Lazaron Shyta

“HELLO WORLD” Contest

Problem statements: https://www.hackerrank.com/hello-world-contest-here

Problem 1. “Marathon”

Solution:

```
int h1,h2,m1,m2;
    char t;
    cin>>h1>>t>>m1;
    cin>>h2>>t>>m2;

    // Calculate the time “middle contest time” in minutes
    int tMin1=h1*60+m1;
    int tMin2=h2*60+m2;
    int totMin1 = (tMin1+tMin2)/2;

    // convert the minutes into hours
    int tHours=totMin1/60;
    int tMins=totMin1%60;

    // prepare the output
    string output;
    if(tHours<10)output+='0'; // leading zeroes, if any
    output+=to_string(tHours);
    output+=':';
    if(tMins<10)output+='0';  // leading zeroes, if any
    output+=to_string(tMins);
    cout<<output;
```

Problem 2. “Arithmetic Expressions”

- One possible short solution is to get the input as a string and to form a string stream to get the numbers one by one.

Solution:

```
string in;
cin>>in;
stringstream ss(in); // make the stream input
                     //    from the string
int res = 0;
int nxt;
while(ss>>nxt)res+=nxt; // get all the integers
                        //  and add to result
cout<<in<<"="<<res;
```

Read about string stream here: https://www.geeksforgeeks.org/stringstream-c-applications/

Problem 3. “Stars Shape”

Solution: Just do as asked 😊

```
    int n,a;
    cin>>n>>a;
    int spc=0;
  // UPPER HALF
    for(int i=0;i<n;i++){
        for(int k=0; k<spc; k++)cout<<" "; //spaces
        spc++; // increment for next line
        for(int j = 0; j < a ; j ++)cout<<"*";
        cout<<"\n";
    }
  // LOWER HALF
    spc-=2; //print less spaces in lower part
    for(int i=0;i<n-1;i++){
        for(int k=0; k<spc; k++)cout<<" "; //spaces
        spc--; //decrement for next line
        for(int j = 0; j < a  ; j ++)cout<<"*";
        cout<<"\n";
    }
```

Problem 4. “Scoring System”.

- The first player to win 5 points wins the set. The first player who win 3 sets win the game.
- One single loop, where you keep count of each of player’s points. Once one of them reaches 5, sets the points to 0 and increment the sets. Once on of the players reaches 3 in sets, break the game and output the winner.

Solution:

```
    string s;
    cin >>s ;
    int a=0,b=0,aa=0,bb=0;

    for(char x : s){
        if(x=='A')a++;
        else b++;
        // check points
        if(a==5){aa++;a=0,b=0;}
        if(b==5){bb++,a=0;b=0;}
        //check sets
        if(aa==3){cout<<"Ana";return 0;}
        if(bb==3){cout<<"Ben";return 0;}
    }
```

Problem 5. “Round Trip”

A greedy approach would be correct for this problem.

- You step only once in Small Rock.
- You always step in a Big Rock.
- If there are two contiguous Small Rocks, always skip one of them, in order to step there in the second trip, otherwise the leap would be bigger in the second trip.
- Why? Let’s say Array[i] and Array[i+1] are small rocks. If you don’t skip one of them when you go from left to right, then you have to make a leap from A[i+2] to A[i-1], which clearly is a greater distance than from A[i] to A[i+1]. (A is sorted).

Solution:

```
int n,d;
 cin>>n>>d;
 vector<pair<int,char>> river;
 for(int i=0;i<n;i++){
     char a;int place;
     cin>>a>>place;
     river.push_back(make_pair(place,a));
 }
 sort(river.begin(),river.end());
 river.push_back(make_pair(d,'B'));

 int result=0;
 map<int,bool>used;
 int lastPlace=0;
 bool jump=0;
 for(int pos=0;pos<=n;pos++){
     if(river[pos].second=='B'){
             result=max(result,river[pos].first-lastPlace);
             lastPlace=river[pos].first;
     }
     else{
         if(jump==0)jump=1;
         else{
             result=max(result,river[pos].first-lastPlace);
             lastPlace=river[pos].first;
             jump=0;
             used[pos]=1;
         }
     }
 }
 lastPlace=0;
 for(int pos=0;pos<=n;pos++){
     if(!used[pos]){
         result = max(result,river[pos].first-lastPlace);
         lastPlace=river[pos].first;
     }
 }
```

```
        printf("%d\n",result);
```

Problem 6. “Inside the Rectangles”

One approach solving this problem, is first sorting all the rectangles by its area.
In that way, we make sure that only the rectangles form the left can be inside the rectangles in the right.

Solution:

```
struct Rect{
    int area;
    int X,Y,Xd,Yd;
    bool operator < (Rect other){
        return area<other.area;
    }
};
int main(){
    vi depths(100000);
    map<int,bool>visited;
    int n;cin>>n;
    Rect all[n];
    for(int i=0;i<n;i++){
        int x,y,xd,yd;
        cin>>x>>y>>xd>>yd;
        Rect newRec;
        newRec.X=x;newRec.Y=y;
        newRec.Xd=xd;newRec.Yd=yd;
        newRec.area=xd*yd;
        all[i]=newRec;
        visited[i]=0;
    }
    int maxdepth=0;
    sort(all,all+n);
    int W=0;
    while(W<n){
        int cP=W;
        int currDepth=1;
        for(int i=W+1;i<n;i++){
            // check if it is inside
            if(all[cP].X > all[i].X)
            if(all[cP].Y > all[i].Y)
            if(all[cP].Xd+all[cP].X<all[i].Xd+all[i].X)
            if(all[cP].Yd+all[cP].Y<all[i].Yd+all[i].Y)

            {
                cP=i;
                currDepth++;
            }
        }
       maxdepth=max(maxdepth,currDepth);
       depths[W]=currDepth;
       W++;
    }
    }
    cout<<maxdepth<<endl;}
```