```
long double sum = 0;
long double x;
while (scanf("%Lf", &x) != EOF){sum += x;}
printf("%.2Lf\n", sum);
```

```
char str[1000];
scanf("%[^\n]s", str);
while(strcmp(str, "TOTAL")){
    scanf("%d %d \n", &p, &q);
    S += p*q;
    scanf("%[^\n]s", str);
}
```

```
while(getline(cin, monument)){
    string::size_type pos;
    pos = monument.find(' ',0);
    monument = monument.substr(pos+1);
    monuments.insert(monument);
}
```

```
double ceil(double x);
double floor(double x);

double z = static_cast<double>(x)/y;
std::cout << z;
```

```
    string sequence;
    cin >> sequence; //cin reads until whitespace
↪   --> use getline to read until newline
    printf("%s\n", sequence.c_str()); //When we have
↪   string type use .c_str() to print or c out !
```

```
map<string, int> species;
    for (int i = 0; i < N; i++) {
        string name; cin >> name;
        species[name]++;
    }
    // iterate through the values of the map
    for (auto it = species.begin(); it !=
↪   species.end(); it++) {
        if (it->second > sum - (it->second)) {}
```

```
// How to use struct ? to access cow.x, cow.adj
↪   ..etc
struct Cow {
    int x, y, p;
    vector<int> adj; // adjascent cows are the
↪   reachable cows from this cow
};
```

```
// How to use complex, coordinates, Cross product :
↪   produit vect
#include<complex>
complex<long int> p = complex<long int>(x, y) //make
↪   complex
signed_created_area = p1.real()*p2.imag() -
↪   p1.imag()*p2.real();
 vector = p1 - p2 //make vector
```

```
long long fib(int n) {
    // Check if the value is in the cache
    if (fibonacci_cache.find(n) !=
↪   fibonacci_cache.end()) {
        return fibonacci_cache[n];
    } else {
        // Compute the Fibonacci value and store it
↪   in the cache
        if (n < 2) {
            fibonacci_cache[n] = n;
        } else {
            fibonacci_cache[n] = fib(n - 2) + fib(n
↪   - 1);
        }
        return fibonacci_cache[n];
    }
}
```

.

Please help the cows determine the maximum number of cows that can be reached by a broadcast originating from a single cow :

```
vector<pair<pair<int,int>, int>> a;
void dfs(int node, vector<int> g[], pair<int, int> state[] ,int);
long double dist(int x1, int y1, int x2, int y2){
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
int max_count = 0;

int main(){
    int N;
    cin >> N;
    a.resize(N);
    for(int i = 0; i < N; i++){
        int x, y, z;
        cin >> x >> y >> z;
        a[i] = {make_pair(x, y), z}; // z is the distance that can be reached
    }

    vector<vector<int>> g(N, vector<int>());
    // le graph
    if (N == 1){  cout << 1 << endl;   return 0;}
    for (int i = 0; i< N; i++){
        // for each edge, see if it->second can reach another edge
        for (int j = i+1; j < N; j++){ // --- we can just check the upper triangle
            if(dist(a[i].first.first, a[i].first.second, a[j].first.first, a[j].first.second)
            <= a[i].second){
                g[i].push_back(j); // cow i can reach cow j
            }
            if(dist(a[i].first.first, a[i].first.second, a[j].first.first, a[j].first.second)
            <= a[j].second){ g[j].push_back(i); // cow j can reach cow i}
        }
    }
    // BFS is better
    int result = 0, count = 0;
    for (int i = 0; i < N; i++){ // iterate over all the cows
        if(g[i].size() == 0){ // if the cow cannot reach any other cow
            continue;
        }
        queue<int> q;
        q.push(i);
        vector<bool> visited(N);
        visited[i] = true;

        count = 0;
        while (!q.empty()){
            int cur = q.front();
            q.pop();
            count++;

            for (int nb : g[cur]){
                if (!visited[nb]){
                    q.push(nb);
                    visited[nb] = true;
                }
            }
        }

        if(count > result)
            result = count;
    }
    cout << result << endl;
    return 0;
}
```

