UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

DEPARTEMENT D'ENSEIGNEMENT
EEA
ELECTRONIQUE - ELECTROTECHNIQUE - AUTOMATIQUE

FACULTÉ DES SCIENCES
MONTPELLIER

Travaux Pratiques :

« Génie Informatique »

HLEE303 - Licence EEA - Semestre 3

Mikhaël MYARA
mikhael.myara@umontpellier.fr

23 octobre 2019

Table des matières

Travaux Pratiques 1

Pointeurs, Fonctions et Allocation Mémoire

Sommaire

1.1 Allocation Statique ou Dynamique ?

Q-1.1.1 : En réservant la mémoire par allocation statique, écrivez un programme qui remplit un tableau de 5.000 `float` avec le carré de l'indice (= du numéro de la case), et qui affiche ensuite le contenu des cases dont l'indice est un multiple de 100.

Q-1.1.2 : Même exercice en allouant la mémoire du tableau dynamiquement.

Q-1.1.3 : Reprenez le premier programme, et multipliez par 10 le nombre de cases allouées autant de fois que nécessaire pour le faire planter. Puis, reportez le même nombre de cases dans le second programme. Concluez sur l'intérêt de l'allocation dynamique.

1.2 Carnet de Notes

Q-1.2.1 : Ecrivez un programme qui :

1. Demande le nombre de notes à gérer, (ce nombre est éventuellement grand)
2. Crée un tableau permettant de contenir les notes (elles sont potentiellement décimales),
3. Permet à l'utilisateur d'entrer les notes dans le tableau,
4. Calcule la moyenne des notes et affiche ce résultat.

Q-1.2.2 : Justifiez le type d'allocation mémoire que vous utilisez.

1.3 Jours, Heures, Minutes et Secondes

Q-1.3.1 : Ecrivez un programme qui, à partir d'un nombre en secondes saisi au clavier, affiche sa décomposition en jours, heures, minutes et secondes.

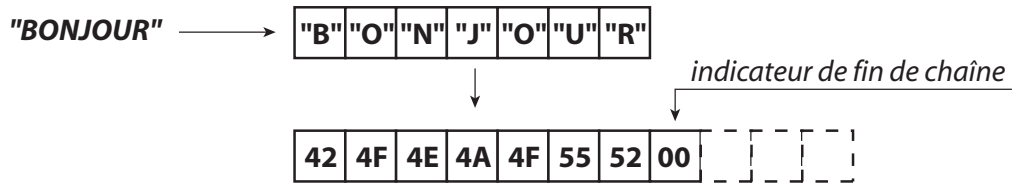
Q-1.3.2 : Ecrivez maintenant une fonction appelée `jhms` qui prendra en paramètre le nombre de secondes et fournira en sortie le nombre de jours, d'heures de minutes et de secondes.

Q-1.3.3 : Ecrivez une fonction `main` qui :

1. propose la saisie au clavier du nombre initial exprimé en secondes,
2. fait appel à la fonction `jhms`,
3. affiche les résultats fournis par la fonction `jhms`.

1.4 Chaînes de Caractères en Langage C

Une Chaîne de Caractères en Langage C n'est ni plus ni moins qu'un tableau de `char` dont le dernier caractère a pour codage la valeur `0`, comme celà est représenté ci-dessous :



Q-1.4.1 : Ecrivez **un programme** qui affiche la taille d'une chaîne de caractères saisie au clavier en utilisant la notation tableau. On supposera que la chaine fait au maximum `200` caractères.

Note : vous **n'utiliserez pas** la fonction `strlen`, vous calculerez la taille de la chaine avec votre propre algorithme.

Q-1.4.2 : Ecrivez **la fonction** `TailleChaine` qui renvoie la taille d'une chaîne de caractères fournie en paramètre en utilisant la notation tableau. Ecrivez également la fonction `main` qui permet d'appeler cette fonction et d'afficher son résultat.

Q-1.4.3 : Même question en utilisant la notation pointeur.

Q-1.4.4 : Ecrivez une fonction qui renverse une chaine de caractères passée en paramètre dans un nouveau tableau qu'elle devra renvoyer. Quel type d'allocation mémoire allez-vous choisir ?

Q-1.4.5 : Ecrivez une fonction qui renverse une chaine de caractères passée en paramètre directement dans le tableau d'origine en utilisant la notation tableau. Expliquez pourquoi cela est possible en termes d'entrées/sortis de la fonction.

Q-1.4.6 : Ecrivez enfin une fonction qui réalise la concaténation de deux chaines de caractères.

1.5 Conversion de Pointeur

Q-1.5.1 : On donne programme suivant :

```
#include <stdio.h>

int main(void)
{
unsigned int i;
int T[20];
int* pL;
unsigned short* pS;
unsigned char* pC;

pL=T;
pS=(unsigned short*)T;
pC=(unsigned char*)T;



for(i=0;i<5;i++)
{
T[i]=253+i;
}

printf("\nResultats :\n");
for(i=0;i<20;i++)
{
printf("\n%20d%20d%20d",pL[i],(int)pS[i],(int)pC[i]);
}

return 0;
}
```

Ce code est disponible ici :

`conversionPointeur.c`
(attaché au PDF)

ou

`conversionPointeur.c`
(via internet)

Note : dans `printf`, la syntaxe %20d sert à réaliser l'affichage d'un entier (exactement comme %d) mais en forçant l'utilisation de 20 caractères, en ajoutant des espaces. C'est ce qui permet ici d'afficher des colonnes régulières.

Expliquez l'utilité de `(unsigned short*)` et `(unsigned char*)` lignes 12 et 13. Expliquez pourquoi on n'a pas de `(int*)` ligne 11.

Q-1.5.2 : Expliquez et commentez son fonctionnement à travers ce qui est affiché à l'écran.

