LANGAGE DE PROGRAMMATION :
PROJET DE FIN D'ANNÉE EN LANGAGE C.

1. AVANT-PROPOS ET REMERCIEMENTS

Nous y sommes, c'est la dernière ligne droite ! Au moment où nous présenterons ce travail, nous attaquerons la dernière ligne droite de l'année scolaire avec en ligne de mire les examens et les vacances. C'est non sans une certaine appréhension que nous commencions ce travail en C au début du mois de mars en sachant très bien que nous étions partis pour un petit bout de chemin et que la route pour arriver à destination pourrait être semée d'embûches. Mais étant donné que ce rapport vous est parvenu en temps et en heure, on peut en déduire que le projet est terminé et prêt à être défendu. C'est bien entendu le cas alors entrons dans le vif du sujet. Mais avant, une petite remise en contexte s'impose.

Tout d'abord le groupe entier tient à remercier Mesdames Masson et Vroman pour l'aide prodiguée au cours de ce travail en n'hésitant pas, parfois, à rester jusque 5h de l'après-midi un mercredi. Bien que les aides fournies puissent paraitre insignifiantes, il est très facile de rester bloquer sur un petit détail pendant assez longtemps pour s'arracher les cheveux et une fois cette étape passée, le reste devient tout de suite plus facile.

Ensuite, vous constaterez au premier contact que le programme peut sembler un peu brouillon et n'utilise pas les différents outils de C mis à disposition et, pour certains, abordés en cours. La raison de ces manquements est finalement assez simple. Notre volonté de prendre de l'avance et de finir ce projet dans les meilleures conditions nous a poussé à réaliser des procédures de tri avant même de voir le « qsort » pour ne citer que cet exemple-là. Il va de soi que tout ce qui est vu en cours est probablement plus simple et qu'ils seront utilisés lors de l'examen. En revanche, vous conviendrez que changer une procédure de tri faite « à la main » pour une procédure de tri automatique ne représente un intérêt que si le programme doit être « générique » ce qui n'est pas le cas ici.

Voilà nous pouvons aborder la présentation du projet proprement dite ! Notez bien que toutes les fonctions et les procédures ne sont pas expliquées dans ce rapport. En cas de de doute, n'hésitez pas à poser des questions.

2. CONSIGNES DE TRAVAIL

Le but du programme est de gérer les étudiants d'une école supérieure de type « EPHEC ». Attention, il ne s'agit pas de gestion de base de données même si la manière dont le programme fonctionne peut de temps en temps le laisser croire.

Une école supérieure contient bien entendu plusieurs années (le cycle court va jusque 3 ans) et plusieurs sections. Bien que le programme soit à même de gérer indifféremment toutes les « année/section », seule une A/S est à la fois. En effet, dans la plupart des cas, on ne gère en même temps qu'une seule classe et il est donc inutile de travailler avec l'ensemble des classes.

Les opérations possibles sur les étudiants sont l'ajout, la modification, la suppression et l'ajout de cotations. Concernant l'ajout (ou encodage) d'un étudiant, il faut bien entendu lui attribuer un matricule unique (et non réutilisable) qui servira lors de la recherche par exemple. Ensuite, il faut faire en sorte que l'utilisateur ne fasse pas d'erreur lors de l'encodage de la date et du choix de la classe de l'étudiant. L'utilisateur doit pouvoir à tout moment modifier les informations entrées pour un étudiant. Bien que les tests empêchent d'entrer des bêtises, il est très facile de faire une faute de frappe ou de lecture… Un étudiant peut tout aussi bien être supprimé. Dans ce cas il n'apparaitra plus dans les listes mais son matricule ne sera pas réutilisable. Enfin, il faut pouvoir ajouter des cotes à un étudiant et en fonction de ces cotes calculer sa moyenne et définir si l'étudiant a réussi ou échoué.

Concernant les fonctionnalités demandées, le programme doit être transparent au niveau de l'utilisateur et la gestion de l'année/section en cours doit donc être demandée en tout début de programme. Lors du premier démarrage, il n'existe aucun fichier (binaire ou texte). Un programme d'initialisation se charge alors de créer ces fichiers (qui devront être supprimés manuellement afin d'éviter les erreurs) contenant le nombre de classes, de cours ainsi que la liste des cours. Une fois cette initialisation effectuée, le programme principal peut démarrer.

