Département IMA / 3A (S6) / 2013 **Programmation Avancée**
Jonathan Aceituno, Vincent Aranéga, Nathalie Devésa, Rachid Elkouri et Laure Gonnord

Évaluation de TP sur Machine (1h45) - Liste de points
Manuel en ligne autorisé (man). Tout autre document/support interdit

Contexte

Soit un fichier texte contenant des points sous la forme *abscisse*, *ordonnee*, le but est de construire une liste des points triée sans doublons. On pourrait ensuite utiliser une librairie pour afficher des segments de droite, mais c'est hors du cadre de cette évaluation.

> **IMPORTANT !** Vous nous rendrez un contrôle TP qui **COMPILE correctement** (et avec des commentaires !), dans lequel il est facile de voir ce que vous avez réussi à faire. Les fonctions qui ne fonctionnent pas seront commentées avec la mention : "ne fonctionne pas !" (et éventuellement des explications . . .). **Un contrôle TP qui ne compile pas sera largement sanctionné!**

1 Préparation et consignes

Avant de commencer, copier le répertoire (entier !) :

`/home/imaEns/lgonnord/PA2013/controletp`

sur votre compte examen (pas sur le bureau !). Vous travaillerez uniquement dans ce répertoire de votre compte examen (`controletp`, donc !). Les fichiers situés à l'extérieur de ce répertoire seront ignorés. **Sauvez souvent vos codes !**

Le répertoire contient :

- `text.txt` : un fichier texte exemple.
- `result.txt` : un fichier montrant ce que votre programme doit produire. *Faites attention à ne pas écraser ce fichier lors de vos travaux!*
- `listepoints.h` : ce fichier contient les spécifications de la SD à implémenter.
- `main.c` : contient le squelette du programme à réaliser.
- `Makefile` : permet de réaliser la compilation séparée de listepoints+main.

Structure de donnée Liste La SD que l'on vous demande d'utiliser est une liste chaînée de `Points` (couple abscisse/ordonnée). La liste doit être (construite) triée par ordre lexicographique croissant, c'est-à-dire selon l'ordre suivant: :

- p_1 et p_2 sont égaux si leurs abscisses et leurs ordonnées sont égaux.
- le point p_1 est strictement inférieur au point p_2 si son abscisse est strictement inférieure , ou alors que les abscisses sont égales et l'ordonnée de p_1 est strictement inférieure à celle de p_2.

Par exemple, $(0, 1) < (0, 2)$ et $(0, 1) < (1, 100)$.

2 Travail à réaliser

Vos fonctions pourront être récursives ou itératives, au choix.

- Écrire dans un fichier nommé `listepoints.c` les fonctions d'entête fournies dans `listepoints.h`. Les spécifications de ces fonctions (ce qu'elles doivent faire!) sont fournies en commentaire dans `listepoints.c`.
- Dans le même temps, tester en complétant le programme `main.c`.
- Votre programme devra compiler en utilisant le Makefile fourni (sans modification).
- Déposer sur le moodle de PS les deux fichiers .c :https://moodle.polytech-lille.fr/

3 Annexe : le .h fourni

```
/*
 *  listepoints.h
 */
#include <stdio.h>

// Data structure
typedef struct point {
  int abs;
  int ord;
} Point;

typedef struct cell {
  struct cell *next ;
  Point value;
}  Cell;

typedef Cell *PtCell, *List;

//if p1 is lexicographically strictly less to p2
//returns -1 else if p1 is equal to p2 then returns 0
//otherwise returns 1
int compare_points(Point p1,Point p2)

// Updates the list of points
// if the point already exists : do nothing
// otherwize : adds the point in the lex order
void update(Point apoint, List* plist);

// Builds a SORTED (according to compare_points)
// list of points from a given file
void load(FILE* fp, List* plist);

// Prints the list
void print_list(List alist);

// Frees the list
void free_list (List* plist);
```