U.E. SYSTEME ET PROGRAMMATION PROCEDURALE

GUIDE DU PROJET

To be “*ScreenSaver*”

or not to be.

This is the project !

PROJET SYSTEME & P.P.
Julio Santilario jsantilario@cesi.fr
A1 2016/2017

CONTENU

1. Introduction

France, 07/12/2016.

MATRIX International Company est leader mondial dans la création et la diffusion auprès du grand public des écrans de veille multi-OS. Coté en bourse, son action, depuis 20 ans, n'a pas cessé d'augmenter considérablement. Avec l'arrivée massive des smartphones, elle s'est diversifiée également pour les écrans de veille de ces nouveaux appareils.

Dans le monde graphique (sur n'importe quel type d'OS), nous sommes assez habitués à voir des écrans comme celui-ci qui s'affichent quand l'utilisateur passe un certain temps sans utiliser le dispositif.

Le succès de ces types d'écrans a donné l'idée à certains chercheurs de la compagnie de se pencher sur le monde Linux également et en particulier sur le mode console. Des milliers d'utilisateurs de la production, dite habituellement « *la prod informatique* », travaillent jour et nuit (surtout la nuit !) sur des systèmes en ligne de commande avec comme seul outil la console ou le terminal.

MATRIX a lancé un appel d'offre mondial pour aider ses chercheurs à la réalisation d'un nouveau type d'écran de veille sur Linux en mode console.

Les Unités d'Enseignements « Programmation Procédurale » et « Systèmes Linux » réalisées à l'EXIA.CESI depuis le mois d'octobre, vous permettent de répondre à cet appel d'offre. Plusieurs équipes ont été sélectionnées dans toute la France pour y répondre et présenter le 16/12/2015 un prototype opérationnel des nouveaux écrans de veille.

CESI exia
ÉCOLE D'INGÉNIEURS INFORMATIQUE

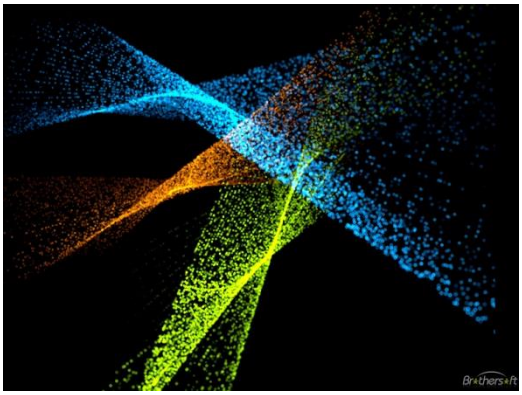
2. Historique du ScreenSaver

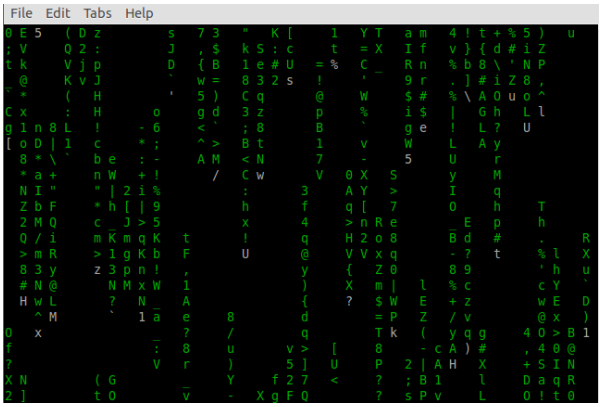
Un **écran de veille** est un programme informatique dont le but originel était de préserver la qualité d'image des écrans d'ordinateurs de type cathodique, qui conservaient "l'empreinte" des fenêtres de logiciels, en stoppant l'affichage de l'écran (écran noir), ou en changeant les couleurs affichées sur chaque pixel (motifs et animations).

Le mot anglais pour les désigner est **screensaver**, qui littéralement signifie « économiseur d'écran ». Néanmoins, cette désignation est trompeuse : les écrans de veille ne sont pas forcément aptes à économiser de l'énergie.

Le premier écran de veille a été écrit pour l'IBM PC par John Socha, plus connu pour avoir créé le gestionnaire de fichiers Norton Commander. Il est aussi à l'origine du terme screensaver. Ce premier écran de veille (nommé **scrnsave** puisque les noms de fichiers à l'époque ne pouvaient dépasser 8 lettres !) a été publié en décembre 1983 dans le magazine Softalk. Il se contentait alors à l'époque de rendre tout simplement l'écran noir au bout de trois minutes (laps de temps qui ne pouvait être modifié qu'en recompilant le programme !).

À l'origine, les écrans de veille avaient pour but de protéger les écrans en changeant périodiquement les images affichées à l'écran lors de périodes de non utilisation de celui-ci. Plus tard, le but d'origine a été transformé pour afficher du contenu artistique ou du divertissement.





Nous retrouvons des écrans de veille de toute sorte et sur tous les OS. Certains sont statiques (simple image), d'autres dynamiques (images en mouvement, déplacement d'objets, …).