Le programme principal va donc lire les informations dans le fichier texte et doit impérativement allouer la mémoire nécessaire à son bon fonctionnement. Après utilisation, l'utilisateur à la possibilité de sauvegarder les changements sur un fichier binaire indépendant du fichier texte.

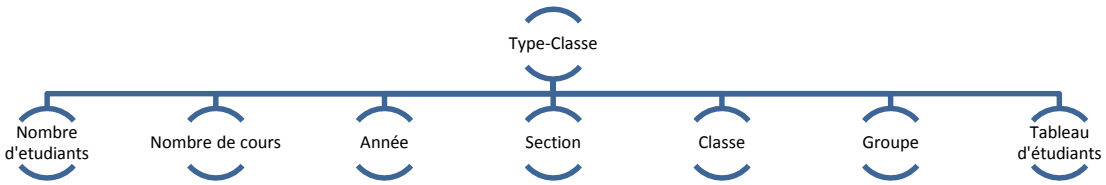
En résumé, ce programme doit gérer la lecture et l'écriture sur différents types de fichiers (binaire et texte) afin de conserver les données après la fermeture de programme. Il doit gérer les fonctions essentielles telles que le tri, l'affichage stylisé permettant un visuel clair pour un utilisateur lambda, la saisie « sécurisée » des données au clavier et l'allocation dynamique de mémoire. Pour y arriver, il faut bien entendu user des structures de données personnalisées et de beaucoup de patience !

3. STRUCTURE DES DONNÉES

Avant d'attaquer le codage proprement dit, attaquons nous à la structure. Ces structures sont utilisées dans les différentes procédures et la taille d'un type donnée dépend bien entendu des types définis qu'il contient. La taille de chaque élément à de l'importance dans l'allocation de la mémoire et donc dans la taille du fichier binaire généré.

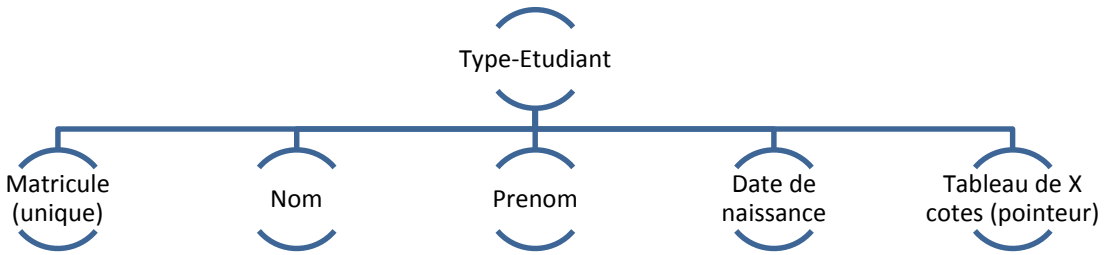
a) Une classe

Une classe contient bien entendu une année et une section afin par exemple de paramétrer l'affichage des listes d'étudiant par année et par section. C'est également utile lors de l'utilisation des fichiers de cours externes au programme et chargé lors de l'encodage et de la modification des cotes d'un étudiant. La classe et le groupe sont eux utiles pour le calcul de la moyenne d'une classe par exemple. Enfin les deux premiers entiers, le nombre d'étudiants et le nombre de cours permettent de calculer avec précision la taille en mémoire de la classe.



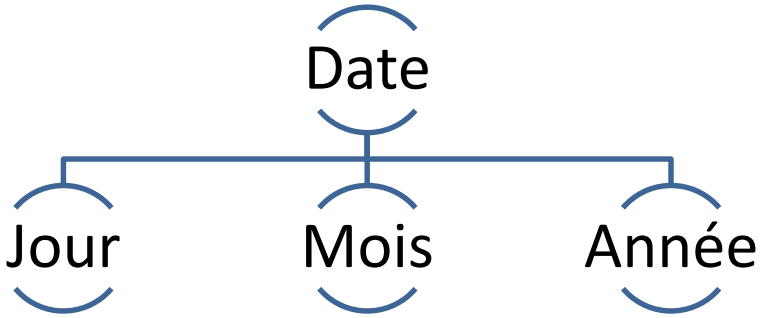
b) Un étudiant

L'étudiant est l'entité de base. L'énorme majorité des données traitées par le programme concerneront les étudiants. Le matricule est un numéro unique de l'étudiant. Il contient également un nom, un prénom et une date de naissance (type personnalisé). Le tableau de cotes est en réalité un pointeur vers un tableau de nombres réels (les float). Ce dernier n'est donc pas directement intégré dans la structure étant donné que le nombre de cours varie en fonction des années/sections. Le fichier de cours contenant les noms de cours ainsi que leurs pondérations est lu une seule fois en début de programme et est passé comme paramètre autant de fois que nécessaire.

