Rapport de projet 1A
2010-2011

Gestionnaire d'emplois du temps

Projet et rapport réalisés par
Simon Renoult et Kévin Renault

SOMMAIRE

Préambule.....3

Première partie

Sujet et fonctions implémentées.....4

Deuxième partie

Organisation et répartitions des tâches.....5

Troisième partie

Fonctions et structures.....6

Focus.....7

Quatrième partie

Mode d'emploi.....10

Conclusion.....11

Annexes.....12

PREAMBULE

Vous pourrez trouver au sien de ces pages le rapport complet de ce mois de travail conjugué entre Kévin Renault et moi-même.

Il est essentiel de comprendre que ce projet est le premier auquel nous avons participé, il a donc été nécessaire (et difficile) de mettre nos connaissances et notre organisation à niveau, de sorte que nous puissions partager tâches et apprentissages sans que l'un ou l'autre ne soit débordé ou lésé d'une quelconque manière.

Il est aussi à noter que le programme dont ce document fait un rapport a été entièrement rédigé en langage C, sous environnement GNU/Linux (*debian-based*) à l'aide de l'éditeur de texte libre Gedit (licence GPL) et compilé *via* GCC (licences GNU GPL et GNU LPL). Nous nous sommes aidés de quatre différentes bibliothèques standards du C à savoir *stdio*, *stdlib*, *string* et *ctype.*

S'il n'est pas précisé une quelconque licence d'utilisation c'est que nous estimons notre travail encore trop immature pour prétendre à un quelconque intérêt extérieur au delà du simple apprentissage scolaire d'un langage de programmation et de la gestion d'un projet. Néanmoins si une décision devait être prise sur ce point, la licence GNU GPL dans sa version 3 serait privilégiée afin de partager avec d'autres les mêmes connaissances que nous avons nous-même pu acquérir dans cette entreprise.

PREMIÈRE PARTIE

1. Sujet

Le sujet qui nous a été proposé exigeait de nous la réalisation en mode console (*shell)* d'un outil de gestion d'emplois du temps. Le langage à utiliser était le langage C que nous avions étudié pendant un peu plus de trois mois au premier semestre.

La *deadline* se situait dans le courant du mois de janvier. Nous avions donc un mois pour réaliser un projet dont nous ignorions alors l'ampleur et le temps de travail effectif qu'il allait exiger, ignorance d'autant plus évidente que ce projet était notre premier, aucune expérience passée ne pouvant nous aider d'une manière ou d'une autre.

2. Fonctions implémentées

Plusieurs exigences en termes de fonctionnalités étaient imposées directement depuis le sujet sans réflexion préalable de notre part.

Nous devions inclure la possibilité de planifier ou annuler un cours directement depuis le logiciel selon plusieurs critères, extraire l'emploi du temps dans un fichier afin de l'exploiter ultérieurement ou même pouvoir charger ce fichier directement en mémoire. Nous devions aussi envisager l'affichage de cet emploi du temps en précédemment saisi/ importé. Tout cela à l'aide d'un menu aidant à la navigation dans le menu, à la saisie des critères d'affichage, de modification, de planification, etc.

L'ensemble de ces saisies devaient se soustraire à un ensemble de contraintes logiques, invisibles pour l'utilisateur mais renvoyant une erreur en cas de conflit.

Nous avons aussi décidé d'optimiser l'occupation de la mémoire par certaines fonctionnalités du langage C en ayant recours aux fonctions d'allocations mémoires (*malloc*, *calloc*, *realloc* et *free*) permettant un ajustement optimal de l'espace occupé par le programme en fonction de la taille de l'emploi du temps.

DEUXIÈME PARTIE

Organisation et répartition des tâches

Une fois notre groupe formé avec le choix du binôme (ce choix s'étant fait avant tout sur la base d'affinités plutôt que sur les capacités et connaissances techniques de chacun) nous nous sommes penchés sur le sujet afin de déterminer les difficultés que représenteraient la réalisation du programme. Estimations qui se sont révélées très largement erronées comme nous devions le découvrir un peu plus tard.

