1 Modalités

Le projet est à réaliser en binôme (ou, éventuellement, en monôme). Les soutenances auront lieu début janvier, la date précise sera donnée ultérieurement. Pendant la soutenance chaque membre du binôme doit montrer la maîtrise de la totalité du code.
Le projet doit être divisé convenablement en plusieurs fichiers accompagnés de fichiers *.h pour les fichiers source ne contenant pas `main()`.
Il faut fournir un `Makefile` qui permet de compiler le projet par un simple `make`.
`make clean` supprimera tous les fichiers *.o, tous les exécutables et le fichier `*~` laissé par `emacs` de telle sorte que `make` suivant permettra de recompiler la totalité du projet.
Tous les fichiers du projet doivent être **obligatoirement** compilés avec les options

`-g -Wall`

et **cette compilation ne doit produire aucun message d'avertissement** (aucun warning). Ce point sera vérifié pendant la soutenance et les messages « warning » auront un impact négatif sur la note. Vous pouvez ajouter librement d'autres options de compilation.

Vous devez vérifier avec `Valgrind`

`http://valgrind.org`.

l'absence de fuites de mémoire. C'est très simple, voyez le tutoriel

`https://openclassrooms.com/courses/debuguer-facilement-avec-valgrind`

Si vous travaillez sur un mac il vous faudra faire cette vérification sur les machines de l'ufr (sauf si vous réussissez à installer Valgrind sur mac).

Vous pouvez utilisez toutes les fonctions du C standard à l'exception de la fonction `system()`. Plus largement, vous pouvez utiliser les fonctions POSIX (The Single Unix Specification `http://www.unix.org/online.html`).
Les fonctions propres à un système unix particulier sont à bannir et au final le projet doit compiler et s'exécuter sur les machines de l'UFR.
Toutes les fonctions que vous écrivez doivent être commentées dans les fichiers source : expliquer ce que la fonction est sensée faire, décrivez les paramètres et les valeurs retournées. (Si vous avez du mal à décrire une fonction alors sans doute elle est mal conçue, si votre projet n'est pas structuré convenablement et ne contient que les fonctions `main()` alors c'est le projet tout entier qui est mal conçu). Si vous utilisez des structures de données compliquées alors décrivez les aussi dans les commentaires.

Vous mettrez le projet dans le répertoire `nom1_prenom1_nom2_prenom2` où les noms/prénoms sont les noms et prénoms de membres du binôme. Dans un sous-répertoire CSM vous

mettrez quelques fichiers en format csm (vois ci-dessous) qui permettront de tester vos commandes.
Le répertoire principale du projet doit obligatoirement contenir le fichier README dans lequel

- vous décrivez brièvement le contenu de chaque fichier source,
- et, surtout, vous décrivez **en détail** quelles fonctionnalités vous avez réussi/échoué à implémenter.
  L'exemple de l'information à mettre dans README : « la commande `toto` implémentée mais l'option `-x` ne marche pas et l'option -z produit les résultats escomptés seulement si ... »).

Vous compresserez le répertoire pour obtenir le fichier `nom1_prenom1_nom2_prenom2.tar.gz` que vous déposerez sur moodle, avant 20h00 la veille de la soutenance. Tout autre type de fichier d'archive (zip, rar etc.) est prohibé. Le temps supplémentaire que je passerai pour décompresser les fichiers qui ne respectent pas cette consigne sera décompté de votre note.

2 Système de gestion de notes

Dans ce projet il s'agit d'écrire plusieurs commandes (programmes) qui implémentent les opérations les plus fréquentes sur les fichiers texte qui respectent le format décrit ci-dessous. Les commandes sont conçues dans le but de faciliter la gestions de notes d'étudiants.

2.1 Format de fichiers

Les fichiers auront un format vaguement similaire au format CSV : chaque ligne de données contient plusieurs données (champs) séparées par un caractère séparateur. Nous appelons le format décrit dessous CSM et on assume que les noms de fichiers de données portent l'extension csm.