Please help the cows determine the minimum integer value of X such that a broadcast from any cow will ultimately be able to reach every other cow:

```
vector<pair<int,int>> a;

int find(int x);
bool unite(int a, int b);

long long dist(int x1, int y1, int x2, int y2){
    return (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
}
int repr[25000]; // initiliazed to -1
int main(){
    int N;
    cin >> N;
    a.resize(N);
    for(int i = 0; i < N; i++){
        int x, y;
        cin >> x >> y;
        a[i] = make_pair(x, y);
    }

    set<pair<long long, pair<int,int>>> g;
    // le graph [TODO]
    if (N == 1){
        cout << 1 << endl;
        return 0;
    }

    for(int i = 0; i< N; i++){
        for(int j = i+1; j < N; j++){
                g.insert({dist(a[i].first, a[i].second, a[j].first, a[j].second), {i,j}});
        }
    }
    fill(repr, repr+25000, -1);

    long long X = 0;
    for (auto itr = g.begin(); itr != g.end(); itr++) {
        if(find(itr->second.first) != find(itr->second.second)){
            unite(itr->second.first, itr->second.second);
            X = itr->first;
        }
    }

    cout << X << endl;

    return 0;
}

int find(int x) {
    if(repr[x] < 0 ) return x;
    return repr[x]=find(repr[x]); // path compression
}

bool unite(int a, int b) {
    a = find(a);
    b = find(b);
    if(a==b) { return false; }
        if(repr[a] > repr[b]) { swap(a,b); } // rank
            repr[a] += repr[b] ;
            repr[b] = a;
    return true;
}
```

.

BFS to find shortest path :

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

// Function to perform BFS and find the shortest path
std::vector<int> bfsShortestPath(const std::vector<std::vector<int>>& graph, int start, int target) {
    int n = graph.size();
    std::vector<int> distance(n, INT_MAX);  // Distance from start to each node
    std::vector<int> parent(n, -1);         // To reconstruct the path
    std::queue<int> q;

    distance[start] = 0;
    q.push(start);

    while (!q.empty()) {
        int current = q.front();
        q.pop();

        for (int neighbor : graph[current]) {
            if (distance[neighbor] == INT_MAX) {  // If neighbor not visited
                distance[neighbor] = distance[current] + 1;
                parent[neighbor] = current;
                q.push(neighbor);

                // Stop if we reach the target
                if (neighbor == target) {
                    break;
                }
            }
        }
    }

    // Reconstruct the path from start to target
    std::vector<int> path;
    for (int at = target; at != -1; at = parent[at]) {
        path.push_back(at);
    }
    std::reverse(path.begin(), path.end());

    // Check if a path was found
    if (path.size() == 1 && path[0] != start) {
        path.clear();  // No path found
    }

    return path;
}

int main() {
    // Example graph represented as an adjacency list
    std::vector<std::vector<int>> graph = {
        {1, 2},      // Neighbors of node 0
        {0, 3, 4},   // Neighbors of node 1
        {0, 4},      // Neighbors of node 2
    };

    int start = 0;
    int target = 5;

    std::vector<int> path = bfsShortestPath(graph, start, target);
```

```
    if (!path.empty()) {
        std::cout << "Shortest path from node " << start << " to node " << target << " is: ";
        for (int node : path) {
            std::cout << node << " ";
        }
        std::cout << std::endl;
    } else {
        std::cout << "No path found from node " << start << " to node " << target << std::endl;
    }

    return 0;
}
```

.

0.0.1 I/O , STL problems

- Big Money sum :

```
#include <cstdio>
#include <string>

using namespace std;

int main() {
    long double total_expenses = 0.0;

    while (1) {
        long double expense;
        int ret = scanf("%Lf", &expense);
        if (ret != 1) {
            break;
        }
        total_expenses += expense;
    }
    printf("%.2Lf\n", total_expenses);
    return 0;
}
```

- Shattered Cake :

```
#include <cstdio>
int main()
{
    while (1)
    {
        long long W, N;
        int ret = scanf("%lld %lld", &W, &N);
        if (ret != 2) {
            break;
        }
        long long area = 0;
        for (int i = 0; i < N; i++) {
            long long w, l;
            scanf("%lld %lld", &w, &l);
            area += w*l;
        }
        printf("%lld\n", area/W);