L'organisation fut à l'origine assez anarchique, ne sachant pas vraiment comment nous y prendre par manque d'expérience passée. Nous avons donc commencé par programmer seulement quand nous pouvions nous réunir, c'est-à-dire la journée à l'IUT ou bien en nous donnant rendez-vous chez l'un ou chez l'autre. Nous travaillions alors chacun sur le même problème, mettant au service de la résolution des problèmes qui se posaient à nous nos deux personnes. Néanmoins ce système avait ses limites puisque nous perdions un temps important à trouver des créneaux communs parfois trop courts pour effectuer le travail nécessaire.

Nous nous sommes donc décidés à travailler chacun de notre côté, nous attardant respectivement sur des points précis différents. Autant Kévin effectuait des recherches de contenu, de liens, de conseils d'utilisation afin de créer une nouvelle fonctionnalité, autant j'implémentais directement les fonctions issues de ces recherches dans le code source et me confrontait aux difficultés de syntaxes qu'impose le langage C.

Cette répartition était aussi motivée par l'utilisation des qualités de chacun. En effet, étant très pointilleux dans la rédaction, j'ai rédigé et organisé le code source de manière claire et homogène, créé un ensemble de fonctions usuelles évitant les redondance de lignes de codes similaires. De son côté Kévin s'attaquait à la rédaction des contraintes d'intégrité, à l'élaboration de quelques algorithmes nécessitant un temps plus important de réflexion.

Nous avons en outre organisé les sources du programme sous forme de versions, sans pour autant ressentir le besoin d'utiliser un logiciel de gestion de versions (VCS). À chaque nouvelle fonctionnalité implémentée était créée une archive considérée comme version stable. Une copie de cette version était créée afin de travailler sur une version de développement.

TROISIÈME PARTIE

1. Fonctions

Une quarantaine de fonctions sont utilisées dans notre programme afin de minimiser les redondances de lignes de codes. Ces fonctions sont organisées dans des *headers* particuliers selon la fonctionnalité à laquelle ils correspondent. En voici la liste suivie d'une rapide description :

int main (void)
Fonction principale du programme de type entier initialisant le pointeur sur le tableau d'emploi du temps, affichant le portail de présentation du programme et lançant le menu principal depuis lequel les fonctions s'enchaîneront. Il libère la mémoire puis retourne une valeur en fin de programme.

int menu_principal (void)
Présente le menu principal, saisit le choix de l'utilisateur et lui renvoie le menu correspondant. Retourne une valeur utilisée pour quitter le programme.

void menu_affichage (void)
Présente le menu d'affichage et l'option d'affichage (étudiant ou professeur), saisit le choix de l'utilisateur et renvoie le menu correspondant.

void menu_affichage_eleve (void)
Présente le menu d'affichage d'un emploi du temps d'un étudiant selon plusieurs critères (promotion, TD, TP) puis appelle la fonction correspondante.

- **void affichage_promotion (void)**
  Saisit la promotion et affiche l'emploi du temps en conséquence.

- **void affichage_td (void)**
  Saisit la promotion et le TD et affiche l'emploi du temps en conséquence.

- **void affichage_tp (void)**
  Saisit la promotion, le TD, le TP et affiche l'emploi du temps en conséquence.

- **void affichage_professeur (void)**
  Saisit le professeur et affichage l'emploi du temps en conséquence.

Void menu_edition (void)
Présente les options d'édition (insertion, suppression, suppression complète), saisit le choix de l'utilisateur et renvoie la fonction correspondante.

- **Void insertion (void)**
  Saisit à l'aide de questions les informations du cours à insérer.

- **Void suppression (void)**
  Saisit à l'aide de questions les informations du cours à supprimer.