Q-1.5.3 : On remplace la ligne 25 par :

```
1         printf("\n%p\t%p\t%p",pL+i,pS+i,pC+i);
```

Sachant que %p commande à `printf` d'afficher l'adresse à laquelle pointe le pointeur, expliquez et commentez ce nouveau programme en utilisant ce qui est affiché à l'écran.

1.6 Matrices Carrées

Q-1.6.1 : Ecrivez un programme qui permet de :
— Demander à l'utilisateur le dimension des matrices qu'il veut gérer,
— Propose la saisie au clavier des valeurs de deux des matrices,
— Calcule le produit des deux matrices saisies,
— Affiche les trois matrices en présence.
Vous écrirez 4 fonctions pour vous aider :
— `float* CreerMatrice(int dimension);`
— `void SaisieMatrice(float* M,int dimension);`
— `void ProduitMatrice(float* A,float* B, float* C,int dimension);`
— `void AfficheMatrice(float* M,int dimension);`
et enfin la fonction `main`

Q-1.6.2 : Même programme avec les prototypes :
— `float** CreerMatrice(int dimension);`
— `void SaisieMatrice(float** M,int dimension);`
— `void ProduitMatrice(float** A,float** B, float** C,int dimension);`
— `void AfficheMatrice(float** M,int dimension);`

Travaux Pratiques 2

Structures de Données Simples

Sommaire

2.1 Complexes

On veut gérer les nombres complexes en lagage C. Pour celà :

Q-2.1.1 : Ecrivez un type `complexe` pour représenter convenablement les nombres complexes en Langage C.

Q-2.1.2 : Déclarez trois variables de type `complexe` nommées `c1`, `c2` et `c3`.

Q-2.1.3 : Effectuez le calcul $C3 = C2 \times C1$ et affichez le résultat à l'écran.

2.2 Matrices 2×2

On souhaite écrire en Langage C quelques opérations de base pour la manipulation des matrices 2×2, telles que le produit et la somme. Pour celà, on va créer un type `matrice22`. On propose le cheminement suivant :

Q-2.2.1 : Ecrivez le type `matrice22`,

Q-2.2.2 : Ecrivez deux fonctions permettant de saisir les valeurs contenues dans une `matrice22` :
— l'une aura pour prototype :

```
matrice22 saisie1(void);
```

— l'autre aura pour prototype

```
void saisie2(matrice22* resultat);
```

Q-2.2.3 : Vous expliquerez pourquoi le second prototype est possible.

Q-2.2.4 : Vous expliquerez et justifierez en quoi l'une des deux solution est avantageuse sur l'autre en terme de temps machine.

Q-2.2.5 : Ecrivez deux fonctions permettant d'afficher une `matrice22` :

— l'une aura pour prototype

```
void affichage1(matrice22 M);
```

— l'autre aura pour prototype

```
void affichage2(matrice22* M);
```

Q-2.2.6 : Ecrivez maintenant trois fonctions permettant d'effectuer la somme de deux `matrice22` :

— l'une aura pour prototype :

```
matrice22 somme1(matrice22 gauche, matrice22 droite);
```

— une autre aura pour prototype :

```
void somme2(matrice22 gauche, matrice22 droite,matrice22*
   resultat );
```

— enfin, une troisième aura pour prototype :

```
void somme3(matrice22* gauche, matrice22* droite,matrice22*
   resultat );
```

Q-2.2.7 : Expliquez pourquoi la seconde possibilité est plus intéressante que la première.

Q-2.2.8 : Ecrivez deux fonctions permettant d'effectuer le produit de deux `matrice22` :

— l'une aura pour prototype :

```
matrice22 produit1(matrice22 gauche, matrice22 droite);
```

— l'autre aura pour prototype

```
void produit2(matrice22* gauche, matrice22* droite,matrice22
   * resultat );
```

Q-2.2.9 : Expliquez pourquoi la deuxieme solution est avantageuse sur la première mais peut produire un résultat faux en fonction du talent du programmeur.

Q-2.2.10 : Ecrivez une fonction `main` permettant à l'utilisateur :

1. d'effectuer la saisie de trois matrices M_1, M_2, M_3,
2. de calculer le résultat de $M_1 \times (M_2 + M_3)$,
3. d'afficher le résultat.

Q-2.2.11 : Ecrivez une nouvelle fonction `main` permettant à l'utilisateur de :

1. saisir un nombre n de `matrice22`, la valeur de n étant décidée par l'utilisateur du programme,
2. de les ranger dans un tableau,
3. d'en effectuer le produit et d'afficher le résultat.

Vous écrirez deux variantes :

— l'une utilisant les fonctions numérottées "1",

— l'autre utilisant les fonctions numérottées "2"

Note : ce genre de programme est utilisé partout ! en optique, en électronique, pour les jeux video en 3D, etc. On a souvent besoin et pour des raisons très différentes de réaliser un grand nombre de produits de matrices 2×2 ou 3×3.

2.3 Carnet d'Adresses

On souhaite gérer un carnet d'adresses de 10 contacts maximum dans lequel les coordonnées des différents contacts seront stockées. Pour chaque contact, on souhaite stocker :

— le Nom, sur 20 caractères maximum,
— le Prénom, sur 20 caractères maximum,
— l'age, sous forme d'un long.

Pour celà, on propose les étapes suivantes :

Q-2.3.1 : Ecrivez une fonction permettant la saisie d'un contact et une permettant l'affichage d'un contact.

Q-2.3.2 : Ecrivez une fonction permettant d'ajouter un contact à la liste des contacts.

Q-2.3.3 : Ecrivez une fonction permettant de supprimer un contact à la liste des contacts.

Q-2.3.4 : Faites une gestion complète avec un menu permettant d'ajouter/supprimer un contact, et d'afficher la liste complète des contacts.

