n; xpos++)
            if(a[xpos]==x) break;
        if(s[0]=='a')//26 and 23 are siblings
        {
            int y;
            cin >> y >> s >> s;
            int flag = 0;
            if(xpos%2==0)
            {
                if(xpos+1<=n && a[xpos+1]==y) flag = 1;
            }
            else if(xpos>1 && a[xpos-1]==y) flag = 1;
            if(flag)cout << "T\n";
            else cout << "F\n";
        }
```

```
        else
        {
            cin >> s;
            if(s[0]=='a')//23 is a child of 10
            {
                int y;
                cin >> s >> s >> y;
                if(xpos/2>0 && a[xpos/2]==y) cout << "T\n";
                else cout << "F\n";
            }
            else
            {
                cin >> s;
                if(s[0]=='r')//24 is the root
                {
                    if(xpos==1) cout << "T\n";
                    else cout << "F\n";
                }
                else
                {
                    int y;
                    cin >> s >> y;//46 is the parent of 23
                    if((xpos*2<=n && a[xpos*2]==y) || (xpos*2+1<=n &&
a[xpos*2+1]==y)) cout << "T\n";
                    else cout << "F\n";
                }
            }
        }
    }
    return 0;
}
```