Void menu_extraction (void)
Présente les options d'extraction (mémoire, professeur, promotion, td, tp) puis appelle la fonction correspondante.

- **Void choix_extraction (int)**
  Enchaînement de if selon le paramètre de la fonction. Ouvre un fichier et appelle la fonction d'extraction correspondante. Ferme le fichier.

- **void type_extraction (FILE*, int)**
  Switch sur l'entier pris en paramètre. Parcours le tableau en mémoire et appelle la fonction d'extraction selon certaines restrictions.

- **Void extraction_edt (FILE*, int)**
  Copie dans le fichier passé en paramètre via un pointeur les informations triées selon les restrictions émises précédemment.

Void menu_chargement
Présente les options de chargement (mémoire, professeur, promotion, td, tp) et appelle la fonction d'import selon certains paramètres.

- **Void import_edt (char[], char[])**
  Copie dans une structure temporaire les informations du fichier, les soumet aux contraintes et les recopie éventuellement en mémoire.

La liste des fonctions qui suit est celle provenant du fichier source *toolbox.c* recensant une multitude de fonctions utilisées tout au long du programme.

Void portail (void)
Portail ouvert en début de programme présentant le logiciel.

Void cloture_pgm (void)
Suite d'instructions à effectuer juste avant la fermeture du programme.

Void verif (void)
Printf de vérification.

void verif_chaine (char[])
Printf de vérification du contenu de la chaine passée en paramètre.

Void verif_entier (int)
Printf de vérification de l'entier passé en paramètre.

Void init_EDT (int)
Modifie la taille de l'emploi du temps via un *realloc*. Le paramètre défini la taille du tableau.

Void saisie_chaine (char[])
Saisie une chaine de caractère via *fgets* et appelle la fonction *clear*.

void clear (char[])
Supprime le caractère '\n' de la chaine passée en paramètre.

Void vider_buffer (void)
Vide le *buffer stdin*.

Void option_sortie (int)
Switch sur le paramètre. Affiche une information selon ce paramètre.

Void confirmation (void)
Printf requérant une confirmation de l'utilisateur.

Void saisie_matiere (char[]), Void saisie_annee (char[]), Void saisie_enseignant (char[]), Void saisie_salle (char[]), Void saisie_jour (char[]), Void saisie_type (char[])
Indique le champs à insérer et appelle la fonction saisie_chaine sur le paramètre saisi en entrée.

Void saisie_debut (int*, int*), Void saisie_fin (int*, int*)
Indique le masque du champs à insérer et vide le *buffer* une fois la saisie effectuée.

Void verif_type (char[], int*, int*)
Remplis les champs td/tp automatiquement selon le type de cours saisi précédemment.

Void saisie_td (int*), Void saisie_tp (int*)
Indique et saisi la valeur du td/tp et appelle la fonction *vider_buffer*.

Void contraintes (struct Cours)
Applique un ensemble de contraintes logiques à une structure passée en paramètre.

Int verif_midi (int, int, int)
Retourne une valeur selon le résultat d'opérations effectuées afin de vérifier si une saisie est bien conforme aux restrictions du midi ou non.

2. Structures

```
typedef struct Enseignement {

        char    matiere[LG_MAX];
        char    enseignant[LG_MAX];
        char    type[LG_MAX];
        char    salle[LG_MAX];
}Ens;

typedef struct Groupe {

        char    annee[LG_MAX];

        int     td;
        int     tp;
}Groupe;
```

```
typedef struct Horaire {

        int     heures;
        int     minutes;
}Horaire;


typedef struct Cours {

        Ens     enseignement;

        Horaire heureDebut;
        Horaire heureFin;
        char    jour[LG_MAX];
        Groupe  groupe;
}Cours;
```

3. Focus

Des différents points que nous avons abordés, plusieurs nous ont paru dignes d'intérêts tant par leur technicité que par leur difficulté de manipulation. Il a nous fallu pour certains d'entre eux plusieurs semaines afin de les mettre en place, depuis la compréhension de leur prototype jusqu'à leur implémentation définitive et stable dans le code.