2.4 Courbes

Q-2.4.1 : Créez une structure, que vous appellerez `point`, qui permette de stocker un point du plan.

Q-2.4.2 : Créez une structure que vous appelerez `courbe` qui permette de définir une courbe formée d'un nombre de points éventuellement grand.

Q-2.4.3 : Créez 2 courbes contenant chacune 5 points avec les valeurs x et y que vous voudrez. Faites en sorte que les x soient différents pour les 2 courbes.

Q-2.4.4 : Créez 2 courbes contenant chacune 5 points avec les valeurs x et y que vous voudrez. Faites en sorte que les x soient différents pour les 2 courbes. De plus, les points devront apparaitre par valeurs de x croissantes.

Q-2.4.5 : Ecrivez une fonction capable de réaliser la somme de ces 2 courbes. Vous aurez à interpoler les valeurs de x qui vous manqueront dans chacune des 2 courbes.

Travaux Pratiques 3

Structures de Données Avancées

Sommaire

3.1 Listes Chaînées

En se basant sur ce qui a été vu en cours, on va écrire le code utile en Langage C pour gérer des listes simplement chaînées.

Q-3.1.1 : Ecrivez le type `maillon` pour une liste chaînée transportant des valeurs de type `float`.

Q-3.1.2 : Ecrivez une fonction d'insertion.

Q-3.1.3 : Ecrivez une fonction de suppression.

Q-3.1.4 : Ecrivez une fonction qui permet à un utilisateur d'ajouter une valeur en fin de liste.

Q-3.1.5 : Ecrivez une fonction qui permet d'afficher toutes les valeurs de la liste.

Q-3.1.6 : Ecrivez un programme qui place toutes les valeurs d'une liste chaînée dans un tableau simple.

3.2 Listes Doublement Chaînées

Réécrivez ou complétez (au choix) l'exercice 3.1 pour une gestion des listes doublement chaînées.

3.3 Traitement de texte

On veut écrire un petit traitement de texte en ligne de commande. Pour cela, on a d'abord besoin d'une fonction permettant de faire de la saisie "en temps réel" ce que ne permet pas scanf. Sous Windows, vous pouvez utiliser les fonctions `getch` et `getche` de `conio.h`. Sous Linux elles n'existent pas par défaut, mais on peut la créer. Pour cela, copiez/collez le code ci-dessous dans un fichier que vous nommerez "getch.h". Par la suite, vous pourrez utiliser la fonction getch dans votre code avec un `#include"getch.h"`.

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

static struct termios old, newn;

/* Initialize new terminal i/o settings */
void initTermios(int echo)
{
 tcgetattr(0, &old); /* grab old terminal i/o settings */
 newn = old; /* make new settings same as old settings */
 newn.c_lflag &= ~ICANON; /* disable buffered i/o */
 newn.c_lflag &= echo ? ECHO : ~ECHO; /* set echo mode */
 tcsetattr(0, TCSANOW, &newn); /* use these new terminal i/o settings now */
}

/* Restore old terminal i/o settings */
void resetTermios(void)
{
 tcsetattr(0, TCSANOW, &old);
}

/* Read 1 character - echo defines echo mode */
char getch_(int echo)
{
 char ch;
 initTermios(echo);
 ch = getchar();
 resetTermios();
 return ch;
}

/* Read 1 character without echo */
char getch(void)
{
 return getch_(0);
}

/* Read 1 character with echo */
char getche(void)
{
 return getch_(1);
}
```

Ce code est disponible ici :

`getch.h`
(attaché au PDF)

ou

`getch.h`
(via internet)

Pour créer notre traitement de texte, on va utiliser une liste chainée. L'idée n'est pas de chainer caractère par caractère mais de chainer paragraphe par paragraphe.

Q-3.3.1 : Expliquez pourquoi il serait impensable de faire un chainage caractere par caractere

Q-3.3.2 : Proposez une structure de donnée permettant de faire du chainage paragraphe par paragraphe

Q-3.3.3 : Ecrivez un programme qui propose la saisie d'un texte et qui le stocke en temps réel dans la liste chainée. La saisie se termine lorsque l'on appuie trois fois d'affilée sur "entrée". A ce moment la, le programme devra réafficher la totalité du texte tapé.

Q-3.3.4 : Complétez le programme pour gérer la touche "backspace"

Travaux Pratiques 4

Gestion des Fichiers

Sommaire

4.1 Sauvegarde en Mode Binaire ou en Mode Texte

On souhaite sauvegarder sous forme d'un fichier les coordonnées des points (x,y) de $\exp(x)$ sur l'intervalle $[0, 1]$ sur 50 points.

Q-4.1.1 : Ecrivez un programme qui crée le tableau en RAM et le remplit avec les bonnes valeurs. On souhaite que les valeurs (x,y) soient intercallées l'une après l'autre dans le fichier. Note : les fonctions mathématiques sont contenues dans `math.h`, et la fonction exponentielle s'appelle `exp` en C.

Q-4.1.2 : Complétez le programme pour sauvegarder les valeurs.

Q-4.1.3 : Editez à l'aide du bloc-notes le contenu du fichier. Qu'observez vous ? Pourquoi ?

Q-4.1.4 : Utilisez la commande `hexump -C nomfichier` du terminal sous linux pour faire l'analyse binaire du fichier. Essayez de reconnaitre une partie des données écrites dans le fichier.

Q-4.1.5 : On souhaite maintenant que le fichier soit directement lisible sous le bloc-notes. Proposez une solution. Note : le fichier obtenu peut être ouvert et exploité sous des logiciels comme Excel.

Q-4.1.6 : Ecrivez le même programme en n'utilisant plus le tableau contenant les valeurs calculées.

4.2 Notes

On souhaite gérer sous forme d'un fichier les notes de chaque étudiant d'une classe. On désire avoir le nom de l'étudiant sur 20 caractères maximum, suivi de 3 notes en format binaire.

Q-4.2.1 : Ecrivez un programme qui permette la saisie puis la sauvegarde des notes d'un étudiant. Utilisez le pour sauvegarder 5 étudiants.

Q-4.2.2 : Ecrivez un programme qui permette de lire dans le fichier puis d'afficher les notes du premier étudiant du fichier.

Q-4.2.3 : Ecrivez un programme qui permette de lire dans le fichier puis d'afficher les notes du n^{ième} étudiant.

Q-4.2.4 : Ecrivez un programme qui permette de lire dans le fichier les notes d'un étudiant dont on connait le nom.

Travaux Pratiques 5

Bibliothèques

Sommaire

5.1 GLUT/OpenGL - Damier

On donne un exemple basique sous GLUT/OpenGL pour démarrer. Repérez dedans l'initialisation, la mise en place du callback pour le dessin, et la fonction de dessin elle-même.

