Feuille d'exercices n°7 de l'option d'informatique.

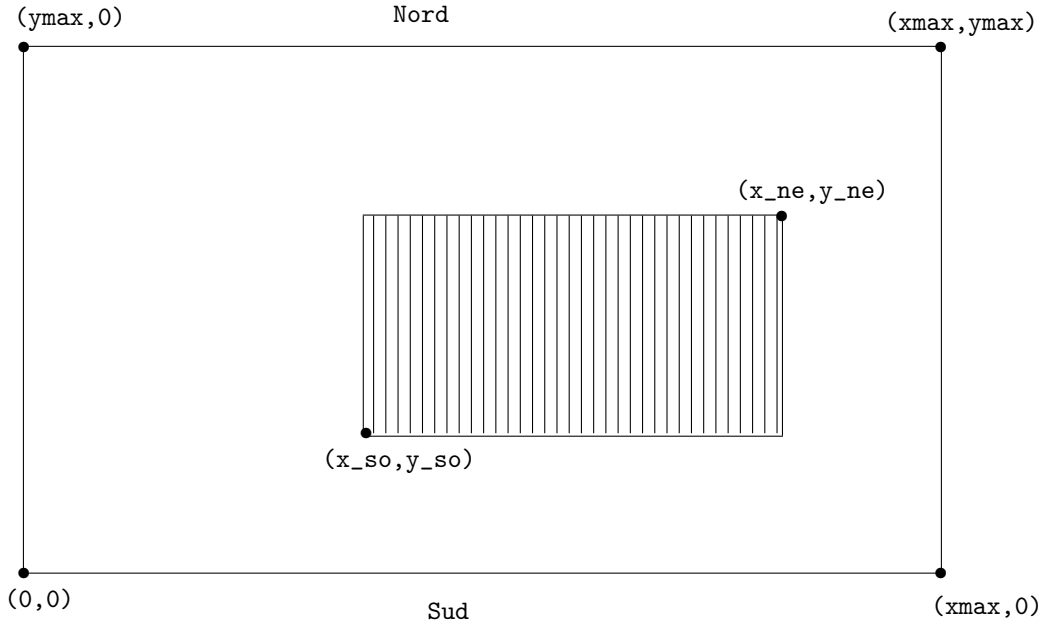
Un point de l'écran de l'ordinateur est caractérisé par ses coordonnées entières (x, y). On considère les rectangles de l'écran qui ont leurs cotés parallèles aux bords. Ils sont donc entièrement déterminés par deux points, le sommet «sud-ouest» et le sommet «nord-est» . On définit les points comme des couples d'entiers :

```
#let point1 = (10,24);;
point1 : int * int = 10, 24
```

et les rectangles comme des tableaux de deux points :

```
#let rect1 = [| (10, 10); (30, 20) |];;
rect1 : (int * int) vect = [| 10, 10; 30, 20 |]
```

rect.(0) représentant le point "sud-ouest" et rect.(1) le point nord-est.



a) Écrire une fonction **verif** qui vérifie si un rectangle est défini correctement, c'est à dire si le premier élément du tableau correspond à un point au sud-ouest du point associé au deuxième élément. **verif** retourne le booléen **true** si c'est le cas et le booléen **false** sinon.

b) Écrire une fonction "**EstIntérieur P rect**" qui retourne le booléen **true** si P est à l'intérieur de *rect* et le booléen **false** sinon (on prendra comme convention qu'un point situé sur la frontière du rectangle est à l'intérieur).

c) Écrire une fonction **Int_list** *récursive* qui étant donné un point P et une liste de rectangles l_rect retourne la liste des rectangles contenant le point P (on pourra utiliser la fonction **EstIntérieur**).

d) Écrire une fonction *itérative* qui à partir d'une liste de rectangles donne la somme des aires de chaque rectangle :

```
#SommeAires;;
- : (int * int) vect list -> int = <fun>
```

e) Écrire une fonction **disjoint** permettant de savoir si 2 rectangles sont disjoints (c'est à dire qu'ils n'ont pas de points en commun sauf éventuellement ceux du bord).

```
#disjoint;;
- : ('a * 'a) vect -> ('a * 'a) vect -> bool = <fun>
```

f) À l'aide de la fonction **disjoint**, écrire une fonction qui détermine si une liste de rectangles est formée de rectangles 2 à 2 disjoints (elle retourne **true** si c'est le cas et **false** sinon). Donner le nombre d'appels à la fonction **disjoint** et préciser la complexité dans le pire des cas de votre algorithme en fonction de la longueur n de la liste (on prendra en compte le nombre d'appels de la fonction **disjoint**).

g) Écrire une fonction **intersect** qui détermine le rectangle intersection de deux rectangles (on retournera un message d'erreur en cas de rectangles disjoints).

h) Écrire une fonction **intersect_list** qui à partir d'une liste de rectangles retourne le rectangle de l'intersection.

i) Écrire une fonction **dessine_list** qui à partir d'une liste de rectangles dessine les rectangles de la liste dans une fenêtre graphique.