Pour plus d'information : https://fr.wikipedia.org/wiki/%C3%89cran_de_veille

3. L'appel d'offre – eXiaSaver

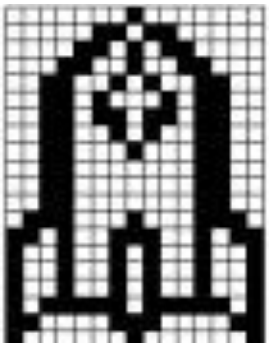
3.1.INTRODUCTION

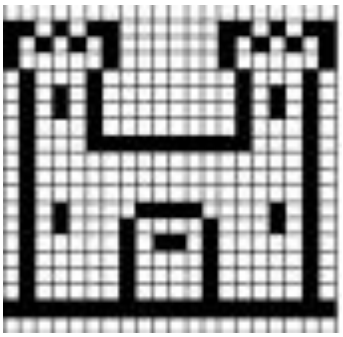
eXiaSaver devra être un exécutable qui lancera « *un jolie termSaver* ». Il sera lancé depuis la ligne de commande d'un terminal GNU/Linux.

3.2. SPECIFICATIONS FONCTIONNELLES

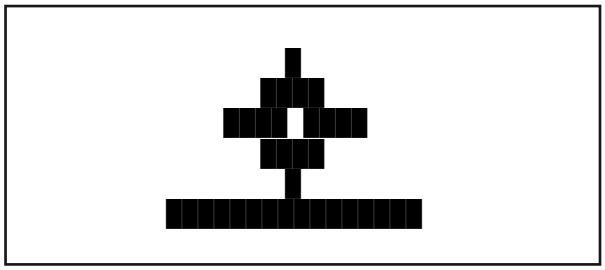
Le termSaver peut être de 3 types différents :

- **Type statique** : il s'agit d'afficher à l'écran le contenu d'un fichier (avec un format prédéfini) qui représente un dessin.





L'affichage devra toujours **être centré sur la taille de la console** (dans la plupart des cas, la taille de la console est de 24 lignes de 80 caractères par ligne). Ces dessins sont représentés sur une grille millimétrée mais celle-ci n'est pas demandée de la représenter. On devrait obtenir un dessin comme par exemple celui-ci dans la console :

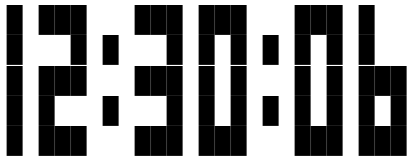


Le format du fichier et l'affichage de caractères sont expliqué dans le point suivant – **spécifications techniques**. Il est demandé de présenter un jeu de tests important.

L'écran de veille se débloque et rend la main au shell quand l'utilisateur clique sur n'importe quelle touche du clavier. Sinon, il reste affiché de façon permanente.

- **Type dynamique** : il s'agit d'afficher une information et que, de façon automatique, cette information se mette à jour sans l'intervention de l'utilisateur.

  L'information que vous devez afficher est l'heure courante, **centrée dans la console**, dans le format classique «**HH:MM:SS**» avec HH de 0 à 23 et MM et SS de 0 à 59.



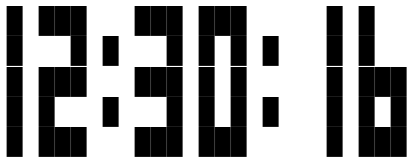
  En bas de la console vous devez faire apparaître un message disant :

  Cet écran sera actualisé dans quelques secondes

  Et toutes les secondes afficher un point (« **.** ») de plus sur la même ligne, SANS EFFACER L'HEURE, ni rajouter une ligne supplémentaire :

  Cet écran sera actualisé dans quelques secondes ……..

  Au bout de N secondes (N doit être paramétrable), réafficher la nouvelle heure courante (par exemple après 10 secondes) :



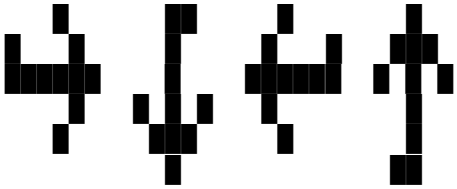
Cet écran sera actualisé dans quelques secondes ..

  L'écran de veille se débloque et rend la main au shell quand l'utilisateur tue le processus avec **Ctrl^C**.

  Le format du fichier pour les chiffres, la taille de chiffres et l'affichage de caractères seront expliqués, de manière plus détaillée, dans le point suivant – **spécifications techniques**.

- **Type Interactif** : après un affichage initial, l'écran de veille demande une action à l'utilisateur. En fonction de cette action (clavier), l'affichage doit être modifié. C'est le même principe qu'un jeu interactif mais dans un termSaver.

  L'idée est de représenter un avion sur la console et le faire voler. L'avion sera représenté en 4 positions de la façon suivante (tailles 5x6 ou 6x5). Pour simplifier, il est possible de prendre une seule dimension de 6x6. Il est demandé de stocker ces 4 positions, chacune dans un fichier PBM et de les charger au démarrage de l'écran de veille :



  L'avion avance selon les commandes de l'utilisateur. Pour piloter l'avion, vous devez taper une des lettres H(aut), B(as), D(roit) ou G(auche) puis taper Entrée. Si l'utilisateur ne rentre aucun caractère ou s'il se trompe de caractère lors de la saisie, l'avion avance avec la dernière commande donnée. Il est possible de piloter l'avion également avec n'importe quelles autres touches (flèches, chiffres 1234, …)

  L'avion vole dans un « espace aérien » de la taille de la console (80x23 car on gardera la dernière ligne pour la saisie des commandes). Quand il arrive aux extrêmes de l'espace aérien, il doit ressortir du côté opposé. L'avion devra apparaître petit à petit, et non d'un seul bloc. Ainsi lorsque l'avion passe sur le bord horizontal ou vertical de l'écran, le nez de l'avion doit être visible de l'autre côté, tandis que l'arrière est encore visible

  Quand l'avion tourne et change de direction ou de sens, il pivote sur son point central (le point (3,3) peut tout à fait convenir). La position initiale sera passée en paramètre de l'écran. La direction et le sens du vol doivent être choisis par le programme de façon totalement aléatoire à chaque démarrage de l'écran. Plus de précisions seront données dans les **spécifications techniques**.