- Le fichier CSM peut commencer par les lignes de commentaires dont le premier caractère est ♯. Ces lignes ne sont pas utilisées dans le traitement, ne contiennent pas de données et leur contenu peut être quelconque. Il est possible d'avoir 0 lignes de commentaires.
- Après les commentaires suivra une ligne qui définit le caractère séparateur. Elle contient juste un caractère (suivi immédiatement par le caractère de la nouvelle ligne). Le premier caractère de cette ligne sera utilisé comme le caractère séparateur dans les lignes de donnée.
  Le caractère séparateur par défaut est le point-virgule et si le fichier utilise le point-virgule comme séparateur alors la ligne qui définit le caractère séparateur peut être supprimée.
- Les lignes suivantes sont des lignes de données, par exemple la ligne
  002345434 ; Martin ; Thomas ; 5.9 ; abs ; 12.9
  contient 6 champs de données séparés par le caractère par défaut " ;" On devine que cette ligne donne le numéro d'étudiant, nom, prénom, la note du premier contrôle,

indique l'absence au deuxième contrôle et la note du troisième contrôle.

On préfère **fortement** que vos commandes soient capables de traiter les lignes de longueur quelconque et que vous n'assumez aucune limite sur le nombre de champs par ligne.

2.2 Commandes

Toutes les commandes ont la forme suivante :

```
nom_commande [options] [fichier ...]
```

où les fichiers qui suivent les options sont les fichiers CSM d'entrée de la commande (dans la plupart de commande il en faut au moins un, pour certaines commandes au moins deux). On suppose que les options précèdent les noms de fichiers d'entrée et l'ordre des options peut être quelconque. Pour traiter les options vous pouvez vous servir de la fonction `getopt()` mais ce n'est as obligatoire.

Il y a trois options qui peuvent être utilisées avec toutes les commandes :

(1) L'option

```
-o fichier_sortie
```

fournit le nom du fichier sortie qui sera créé par la commande. Si cette option n'est pas spécifié alors la commande envoie la sortie vers la sortie standard. Si le nom de fichier sortie n'a pas d'extension `.csm` alors la commande ajoutera cette extension.

Indication : si le fichier sortie est un de fichiers d'entrée alors il faut éviter la modification du fichier tant qu'il est utilisé comme le fichier d'entrée. Dans ce cas il convient d'utiliser un fichier temporaire (voir `tmpfile, tmpnam`) pour stocker les résultats de la commande et le recopier à la fin dans le fichier sortie. Si vous créez vous même le fichier temporaire (`tmpnam()` et `fopen()`) n'oubliez pas de le supprimer.

Attention : Si `fichier_sortie` existe et ce n'est pas un de fichiers d'entrée alors la commande doit demander à l'utilisateur une confirmation avant d'écraser ce fichier.

(2) L'option

```
-s c
```

indique que le caractère `c` doit être utilisé comme séparateur pour le fichier sortie. Si l'option `-s` n'est pas spécifiée alors on utilisera le séparateur par défaut. (Sur le terminal si le caractère séparateur est interprété par le shell il faut le mettre entre les quotes, comme `-s ','`.)

(3) L'option

```
-h
```

affiche l'aide de la commande qui détaille les options obligatoires et facultatives [1] de la commande.

Le fichier de sortie n'aura jamais de commentaires, les lignes de commentaires sont possibles dans les fichiers d'entrée mais pas à la sortie.

1. L'option facultative dans le sens que l'utilisateur n'est pas obligé de l'utiliser, par contre, pour vous, l'implémentation de chaque option est obligatoire.

2.2.1 Valeur de champ

Dans la suite on parle parfois de la valeur de champ. On suppose que la valeur d'un champ est une chaînes de caractères obtenue à partir du contenu de champ en supprimant les caractères blancs, au sens de `ispace()`, au début et à la fin de la chaîne.

2.2.2 union

La commande

```
union fichier fichier [fichier...]
```

produit la concaténation de plusieurs (au moins deux) fichiers d'entrée, i.e. le résultat de cette commande est le fichier qui contient toutes les lignes de données des fichiers d'entrée. Cette commande échoue si parmi les fichiers d'entrée il y en deux qui possèdent le nombre de champs différent par ligne.

2.2.3 select

```
select -l  n1,...,nk   [fichier]
```

L'option `-l` est obligatoire. La commande sélectionne les champs n1,. . . ,nk. Par exemple `select -l 0,6,4,0 f` produit le fichier qui contient dans chaque ligne les champs 0, 6, 4, et encore une fois 0 de chaque ligne de données du fichier d'entrée `f`. On suppose que les champs de chaque lignes sont indexés à partir de 0. Notez qu'un champ peut être sélectionné plusieurs fois. Si le fichier d'entrée n'est pas spécifié alors la commande prend l'entrée depuis l'entrée standard.

