1 全体のテンプレ

別にこれに固定しなくてもよい

```
/*
input data
*/

void init(){
}

bool input(){
}


/*
dp とか
*/
void init_solve(){
}

void q_input(){
}

int solve(){
        return 0;
}


int main(){
        int m;
        for(int i=0;i<m;i++){
                cout<<solve()<<endl;
        }
}
```

2 include

```
#include<cmath>
#include<string>
#include<vector>
#include<queue>
#include<stack>
#include<set>
#include<map>
#include<algorithm>
#include<utility>
#include<climits>
#include<cstdio>
#include<cassert>
using namespace std;
```

3 素数

```
vector<int> prime;
vector<int> prime_list;
void prime_set(int n){
        n+=100;
        prime.resize(n);
        for(int i=0;i<n;i++){
                prime[i]=1;
        }
        prime[0]=prime[1]=0;
        for(int i=0;i*i<n;i++){
                if(prime[i]){
                        for(int j=i*2;j<n;j+=i){
                                prime[j]=0;
                        }
                }
        }
        for(int i=0;i<n;i++){
                if(prime[i])prime_list.push_back(i);
        }
}
```

4 文字列

```
string revStr(string s){
    return string(s.rbegin(),s.rend());
}
```

5 二次元幾何

```
#define X first
#define Y second

typedef long double ld;
typedef long long ll;
typedef ll P_type;
typedef pair<ll,ll> P;

const ld INF = 1e39;
const ld EPS = 1e-8;
const ld PI = acos(-1);

P_type out_pro(P a,P b) {
  return (a.X * b.Y - b.X * a.Y);
}

P_type in_pro(P a,P b){
  return (a.X * b.X + a.Y *b.Y);
}

//以下2つは小数点以下が必要なことがあるのでにする ld
ld pow_len(P a) {
  return a.X * a.X + a.Y * a.Y;
}

ld len(P a){
  return sqrt(pow_len(a));
}

P make_v(P a,P b) {
  return P(a.X - b.X, a.Y - b.Y);
}
//凸法
vector<P> convex(vector<P> list) {
  int m=0;
  vector<P> res(list.size()*2);
  sort(list.begin(),list.end());
  for(int i=0; i<list.size(); res[m++]=list[i++]){
    for(;m>1&&out_pro(make_v(res[m-1],res[m-2]),make_v(list[i],res[m-2]))<=EPS;--m);
  }
  for(int i=list.size()-2,r=m; i>=0; res[m++]=list[i--]){
    for(;m>r&&out_pro(make_v(res[m-1],res[m-2]),make_v(list[i],res[m-2]))<=EPS;--m);
  }
  res.resize(m-1);
  return res;
}
```

6 UnionFind

```
class union_find{
private:
	vector<int> parents;
	vector<int> rank;
public:
	union_find(int n){
		parents.resize(n);
		rank.resize(n);
	}
  void init(){
		for(int i=0;i<parents.size();i++){
			parents[i]=i;
			rank[i]=0;
    }
  }
	int find(int x){
		if(parents[x]==x){
			return x;
		}else{
			return parents[x]=find(parents[x]);
		}
	}
	void unite(int x,int y){
		x=find(x);
		y=find(y);
		if(x==y)return;
		if(rank[x]<rank[y]){
			parents[x]=y;
		}else{
			parents[y]=x;
			if(rank[x]==rank[y])rank[x]++;
		}
	}
	bool same(int x,int y){
		return (find(x)==find(y));
	}
};
```