TRAVAIL PRATIQUE 1

 Hiver 2017

Introduction

Le but de ce TP est de vous faire pratiquer les concepts vus au cours des 3 premiers chapitres du cours (pour ceux qui connaissent déjà le C, vous devez vous limiter à ces concepts). Vous allez devoir programmer une petite application qui calcule les notes finales lettrées associées aux notes obtenues par des étudiants à divers travaux. Cette application est une version très simplifiée du calcul fait par le logiciel Résultats.

Paramètres du programme

La ligne de commande du programme est la suivante, on supposant que l'exécutable s'appelle `tp1` :

`tp1 seuilMinA+ seuilMinA … seuilMinD < ficNotes`

- tout ce qui est en italiques doit être remplacé par une valeur appropriée, i.e. les valeurs seuils requises pour obtenir chacune des notes de A+ à D, dans cet ordre. Il ne doit pas manquer de valeur seuil, et elles doivent être spécifiées en ordre décroissant. Deux notes ne peuvent partager le même seuil. Les seuils peuvent comporter des virgules. *50.5*, *50.0* et *50* sont donc des valeurs seuil acceptables.

Le fichier des notes est envoyé sur l'entrée standard et doit respecter les consignes suivantes :

- Il commence par le nombre d'évaluations associées à la note finale (exemple : 4 s'il y a deux TPs et deux examens). Ce nombre doit être supérieur ou égal à 1 et inférieur ou égal à la limite spécifiée par le programme (voir ci-dessous).

- Suivent ensuite les pondérations associées à chacun des travaux. Chaque pondération est un entier compris entre 1 et 100 inclusivement et exprime un pourcentage de la note finale Naturellement, il faut que le nombre de pondérations soit égal au nombre de travaux et que la somme des pondérations vaille 100.

- Suivent ensuite les notes maximales potentielles pour chaque travail, qui sont des entiers supérieurs ou égaux à 1. Pour un travail noté sur 20, la note maximale sera de 20. Pour un travail noté sur 100, la note maximale sera de 100. Pour un travail noté en Succès/Échec, la note maximale sera de 1.

- On spécifie ensuite pour chaque étudiant son code permanent suivi des notes qu'il a obtenues pour chaque travail. Vous ne devez pas limiter a priori la taille du groupe (votre programme devrait pouvoir aussi bien traiter des petits groupes que des super groupes contenant des millions d'étudiants). Vous limiterez votre validation des codes permanents au fait qu'ils soient bien constitués de 4 lettres majuscules non accentuées suivies de 8 chiffres. Les notes doivent être comprises entre 0 et 100 inclusivement et peuvent comporter des décimales. Vous validerez que la note est bien spécifiée pour chaque travail.

Chaque élément du fichier est séparé des précédents et des suivants par un ou plusieurs caractères blancs, tels que définis par la fonction *isspace()*.

Dès que vous constatez un problème au niveau du contenu du fichier, vous devez afficher un message d'erreur (sur le canal d'erreur) et arrêter les traitements.

Traitement à effectuer

Vous devez afficher l'histogramme des notes finales lettrées obtenues par les étudiants. Pour cela, vous devez calculer pour chaque étudiant sa note finale pondérée sur 100 à partir des notes obtenues à chacun des travaux, puis convertir cette note finale numérique en une note finale lettrée en utilisant le barème de conversion fourni par l'usager sur la ligne de commande.

L'histogramme sera affiché en utilisant une ligne pour chaque note lettrée entre E et A+, en commençant par E et en finissant par A+. La note finale sera affichée sur 5 colonnes et cadrée à gauche. La barre d'histogramme sera affichée sur 50 colonnes maximum (donc une ligne ne contiendra jamais plus de 55 caractères lisibles). On utilise le caractère X (x majuscule) pour représenter la barre. Le nombre de X pour la note *note* est déterminé par la formule :

`sup(nbÉtudiants(`*note*`)/sup(nbMaxÉtudiants/largeurHistogramme))`

où :

- *sup(x)* représente l'entier immédiatement supérieur ou égal à *x,*
- *nbÉtudiants(note)* est le nombre d'étudiants associés à la note *note*,
- *nbMaxÉtudiants* est le nombre d'étudiants maximal associés à une note parmi les notes de E à A+,
- *largeurHistogramme* vaut 50.

Spécifications additionnelles

Votre programme doit spécifier un nombre maximal de 5 travaux par défaut, mais il faut pouvoir changer ce nombre maximal à la compilation au moyen de l'option facultative de gcc `-DNBMAXTRAVAUX=`*`nouvelleValeur`*

Exemples :

2 exemples sont fournis dans Moodle, *ficEnonce* et *ficEnonce1*. Ils sont associés à leur résultat par les lignes de commande suivantes, en supposant que l'exécutable s'appelle tp1 :

```
./tp1 90 85 80 76 73 70 66 63 60 55 50 < ficEnonce > resHisto

./tp1 90 86 82 78 74 70 66 62 58 54 50 < ficEnonce1 > resHisto1
```

Je vous fournis à titre d'information les résultats intermédiaires *resInterm* (associé à *resHisto*) et *resInterm1* (associé à *resHisto1*) afin de vous permettre de mieux valider vos traitements.