2.2.4 tri

```
tri -c [-n]  [fichier]
```

effectue le tri de lignes de données. L'option `-c` où `c` est un nombre entier non négatif indique que le tri doit être effectué par les valeurs du champs `c` (on indexes les champs à partir de 0).
Par exemple `tri -2 f` fait le tri du fichier `f` en utilisant les valeurs du champ 2 de chaque ligne.

(1) Par défaut il s'agit de tri dans l'ordre lexicographique, le champ de tri est considéré comme un champ de texte et la valeur du champ est la valeur dans le sens de la section 2.2.1.

    Les lignes avec le champ `c` vide (ne contenant que les caractères blancs) seront placées à la fin du fichier de sortie.

(2) L'option facultatif `-n` indique que le tri doit se faire par les valeurs numériques du champ `c`. Dans ce cas on suppose que le champ `c` contient soit une valeur numérique soit le mot `"abs"` (qui peut être précédé et suivi de caractères blancs).

Les lignes avec `abs` seront placées après les lignes avec le champ `c` contenant les valeurs numériques et mais avant les lignes avec le champ `c` vide.

Si le fichier d'entrée n'est pas spécifié alors la commande prend l'entrée depuis l'entrée standard.

2.2.5 moyenne_ponderee

```
moyenne_ponderee  -m formule  [fichier]
```

produit un fichier qui reprend toutes les lignes de données du `fichier` en ajoutant à la fin de chaque ligne un nouveau champ numérique. La valeur du nouveau champ est une somme pondérée calculée selon la formule qui suit `-m`. La formule aura la forme $c1 * k1 + ... + cj * kj$ où chaque `ci` est un numéro de champ et chaque `ki` un coefficient. Par exemple

```
moeyenne_ponderee -m '1*1.5+3*0.5+4*2' fichier
```

calcule pour chaque ligne la valeur

$$\frac{c_1 * 1.5 + c_3 * 0.5 + c_4 * 2}{1.5 + 0.5 + 2} \tag{1}$$

où c_1, c_3, c_4 sont les valeurs numériques de champs $1, 3, 4$. Notez que le dénominateur est la somme de coefficients.
Les champs vides sont traités comme s'ils contenaient 0.
Les champs contenant `"abs"` à la place d'une valeur numérique ne seront pas pris en compte dans le calcul de la moyenne, c'est-dire ces champs sont omis aussi bien dans le numérateur que dans le dénominateur. Par exemple si le champ 3 d'une ligne contient `"abs"` alors la formule (1) pour cette ligne prendra la forme $\frac{c_1 * 1.5 + c_4 * 2}{1.5+2}$. Si tous les champs de calcul ont la valeur `"abs"` la somme pondérée prendra aussi la valeur `"abs"`.
Si le fichier d'entrée n'est pas spécifié alors la commande prend l'entrée depuis l'entrée standard.

2.2.6 fusion

La commande

```
fusion -c n:m fa fb
```

fait fusionner deux fichiers `fa` et `fb`. Chaque ligne du fichier de sortie est obtenue en fusionnant une ligne de données du fichier `fa` et une ligne de données du fichier `fb`, les deux lignes fusionnent seulement si le n-ème champ de la ligne de `fa` et le m-ème champ de la ligne de `fb` ont la même valeur au sens de la section 2.2.1.

Par exemple `fusion -c '3:2' fa.csm fb.csm` fusionne les fichiers `fa.csm` et `fb.csm` en utilisant le champ 3 du premier fichier et le champ 2 du deuxième.
Les lignes de données du fichier `fa` qui ne peuvent pas être fusionnées (pas de ligne appropriée dans le fichier `fb`) doivent être envoyées sur la sortie d'erreurs standard (chaque ligne précédée par le nom du fichier `fa`). Le même pour pour chaque ligne du fichier `fb` qui ne peut pas être appariée à une ligne de `fa`.
Remarque : Vous pouvez assumer que le fichier `fa` n'a pas deux lignes avec la même valeurs du n-ème champ. On assume que la même condition est satisfaite pour les valeurs du champ m de `fb`.
Indications : Triez les fichiers `fa` et `fb` selon les champs correspondant avant de le fusionner.

2.2.7 fomatage

Pour toutes les commandes ci-dessus on a assumé que toutes les lignes de données ont le même nombre de champs. La commande

```
formatage fichier
```

s'applique à un fichier qui ne satisfait pas cette condition. La commande repère la ligne de données avec le plus grand nombre de champs n et ajoute à la fin de chaque ligne les champs vides pour obtenir toutes les lignes avec le même nombre n de champs.