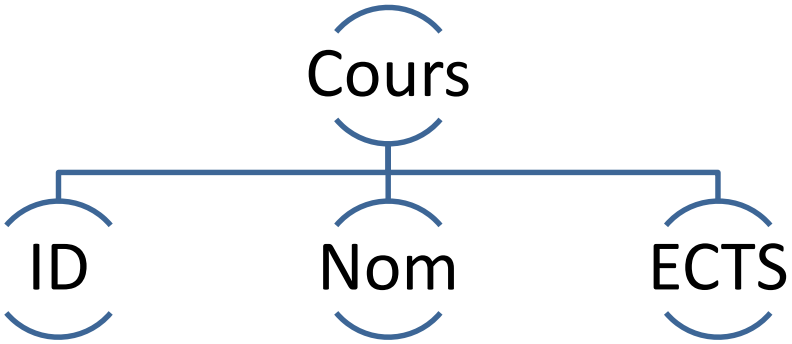


c) Une date

Enfin une date reste toujours utile à programme mais il est vrai que ce type de sert pas à grand-chose mais bon, soyons tolérants… et codons…



d) Un cours



Ce type est un peu spécial car il est chargé dans le programme en tant que structure mais les données qui utilisent ce type se trouvent dans un fichier texte généré par un autre programme. Ce petit programme génère un fichier texte avec comme première ligne le nombre de cours du fichier et ensuite un cours par ligne. Ce fichier texte est chargé une fois qu'on en arrive à encoder les cotes ou à les modifier. Attention que ce type varie légèrement d'un programme à l'autre (absence ou non de l'identifiant).

e) Les variables principales et constantes

Tous les programmes du projet ont comme constantes le maximum d'étudiants dans une classe (maxEtu) et la taille d'un nom utilisée dans le programme (tNom). En plus, le programme principal défini le nom de fichier des métadonnées (cf. matricule) sous le nom META.

Du point de vue des variables, tous les programmes ont comme variable principales un pointeur sur un tableau de classes (*tabclasses), l'année et la section actuellement traitée ainsi que le nombre de classes et de cours lus dans le fichier texte. Enfin, le programme principal et le programme pour les moyennes ont aussi comme variable principale un tableau de cours (*tabcours).

Les noms des fichiers binaires et textes sont également très importants dans la cadre de la lecture et de la sauvegarde mais ne sont bien entendu pas contant donc…

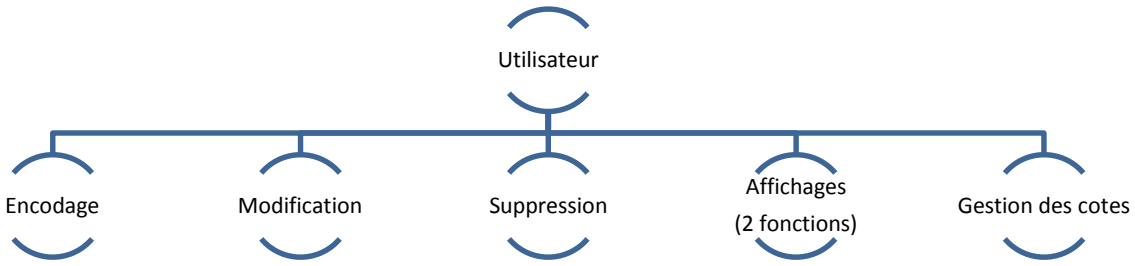
4. STRUCTURE DU PROGRAMME… OU PLUTÔT DES PROGRAMMES !

Après avoir évoqué la partie OSD du projet qui est commune à toutes les sections de notre projet, nous pouvons aborder l'aspect programme et structure du programme en particulier. Le projet (si l'on peut l'appeler comme ça) est composé de trois programmes à utiliser dans un ordre bien spécifique.