Allocation dynamique :

Ce premier focus porte sur des fonctions de la bibliothèque *stdlib.h* permettant la gestion dynamique de la mémoire à savoir *malloc*, *realloc* et *calloc*.

La question de l'utilisation de ces fonctions s'est posée assez rapidement. En effet tout le programme se basait sur l'utilisation de l'emploi du temps, ainsi nous devions soit déclarer un tableau aussi grand que le nombre de possibilités soit gérer de manière dynamique cette taille. Chacun de ces choix avait ses avantages et inconvénients. Dans un soucis d'optimisation, nous avons pris la décision de nous essayer à la gestion dynamique de la mémoire.

L'allocation dynamique s'est faite dans notre programme à l'origine par l'utilisation de la fonction *malloc* (*void *malloc(size_t size)*). Nous l'utilisons à l'ouverture du programme afin d'initialiser le pointeur sur l'emploi du temps (de type *struct Cours*). La taille passée en paramètre est une variable globale (appelée *size*) nous permettant de faire varier la taille du tableau, lui même déclarer en variable globale.
On trouve dans un deuxième temps l'utilisation de la fonction *realloc*(*void *malloc(size_t size)*) qui nous permet à chaque insertion d'un cours ou import depuis un fichier de créer une nouvelle case dans ce tableau. Dans le cas de la fonction de suppression ce *realloc* est utilisé en lui passant en paramètre *size*-1.
La fonction *calloc* (*void *malloc(size_t size*)) quant à elle nous permet de remettre à zéro tous les bits en mémoire et de vider ainsi tout l'emploi du temps. Suivie d'un *realloc*, cette fonction nous permet de réinitialiser le tableau.

Nous avons mis un certain temps avant que cette fonctionnalité soit opérationnelle, en effet si son utilisation nous paraît aujourd'hui relativement aisée, elle nécessite une certaine rigueur dans la rédaction du code.

Lecture/Écriture dans un fichier :

Ce point était un pivot essentiel dans l'élaboration de notre programme. En effet son implémentation était nécessaire pour l'export des saisies mais aussi pour leur import ultérieur. En outre, il confrontait les notions vues en cours avec une utilisation concrète dans un programme ce qui nous imposait son utilisation.

Le plus difficile ici n'était pas tant la mise en place que la syntaxe. En effet nous avons éprouvé de nombreuses difficultés à élaborer la fonction d'import. Autant la fonction d'export via *fprintf* (*int fprintf(FILE *stream, const char *format, …)*) s'est faite aisément, autant la fonction d'import a connu une gestation difficile. Les questions de masque, de longueur de chaînes, de caractère non reconnu nous ont largement compliqués la tâche.

Nous avons finalement résolu le problème avec un *fgets* de chaque ligne dans une chaîne puis une copie de cette ligne en mémoire par la fonction *sscanf* que nous n'avions pas envisagé à l'origine.

QUATRIÈME PARTIE

Mode d'emploi du logiciel

Après avoir lancé le programme, un portail est affiché qui vous informe des conditions d'utilisation, de quelques astuces d'utilisation comme l'option de sortie ou les contraintes à la saisie d'un champs.

(Annexe 1.1)

Arrivé au menu principal plusieurs options se proposent à nous. On peut au choix :

– Afficher un emploi du temps en mémoire ;
– Éditer l'emploi du temps (insérer ou supprimer) ;
– Extraire ce qui se trouve en mémoire dans un fichier ;
– Importer des données depuis un fichier ;

(Annexe 1.2)

Si l'on décide d'insérer un cours, direction EDITION puis le choix « 1 : Insertion d'un champs ». Reste à remplir les différentes champs nécessaires et valider la saisie.