    }
    return 0;
}
```

.

- How to use sstream, getline, .ignore in I/O ?? Example of Extreme temperature search. Input :
2020-01-15 5 4 6 8 12 13 12 9 7
2020-01-16 6 3 4 6 10 12 11 7

```
#include<sstream> //using namespace std;
int main() {
    int N;
    cin >> N; // this is to read the number of days in the input
    cin.ignore(); // Ignore the newline character after N
    int min_temp = 51, max_temp = -51; // Initialize with extreme values
    for (int i = 0; i < N; i++) {
        string line;
        getline(cin, line); // Read the entire line
        istringstream iss(line);
        string date;
        iss >> date; // Read the date
        int temp;
        while (iss >> temp) { // Read each temperature
            min_temp = min(min_temp, temp);
            max_temp = max(max_temp, temp);
        }
    }
    cout << min_temp << " " << max_temp << endl;
    return 0;
}
```

- WhereAmI : find the smallest length K of substrings in a sequence so that a substring of this length **determine** our position.

```
int main() {
    int N;
    string sequence;
    scanf("%d", &N);
    cin >> sequence; //cin reads until whitespace --> use getline to read until newline
    for (int k=1; k<=N; k++) {
        vector<string> substrings;
        for (int i=0; i<=N-k; i++) {
            //take a substring of length k starting from i
            string sub = sequence.substr(i, k);
            substrings.push_back(sub);
        }
        //check if there are any duplicates
        bool unique = true;
        for (int i=0; i<substrings.size(); i++) {
            for (int j=i+1; j<substrings.size(); j++) {
                if (substrings[i] == substrings[j]) {
                    unique = false;
                    break;}
            }
            if (!unique) {break;}
        }
```

.

```
    {
        if (unique) {
            printf("%d\n", k);
            break;
        }
    }
    return 0;
}
//remarks:
//---------
// Here we know that N < 1000 so we allow ourselves to use such a solution that is O(N^2) in the worst case.
// We can also use a hash table to store the substrings and check if they are unique in O(1) time.
// This will reduce the time complexity to O(N) in the worst case.
// we can also procede with a binary search to find the smallest k that satisfies the condition.
```

.

0.0.2 Graph/paths problems

- Find a valid path (not necessarily the shortest path) in a grid maze. Rabbit

```
#include <iostream>
char grid[501][501]; // 500*500 for storing the grid
char out[501 * 501]; // 500*500 for stroing the path
char directions[4] = {'L', 'D', 'R', 'U'}; // directions
int diri[4] = {0, 1, 0, -1}, dirj[4] = {-1, 0, 1, 0};
int C, R;
bool dfs(int, int, int);

int main(){
    cin >> C >> R;
    int start_i, start_j;
    for (int i = 0; i < R; i++){
        for (int j = 0; j < C; j++){
            cin >> grid[i][j];
            if (grid[i][j] == 'R'){
                start_i = i;
                start_j = j;
            }
        }
    }
    dfs(start_i, start_j, 0);
    cout << out << endl; // print the path ----> dont forget to add endl because the output is not flushed.
    return 0;
}

bool dfs(int i, int j, int prof){
    if (i < 0 || j < 0 || i >= R || j >= C) // if we are out of the grid{
        return false;
    }
    else if (grid[i][j] != '.' && grid[i][j] != 'R') {
        return grid[i][j] == 'D'; // if we are in the destination return true else false
    }
    grid[i][j] = 'V'; // mark the cell as visited
    // deep
    for (int k = 0; k < 4; k++){// for the four directions we test if in the end of the recursion we get to D
        if (dfs(i + diri[k], j + dirj[k], prof + 1)){
            out[prof] = directions[k];
            return true;
        }
    }
    return false;
}
```

.

- Iterative DFS (from Moocast)

```
for(int i=0; i<N; i++){
    vector<bool> visited(N, false);
    vector<int> q;
    q.push_back(i);
    visited[i] = true;
    int count = 0;
    while(!q.empty()){
        int cur = q.back();
        q.pop_back();
        count++;
        for(int j=0; j<cows[cur].adj.size(); j++){
            int next = cows[cur].adj[j];
            if(!visited[next]){
                visited[next] = true;
                q.push_back(next);
            }
        }
    }
    maxCows = max(maxCows, count);
}
```

.

- [Satpixels] Find the bigget connected chain in a grid :