- Le programme d'initialisation qui se charge de générer les fichiers textes et binaires permettant au programme principale de ne pas travailler dans le vide. Ce programme doit donc être utilisé en premier
- Le programme principal offre les fonctionnalités primaires (aux yeux des utilisateurs) d'un programme de gestion d'étudiants. C'est à partir de ce programme qu'on peut modifier les étudiants, les cotes, les classes,… Bref tout ce qui fait la vie estudiantine !
- Le programme qui gère les étudiants une fois le travail d'encodage et de modification terminé. Ce programme permet tout simplement de créer un fichier texte reprenant la liste des étudiants classés dans l'ordre décroissant des moyennes et par la suite de l'imprimer (enfin je suppose…).

Les fonctions utilisent le même socle commun de fonctions et de procédures et ces fonctions peuvent donc être incluses dans une bibliothèque c chargée en début du programme et qui allège (et pas qu'un peu) la code du programme qui se résume alors à sa fonction main(). On peut ensuite classer les fonctions en deux sous-catégories aux usages et aux noms bien différents.

Les fonctions « utilisateurs » correspondent à des fonctionnalités directement visible et compréhensibles par l'utilisateur. Ce sont par exemple, l'encodage, l'affichage, la modification ou encore la suppression. Ces fonctions contiennent les textes qui informeront l'utilisateur sur ce qu'il doit faire ainsi que la saisie au clavier.



Les fonctions « outils » sont toute une série de fonctions transparentes aux yeux de l'utilisateur mais indispensable ou bon fonctionnement du programme. Elles sont utilisable à loisir dans tout le programme et permettent par exemple la recherche d'un étudiant, la vérification d'une date, la modification d'une classe, le tri (et oui, nous avions fini le tri avant de voir le qsort) ou encore la possibilité de lire une chaine en supprimant en ENTER final (utile dans le cadre de l'affichage).

5. POUR L'UTILISATEUR…

a) Le menu principal

⇨ int menu_principal(Tclasse * tabclasses)

Le menu principal est la fonction utilisateur par excellence. C'est la première interface utilisateur de programme et aussi la dernière (en cas de crash système mais ne soyons pas pessimistes). Cette fonction affiche les entrées du menu numérotées de 1 à 7 et permet la saisie d'une option sous forme d'un nombre entier avec les tests afin d'éviter à l'utilisateur d'entrer le nombre 8 par exemple. Une fois la bonne entrée sélectionnée, la valeur retournée par la fonction est utilisée par le switch principal et le programme poursuit son déroulement normal. A la fin de chaque tâche, le programme retourne automatiquement au menu et il est possible de quitter le programme et de sauvegarder depuis l'interface via une boucle « while ».

b) L'encodage des étudiants

⇨ void encodage(Tclasse*classes,int nbClasses)

La fonction d'encodage est une fonction qui ne renvoi rien. Elle permet à l'utilisateur de saisir au clavier les données classiques d'un étudiant qui sont le nom, le prénom, la date de naissance et la classe dans laquelle l'étudiant doit être inscrit. Le matricule est attribué par le programme sous la forme HExxxxxx et n'est jamais réutilisable. Le nom est le prénom ne présentent pas de risques particuliers et peuvent bien entendu contenir des espaces. La date de naissance est vérifié au moyen d'une procédure assez complexe en long, en large et en travers. Le choix de la classe s'effectue manuellement via une fonction dédiée car il est en effet possible qu'un étudiant aie fait la demande d'une classe en particulier. Cependant, afin de faciliter l'encodage, le programme choisi automatiquement la classe encodée précédemment pour tous les étudiants Lors de l'encodage, les cotes de l'étudiant sont initialisées à -1. Cela permet au programme de faire la différence entre une cote valant zéro (ce qui n'est pas très cool) et une cote non présente (qui peut poser un problème malgré tout).

c) L'affichage par liste d'étudiants

⇨ void affichage_etu(Tclasse * tableau, int nbClasses,int nbCours,Tcours *tabcours);

L'affichage est une procédure utile pour plusieurs raisons. Premièrement, c'est pour l'utilisateur le moyen facile de voir le nombre d'étudiants dans les différentes classes ainsi que le nombre de classe correspondant à son année/section. Ensuite, il est possible pour l'utilisateur d'afficher la liste des étudiants d'une classe et affichant alors une liste détaillé des étudiants classés par ordre alphabétique contenant le nom et le prénom (ah bon…), le matricule, la date de naissance et la moyenne de l'étudiant (en pourcentage).