  Prévoir une sortie de l'écran de veille à tout moment en tapant un caractère bien prédéterminé (par exemple, 'q' , 'x', …)

Le choix d'un des écrans de veille doit se faire de façon automatique et sans l'interaction de l'utilisateur. Pour cela, il est demandé de créer un autre exécutable qui fera la tâche « lanceur » : eXiaSaver. Voir spécifications techniques pour plus de détails.

eXiaSaver doit proposer également une fonctionnalité de historique qui permettra de faire de statistiques sur le nombre de types d'écrans lancés, sur les dates de lancement triées par ordre chronologique ou l'inverse. Cette fonctionnalité sera accessible en lançant le même exécutable avec le paramètre « ***-stats*** ». Voir **spécifications techniques** pour plus de détails sur cette fonctionnalité.

3.3.SPECIFICATIONS TECHNIQUES

1. Structures de données

Vous devez réfléchir aux structures de données (struct, tableaux, listes, listes chaînées, files, pile, …) qui seront nécessaires pour les différentes fonctionnalités de votre application.

Le choix de la structure de données est le plus important et doit être fait au début du projet de façon très rigoureuse. De cela dépendront tous les algorithmes que vous allez implémenter par la suite.

Vous trouverez une grille en annexe qui vous aidera à formaliser les structures des données ainsi que les prototypes des fonctions. **Cela vous sera demandé par vos tuteurs 2 jours après le démarrage du projet.** Ce travail vous servira également comme base pour la présentation et la documentation technique à la fin du projet.

Voici quelques conseils :

- Pensez à des tableaux à deux dimensions (statiques ou dynamiques)
- Pensez à regrouper des informations sur un même « objet » à travers de « struct ».
- L'utilisation des pointeurs peut sans doute simplifier l'implémentation de certains algorithmes et faciliter la manipulation de ces structures de données complexes.
- Choix entre les piles et les files en fonction des algorithmes à coder.

2. Format fichier PBM

Pour chaque image utilisée dans les écrans de veille (image statique, tous les chiffres de 0 à 9, les avions), vous devez construire un fichier PBM ASCII avec la syntaxe suivante **à respecter scrupuleusement** :

- Le nombre magique du format : il dépend du format et de la variante (binaire ou ASCII).
- Un caractère d'espacement (espace, tabulation, nouvelle ligne)
- La largeur de l'image (codée en caractères ASCII)
- Un caractère d'espacement
- La hauteur de l'image (codée en caractères ASCII)
- Un caractère d'espacement
- Les données binaires de l'image :
  - L'image est codée ligne par ligne en partant du haut
  - Chaque ligne est codée de gauche à droite
- Toutes les lignes commençant par # sont ignorées.

Le format de fichier PBM est utilisé pour des images noir et blanc. Un pixel noir est codé par un caractère 1, un pixel blanc est codé par un caractère 0.

Un fichier pbm ASCII a pour nombre magique P1. Dans les données ascii de l'image, les caractères d'espacement à l'intérieur sont ignorés. Aucune ligne ne doit dépasser 70 caractères.

Exemple

```
P1
# Un exemple bitmap de la lettre "J"
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

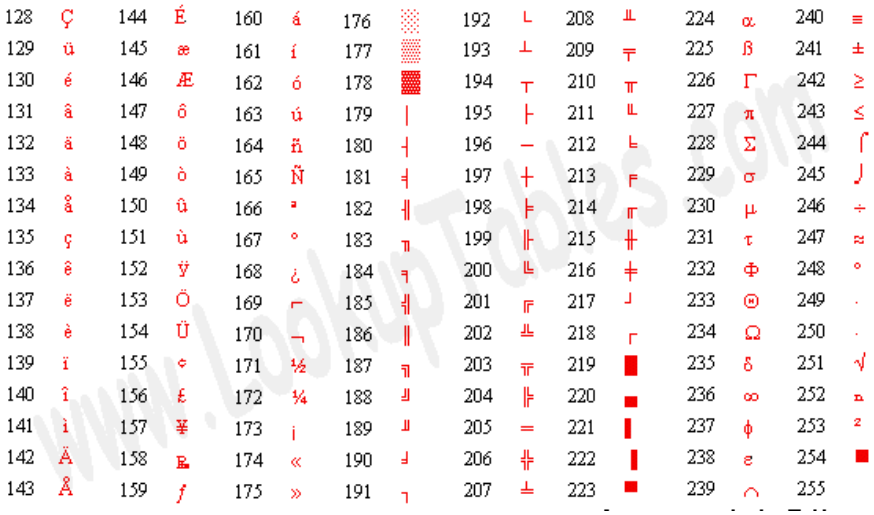
Pour plus d'information : http://fr.wikipedia.org/wiki/Portable_pixmap

Vous trouverez plusieurs fichiers image PBM dans les ressources. Vous pouvez en créer d'autres. Commencez par exemple pour créer le fichier pour la fusée et le château de la page 5, de la même taille ou à l'échelle. N'hésitez pas à montrer votre créativité et présenter au jury un jeu de tests conséquent !

Le module de lecture d'un fichier PBM doit être partagé par les 3 exécutables.

3. Page de code et affichages

Pour les affichages, vous devez utiliser certains caractères ASCII de cette table (**Attention : cette table peut être différente en fonction de l'OS et de la langue de l'OS**). Si vous ne trouvez pas les mêmes caractères, vous pouvez les remplacer par d'autres; par exemple, si vous ne trouvez pas le caractère 219, remplacez-le par un X. Le caractère utilisé n'influera en rien dans la note finale du projet.



Vous pouvez tester quel code ASCII est utilisé dans votre ordinateur en exécutant ce petit programme :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int  i = 0;

   for (i = 128; i < 256; i++)
      printf("%d = %c\n", i , (char)i);

   return 0;
}
```