```
#include <stdio.h>
#include <GL/glut.h>

void display(void)
{
glClear( GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_POLYGON);
glVertex3f(2.0, 4.0, 0.0);
glVertex3f(8.0, 4.0, 0.0);
glVertex3f(8.0, 6.0, 0.0);
glVertex3f(2.0, 6.0, 0.0);
glEnd();
glFlush();
}

int main(int argc, char **argv)
{
printf("hello world");
glutInit(&argc, argv);
glutInitDisplayMode ( GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

glutInitWindowPosition(100,100);
glutInitWindowSize(300,300);
```

```
glutCreateWindow ("square");

glClearColor(0.0, 0.0, 0.0, 0.0); // black background
glMatrixMode(GL_PROJECTION); // setup viewing projection
glLoadIdentity(); // start with identity matrix
glOrtho(0.0, 10.0, 0.0, 10.0, -1.0, 1.0); // setup a 10x10x2 viewing world

glutDisplayFunc(display);
glutMainLoop();

return 0;
}
```

Ce code est disponible ici :

damierGL.c
(attaché au PDF)

ou

damierGL.c
(via internet)

Q-5.1.1 : Ecrivez une fonction qui dessine un carré de taille et de position passés en paramètre. Utilisez cette fonction pour dessiner un damier bleu et rouge à l'écran.

Q-5.1.2 : Ecrivez une fonction qui dessine un cercle de rayon et de position passés en paramètre. Note : on ne peut pas dessiner directement un cercle en openGL. En revanche, on peut dessiner une ligne brisée avec suffisamment de segments pour donner l'illusion qu'il s'agit d'un cercle. On pourra utiliser simplement des projections pour les coordonnées du cercle :

$$x_k = R \times \cos(k \times \Delta\alpha)$$
$$y_k = R \times \sin(k \times \Delta\alpha)$$

où k est un entier $< N$ et $\Delta\alpha = 2\pi/N$ avec N le nombre de points. Utilisez cette fonction pour dessiner un damier bleu et rouge à l'écran. Testez cette fonction en dessinant quelques "pions" sur le damier.

Q-5.1.3 : Trouvez sur internet comment on redéfinit le "callback" pour la gestion de la souris avec GLUT. Ecrivez une fonction qui détermine dans quelle case du damier la souris a cliqué. Affichez le résultat dans la console (avec `printf`). Dessinez un cercle dans la case qui a été cliquée.

Q-5.1.4 : Définissez maintenant une structure qui vous permet de gérer des pions (les ronds) sur un damier. Remplissez cette structure avec les conditions de départ d'une partie de dames, et faites afficher à la fonction de dessin l'état du damier à partir de cette structure. Ecrivez une fonction qui permet de gérer le déplacement des pions à la souris. Vous aurez besoin de `glutpostredisplay` pour forcer openGL à redessiner.

Q-5.1.5 : On veut remplacer les couleurs uniformes du damier par des textures. Pour vous y aider, on donne le code permettant de charger une image au format BMP, que vous ajouterez à votre projet sous la forme de 2 fichiers :

LoadBmpTexture.c

```
#include "LoadBmpTexture.h"
#include <stdio.h>

GLuint loadBMPTexture(const char * imagepath)
{
// Data read from the header of the BMP file
unsigned char header[54]; // Each BMP file begins by a 54-bytes header
unsigned int dataPos;     // Position in the file where the actual data begins
unsigned int width, height;
unsigned int imageSize;   // = width*height*3
// Actual RGB data
unsigned char * data;

FILE * file = fopen(imagepath,"r");
if (file==NULL)                    {printf("Image could not be opened\n"); return 0;}

if ( fread(header, 1, 54, file)!=54 ){ // If not 54 bytes read : problem
    printf("Not a correct BMP file\n");
    return 0;
}

if ( header[0]!='B' || header[1]!='M' ){
    printf("Not a correct BMP file\n");
    return 0;
}

// Read ints from the byte array
dataPos    = *(int*)&(header[0x0A]);
imageSize  = *(int*)&(header[0x22]);
width      = *(int*)&(header[0x12]);
height     = *(int*)&(header[0x16]);

if (imageSize==0)   imageSize=width*height*3; // 3 : one byte for each Red, Green and Blue
if (dataPos==0)      dataPos=54; // The BMP header is done that way

// Create a buffer
data = malloc(imageSize*sizeof(char));

// Read the actual data from the file into the buffer
fread(data,1,imageSize,file);

//Everything is in memory now, the file can be closed
fclose(file);

// Create one OpenGL texture
GLuint textureID;
glGenTextures(1, &textureID);
```

```
// "Bind" the newly created texture : all future texture functions will modify this textur
glBindTexture(GL_TEXTURE_2D, textureID);