Deuxième option : importer un cours se trouvant dans les fichiers de la bibliothèque. Pour cela, allez depuis le menu principal dans le sous-menu « 4 : CHARGEMENT » . Choisissez le fichier selon son contenu (éditable directement depuis le fichier).

(Annexe 1.3 et 1.4)

Une fois le tableau rempli, vous pouvez combiner ces deux méthodes afin de créer un emploi du temps complet.

Vous pourrez ensuite le consulter directement depuis le logiciel via la fonction d'affichage disponible dans le menu principal puis « 1 : AFFICHAGE » et le remplissage de quelques critères. À noter qu'il est possible de consulter autant l'emploi du temps d'un professeur que celui d'une promotion, ou TD ou TP.

(Annexe 1.5)

La consultation peut aussi se faire directement depuis les fichiers extraient précédemment. Ils se trouvent par défaut dans le répertoire source du logiciel puis « bibliotheque ».

(Annexe 1.6)

Si une saisie est erronée vous pouvez à tout moment supprimer un cours depuis le menu d'édition ou encore effacer tout ce qui se trouve en mémoire.

(Annexe 1.7)

CONCLUSION

Pour clore ce rapport nous pouvons avouer que de tout ce qui nous a été proposé durant ce semestre de programmation, l'élaboration de ce programme a été l'expérience la plus prodigue d'enseignements, de techniques et savoir faire.

Sur le plan technique, nous nous sommes rendus compte des difficultés que présentait le langage C en terme de syntaxe et d'utilisation. La rigueur tant syntaxique que structurale est une vertu essentielle dans le développement d'un programme quand bien même sa longueur n'excède le millier de lignes. Et au delà du simple apprentissage en amphithéâtre, il semble essentiel de posséder une capacité d'abstraction afin d'envisager toutes les possibilités que peuvent recéler une simple fonction standard. Il est certain que nous n'avons jamais autant appris que durant ce mois de programmation. Il reste cependant regrettable que l'expérience n'ait pas été menée plus tôt dans l'année avec des projets de moindre envergure.

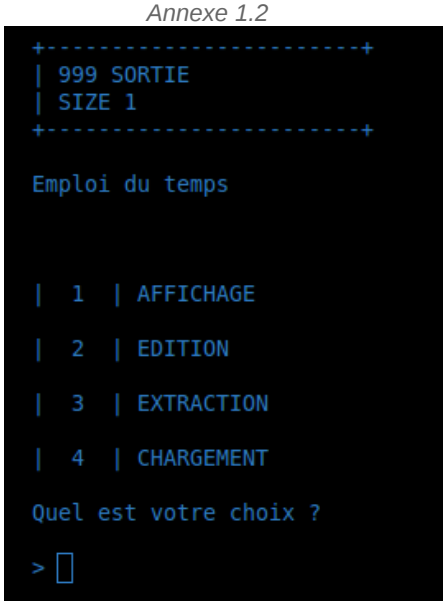
Une autre facette à aborder est celle de la gestion de l'effectif. En effet nous nous sommes très rapidement rendus compte que la gestion d'un simple binôme était chose ardue, d'autant plus qu'une période de fête s'intercalait dans le développement du programme et que, de fait, les interactions entre chaque membre de la pair en devenait encore moins aisée. On découvre alors la nécessité de travailler avec des gens motivés, au delà de leur simples connaissances techniques.

On pourra cependant regretter l'absence de challenge scientifique dans l'élaboration d'algorithmes plus compliqués. En effet les difficultés rencontrées (qui furent légion tant cet exercice était nouveau) étant pour la grande majorité d'ordre syntaxique et non d'ordre algorithmique. Mais reste que cette expérience nous a donné goût au développement de projets de groupes, et, si la gestion d'un effectif peut parfois sembler malaisé, elle n'est rien comparée à la rudesse d'un développement solitaire.