Le format de fichier PBM est utilisé pour des images noir et blanc. Un pixel noir est codé par un caractère 1, un pixel blanc est codé par un caractère 0. Néanmoins, sur la plupart des consoles Linux, les couleurs sont inversées avec le fond noir et les caractères blancs. On considère que le caractère à afficher du fichier PBM est celui codé par 1 et rien à afficher avec la valeur 0.

Pour l'affichage de l'heure pour le niveau dynamique, la taille de chiffres est par défaut de 5 lignes x 3 colonnes pour tous les chiffres comme dans l'exemple de la page 6. Le séparateur entre les heures, les minutes et les secondes est « : » tenant en compte qu'il doit s'adapter à la taille de l'affichage. N'oubliez pas que **l'affichage de l'horloge doit être centré sur la console en largeur et en hauteur.**

La taille de l'affichage sera paramétrable grâce à une variable d'environnement (voir section juste après). Les tailles possibles sont les suivantes : 5x3, 7x4, 9x5, 11x6, etc … **jusqu'à un maximum raisonnable pour rentrer tous les chiffres dans une console avec une taille de 80x23**.

Vous ne devez stocker que la modélisation des chiffres pour 5x3 : c'est une contrainte technique imposée. Toutes les autres tailles ne doivent être affichées que par des calculs à partir de la modélisation de base. Le stockage d'autres tailles en mémoire vous pénalisera.

ASTUCES :

- la taille de base d'un chiffre, peut être modélisée par un tableau à deux dimensions avec des 1 et des 0. L'ensemble des chiffres peut être représenté par un tableau des pointeurs vers un tableau à 2 dimensions.
- Si vous avez défini votre structure de données, vous pouvez créer une fonction de remplissage « statique » en attendant que le module de lecture des fichiers PBM soit opérationnel. Cela vous permettra de réaliser des tests d'affichage plus rapidement.
- L'affichage ligne par ligne peut être une bonne idée. Implémentez des algorithmes efficaces, sans créer des structures « lourdes ».

4. Paramétrage et Variables d'environnement

Le paramétrage de toutes les applications se fera à travers des variables d'environnement Linux qui peuvent être saisies directement dans le terminal ou être insérées dans le fichier « *.profile* » de la session utilisateur.

Sur Debian : https://wiki.debian.org/EnvironmentVariables
Sur Ubuntu : https://doc.ubuntu-fr.org/variables_d_environnement

Vous devez définir les variables suivantes (dans le console ou dans le .profile):

- **EXIASAVER_HOME** : répertoire où se trouvent les 3 exécutables. Si cette valeur n'est pas définie, le répertoire par défaut est le répertoire courant.
- **EXIASAVER1_PBM** : répertoire où se trouvent tous les fichiers PBM pour le niveau statique. Si cette valeur n'est pas définie, le répertoire par défaut est le répertoire courant. Elle sera utilisée par le lanceur pour parcourir tous les fichiers du répertoire et en prendre un de façon aléatoire.
- **EXIASAVER2_PBM** : répertoire où se trouvent tous les fichiers PBM pour le niveau dynamique. Si cette valeur n'est pas définie, le répertoire par défaut est le répertoire courant.
- **EXIASAVER2_TAILLE** : taille d'affichage de chiffres dans l'horloge numérique. Si cette variable n'est pas définie, la valeur par défaut est 5x3
- **EXIASAVER2_SLEEP** : nombre (positif) de secondes entre 2 rafraichissements de l'horloge. Si cette variable n'est pas définie, la valeur par défaut est 10 secondes.
- **EXIASAVER3_PBM** : répertoire où se trouvent tous les fichiers PBM pour le niveau interactif. Si cette valeur n'est pas définie, le répertoire par défaut est le répertoire courant.

Le lanceur eXiaSaver

- « videra » la console avec l'instruction **system(''clear'') ;**
- fera le choix de façon aléatoire entre un des 3 types d'écran
- lira les variables d'environnement correspondantes au type choisi
- choisira un fichier image dans le répertoire des images de façon aléatoire si type statique. Prévoyez un jeu de tests conséquent pour mieux voir le coté aléatoire. Utilisez les primitives de parcours d'un répertoire pour connaître son contenu et en choisir un de façon aléatoire.
- lancera l'exécutable correspondant au niveau choisi avec « **exec ou une de ses variantes** » en passant les paramètres nécessaires (nom image pour le type écran de veille statique, position initiale de l'avion pour le type interactif).

Voici un exemple de lancement de chaque type :

exiasaver1 ex1.pbm	// nom du fichier choisi passé en paramètre
exiasaver2	// sans paramètres
exiasaver3 10x20	// position initiale, ligne 10, colonne 20 où mettre le centre de gravité de l'avion. La position et le sens seront choisis aléatoirement par exiasaver3.