// Give the image to OpenGL
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, data)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
return textureID;
}
```

Ce code est disponible ici :

LoadBmpTexture.c
(attaché au PDF)

ou

LoadBmpTexture.c
(via internet)

LoadBmpTexture.h

```
#ifndef __LOADBMPTEXTURE_H__
#define __LOADBMPTEXTURE_H__
#include <GL/glut.h>            /* Open GL Util    OpenGL*/

GLuint loadBMPTexture(const char * imagepath);
#endif
```

Ce code est disponible ici :

LoadBmpTexture.h
(attaché au PDF)

ou

LoadBmpTexture.h
(via internet)

La texture s'utilise comme suit (c'est à répéter pour chaque texture voulue) : Vous devez d'abord déclarer une variable globale :

```
GLuint maTexture;
```

Puis, dans le main, vous devez charger la texture :

```
maTexture = loadBMPTexture("/home/myara/Bureau/opengl/test00/basic.bmp");
```

Ensuite, renseignez vous sur internet sur les fonctions `glEnable` , `glBindTexture` et `glTexCoord2d` pour plaquer la texture.

Q-5.1.6 : (facultatif) Programmez un jeu de dames à 2.

5.2 GLUT/OpenGL - Logiciel de dessin

Ecrivez un logiciel de dessin qui permet de dessiner un polygone. Utilisez une liste chainée pour stocker les coordonnées des points. Réalisez la sauvegarde et le chargement d'un dessin via un fichier.

5.3 GLUT/OpenGL - Traceur de courbes

Q-5.3.1 : Ecrivez un programme capable de tracer une courbe (par exemple $y = sin(x)$ sur $[-1 \quad 1]$) en ajoutant les chiffres sur les axes. Pour afficher du texte dans la fenêtre OpenGL, essayez le code ci-dessous et adaptez le :

```
glRasterPos2i(0,0);
 glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 'A');
 glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 'B');
```

Q-5.3.2 : Affichez la "grille".

Q-5.3.3 : Gérez les échelles log et semi-log pour les 2 axes.

Compléments

Travaux Pratiques A

Compilation, Projets, Bibliothèques

Sommaire

En deux phases, on va créer une bibliothèque. Cela nécessite d'abord de comprendre ce qu'est la compilation multi-fichiers, comme proposé dans le premier exercice.

A.1 Compilation multi-fichiers

On va partir du code suivant :

```
#include <stdio.h>

int TailleChaine(char* s)
{
int i=0;
while(s[i]!=0)
  { i++;  }
return i;
}

void RenverseChaine(char* s)
{
int i;
int taille = TailleChaine(s);
char pivot;


for(i=0;i<taille/2;i++)
  {
  pivot = s[i];
  s[i] = s[taille-1-i];
  s[taille-1-i] = pivot;
```

```
  }
}

void AfficheChaine(char* s)
{
printf("%s\n",s);
}



int main()
{
char texte[200];
int len;
printf("Quel est le mot ? ");
scanf("%s",texte);
AfficheChaine(texte);
len = TailleChaine(texte);
printf("Cette chaine fait %d caracteres\n",len);
RenverseChaine(texte);
AfficheChaine(texte);

return 0;
}
```

Ce code est disponible ici :

`multi.c` (attaché au PDF) ou `multi.c` (via internet)

Ce code contient 3 fonctions qui concernent le traitement des chaines de caractères + la fonction `main`. On veut placer les 3 fonctions dans un fichier séparé.

Q-1.1.1 : Travail Préparatoire : commencez par créer un projet codeblocks et copiez ce code à l'intérieur du projet. Vérifiez que cela fonctionne correctement

Q-1.1.2 : On va maintenant séparer dans deux fichiers .c les fonctions : un fichier contiendra les 3 premières fonctions (vous l'appellerez "chaines.c"), un autre contiendra la fonction main (vous l'appelerez "main.c"). **Attention : ces deux fichiers devront appartenir au même projet Codeblocks !**

Pour cela : A partir du projet créé à la question précédente, allez dans le menu "File" puis faites "New File", et sélectionnez "Empty File C". Puis une interface vous propose de donner des informations sur le nouveau fichier. Pour le nom de fichier indiquez "chaine.c", vérifiez que la case "add file to active project" est sélectionnée, puis que les "build targets" Debug et Release sont cochées.

Vous devez voir dans votre projet le fichier "main.c" qui existait déjà, et un nouveau fichier vide appelé "chaine.c". Faites un couper/coller des 3 fonctions déjà présentes dans "main.c" (de la ligne 2 à la ligne 30) vers le fichier "chaine.c".

Q-1.1.3 : **Compilation de "chaine.c" :** Vous pouvez alors essayer de compiler chacun

des fichiers séparément. Pour cela, cliquez avec le bouton droit sur l'icone "chaine.c" dans codeblocks puis choisissez l'option "build file". Vous avez alors la liste d'erreurs ci-dessous :

```
chaine.c: In function 'AfficheChaine':
chaine.c:27:1: attention : incompatible implicit declaration of built-in function 'print
/usr/lib/gcc/i686-linux-gnu/4.6/../../../i386-linux-gnu/crt1.o: In function `_start':
(.text+0x18): undefined reference to `main'
```

Il y a 2 erreurs :

- d'abord le compilateur ne trouve pas `printf`. C'est normal, `printf` n'est pas connu par "chaine.c" alors qu'elle est appelée dans `AfficheChaine`. La raison est que nous n'avons pas ajouté `#include <stdio.h>` dans le fichier "chaine.c", il faut le faire.
- Ensuite, le compilateur ne trouve pas de fonction `main` dans ce fichier. C'est normal pour ce que l'on veut faire, il n'y aura qu'une seule fonction `main` dans notre projet (et il ne peut n'y en avoir qu'une et une seule par projet !), et elle sera dans `main.c`.

A ce stade, si vous recompilez "chaine.c" il doit se recompiler sans erreur.

Q-1.1.4 : Compilation de "main.c" : Faites de même que précédemment pour compiler main.c. Vous obtiendrez la liste d'erreur ci-dessous :