d) La fiche signalétique

⇨ void affichage_etu(Tclasse * tableau, int nbClasses,int nbCours,Tcours *tabcours);

Comme le nom l'indique, la fiche signalétique permet l'affichage d'un étudiant dans ses moindres détails. S'y trouve le matricule, le nom, le prénom ainsi que la date de naissance de l'étudiant. L'utilisateur peut également avoir un aperçu net et précis des cotes déjà encodées avec l'affichage du cours, du nombre de crédits et de la cote attribuée. De plus, si toutes les cotes de l'étudiant ont été correctement entrées, la moyenne ainsi que le statut de l'étudiant sont affichés.

e) Modifier un étudiant et le supprimer

⇨ void modif_etu(Tclasse * tableau,int nbClasses);

Après avoir encodé un étudiant et en avoir affichés les détails, il peut très vite devenir nécessaire de le modifier. Une entrée du menu est spécialement prévue à cet effet en proposant à l'utilisateur de tout modifier (à l'exception du matricule) concernant les informations de base de l'étudiant. Il est ainsi possible de modifier le nom, le prénom, la date de naissance ainsi que de changer un étudiant de classe. Afin de faciliter la tâche à l'utilisateur, ce dernier ne modifie une information qui s'il entre une autre information à la place. Au cas où l'utilisateur ne désire pas modifier un champ, il lui suffit de le laisser vide.

⇨ void suppr_etu(Tclasse * tableau,int nbClasses);

Concernant la suppression, le programme effectue une recherche et supprime l'étudiant concerné. Ensuite, via une fonction particulière, il remonte les éléments qui se trouvaient dans le tableau afin de ne pas avoir à gérer l'espace vide entre les étudiants.

f) Ajouter des cotes

⇨ void ajout_cotes(Tclasse * tableau, int nbClasses,int nbCours,Tcours *tabcours) ;

Outre la modification, il est également possible d'ajouter des cotes à un étudiant. La procédure fait entrer comme paramètre le tableau de cours et affiche ensuite les cours correspondant à l'étudiant ainsi que les cotes déjà acquises par l'étudiant. Il est également possible de n'encoder que certaines cotes en initialisant la cote a -1. Ainsi, le programme pourra faire la différence entre un zéro et une cote pas encore encodée. Attention cependant au fait qu'une moyenne ne peut être calculée que sur un étudiant dont toutes les cotes ont été encodées. Si pour une raison ou une autre un étudiant n'a pas pu présenter un examen (par exemple), il obtient soit zéro si ce n'est pas justifié soit il doit le représenter avant de poursuivre son cursus.

6. C'EST MIEUX AVEC LES BONS OUTILS !

Les outils sont des fonctions ou procédures utilisée dans tout le programme sans ordre ni logique précise. Ces fonctions ont des usages aussi vastes que le retour du nombre de classe jusqu'au tri en passant par le calcul de la moyenne. Attention donc, l'ordre dans lequel ces différentes fonctions sont disposées dans les programmes du projet (et accessoirement dans ce rapport) peut sembler incohérent car elles sont programmées en fonctions des usages et des avancées dans l'énoncé. Pour s'y retrouver plus facilement, vous pouvez aller voir le sommaire ! Toutes ne seront pas expliquées en détails car il y en a près de 18 mais certaines valent le détour…

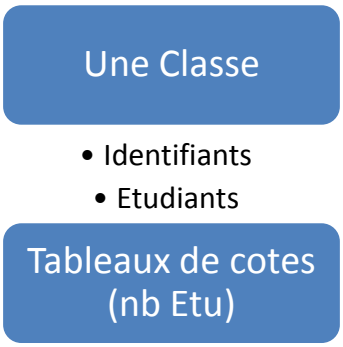
a) Initialisation du programme

- ⇨ char choix_section();
- ⇨ int ret_annee();

Les fonctions d'initialisation sont une étape essentielles car elles permettent à l'utilisateur de choisir quelle année/section sera traitée par le programme. Ces fonctions permettent donc à l'utilisateur de choisir l'année et la section en prenant bien soin de faire les tests adéquats pour générer les bons noms de fichiers.

b) La lecture et la sauvegarde sur le fichier binaire

La lecture et la sauvegarde sur le fichier binaire ne sont pas des procédures proprement dites mais font bien entendu partie intégrante du programme. Elles sont placées avant et après la boucle while du menu principal dans la fonction main. Attention au fait que la manière dont le fichier binaire et écrit (et donc lu) doit être cohérente entre les programme principal et le programme d'initialisation. Sans quoi, cela ne peut pas fonctionner !



