#define 的使用

1. 宏定义MAXN，MINN，PI等（常用大写）
2. 对数组的边界进行定义

```
#define MAXN 99999
#define NINN -99999
//常量的声明一般用const
int a[MAXN][MAXN];
```

3. 代入公式计算，避免调用函数带来的开销

```
#define Abs(x) ((x)>=0 ? (x):(-(x)))
```

强调：每项运算都要加括号，因为括号的优先级最高，保证表达式的运算顺序不会发生改变

std::cin

1. 取消同步缓冲区，加速读取

```
std::ios::sync_with_stdio(false);

#define IOS ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
```

2. 函数返回值

getline()

1. 作用：读取一行中若干用空格隔开的整数
2. 头文件：`#include <string>`
3. 用法：`istream& getline ( istream &is , string &str , char delim );`

   其中，`istream &is` 表示一个输入流，譬如`cin`；

   `string&str`表示把从输入流读入的字符串存放在这个字符串中（可以自己随便命名，str什么的都可以）；

   `char delim`表示遇到这个字符停止读入，在不设置的情况下系统默认该字符为'\n'，也就是回车换行符（遇到回车停止读入）。
4. 返回值：getline 遇到换行符，将停止读入并返回。getline 函数将 istream 参数作为返回值，和输入操作符一样也把它用作判断条件。就是`getline (cin, stri)`返回cin，cin再被转换为bool值true（读入成功）或者false（读入失败）

举例（紫书P104） 计算每行中所用整数的和

```

```

```cpp
#include<iostream>
#include<string>
#include<sstream>
using namespace std;

int main(){
    string line;
    while(getline(cin, line)){//每行读取（全部一并读取）
        int sum = 0,x;
        stringstream ss(line);//省去清空缓存等操作
        while(ss >> x) sum += x;//逐个输出并转换数据类型
        cout << sum;
    }
    return 0;
}
```

do...while...

先执行语句，后做判断

语法：

```cpp
do{
    Statement _1;
    Statement _2;
}while(Exp_cntrl);//分号不可丢
```

应用 next_permutation()

inline

- 用途：解决一些频繁调用的小函数大量消耗栈空间（栈内存）的问题
- 限制：不能包含复杂的结构控制语句例如 while、switch，不能是直接递归函数
- 对比：内联函数和普通函数的区别在于：当编译器处理调用内联函数的语句时，不会将该语句编译成函数调用的指令，而是直接将整个函数体的代码插人调用语句处，就像整个函数体在调用处被重写了一遍一样。
- 优劣：使用内联函数会使最终可执行程序的体积增加。以时间换取空间，或增加空间消耗来节省时间

vector 与 unique()

```cpp
//unique讲解示例
#include<iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
    vector<int> v{3,4,5,1,2,5,3};
```

```
    //排序
    sort(v.begin(), v.end());
    //1 2 3 3 4 5 5

    //pos是去重以后vector中没有重复元素的下一个位置的迭代器。
    vector<int>::iterator pos = unique(v.begin(), v.end());

    //去重后整个容器
    for (int i = 0; i < v.size(); i++)
    {
        cout << v[i] << ' ';
    }
    cout << endl;
    //1 2 3 4 5 5 5

    //从容器开始到pos：去重后的容器元素
    for (vector<int>::iterator i = v.begin(); i < pos; i++)
    {
        cout << *i << ' ';
    }
    cout << endl;
    //1 2 3 4 5

    //从pos到容器结束：无意义的元素
    for (vector<int>::iterator i = pos; i < v.end(); i++)
    {
        cout << *i << ' ';
    }
    cout << endl;
    //5 5

    return 0;
}
```

problem

Fib

C++保留指定位数小数（四舍五入）

```
#include<iomanip>
cout<<fixed<<setprecision(6)<<l;
```

进一法、四舍五入

e:计算结果 a：被除数 b：除数:
1(四舍五入)： e=(a+(b/2))/b
2(进一法)： e=(a+(b-1))/b

取模运算

模运算与基本四则运算有些相似，但是除法例外。其规则如下：

1. (a + b) % p = (a % p + b % p) % p
2. (a - b) % p = (a % p - b % p) % p
3. (a * b) % p = (a % p * b % p) % p
4. a ^ b % p = ((a % p)^b) % p

结合律：

((a+b) % p + c) % p = (a + (b+c) % p) % p

((a*b) % p * c)% p = (a * (b*c) % p) % p

交换律：

(a + b) % p = (b+a) % p

(a * b) % p = (b * a) % p

分配律：

(a+b) % p = (a % p + b % p) % p

((a +b)% p * c) % p = ((a * c) % p + (b * c) % p) % p

重要定理

若a≡b (% p)，则对于任意的c，都有(a + c) ≡ (b + c) (%p)；

若a≡b (% p)，则对于任意的c，都有(a * c) ≡ (b * c) (%p)；

若a≡b (% p)，c≡d (% p)，则 (a + c) ≡ (b + d) (%p)，(a - c) ≡ (b - d) (%p)，

(a * c) ≡ (b * d) (%p)，(a / c) ≡ (b / d) (%p)；

sort（cmp）

1、重载运算符<

```
struct node
{
    int dat;
    friend bool  operator<(node a,node b)
    {
        return a.dat>b.dat;
    }
};

int main()
{
    int n,m;
    while (cin>>n>>m)
    {
        priority_queue<node> q;
        int data;
        for(int i=0;i<n;i++)
        {
            string str;
            cin>>str;
```

```
            if(str=="I")
            {
                cin>>data;
                node no;
                no.dat=data;
                q.push(no);
                if(q.size()>m)
                    q.pop();
            }
            else if(str=="Q")
            {
                cout<<q.top().dat<<endl;
            }
        }
    }
    return 0;
}
```

在上面的代码中我们声明的优先队列是升序的，与以往相反，在此处node中重载了运算符<，之前一直不理解升序降序的规则。

```
friend bool  operator<(node a,node b)
    {
        return a.dat>b.dat;
    }
```

我们的参数为a,b 正常的小于应该是a<b，也就是a.dat<b.dat 这样的优先队列是正常的降序，但是我们想得到相反的结果应该a.dat>b.dat。

2、sort中的cmp()函数

```
#include <iostream>
#include<queue>
#include<string>
#include<algorithm>
using namespace std;
struct node
{
    int dat;
};
bool cmp(node a,node b)
{
    return a.dat <b.dat;//升序
}
int main()
{
    node a[5];
    for (int i = 1; i <= 5; i++)
        a[i - 1].dat = i;
    sort(a, a + 5, cmp);
    for (int i = 1; i <= 5; i++)
        cout<<a[i - 1].dat << endl;
    return 0;
}
```

运行结果按升序排序：

```
1 2 3 4 5
```

其中关键代码个人理解如下：

```
bool cmp(node a,node b)
{
    return a.dat <b.dat;//升序
}
```

两个节点a,b 当a<b时返回true，意味着要求a<b，这样假设数组中的数是a、b、c、d、e时，当a<b<c<d<e时才满足cmp规则true，这样是升序序列。