```
main.c:(.text+0x45): undefined reference to `AfficheChaine'
main.c:(.text+0x51): undefined reference to `TailleChaine'
main.c:(.text+0x76): undefined reference to `RenverseChaine'
main.c:(.text+0x82): undefined reference to `AfficheChaine'
```

Ici, "tout est normal" : le compilateur ne connait pas les fonctions que l'on a enlevé de "main.c", et lorsqu'il compile "main.c" cela pose un problème. Pour compiler "main.c", il a au moins besoin de connaitre le prototype de ces fonctions. Ainsi, pour "faire passer" la compilation, vous pourriez ajouter les lignes suivantes à main.c, au dessus du main :

```
int TailleChaine(char* s);
void RenverseChaine(char* s);
void AfficheChaine(char* s);
```

Cette fois, vous pouvez relancer la compilation complète du projet, il va fonctionner exactement comme initialement. Ceci est parfaitement correct mais on préfère souvent organiser le projet autrement.

Q-1.1.5 : Création du fichier "chaine.h" : En pratique, on préfère placer les déclarations des fonctions définies dans un fichier .c à l'intérieur d'un fichier .h correspondant. Par exemple, pour "chaine.c", on va créer un fichier "chaine.h". Comme précédemment, créez un nouveau fichier pour le projet et appelez ce fichier "chaine.h" (son type sera "C/C++ header"). **Supprimez** les 3 déclarations de fonctions que l'on a ajouté à "main.c" précédemment, et placez les dans le nouveau fichier "chaine.h".

Note : lorsque vous créez chaine.h, codeblocks ajoute automatiquement du texte dans le fichier, si bien que votre fichier chaine.h ressemble à :

```
#ifndef CHAINE_H_INCLUDED
#define CHAINE_H_INCLUDED

int TailleChaine(char* s);
void RenverseChaine(char* s);
```

```
void AfficheChaine(char* s);

#endif // CHAINE_H_INCLUDED
```

Ce texte supplémentaire sert à éviter certains problèmes de compilation (problème des "inclusions multiples") liés aux fichiers .h.

Ajoutez ensuite `#include "chaine.h"` en haut du fichier main.c et relancez la compilation : tout doit fonctionner!

A.2 Création d'une bibliothèque statique

On va maintenant créer une bibliothèque contenant les 3 fonctions. En fait, le travail le plus difficile a été fait à l'exercice ci-dessus : la séparation des fichiers. Pour créer une bibliothèque, on va réutiliser les fichiers chaine.c et chaine.h que l'on a créé ci-dessus.

Nous allons nous placer dans le cas des bibliothèques dites "statiques", c'est à dire les bibliothèques qui seront inclues pendant la compilation du projet. Il existe d'autres types de bibliothèques (dites "dynamiques") qui sont chargées pendant l'exécution du programme, qui ne sont pas traitées ici (leur traitement change d'un système d'exploitation à l'autre).

La première chose à faire est de créer un projet de type "bibliothèque statique" avec Codeblocks. Créez donc un projet de type "Static library" avec Codeblocks. Appelez ce projet "chainelib". Dans le "main.c" de ce projet, copiez exactement ce que l'on avait mis dans "chaine.c" précédemment, puis compilez le projet. **Notez que dans ce projet il n'y a pas de fonction main**. C'est absolument normal! c'est une bibliothèque, donc une liste de fonctions, pas un programme avec un point de départ identifié.

C'est terminé! La bibliothèque est prête à l'emploi dans le répertoire de votre projet, et s'appelle "libchainelib.a" (sous linux). Le nom peut changer d'un système d'exploitation à l'autre (libchainelib.lib sous windows).

A.3 Utilisation de la bibilothèque statique

Créez un nouveau projet "console application" avec Codeblocks, appelez le "ExempleChaineLib". Puis allez dans le gestionnaire de fichiers (sortez de codeblocks), et recopiez les fichiers "libchainelib.a" et le fichier "chaine.h" dans le répertoire du projet "ExempleChaineLib". Par soucis d'homogénéité, renommez alors la copie de "chaine.h" en tant que "libchainelib.h". Ensuite, il vous faut ajouter la librairie au projet. Pour cela, allez dans le menu "Project", rubrique "Properties". Une interface s'ouvre, cliquez sur "Project's build options". Allez alors dans l'onglet "Linker Settings" et cliquez sur "Add" sous "Link Libraries". Naviguez alors jusqu'à la copie de la librairie (dans votre répertoire/projet) et sélectionnez "libchainelib.a". A l'invite "keep this as relative path", choisissez "non". Ensuite validez toutes les interfaces de dialogue jusqu'à revenir au niveau du projet.

Dans le main.c, recopiez le code du main.c du projet créé pendant le premier exercice. Remplacez le `#include "chaine.h"` par `#include "libchainelib.h"`.

Compilez, cela doit fonctionner. Vous avez utilisé la librairie que vous aviez créé.

Travaux Pratiques B

Exercices complémentaires

Sommaire

Ce TP contient un ensemble d'exercices que vous pouvez faire chez vous. Il y a deux catégories d'exercices : les exercices "débutant", qui sont pensés pour aider les étudiants qui n'ont pas le niveau L2 à démarrer, et les autres exercices qui sont des compléments aux exercices faits pendant les séances, pour vous entrainer, et sont classés parr thème.

Certains exercices niveau débutant sont issus ou adaptés de `https://www.ltam.lu/cours-c/prg-c_c.htm`, qui peut aussi vous servir de cours de C de base.

■ Débutant

B.1 Maximum de 3 valeurs

Q-2.1.1 : Ecrivez un programme qui calcule le maximum de 3 entiers, préalablement saisis au clavier.

Q-2.1.2 : Même question avec des `float`

Q-2.1.3 : Même question avec des `char`

Q-2.1.4 : Même question avec des `double`

B.2 Permutations

Ecrire un programme qui permute et affiche les valeurs de trois variables A, B, C de type entier qui sont entrées au clavier :