Pour la lecture des variables d'environnement en C et le tirage aléatoire, regardez la documentation sur les fonctions suivantes :

char * getenv(char * name) ;
int rand(void) ;
void srand(unsigned int seed);

5. Historique et statistiques

Vous devez gérer un fichier qui enregistrera l'historique des lancements. Vous devez stocker dans ce fichier :

- La date et l'heure du lancement.
- Le type d'écran lancé :
  - niveau statique=1
  - niveau dynamique=2
  - niveau interactif=3
- En fonction du niveau :
  - Nom du fichier image affiché pour le niveau statique
  - Taille de l'horloge pour le niveau dynamique
  - Position initiale de l'avion

Voici un exemple de fichier à construire :

```
#Format de chaque ligne avec séparateur ;
#Date format jj/mm/yyyy HH:MM:ss;
#Niveau : 1=statique, 2=dynamique, 3=interactif;
# Si 1, nom fichier ; Si 2, taille d'affichage ; Si 3, position initiale
avion sur la console
10/10/2014 15:15:10;1;ex1.pbm
10/10/2014 05:25:45;2;5x3
11/12/2014 13:07:23;3;10x10
11/12/2014 15:15:12;2;7x4
15/01/2015 15:15:04;1;ex2.pbm
15/01/2015 15:15:30;3;20x20
```

L'enregistrement se fait par le lanceur eXiaSaver juste avant de lancer le *termSaver* avec la commande **exec**. Les statistiques se feront à partir des informations contenues dans ce fichier. Présentez un menu principal avec 3 ou 4 possibles affichages selon le critère choisi (date, type, nom du fichier, tailles utilisés, …)

NOTA : ce format est donné à titre d'exemple. Vous pouvez vous en inspirer ou créer le vôtre si celui-ci ne vous convient pas.

6. Utilisation de primitives et autres fonctions en C

Voici quelques contraintes techniques :

- Le lanceur doit exécuter un des 3 écrans de veille à travers la commande « **exec ou une de ses variantes** »
- La **lecture des fichiers doit se faire** dans un **processus fils** (**fork**, **getpid**, …)
- Le processus père doit attendre la fin du processus fils pour continuer (**wait**)
- La manipulation des fichiers devra se faire avec : **open, close, fopen, fclose, fgetc, fgets, fprintf …**
- Le parcours d'un répertoire (pour le choix d'une image, choix aléatoire) se fera avec : **opendir, readdir, closedir**
- La manipulation des chaînes de caractères se fera avec **: strcpy, strlen, strcmp**, **sscanf,**…
- L'allocation de mémoire dynamique avec **malloc et free (**tableau des pointeurs par exemple**).** Intéressant de l'utiliser sachant que les tailles des images PBM sont différentes.

Voici une liste d'autres fonctions et primitives qui vous seront, sans doute, très utiles :

PRIMITIVES/APPELS SYSTEME	FONCTIONS C
• access • stat • chdir • time • kill	• memset • perror • sleep • « tests sur des caractères » : isalpha, isupper, … • « conversion de types » : atoi, atof, atol • system – appelle le shell et le passe la commande, par exemple `system("clear")` pour effacer la console

Vous pouvez trouver toutes les primitives et fonctions dans les manuels de Linux. Voici un lien qui peut vous être utile (*Ressource également disponible sur Moodle)*

4. Fonctionnalités optionnelles

4.1. Format des images en couleur

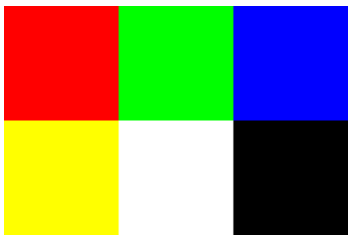
Au lieu d'utiliser des fichiers en noir et blanc, vous pouvez faire le choix de travailler avec des fichiers en couleur. Le mécanisme est le même qu'avec les fichiers PBM sauf que le format utilisé sera le PPM. Pensez si la structure de base définie pour les fichiers PBM servira pour ce type de fichiers.

Le format de fichier PPM est utilisé pour des images couleur. Chaque pixel est codé par trois valeurs (rouge, vert et bleu). Comme le format PGM, en plus des caractéristiques de largeur et hauteur, une valeur maximale utilisée pour coder les niveaux de couleur ; cette valeur doit être inférieure à 65536.

Un fichier ppm ASCII a pour nombre magique P3. Chaque pixel est codé en caractères ASCII, précédés et suivis par un caractère d'espacement. Aucune ligne ne doit dépasser 70 caractères.

Exemple

```
P3
# Le P3 signifie que les couleurs sont en ASCII,
# par 3 colonnes et 2 lignes,
3 2
# ayant 255 pour valeur maximum, et qu'elles sont en RGB.
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```



http://fr.wikipedia.org/wiki/Portable_pixmap

Ce format réduit considérablement la taille d'une image (surtout la largeur).Si cela vous paraît très réducteur, vous pouvez imaginer un autre type de format alternatif au PPM en remplaçant les codes RGB par des séquences de couleur par pixel (**voir ci-après**).

```
P4
# Le P4 est le format invité par nous
3 2
2230 2231    0
0130 0132 0133
```

4.2.Gestion de la couleur sous linux

Cet article sur les affichages en couleur sur Linux vous sera utile.

http://www.linuxforums.org/forum/programming-scripting/88-color-console.html

"Text color output is not defined in ANSI C/C++. Instead the creators of the language left that to be operating system dependent. In Linux, to change text color you must issue what are known as terminal commands. To do this you just change your output statement to contain a terminal command."