```
10 5
..*.....**
.**..*****
.*...*....
..****.***
..****.***
```

```cpp
#include <iostream>
#include <cstdio>
using namespace std;
char grid[1000+1][80+1]; //MAX VALUE
int W, H;

const int NOT_VISITED = 0, IN_VISIT = 1, VISITED = 2;
int state[1000+1][80+1];
int diri[4] = {0, 1, 0, -1}, dirk[4] = {-1, 0, 1, 0};
void dfs(int i, int k, int &increm);

int main() {
    cin >> W >> H;
    for (int i = 0; i < H; i++) {
        for (int k = 0; k < W; k++) {
            cin >> grid[i][k];
            state[i][k] = NOT_VISITED;
        }
    }
    int largest_pasture = 0;
    for (int i = 0; i < H; i++) {
        for (int k = 0; k < W; k++) {
            // if the cell has been already visited (through another one or stating from it) there is no need
↪    to revisit it !!!!
            if (grid[i][k] == '*' && state[i][k] == NOT_VISITED) {
                int length_pasture = 1;
                dfs(i, k, length_pasture);
                if (length_pasture > largest_pasture) {
                    largest_pasture = length_pasture;
                }
            }
        }
    }cout << largest_pasture << endl;return 0;
}
void dfs(int i, int k, int &increm) {
    state[i][k] = IN_VISIT;
    for (int m = 0; m < 4; m++) {
        int x = i + diri[m];
        int y = k + dirk[m];
        if (x >= 0 && x < H && y >= 0 && y < W && grid[x][y] == '*' && state[x][y] == NOT_VISITED) {
            increm++;
            dfs(x, y, increm);
        }
    }state[i][k] = VISITED;}
```

.

0.0.3 Geometry

- [Iceberg] Calculate the total area of N polygons with P_i points each.

```
#include<iostream>
#include<vector>
#include<complex>
#include <cstdlib> // dont forget std namespace
long int total_area = 0;  // here i will store the areas of the icebergs
int main() {
    int N; // number of ICEBERGS
    cin >> N;
    for (int i = 0; i < N; i++) {
        int P; // number of points
        cin >> P;
        vector<complex<long int>> iceberg;
        for (int j = 0; j < P; j++) {
            long int x, y;
            cin >> x >> y;
            iceberg.push_back(complex<long int>(x, y));
        }
        // calculate the area of the iceberg
        long double area = 0;
        for (int j = 1; j < P-1; j++) {
            complex<long int> p1 = iceberg[j] - iceberg[0];
            complex<long int> p2 = iceberg[j+1] - iceberg[0];

            area += (p1.real()*p2.imag() - p1.imag()*p2.real());
        }
        total_area += abs(area);
    }
    total_area = total_area / 2;
    cout << total_area << endl;
    return 0;
}
// Attention ! La division doit s effectuer a la fin.
// Sinon, on aura des erreurs de precision.
```

.

[Clockwise_fence] Determine if an enclosed fence is enclosed clockwise or counterclockwise –¿ we use the sign cross product to count the orientations.

```
#include<iostream>
#include<vector>
#include<complex>
#include <cstdlib>
using namespace std;

const char directions[4] = {'N','E','S','W'};
const complex<int> vectors[4] = {complex<int>(0,1),complex<int>(1,0),complex<int>(0,-1),complex<int>(-1,0)};
↪  // corresponds to N,E,S,W
int main() {
    int N;
    cin >> N;
    for(int i = 0;i<N;i++){
        string fence;
        cin >> fence;

        int cw_counter = 0;
        int ccw_counter = 0;

        for (int j = 0; j < fence.size() - 1; j++) {
            complex<int> v1;
            complex<int> v2;
            for (int k = 0; k < 4; k++) {
                if (fence[j] == directions[k]) {
                    v1 = vectors[k];
                }
                if (fence[j + 1] == directions[k]) {
                    v2 = vectors[k];
                }
            }
            int cross_product = v1.real() * v2.imag() - v1.imag() * v2.real();
            if (cross_product > 0) {
                ccw_counter++;
            } else if (cross_product < 0) {
                cw_counter++;
            }
        }
```

.