La lecture dans le fichier binaire utilise les éléments se trouvant dans le fichier texte qui sont le nombre de classes ainsi que le nombre de cours de l'année/section. Une fois ces deux informations encodées, le programme utilise une boucle pour lire le fichier. La structure de fichier binaire est assez simple. Le programme lit d'abord une classe en entier avec ses identifiants ainsi qu'un tableau de 25 étudiants (statiques). Ensuite, en fonction du nombre d'étudiants réels de la classe, le programme lire les tableaux de cotes (des réels) le nombre de fois qu'il est nécessaire et mettra à jour le pointeur de l'étudiant lu précédemment lors de la lecture de la classe.

La sauvegarde utilise le même principe hiérarchique pour écrire les données. Lors de l'initialisation du fichier binaire, le programme écrit dans le fichier une classe contenant ces 25 étudiants avec pour chaque étudiant le pointeur vers un tableau de cotations. Alors bien entendu ce pointeur ne redirige vers aucun tableau mais étant donné le champ « nbEtu » présent dans chaque classe, le programme ne tentera pas d'initialiser un tableau inexistant. Ensuite, lors des autres démarrages, le programme pourra se fier au nombre d'étudiants et au nombre de cours pour charger en mémoire un tableau de la bonne taille et de lire le bon nombre de tableaux par classe.

c) Lecture et création du fichier texte

Pour bénéficier de certaines informations, le programme doit aller lire sur un fichier texte le nombre de classes, le nombre de cours ainsi qu'un tableau de cours à charger en mémoire au début de programme. Tout d'abord, il a fallu établir une convention de nommage pour les fichiers générés (cette convention s'applique également aux fichiers binaires avec l'extension .dat) qui consiste à créer une chaine de caractère manuellement avec comme premier caractère l'année et comme section la section en majuscule. Un petit tour de passe-passe permet de transformer l'année qui est en entier en en caractère en une ligne. Ensuite, il faut clairement définir ce que contient le fichier texte et la manière dont les données y seront formatées. Dans notre cas, la première ligne du fichier texte contient le nombre de classes et la seconde le nombre de cours. Les lignes suivantes affichent les cours avec un identifiant entier, une pondération réelle et un nom de cours en séparant les éléments par une tabulation.

```
4
14
0	6.000000	Mathematiques
1	4.000000	Principes de programmation
2	7.000000	Langage de programmation
3	5.000000	Electricite
4	5.000000	Electricite : Labo
5	5.000000	Phenomenes periodiques et TP
6	5.000000	Initiations aux systemes informatiques et TP
7	4.000000	Organisation et strcutre de donnees
8	2.000000	Electronique
9	4.000000	Electronique : Labo
10	1.000000	Connectique et cablage
11	3.000000	Telecommunication
12	8.000000	Anglais
13	1.000000	Anthropologie
```

Exemple d'un fichier texte formaté, le fichier 1T.txt

Les procédures d'écriture formatent donc le fichier de cette manière à partir de tableau dynamique créé lors du programme d'initialisation. Les fichiers textes et binaires ne peuvent être initialisés qu'une seule fois depuis le programme. Il est en effet impossible de modifier un fichier depuis ce programme car cela peut causer des dégâts irréversibles sur les informations contenues dans les fichiers. Seul une suppression manuelle des deux fichiers peut régler le problème !