```
#include <stdio.h>

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"

int main (int argc, char const *argv[]) {

  printf(ANSI_COLOR_RED     "This text is RED!"     ANSI_COLOR_RESET "\n");
  printf(ANSI_COLOR_GREEN   "This text is GREEN!"   ANSI_COLOR_RESET "\n");
  printf(ANSI_COLOR_YELLOW  "This text is YELLOW!"  ANSI_COLOR_RESET "\n");
  printf(ANSI_COLOR_BLUE    "This text is BLUE!"    ANSI_COLOR_RESET "\n");
  printf(ANSI_COLOR_MAGENTA "This text is MAGENTA!" ANSI_COLOR_RESET "\n");
  printf(ANSI_COLOR_CYAN    "This text is CYAN!"    ANSI_COLOR_RESET "\n");

  return 0;
}
```

All the colors that I have found are:

\033[22;30m - black
\033[22;31m - red
\033[22;32m - green
\033[22;33m - brown
\033[22;34m - blue
\033[22;35m - magenta
\033[22;36m - cyan
\033[22;37m - gray
\033[01;30m - dark gray
\033[01;31m - light red
\033[01;32m - light green
\033[01;33m - yellow
\033[01;34m - light blue
\033[01;35m - light magenta
\033[01;36m - light cyan
\033[01;37m - white

Toute la gamme de possibilités RGB n'existant pas sur cette liste, vous pouvez faire une fonction « decoderRGB » qui trouvera la couleur la plus proche à celle qui est codée en RGB.

4.3. Gestion du multiprocessus

Pour « paralléliser » certaines tâches, vous pouvez mettre en pratique le multiprocessus :

- Lecture de plusieurs fichiers contenant les chiffres du type dynamique sur deux, trois ou N processus fils. C'est-à-dire, que chaque processus fils charge un chiffre.

- Coder 2 processus pour le type interactif : un processus qui gère l'affichage et un processus qui gère la gestion des touches clavier. Dès que l'utilisateur appuie sur une touche, le processus envoi cette information à l'autre processus pour rafraichir l'écran. Cela rendra l'affichage de l'avion plus fluide (pas besoin de taper sur une touche + Entrée)

- Calculs de la grille d'affichage sur plusieurs processus fils. Mais attention, cela est possible uniquement pour les calculs mais pas pour l'affichage. Vous devrez attendre la fin de tous les processus fils pour réaliser l'affichage.

Attention, lors du lancement de plusieurs processus fils! Le lancement du **fork** dans une boucle **for** ne fera pas forcément le nombre des fils que vous avez pu imaginer. Un compteur et une boucle **do … while** est certainement plus adapté pour lancer le nombre de processus fils souhaité.

5. Informations diverses

Tous les développements doivent être réalisés sur un ***environnement Linux***. Lors de la soutenance vous devrez montrer tous les développements et les démonstrations sur Linux.

ATTENTION : aucun projet ne sera recevable s'il est développé sur un autre environnement que Linux. La note de groupe sera automatiquement « **D** » pour tous les membres de l'équipe.

Vous êtes libres d'utiliser ou pas un IDE sur Linux. Vous pouvez utiliser la VM du module ou vous pouvez utiliser celle qui vous convient le mieux (à condition de ne pas perdre 3 jours à installer une version qui fonctionne !!).