Vérification

- Le programme remis doit s'appeler *tp1.c* et se compiler sur le serveur *malt* avec la commande `gcc -W -Wall -lm` sans générer d'avertissements (ni de messages d'erreurs évidemment).
- Le TP se fait de préférence en équipe de deux étudiants, ou seul(e) sinon. Les noms des membres de l'équipe doivent aussi être indiqués au début du fichier *tp1.c*.

Évaluation (sur 100)

Fonctionnalité : 50 pts : le programme compile et fonctionne correctement.

Plus spécifiquement :

- Un programme qui ne compile pas avec le compilateur gcc installé sur *malt* aura nécessairement 0 du point de vue des fonctionnalités. CE N'EST PAS LE RÔLE DU CORRECTEUR DE CORRIGER VOTRE PROGRAMME POUR EN PERMETTRE LA COMPILATION.
- Si vous nommez votre fichier autrement que ce qui est demandé, vous perdrez des points.
- En cas d'erreur sur la ligne de commande, vous devez vous assurer que votre programme n'effectue aucun calcul ni entrée-sortie, qu'il affiche des messages d'erreurs **appropriés** (et exempts de fautes de français) sur *stderr* **en spécifiant en premier le nom de l'exécutable**, et qu'il se termine proprement.
- Vous devez vous assurer que votre programme fonctionne correctement avec toutes sortes de cas. Vérifiez que votre programme fonctionne correctement avec les exemples fournis, mais SVP ne vous limitez pas à cet exemple.
- Assurez-vous que votre programme réagit correctement avec les cas d'erreur (que ce soit les erreurs associées à une ligne de commande erronée ou celles associées à un contenu erroné dans le fichier des notes transmis sur l'entrée standard).
- Assurez-vous que chaque ligne affichée sur *stdout* ou sur *stderr* se termine par un retour chariot.

Structure et lisibilité du code: 50 pts.

Plus spécifiquement (cette liste n'est pas nécessairement exhaustive) :

- Le code imprimé doit évidemment correspondre au programme livré dans Moodle. La version imprimée doit être précédée d'une page de garde.
- Bon découpage en fonctions le cas échéant. Attention : c'est le rôle de chaque fonction de valider ses arguments. C'est donc une très mauvaise pratique d'avoir une fonction du style *valideArgumentsMain()* pour valider les arguments de main(). Chaque fonction doit être responsable d'une seule tâche, et doit donc pouvoir être nommée selon le patron *verbeComplements()*.
- Essayez de limiter au maximum l'utilisation de variables globales ou de valeurs fixées comme constantes du préprocesseur par la commande #define, de façon à permettre au besoin la réutilisation ultérieure de la fonction dans d'autres applications. Donc idéalement, tout ce dont une fonction a besoin pour fonctionner devrait lui être passé en paramètre.
- Utilisation de structures de données et de contrôle appropriées.
- Bonne disposition du code
- Commentaires pertinents. En particulier, vous mettrez un commentaire avant chaque déclaration de fonction pour spécifier son rôle. Mais vous devrez également commenter votre code lui-même de façon adéquate au besoin.

- Utilisation d'identificateurs significatifs spécifiés en français (pas de franglais).
- Présence d'un en-tête dans le fichier (spécifiant entre autres le nom du fichier, son rôle, ses auteurs, la date de création).
- Pas de *#include* inutiles
- Pas de hardcodage de valeurs constantes (ou hardcodage minimal).
- Utilisation de *char * argv[]* plutôt que de *char ** argv*.
- Utilisation des fonctions modernes de conversion en nombres (*strtol…* plutôt que *atoi…*).
- Vous pouvez utiliser d'autres fonctions de la bibliothèque standard du C que celles montrées en classe. Par contre, vous ne devez pas utiliser les tableaux à plusieurs dimensions, ni les structures, ni les entrées-sorties dans des fichiers ni l'allocation dynamique.

Remise

Le travail doit être réalisé seul ou en équipe de deux étudiants.

La remise électronique doit être faite **au plus tard le 20 février 2017 à 13h20**. Elle consiste à remettre le fichier source via Moodle, en le préfixant au préalable du code permanent des étudiants de l'équipe, comme suit : **AAAA99999999_BBBB99999999_tp1.c.** **IMPORTANT :** Une remise en mode brouillon suffit. Si vous faites une remise officielle, vous ne pourrez pas l'écraser avec une nouvelle remise. Seule la dernière remise effectuée sera évaluée, peu importe son statut.

Le dossier papier doit être remis **en mains propres le 20 février 2017 à 13h30**. Il consiste en une impression du code source précédé d'une page de garde, le tout **BROCHÉ**. N'oubliez pas de spécifier le nom des coéquipiers dans le code source et en page de garde de votre dossier papier. Du code informatique s'imprime toujours avec une police non proportionnelle, de type *Courier New*. Si jamais vos lignes de code sont fréquemment plus longues que le nombre de colonnes de votre page, SVP pensez à imprimer votre code avec une police plus petite (mais de taille 10 au minimum) et/ou à l'imprimer en mode paysage.

Les retards ne sont pas acceptés.