```
A ==> B , B ==> C , C ==> A
```

B.3 Trinomes

Q-2.3.1 : Ecrivez un programme qui calcule les solutions réelles d'une équation du second degré :

$$ax^2 + bx + c = 0$$

en discutant la formule :

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \qquad \text{avec} \qquad \Delta = b^2 - 4ac.$$

Utilisez une variable d'aide D pour la valeur du discriminant et décidez à l'aide de D, si l'équation a une, deux ou aucune solution réelle. Utilisez des variables du type int pour A, B et C.

Considérez aussi les cas où l'utilisateur entre des valeurs nulles pour A ; pour A et B ; pour A, B et C. Affichez les résultats et les messages nécessaires sur l'écran.

B.4 Resistance Equivalente

Ecrire un programme qui affiche la résistance équivalente à trois résistances R1, R2, R3 (type double),

— pour les résistances sont branchées en série :

$$R_{ser} = R_1 + R_2 + R_3$$

— pour les résistances sont branchées en parallèle :

$$R_{par} = \frac{R_1 R_2 R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3}$$

B.5 Triangle

Ecrire un programme qui calcule et affiche l'aire d'un triangle dont il faut entrer les longueurs des trois côtés. Utilisez la formule :

$$A = P(P - A)(P - B)(P - C)$$

où A, B, C sont les longueurs des trois côtés et P le demi-périmètre du triangle.

B.6 Distance

Ecrire un programme qui calcule et affiche la distance DIST (type double) entre deux points A et B du plan dont les coordonnées (XA, YA) et (XB, YB) sont entrées au clavier comme entiers.

B.7 Tables de multiplication

Q-2.7.1 : Ecrivez un programme qui demande un entier entre 1 et 12 à l'utilisateur. Si la valeur n'est pas correcte le progamme doit s'entêter à redemander, autant de fois que nécessaire. Ensuite, le programme devra afficher la table de multiplication de la valeur saisie.

Q-2.7.2 : Ecrivez un programme qui affiche un tableau global des tables de multiplication pour N variant de 1 à 10 :

```
 X*Y I   0   1   2   3   4   5   6   7   8   9  10
-------------------------------------------------
  0  I   0   0   0   0   0   0   0   0   0   0   0
  1  I   0   1   2   3   4   5   6   7   8   9  10
  2  I   0   2   4   6   8  10  12  14  16  18  20
  3  I   0   3   6   9  12  15  18  21  24  27  30
  4  I   0   4   8  12  16  20  24  28  32  36  40
  5  I   0   5  10  15  20  25  30  35  40  45  50
  6  I   0   6  12  18  24  30  36  42  48  54  60
  7  I   0   7  14  21  28  35  42  49  56  63  70
  8  I   0   8  16  24  32  40  48  56  64  72  80
  9  I   0   9  18  27  36  45  54  63  72  81  90
 10  I   0  10  20  30  40  50  60  70  80  90 100
```

B.8 Palindrome

Q-2.8.1 : Ecrivez un programme qui demande la saisie d'un mot au clavier, puis qui détermine si le mot est un palindrome. Par exemple, BOB et RADAR sont des palindromes. Par simplicité on supposera que le mot est tapé tout en majuscules ou tout en minuscules.

B.9 Exercice de base sur les pointeurs

Q-2.9.1 : Soit P un pointeur qui "pointe" sur un "tableau" A :

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses (relatives à `A` fournissent ces expressions :

1. `*P+2`
2. `*(P+2)`
3. `&P+1`
4. `&A[4]-3`
5. `A+3`
6. `&A[7]-P`
7. `P+(*P-10)`
8. `*(P+*(P+8)-A[7])`

Vous chercherez d'abord, en réfléchissant et dessinant le tableau, quelles sont les réponses théoriques, puis vérifierez en testant dans un `main` que votre idée est juste.

B.10 Table ASCII

Q-2.10.1 : Ecrivez un programme qui affiche la table ASCII de tous les caractères dont le code ASCII est situé entre 32 et 127. La table ASCII devra afficher sur chaque ligne le code ASCII d'une lettre en décimal, en hexadécimal, puis la lettre elle-même. Par exemple pour la ligne du A :

```
65 42 A
```

N'hésitez pas à comparer vos résultats avec une table ASCII trouvée sur internet.

Q-2.10.2 : Même question en affichant la même table ASCII mais présentée en affichant 5 lettres différentes avec leur code ASCII et leur code hexadécimal sur la même ligne. Par exemple une ligne pourrait contenir :

```
65 42 A    66 43 B    67 44 C    68 45 D     69 56 E
```

B.11 Pyramide et Croix

On veut faire afficher à un programme des pyramides dessinées avec des caractèreres dans le terminal. L'idée est de donner au programme le nombre de lignes sur laquelle elle s'étendra, et de la générer ensuite. Par exemple, si on donne `n=5`, le programme doit générer :