Vous êtes également invités à installer les « Addons Invité » de VirtualBox qui vous permettront d'utiliser les répertoires partagés entre la VM et la machine hôte. Tuto disponible en ressources sur Moodle. (*Merci à Baptiste Saclier de A2 LYON qui accepte de le partager pour ce projet*).

Également, vous devez dans votre équipe gérer les sources et partager vos réalisations dans un dépôt Git (connecter la VM au réseau + GitHub) - https://www.atlassian.com/git/

6. Modalités d'évaluation

La note finale sera composée de :

- 40% note de groupe sur la réalisation spécifications techniques & fonctionnelles obligatoires.
- 40% note sur la question individuelle posée par les jurys lors de la soutenance
- 10% note d'évaluation par les pairs
- 10% note donné par le jury en fonction de la prestation orale lors de la soutenance

Voir grille avec les critères d'évaluation sur Moodle.

IMPORTANT : le projet est composé d'une partie OBLIGATOIRE. Ce qui est demandé doit d'être totalement réalisé et présenté le jour de la soutenance. Si cette partie n'est pas réalisée, la note sera C pour chaque personne du groupe (voire D si les jurys constatent un manque important de travail).

Pour avoir B dans la note groupe du projet, il faut réaliser le type dynamique ou interactif. Pour prétendre avoir A, il faudra faire les 2 types.

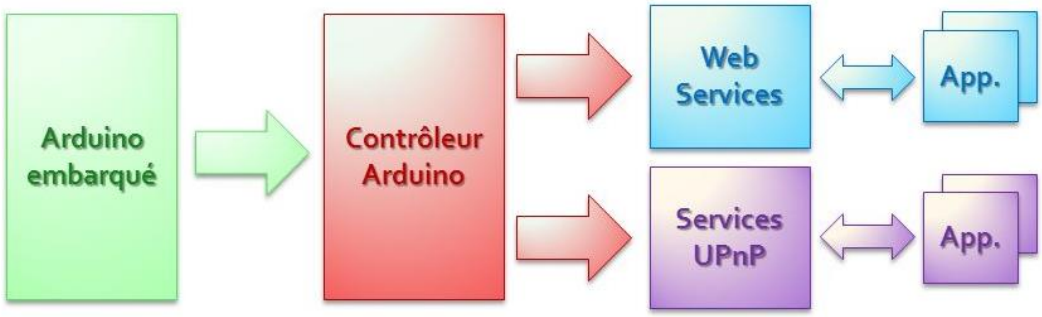
Les fonctionnalités optionnelles peuvent vous aider à compenser certains manques dans vos réalisations. **MAIS** attention à ne pas vous disperser pour autant dans trop de choses à la fois. Elles seront prises en compte seulement si la partie obligatoire est réalisée.

7. Livrables techniques attendus

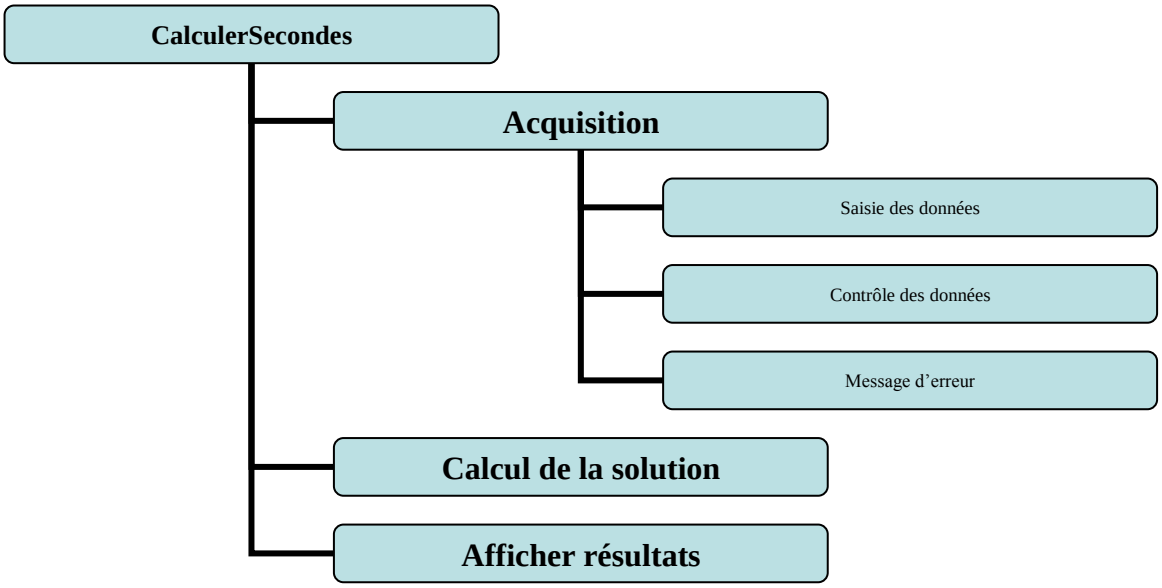
Les livrables à mettre à disposition des tuteurs dans votre dépôt Git sont :

- Le code réalisé (dans les fichiers .c et .h). Dans chaque exécutable, vous devez utiliser au moins 3 fichiers sources différents (et les fichiers .h nécessaires). Pour la bibliothèque de lecture des fichiers PBM, vous devez partager le même code source pour tous les exécutables.
- Une documentation technique sur l'architecture de vos exécutables : structures utilisées, modularisation des fichiers, organisation de fonctions. Un schéma vaut mieux qu'un grand commentaire de 30 lignes. Voir exemples ci-dessous.
- Un manuel d'utilisation sur 2 ou 3 pages maximum.

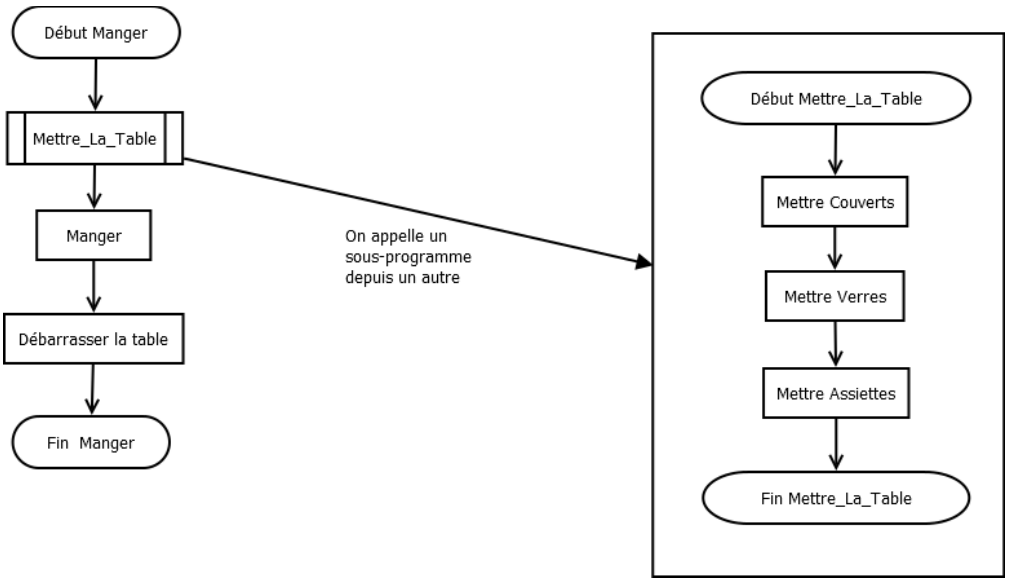
Exemple pour l'architecture logiciel (modularisation) :



Exemple pour la pile d'appels de fonctions :



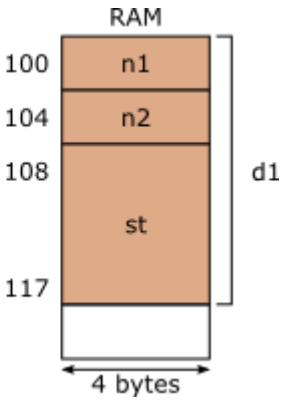
Exemple d'algorigramme:



Exemples de représentation de structures en C :

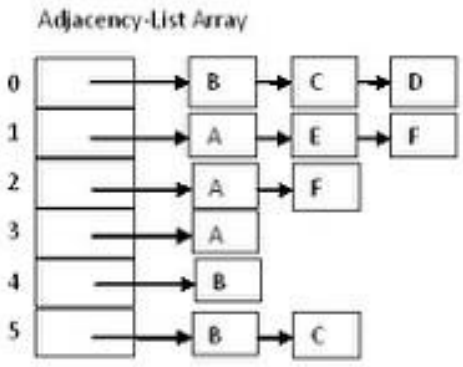
```
struct Etudiant {
	char nom[50];
	char prenom[50];
	char email[255];
	char sexe;
	int groupe;
	Adresse adresse;
	char telephone[100];
}