```
        if(cw_counter > ccw_counter){
            cout << "CW" << endl;
        }else{
            cout << "CCW" << endl;
        }
    }
    return 0;
}
```

- [CowFind] Finding a pattern in a text in the naive way and not KMP hashing, but it is O(2N)

```
#include<iostream>
#include<string>
using namespace std;
const string front = "))";
const string hind = "((";

int main(){
    string grass;
    cin >> grass;
    int N = grass.size();

    // naive pattern searching it gives O(2N) (SEE KMP:Knuth-morris-pratt hashing)
    int possibilities = 0;
    int coeff = 0;
    int j, k;
    for (int i=0; i < N - 2 + 1;i++) {
        j = 0;
        k=0;
        while(j < 2 && grass[i+j] == hind[j]){j++;}
        while(k < 2 && grass[i+k] == front[k]){k++;}
        if (j == 2){
            coeff++;
        }
        if (k==2){
            possibilities += coeff;
        }
    }
    cout<< possibilities <<endl;

}
```

.

Moo Sick :

```
int sizePattern, sizeSong ;
int pattern [10], song[20001], curPattern[10] ;

bool doMatch(int pos) { // O(sizePattern × log(sizePattern))
  copy(song+pos,song+pos+sizePattern,curPattern);
  sort(pattern,pattern+sizePattern) ;
  sort(curPattern,curPattern+sizePattern) ;
  const int diff = curPattern[0] - pattern[0] ; // compare both min
  for(int i = 1 ; i < sizePattern ; i++)
    if(curPattern[i] - pattern[i] != diff) // all need to be equal
      return false ;
  return true;
}

int main () {
  // reading input
  scanf("%d\n",&sizeSong);
  for(int id = 0 ; id < sizeSong ; id++ )
    scanf("%d\n",song+id) ;
  scanf("%d\n",&sizePattern);
  for(int id = 0 ; id < sizePattern ; id++ )
    scanf("%d\n",pattern+id) ;

  // main algorithm check all positions for matches
  // O( sizeSong × sizePattern log(sizePattern ))
  vector<int> matchs ;
  for(int pos = 0 ; pos+sizePattern <= sizeSong ; pos++)
    if(doMatch(pos))
      matchs.push_back(pos+1);

  // print out matches
  printf("%lu\n",matchs.size());
  for(int m : matchs)
    printf("%d\n",m);
  return 0;
}
```

.

Where am I? (Print a line containing a single integer, specifying the smallest value of K)

```
int main()
    int N; cin >> N;
    char str[N+1];  cin >> str;  int K = 1;  bool var = true;
    while(K<N){
        char comp1[K+1], comp2[K+1];
        for(auto i = 0; i != N-K; ++i){
            if(!var){  break;   }else if(i == N-K-1){
                cout << K << "\n";
                return 0;
            }
            else{ strncpy(comp1, &str[i], K);
                 comp1[K] = '\0';
                for(auto j = 0; j != N-K+1; ++j){
                    if(i!=j){
                        strncpy(comp2, &str[j], K);
                        comp2[K] = '\0';
                        if(!strcmp(comp1, comp2)){
                            var = false;
                            break;
                        }
                    }
                }
            }
        }
        K++;
        var = true;
    }
    cout << K << "\n";
    return 0;
```

Splitting dna :

```
#include <iostream>, <vector>, <sstream>, <string>, <climits>

using namespace std;

static long pos_inf = numeric_limits<long>::max();

int main()
{
    int N;
    if (scanf("%d\n", &N) != 1) {
    }

    vector<vector<long> > s;
    vector<long> l;
    l.resize(N);
    s.resize(N);

    string line;

    getline(cin, line);

    stringstream ss(line);
    string c;
    int j = 0;

    while (ss >> c){
        s[j].resize(N);
        s[j][j] = 0;
        l[j] = stol(c);
        j++;
    }
    for (int j=1; j<N; j++){
        for (int i=0; i<N-j; i++){
            long c = 0;
            long min = pos_inf;
            for (int k=i; k<i+j; k++){
                c += l[k];
                if (s[i][k] + s[k+1][i+j] < min){
                    min = s[i][k] + s[k+1][i+j];
                }
            }
            c += l[i+j];
            s[i][i+j] = c + min;
        }
    }
    cout << s[0][N-1] << endl;
    return 0;
}
```