Prog en C
Matthieu Boyer
Planche A

Khôlle MP2I
18/11

Vivement Dimanche

Écrire un code C renvoyant le nombre de dimanche tombés un premier du mois durant le 20ème siècle, le 1er janvier 1900 étant un lundi.

Solution: Vivement Dimanche

Le 20ème siècle commence le mardi (1900 n'étant pas une année bissextile) 1er Janvier 1901 et s'achève le 31 Décembre 2000 Ici, suite à un problème de compilation on écrit or au lieu de la syntaxe C correcte.

```
int dimanches = 0;
int j_act = 1;
int d_act = 0;
int m_act = 0;
int y_act = 1901;

int jour_mois(int N, int y) \{
        if (N == 1 or N == 3 or N == 5 or N == 7 or N == 8 or N == 10 or N == 12) \{
                return 31;
        \}

        else if (N == 4 or N == 6 or N == 9 or N == 11) \{
                return 30;
        \}

        else if (N == 2) \{
                return (y \% 4) == 0 ? 29 : 28; // Suffit car 2000 est bissextile
        \}
\}

void step(int* j, int* d, int* m, int* y) \{
        *j = (*j + 1) \% 7;
        *d = (*d + 1) \% (jour_mois(*m, *y));
        *m = *d == 0? (*m + 1) \% 12 : *m;
        *y = *m == 0? (*y + 1) : *y;
\}

int main() {
        while (y_act < 2001) {
                step(&j_act, &d_act, &m_act, &y_act);
                if (j_act == 6 && d_act == 0) {
                        dimanches++;
                }
        }
}
```

Prog en C
Matthieu Boyer
Planche B

Khôlle MP2I
18/11

Sens Giratoire

Pour des questions de conservatisme, Nicolas S. doit prendre la première sortie à droite lorsque sa voiture arrive à un sens giratoire. On vous donne les coordonées du sens-giratoire R, un tableau de coordonnées de points P_i représentant les destinations de la sortie du sens giratoire, un point d'entrée E (qui est l'une des destinations) et les segments $[RE]$ et $[RP_i]$. Les points sont donnés dans un ordre arbitraire. Écrivez une fonction qui renvoie les coordonnées de la destination de la sortie qui correspond, du point de vue de Nicolas S., à la première sortie du sens giratoire. Pour cela, on propose trois méthodes différentes :

1. Étudier les droites (EP_i)
2. Étudier les coordonnées polaires de P_i dans le repère $(R, [RE))$
3. Étudier géométriquement les déterminants $\left(\overrightarrow{RE}, \overrightarrow{RP_i}\right)$ un par un.

L'objectif final est de ne pas utiliser de flottants.

Solution: Sens Giratoire

Voir `https://prologin.org/static/archives/2015/questionnaire/correction.pdf` : Solution souhaitée :

```
int main(){
        int n, x, y, x1, y1, xd, yd ;
        scanf(%d%d%d%d%d%d%d, &x, &y, &n, &x1, &y1, &xd, &yd);
        for (int i = 0; i < n - 2; ++i){
                int xi, yi ;
                scanf(%d%d, &xi, &yi);
                int a = (x1 - x) * (yi - y) - (y1 - y) * (xi - x);
                int b = (xd - x) * (yi - y) - (yd - y) * (xi - x);
                int c = (x1 - x) * (yd - y) - (y1 - y) * (xd - x);
                if ((a >= 0 && (c < 0 or b < 0)) or (a < 0 && c < 0 && b < 0))
                        xd = xi, yd = yi ;
        }
        printf(%d %d
, xd, yd);
}
```

Coin Supérieur Droit

Étant donné un point $C = (x, y)$ du plan et un tableau $2D$ de cercles (où `cercles[i] = [xi, yi, ri]`), donnez un algorithme permettant de renvoyer vrai si on peut aller de l'origine du repère à C sans croiser de cercle, tout en restant dans le rectangle associé.

Solution: Coin Supérieur Droit

Tiré de `https://leetcode.com/problems/check-if-the-rectangle-corner-is-reachable/description/?envType=problem-list-v2&envId=geometry`. On dispose de cercles numérotés de 0 à $n-1$ et de 4 segments :

$$O(x,0), O(0,y), (0,y)(x,y), (x,0)(x,y)$$

. Remarquons qu'on ne peut passer entre deux cercles/entre un segment et un cercle si et seulement si il existe un point d'intersection entre les deux. On construit alors (en $\mathcal{O}(n)$) le graphe $G = (V, E)$ où V contient un sommet u_{c_i} par cercle et un sommet pour chacun des bords (u_n, u_e, u_s, u_o) et $(u, v) \in E$ si et seulement si les formes associées à u et v s'intersectent. Alors, on renvoie faux si et seulement si on peut trouver un chemin pour relier u_n, u_s, u_e, u_o, u_n, u_e ou u_o, u_s. Pour faire les calculs rapidement, on peut tout simplement utiliser l'algorithme union-find en considérant une partition en composantes connexes à partir du graphe vide ou un parcours en largeur/profondeur.