- ⇨ int retour_nbCours(int *nbClasses,char fitexte[6]);
- ⇨ Tcours* lecture_cours(int nbCours,char fitexte[6]);

Les procédures de lectures sur le fichier texte permettent, dans l'ordre respectif, de renvoyer le nombre de classes ainsi que le nombre de cours et de créer le tableau de cours. Une fois le nombre de cours défini. Il suffit à la procédure de lecture d'allouer dynamiquement de l'espace à un tableau de type Tcours et à lire les cours. Ce tableau est chargé en début de programme et est passé comme paramètres aux fonctions d'affichage, de modification ainsi que pour le calcul de la moyenne.

d) Un matricule automatique

Un matricule généré automatiquement est attribué aux étudiants une fois un nom, un prénom et une date de naissance valide encodés. La procédure concatène une chaine de caractère contenant les deux lettres « HE » avec un entier formaté sur six chiffres. Le matricule prend donc la forme classique HE200882 par exemple. Pour conserver les réglages du matricule, le programme sauvegarde l'entier utilisé dans un fichier binaire nommées metaProg.dat. Ce qui permet de supprimer le fichier des étudiants (on ne sait jamais…) sans pour autant réattribuer les matricules déjà utilisés.

e) La recherche par matricule

⇨ Tetu recherche_mat(char * matricule, Tclasse * tableau,int nbClasses);

On ne peut effectuer une recherche efficace que si l'on recherche quelque chose qui est différent pour chaque étudiant. Cette « chose », vous l'avez deviné, c'est le matricule. Concernant l'aspect pratique, il est vrai qu'une fonction de recherche par nom aurait pu être utile mais étant donné que la procédure d'affichage permet d'obtenir les matricules d'une classe donnée, l'utilisateur pourra se tourner vers cette procédure s'il veut afficher le matricule d'un étudiant. Mais revenons à la recherche !

La fonction de recherche fait entrer un matricule, le tableau de classes ainsi que le nombre de classe et renvoie un étudiant (l'unique étudiant trouvé ou non). C'est une fonction classique qui parcourt tout le tableau avec deux boucles imbriquées (une pour les classes, l'autre pour le tableau d'étudiants). Lorsque la fonction trouve un étudiant, elle le renvoie et lorsqu'aucun étudiant ne correspond au matricule entré, elle renvoie un étudiant dont le matricule vaut « 00000000 » qui est bien entendu une chaine spécifique.

f) Le tri sur les noms et sur les moyennes

⇨ void tri(Tclasse * tableau, int nbClasses);

La fonction de tri est programmée « manuellement ». Autrement dit, elle n'utilise pas le qsort pour la simple et bonne raison que ce dernier a été vu après la programmation de cette fonction. Alors il s'agit d'une fonction de tri classique sous forme de procédure qui prend comme paramètre un tableau de classes ainsi qu'un nombre de classes. Le tri est effectué en majeur sur le nom de famille et ensuite sur le prénom.

⇨ void tri_moyenne(int nbClasses,Tclasse* tabclasses,Tcours *tabcours,int nbCours);

Il existe également une autre fonction de tri qui est utilisée dans le programme qui génère le fichier texte dans l'ordre décroissant des moyennes. C'est le même principe sauf que le tri s'effectue avant tout sur la réussite de l'étudiant, ensuite sur la moyenne obtenue et enfin sur le nom de famille de l'étudiant.

g) La modification d'une classe

⇨ void modif_classe(char * matricule,Tclasse * tableau);

La procédure de modification de classe est utilisée lors de la modification d'un étudiant et lors de sa suppression. Le tri éventuel étant effectué lorsque l'on retourne au menu principal. Cette procédure prend comme paramètre le tableau des classes ainsi que le matricule de l'étudiant concerné. Ensuite, via une recherche (un peu spécialisée car elle ne « renvoie » rien), le matricule donne lieu à l'emplacement de l'étudiant (sa classe et son indice). Avec ses informations, on peut supprimer l'étudiant de la classe et combler l'espace vide en remontant les éléments de 1 à partir de l'indice de l'étudiant en question. Cette procédure peut donc servir pour la suppression comme pour la modification car modifier un étudiant signifie l'enlever (temporairement) du tableau de classe pour le remettre dans une autre qui peut en réalité rester la même.

h) Le calcul des moyennes

⇨ float calcul_moyenne(Tcours *tabcours,float *cotes,int nbCours,float *total_ects)

En faisant entrer le tableau de cours, le tableau de cotations et le nombre de cours, il est assez facile de faire une boucle qui effectue les calculs. On pondère d'abord les cotes en fonction du nombre d'ECTS. Ensuite, on incrémente un nombre total de crédits réussis (supérieurs à 10/20) et une cote totale. Le petit calcul est un jeu d'enfant : cote totale / nombre de crédits. Ce calcul nous donne une cote sur 20 des cours réussis par l'étudiant.

i) Vérifier une date, tout un programme !

