TP10
Programmation C L2.1
Listes de points

Nous allons travailler avec une fenêtre de taille 512 par 512 pixels et avec des tableaux de points d'au plus 256 pixels. Nous allons utiliser la structure suivante, classique pour les points. Dans le but de stocker des polygones de taille variables, nous allons utiliser une liste chaînée de points. Ces structures sont rassemblées dans un fichier structures.h téléchargeable depuis la plateforme d'enseignement.

```
#define TAILLE_X    512
#define TAILLE_Y    512
#define NB_MAX_POINT  256
typedef  struct {
  int x;
  int y;
} Point ;

typedef struct cellulePoint {
  Point p;
  struct cellulePoint *suivant;
}CellulePoint,*ListePoint;
```

L'intégralité du code produit de ce TP devra former un unique programme, dont vous veillerez à ce que l'utilisation en soit agréable.

Rappelons que la fonction

```
int MLV_wait_keyboard_or_mouse(MLV_Keyboard_button* sym,MLV_Keyboard_modifier* mod,
                                                        int *unicode,int* x,int* y)
```

retourne `1` si on appuie sur une touche du clavier et `3` si on clique sur la souris. Par ailleurs, les trois premiers arguments contiendront les informations concernant la touche entrée si on appuie sur un touche du clavier, et les deux dernières les coordonnées du point cliqué si on clique dans la fenêtre. Concrètement, ici, vous n'aurez besoin de n'utiliser que les informations des clics de souris. Il vous est donc conseillé d'utiliser cette fonction sous la forme `MLV_wait_keyboard_or_mouse(NULL,NULL,NULL,&x,&y)`, où `x` et `y` sont des variables de type `int`.

1. **Dessiner un polygone pas à pas.**

   (a) Écrivez un programme qui déclare une `ListePoint` vide, créé une fenêtre, l'affiche, puis la détruit (après avoir attendu un temps raisonnable, ou un clic de souris). À chaque question suivante, vous n'oublierez pas de mettre à jour le programme principal pour qu'il appelle (et donc, pour tester) les fonctions réalisées.

   (b) Écrivez une fonction `ListePoint allouerCellule(Point pval)` qui alloue l'espace mémoire nécessaire pour une nouvelle cellule d'une liste de points et retourne l'adresse de cette nouvelle cellule après avoir affecté le point `pval` au champ `p` et `NULL` au champ `suivant`. Si il n'y a plus de place disponible, la fonction renvoie la valeur `NULL`.

   (c) Écrivez une fonction `int insererEnTete(ListePoint* liste, Point val)` qui insère le point `val` au début de la liste de points `* liste`. La fonction renvoie 0 en cas de problème et 1 sinon.

(d) Écrivez une fonction `int EntrerPolygone(ListePoint* lst)` qui à chaque clic de l'utilisateur ajoute le point entré à une liste de points, affiche la ligne entre le point entré et le point précédent, ceci jusqu'à ce que l'utilisateur appuie sur une touche du clavier. Le programme renvoie 1 si la liste de points a éte correctement créee.

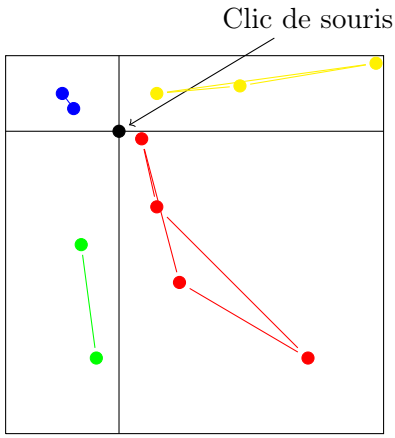
(e) Écrivez une fonction `void libererListe(listePoint* liste)` qui libère tout l'espace mémoire occupé par la liste `*liste`. La liste devra être NULL après l'appel

2. **Découper un polygone.**

(a) Écrivez une fonction `ListePoint extraireListeBasGauche(ListePoint* liste, Point p)` qui extrait de la liste des points contenus dans `liste` ceux qui sont situés en bas et à gauche du point `p` et en forme une nouvelle liste.

(b) De manière similaire, écrivez les fonctions `ListePoint extraireListeBasDrote(ListePoint* liste, Point p)`, `ListePoint* extraireListeHautGauche(ListePoint* liste, Point p)` et `ListePoint* extraireListeHautDroite(ListePoint* liste, Point p)`.

(c) Modifiez votre programme précédent pour qu'après avoir rentré son polygone, l'utilisateur rentre un point à la souris et le programme dessine un polygone de couleur différente dans chacun des quatre rectangles issus de ce point. (Un example est donné dans la figure suivante)



3. **Recoller deux polygones.**

(a) Écrivez une fonction `void concatenerListes(ListePoint* origine, ListePoints *ajout)` qui concatène la liste `ajout` à la fin de `origine`. la liste `*ajout`sera vide après l'appel

(b) Modifiez votre programme précédent pour qu'après un nouveau clic de souris, il y ai maintenant deux polygones : un au-dessus du point cliqué à la question précédente, et un au-dessous.

4. **Tri, le retour.**

(a) Écrivez une fonction `void deplacerEnTete(ListePoint * liste, Point p)` qui déplace la première occurence du point `p` en tête de la liste `*liste`.

(b) Écrivez une fonction `Point plusPres(listePoint* liste, Point p)` qui renvoie le point le plus près de `p` présent dans `liste`.

(c) Écrivez une fonction `void deplacerPlusPresEnTete(ListePoint* liste, Point p)` qui déplace en tête de `liste` le point le plus près de `p`.

(d) Écrivez une fonction qui place en tête le point le plus près de l'origine, puis en deuxième le point le plus près du premier, en troisième le point le plus près du deuxième (qui n'est pas le premier points), etc (sans s'occuper des points déjà traités).

(e) Modifiez votre programme pour trier les listes en fonction de cet ordre avant chaque affichage.