ANNEXES

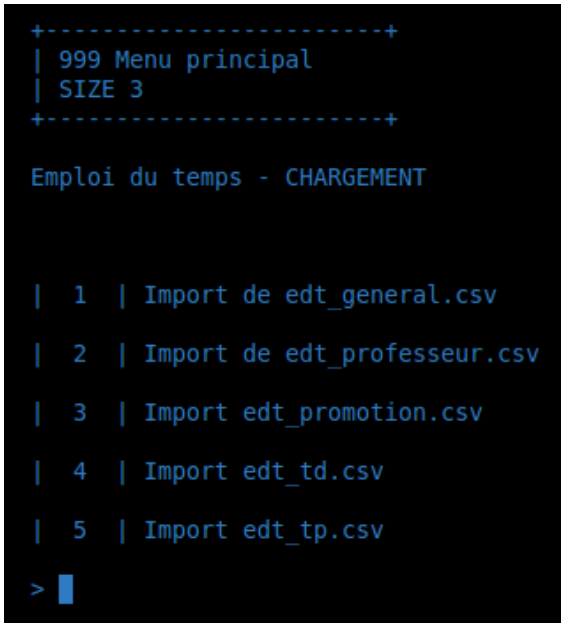
Annexe 1.1

Annexe 1.2



Annexe 1.3

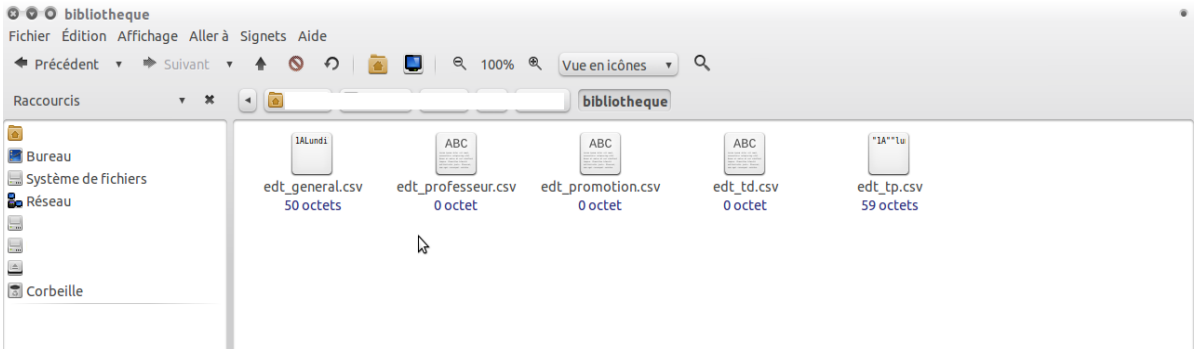
Annexe 1.4



Annexe 1.5

```
+-----------------------+
| 999 Menu principal
| SIZE 3
+-----------------------+

Emploi du temps - AFFICHAGE


|  1  | Afficher l'emploi du temps d'un étudiant

|  2  | Afficher l'emploi du temps d'un professeur

> 
```

Annexe 1.6



Annexe 1.7

```
Emploi du temps - EDITION SUPPRESSION CHAMPS

[*] Promotion concernée (1A ou 2A) : 1A

[*] Jour (Lundi/Mardi/Mercredi/Jeudi/Vendredi/Samedi): Lundi

[*] Type de cours (TD ou TP ou Amphi) : TP

[*] Groupe (TD) concerné : 1

[*] Groupe (TP) concerné : 1.1

[*] Heure de début (HH:MM) : 14:00
```

Annexe 1.8

```
+-----------------------+
| 999 Menu principal
| SIZE 1
+-----------------------+

Emploi du temps - EDITION


| 1 | Insertion d'un champs

| 2 | Suppression d'un champs

| 3 | Suppression de tous les champs

> 3


Etes vous certain de vouloir effectuer cette opération (oui/non) ? oui
```