La fonction de vérification de date est une fonction utilisée une fois sur les trois programmes. Elle est utilisée dans le cadre d'ajout d'un étudiant.
Dans un premier temps, la fonction récupère la date du système afin de l'utiliser comme point de comparaison.
Dans un second temps, il est demandé à l'utilisateur d'entrer un jour, un mois et une année.
Une fois fait, la fonction vérifie tout d'abord l'année. Si celle-ci est ultérieure à l'année du moment ou antérieure à 1900, un message d'erreur s'affichera et demandera de réencoder.
Si la date se situe entre 1900 et est inférieure à la date actuelle, la fonction vérifiera si le jour n'est pas trop grand ou trop petit et indiquera une erreur si tel est le cas.
Enfin, si la date se situe au mois de février, la fonction vérifie s'il s'agit d'une année bissextile ou non et vérifie en fonction de cela si le jour est supérieur à 29 ou 28.
Lorsque l'utilisateur a bien entré toutes les données, la fonction s'arrête.

j) Vérifier un entier

Lors de l'utilisation d'entier, il est toujours difficile de vérifier s'il s'agit d'un entier, d'une chaîne de caractère ou d'un float. Nous avons donc décidé de créer notre propre processus de vérification.
Dans un premier temps, une variable string (verichar) est lue et remplace le « Enter » par « \0 »
pour qu'il n'entrave pas la vérification. Ensuite, notre procédure « verif_integer » vérifie chaque caractère de la chaîne pour voir s'il s'agit bien d'un chiffre et ce grâce à l'utilisation du code ascii. Si tous les caractères sont bien des chiffres, la procédure transformera la chaîne en entier. Si pas, elle donnera comme valeur à la variable « -2 » afin d'indiquer une erreur.
Enfin, la procédure retourne l'entier généré. Pour être certains qu'il n'y ait pas d'erreur, nous
Vérifions que la chaîne entrée n'était pas vide grâce à notre procédure « chaine_vide » qui si elle est positive donne la valeur « -2 » à la variable concernée.

7. CONCLUSION ET AVIS PERSONNELS

Alors ça y est, le projet est fini ! Bien entendu, on ne s'attendait pas que cela soit aussi facile qu'en Pascal lors du projet du premier quadrimestre. Mais il faut avouer que le compilateur peut parfois jouer de bien vilains tours. Il faut en effet se rendre à l'évidence, certaines de ses actions sont parfois tout bonnement incompréhensibles et il faut alors tenter de régler le problème même si il se met à afficher des « warning ». Le fonctionnement du programme a un prix !

Concernant la programmation, tout était assez clair au niveau de la structure du programme et de son fonctionnement. Les difficultés se sont plus concentrées au niveau de la sauvegarde des données (étant donné le comportement aléatoire du compilateur et de son usage de la mémoire) et de la compatibilité des fichiers entres les sous-programmes du projet. Il a fallu persévérer (et demander conseil) pour arriver au projet actuellement présenté dans ce rapport mais ça en vaut sûrement la peine.

Personnellement, il s'agit de mon premier projet en C en deux ans. C'est instructif et quand on y réfléchit bien plus utile que le Pascal. En revanche, ça représente une masse de travail considérable qui peut être source de quelques maux. Je veux bien réessayer dans un autre langage mais pas trop souvent quand même !

Antoine BETAS

Pour ma part, c'est aussi la première fois que je programme en C. Ayant déjà programmé avec d'autres langages, l'apprentissage d'un nouveau fut un plaisir pour moi. Ce projet a été un défi pour nous trois et je pense que c'est une réussite. En effet, après avoir passé de nombreuses heures dessus, il tourne comme on le souhaitait.

Cédric BREMER

De mon point de vue, ce projet fut un fameux défi de par ma non connaissance du C. Lors du projet en Pascal, j'avais dit que j'attendais avec grande impatience le projet en C et me voilà comblé.
J'ai pu découvrir que le C est aussi puissant qu'horrible. J'ai pu passer des nuits entières pour comprendre des erreurs alors que parfois il ne s'agissait que d'un « ; » manquant. Le C nous impose donc une certaine rigueur quant à son utilisation.
Ce projet était un défi et je pense avoir la fierté de pouvoir que c'est un défi réussi.

Alexis GEORGES