檔案讀寫：

1.重新導向
```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
freopen("pa.in","r",stdin);
freopen("pa.out","w",stdout);
return 0;
}
```
適用實機：
將標準輸入從螢幕＆鍵盤改到檔案

2.fstream
```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
int q;
ofstream fout;
fout.open("test.txt");
fout<<123<<endl;
fout.close();

ifstream fin;
fin.open("test.txt");
fin>>q;
cout<<q<<endl;
fin.close();

}
```
適用實機：
需要同時針對鍵盤與螢幕以及檔案做輸入書出的動作

容器通用操作

```
cout << PG.empty() << endl;
//若容器為空 則回傳true

cout << PG.size() << endl;
//回傳容器的大小

PG.clear();
//清空容器內的元素
```

Sort:

Case1: 不需要自訂compare函數

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

int main()
{
int q[]={2,1,4,3,5};
sort(q,q+5);
}
```

Case2: 需要自訂compare函數

```
#include<iostream>
#include<algorithm>
using namespace std;
struct PG
{
PG(){};
PG(int a,string b) : num(a),id(b){}
int num;
string id;
};
bool cp(PG a,PG b)
{
if(a.num == b.num)
return a.id < b.id;
else return a.num > b.num;
//若num一樣 則比較id的字典順序 字典順序小的在前面
//否則比較num大小 大的在前面
}

PG miko[5];
//以下測試用data-------------------------------------
miko[1]=PG(5,"Sona");
miko[2]=PG(4,"Janna");
miko[3]=PG(5,"Morgana");
miko[4]=PG(3,"Tristina");
//---------------------------------------------------
int main()
{
sort(miko+1,miko+5,cp);
for(int i=1;i<=4;i++)cout<<miko[i].num<<" "<<miko[i].id<<endl;
}
```

Queue:

```
#include<iostream>
#include<queue>
using namespace std;
int main()
{
int t;
queue<int> PG; //宣告
PG.push(123);
PG.push(456);
cout << PG.front() << endl; // queue前端元素
PG.pop(); //移除queue前端元素
}
```

Stack:

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
int t;
stack<int> PG; //宣告
PG.push(123);
PG.push(456);
cout << PG.top() << endl; // stack前端元素
PG.pop(); //移除stack前端元素
}
```

Priority Queue:

```
#include<iostream>
#include<queue>
using namespace std;
struct cp
{

bool operator()(int &a,int &b)
{
return a < b ;
//在此自行定義compare函數
}
};
int main()
{
priority_queue<int,vector<int>,cp> PG;
PG.push(2);
PG.push(5);
PG.push(3);
PG.push(1);
PG.push(4);

while(!PG.empty())
{
cout << PG.top() << endl;
PG.pop();
}

}
```

Map:

```
#include<iostream>
#include<map>
using namespace std;
int main()
{
map<string,int> PG;
map<string,int>::iterator iter; //指向元素的指標

PG["Eruru"] = 1;
PG["Fate.T.H"] = 2;
PG["Shamaru"] = 3;
//以上新增三筆資料

cout << PG["Fate.T.H"] << endl;
//取得資料的便捷方法
//但是此方法若該筆資料不存在,會自動新增
//以本例子為例,若"Fate"不存在
//則map會插入"Fate"這筆資料

iter = PG.find("Fate.T.H");
cout << iter->first << " " << iter->second << endl;
//取得資料的傳統方法

if(PG.find("Eruru")==PG.end())cout<<"Not exist";
//判斷某筆資料是否存在

for(iter=PG.begin(); iter!=PG.end(); ++iter)
{
cout << iter->first << " " << iter->second << endl;
}
//把整個map資料印出

system("pause");

}
```

Set:

```
#include<iostream>
#include<set>
using namespace std;
int main()
{
set<int> PG;
set<int>::iterator iter; // 指向元素的指標

PG.insert(123);
PG.insert(456);
//插入資料

if(PG.find(456)==PG.end())cout<<"Not found";
//檢索元素

PG.erase(PG.find(456));
//移除元素

}
```

String:

```
#include<string>
#include<iostream>
using namespace std;
int main()
{
string q="Sona Janna Morgana Tristina";
cout << q.find("Janna") << endl;
cout << (q.find("Katarina")==string::npos) << endl;
//find若有找到 則回傳index若無
// 則回傳string::npos (一個整數)
cout << q.erase(5,4) << endl;
//自index5開始刪除4個字元
cout << q.insert(5,"Jann") << endl;
//自index5插入Jann
cout << q.substr(0,4)<<endl;
//自index0取出長度4的字串
cout << q.replace(q.find("Tristina"),8,"Sona") << endl;
// replace(位址,刪除長度,新字串)

/****************************************
輸出結果：
5
1
Sona a Morgana Tristina
Sona Janna Morgana Tristina
Sona
Sona Janna Morgana Sona
****************************************/
}
```

GCD:

Ver1:

```
#include<iostream>
using namespace std;
int gcd(int a,int b)
{
while( (a%=b) && (b%=a));
return a+b;
}
int main()
{
int a,b;
while(cin >> a >> b)cout << gcd(a,b) << endl;
}
```

Ver2:

```
#include<iostream>
using namespace std;
int main()
{
int a,b;
while(cin>>a>>b)cout<<__gcd(a,b)<<endl;
}
```

Floyd–Warshall:

```
#include < iostream >
using namespace std;
int n;
int map[50][50];
int backtrace[50][50];

void re(int i, int j)
{
    int k = backtrace[i][j];
    if(!k)return;
    re(i, k);
    cout << " => " << k ;
    re(k, j);
}

int main()
{

    n = 10;
    for(int i=1; i<=n; i++)
                for(int j=1; j<=n; j++)
                    map[i][j]=999999999;

    for(int k=1; k<=n; k++)
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
                if(map[i][k] + map[k][j] < map[i][j])
                {
                    map[i][j] = map[i][k] + map[k][j];
                    backtrace[i][j] = k;
                }
    cout << "從3到5的最短路徑是: ";
    cout << "3";
    re(3,5);
    cout << " => 5" << endl;
    cout << "從3到5的最短路徑長度  : " << map[3][5] << endl;

}
```

C++處理輸入輸出

方法.stringstream(最常用的解法)

```
#include < iostream >
#include < sstream >
//記得include sstream
using namespace std;
int main()
{
        int qmax,q[101],m;
        //qmax代表總共有幾筆 q為每筆測資 m為測資大小
        cin>>qmax;
        cin.ignore();
        //ignore會忽略掉目前這一行測資,用來處理qmax後面的換行
        string str_tmp;
        while(qmax--)
        {
                m=0;
                getline(cin,str_tmp);
                //從cin中讀取一行資料到str_tmp

                istringstream cin_v2(str_tmp);
                //利用str_tmp建立輸入字串之資料流
                while(cin_v2>>q[++m])
                {
                        cout<<"read " << q[m] << endl;
                }
                //接下來就可以把cin_v2當作cin使用

        }
}
```

MST Prim:

```
int w[9][9];    // 一張有權重的圖
int d[9];       // 紀錄目前的MST到圖上各點的距離。
int parent[9];  // 紀錄各個點在MST上的父親是誰
bool visit[9];  // 紀錄各個點是不是已在MST之中

void prim()
{
    for (int i=0; i<9; i++) visit[i] = false;
    for (int i=0; i<9; i++) d[i] = 1e9;

    d[0] = 0;   // 可以選定任何點做為樹根，這裡以第零點做為樹根。
    parent[0] = 0;

    for (int i=0; i<9; i++)
    {
        int a = -1, b = -1, min = 1e9;


     for (int j=0; j<9; j++)
            if (!visit[j] && d[j] < min)
            {
                a = j;  // 記錄這一條邊
                min = d[j];
            }

        if (a == -1) break; // 與起點相連通的MST都已找完
        visit[a] = true;

        for (b=0; b<9; b++)
            // 以下與Dijkstra's Algorithm略有不同
            if  (!visit[b] && w[a][b] < d[b])
            {
                d[b] = w[a][b]; // 離樹最近，不是離根最近。
                parent[b] = a;
            }
    }
}
```

其他輔助用函數

cin.ignore();

將目前這行剩餘的資料丟棄

cin.peek();

得到資料流下一個字元 但不會將字元從資料流移除

舉例來說

如果測資是abc

```
string t;
char c=cin.peek();
cout << c << endl;
cin >> t;
cout << t << endl;
```

輸出會得到

a

abc

Union and Find:

```
#include < iostream >
#define UNION_MAX 30000
#include < map >
using namespace std;
class Union{
public:
        int src[UNION_MAX+1];
        //來源 紀錄從屬關係
        int m;
        //紀錄大小
        Union(int q) //int q 代表此set之大小
        {
                m = q;
                for (int i = 1; i <= m; i++){src[i] = -1;}
                //初始 將來源設為-1代表root
        }
        int find(int q)
        {
                if(src[q] == -1)return q;
                //如果來源是-1代表自己就是類別號碼
                int t=q;
                while (src[t] != -1){t = src[t];}
                src[q]=t;
                //將來源設為本次查詢結果 減少下次查詢之時間
                return t;
        }
        void combine(int a,int b)
        {
                a = find(a); b = find(b);//分別找到a b的源頭
                if (a == b)return ;
                else src[b]=a;
                //源頭相等則不用合併 否則將b併入a
        }
};
int main()
{
        int n,m,qmax;
        cin >> qmax;
        while(qmax--)
        {
                cin >> n >> m;
                Union PG(n);
                int a,b;
                for(int i = 1; i <= m; i++)
                {
                        cin >> a >> b;
                        PG.combine(a,b);
                }
                //此題為10608 題目會給朋友人數n 朋友關係數量m
                //接下來有m行 分別代表某某跟某某互為朋友 最後輸出最大的朋友集合
                //（為了減少篇幅 把統計人數部份的code移除了）
        }
}
```