struct Adresse {
	char adresse1[255];
	char adresse2[255];
	char adresse3[255];
	char codePostal[255];
	char ville[255];
}
```

```
struct contact
{
    int n1;
    float n2;
    char st[10];
} d1;
```



Etudiant
char nom[50] char prenom[50] char email[255] char sexe int groupe Adresse adresse char telephone[100]

Adresse
char adresse1[255] char adresse2[255] char adresse3[255] char codePostal[255] char ville[255]



Un autre exemple de structure dynamique

Représentation de la répartition en bibliothèques :

etudiant.h
typedef struct Etudiant Etudiant void afficherEtudiant(Etudiant etudiant) void saisirEtudiant(Etudiant *etudiant)

adresse.h
typedef struct Adresse Adresse void afficherEtudiant(Adresse adresse) void saisirEtudiant(Adresse *adresse)

8. Quelques conseils

Concentrez-vous sur la programmation en C, les structures de données, le découpage du code et les affichages de chacune des demandes et tout se passera très bien. **Concentrez-vous sur toutes les fonctionnalités obligatoires demandées** pour ne pas en oublier une importante.

Même si les spécifications de ce projet vous permettent d'assigner des fonctionnalités distinctes à chaque personne du groupe, on vous recommande vivement d'assurer collectivement la partie de base obligatoire décrite au début de ce projet. Une fois cette fonctionnalité assurée, vous pouvez vous dispatcher les tâches restantes.

Mettez-vous d'accord sur les structures à utiliser au début du projet car plusieurs personnes du groupe pourraient être amenées à les utiliser.

> **ASTUCE :** par exemple, n'attendez pas d'avoir le module d'écriture d'un fichier au point pour faire le module de lecture. Il suffit de se mettre d'accord sur le format et le contenu du fichier avant de commencer. Par exemple, vous pouvez construire à la main un fichier historique avant même que le module d'écriture du fichier soit opérationnel.

Vous devez montrer, lors de la soutenance :

- Que les programmes répondent aux spécifications fonctionnelles et techniques.
- vos connaissances personnelles sur les structures de données et les algorithmes de vos programmes.

Si vous implémentez des algorithmes de tri pour l'affichage des stats, vous pouvez, dans un premier temps utiliser des algos simples (bulle, insertion, sélection) et dans un deuxième temps, s'il vous reste du temps, implémenter un algorithme plus efficace (merge, quicksort). De même pour la recherche (faite d'abord une séquentielle puis une dichotomique). Mais, attention, pour que la recherche dichotomique soit efficace, la liste doit être préalablement triée.

Vous pouvez utiliser n'importe quelle structure de données présentée dans le module ou n'importe quelle autre que vous connaissez si vous êtes à l'aise avec la programmation en C. Dans tous les cas, réfléchissez bien à la plus adaptée à chaque problématique et sachez argumenter le pourquoi de votre choix.

ET SURTOUT….. AMUSEZ-VOUS BIEN !!!

LA PROGRAMMATION DOIT ETRE UN PLAISIR !!!

9. Organisation du projet

Le projet se déroule en groupes de **3 personnes** (maximum 4, en fonction de la taille des promos) du 7 au 16 décembre 2016

- Le vendredi 9 décembre, en fin de journée, chaque groupe doit rendre le document décrivant l'analyse du problème et le choix des fonctions et variables (cf. document "Feuille d'avancement"). Pour cette phase, vous serez assistés par un **étudiant de A5** qui sera votre référent pour le projet.

- Les livrables attendus sont à rendre le jeudi 15 décembre à 12h00 au plus tard en version électronique sur Git, par mail ou en version papier (voir avec votre tuteur responsable du projet).

- Le code source doit être commenté.

- L'après-midi du 15 décembre sera consacré à la préparation de la soutenance avec la réalisation d'un support (ppt, prezi, autre …)

- Dans la journée du 16 décembre, chaque groupe devra présenter et défendre son travail au cours d'une soutenance orale avec démonstration complète de l'application de 10 minutes puis séance de questions individuelles d'une même durée. Chaque membre du groupe devra prendre la parole de manière équitable. Le chef de projet présentera l'organisation de l'équipe, l'organisation et la distribution des tâches pendant la semaine.

Ecran de veille MATRIX

Pour toute question sur le projet, n'hésitez pas de demander à votre tuteur.