```
    *
   ***
  *****
 *******
*********
```

Q-2.11.1 : Ecrivez une fonction qui affiche m espaces. Justifiez son prototype.

Q-2.11.2 : Ecrivez une fonction qui affiche p étoiles. Justifiez son prototype.

Q-2.11.3 : Si on note k le numéro de la ligne :
— déterminez combien il faut afficher d'étoiles en fonction de k,
— déterminez combien il faut afficher de caractères "espace" avant d'afficher la première étoile de chaque ligne en fonction de la valeur de k.

A partir de ces remarques, écrivez une fonction qui affiche une ligne de la pyramide en fonction de la valeur de k. Cette fonction fera appel aux deux fonctions écrites à la question précédente.

Q-2.11.4 : Ecrivez enfin une fonction :
`void pyramide(int n);` qui affiche la pyramide complète.

Q-2.11.5 : Ecrivez un main qui propose à l'utilisateur de saisir la hauteur de la pyramide puis l'affiche.

Q-2.11.6 : Sur le même principe, écrivez des fonctions et un programme qui affiche un "V", puis un "X", comme suit (ici exemple pour 4 dans les deux cas).

Pour le V :

```
*     *
 *   *
  * *
   *
```

Pour le X :

```
*     *
 *   *
  * *
   *
  * *
 *   *
*     *
```

Pointeurs et Tableaux

B.12 Minimum et Maximum d'un tableau

Q-2.12.1 : Ecrivez une fonction qui fournit en sortie le minimum et le maximum d'un tableau de `float`. Son prototype sera :

```
void minmax(float* tab,int nval,float* min, float* max);
```

Q-2.12.2 : Ecrivez une fonction `main` qui remplit un tableau en tirant 20 valeurs au sort et en calcule le min et le max. Vous utiliserez la fonction `rand` du C pour générer les nombres aléatoires.

B.13 Rendre la monnaie

Q-2.13.1 : Sur le même principe que la décomposition en jours-heures-minutes-secondes (exercice 1.3, écrivez une fonction qui prend en paramètre une somme en euros (entière) donnée par un client ainsi que la somme à payer, et calcule comment rendre la monnaie en billets de 100€, 50€, 20€, 10€, et 5€, et en pièces de 2€ et 1€, en donnant la priorité aux grandes coupures. La fonction utilisera le passage par adresse pour chaque type de billet ou piece.

Q-2.13.2 : Utilisez cette fonction sur un exemple dans le main.

B.14 Occurrences

Ecrire un programme qui lit une chaîne de caractères `CH` au clavier et qui compte les occurrences des lettres de l'alphabet en ne distinguant pas les majuscules et les minuscules. Utiliser un tableau ABC de dimension 26 pour mémoriser le résultat et un pointeur PCH pour parcourir la chaîne `CH` et un pointeur PABC pour parcourir ABC. Afficher seulement le nombre des lettres qui apparaissent au mois une fois dans le texte.

Exemple :

```
Entrez un ligne de texte (max. 100 caractères) :
Jeanne
La chaîne "Jeanne" contient :
1   fois la lettre 'A'
2   fois la lettre 'E'
1   fois la lettre 'J'
3   fois la lettre 'N'
```

B.15 Histogramme

On part du code ci-dessous :

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


float ranf()         /* ranf() is uniform in 0..1 */
{
return ((float)rand())/RAND_MAX;
}
// moyenne m, ecart type s
// methode dite de box-muller
float ranNorm(float m, float s)
{         /* mean m, standard deviation s */
float x1, x2, w, y1;
static float y2;
static int use_last = 0;

if (use_last)
{
y1 = y2;
use_last = 0;
}
else
{
do {
x1 = 2.0 * ranf() - 1.0;
x2 = 2.0 * ranf() - 1.0;
w = x1 * x1 + x2 * x2;
} while ( w >= 1.0 );

w = sqrt( (-2.0 * log( w ) ) / w );
y1 = x1 * w;
y2 = x2 * w;
use_last = 1;
}
return( m + y1 * s );
}



int main()
{
srand(time(NULL));

int Nval;
int i;
Nval = 10000;
```

```
float* val = malloc(Nval*sizeof(float));
for(i=0;i<Nval;i++)
{val[i]=ranNorm(5,5);
}

// ### A completer

return 0;
}
```

Ce code est disponible ici :

histo.c
(attaché au PDF)

ou

histo.c
(via internet)

Sans chercher à comprendre les détails, ce programme génère un tableau `val` de 10000 valeurs aléatoires répondant à la loi normale.

Q-2.15.1 : Complétez ce programme pour qu'il calcule l'histogramme du tableau `val` sur N tranches, N étant choisi par l'utilisateur du programme. Cet histogramme sera calculé par une fonction dont le prototype sera : float* histogramme(float* val, int nval, int N);

Vous utiliserez la fonction `minmax` créée lors de l'exercice B.12 pour déterminer l'étendue des valeurs.

Q-2.15.2 : Ajoutez maintenant une fonction qui représente l'histogramme obtenu en mode texte, horizontalement, comme représenté ci-dessous.

```
 1 *
 2 *
 3 *
 4 **
 5 *****
 6 **********
 7 ********************
 8 ********************************
 9 ******************************************
10 **************************************************
11 **************************************************
12 ******************************************
13 **********************************
14 ************************
15 **************
16 *******
17 ****
18 **
19 *
20 *
```

B.16 Cryptage

On veut crypter du texte.

Q-2.16.1 : Ecrivez un programme qui crypte une chaine de caractères en suivant le fonctionnement suivant :

— les minuscules seront échangées telles que `'a'` devient `'z'`, `'b'` devient `'y'`, `'c'` devient `'x'`, et ainsi de suite

— de même, les majuscules sont échangées telles que `'A'` devient `'Z'`, `'B'` devient `'Y'`, `'C'` devient `'X'`, et ainsi de suite.

— Les autres caractères ne sont pas modifiés.

Il devra :

1. proposer la saisie d'une chaine de caractères au clavier (utilisez `scanf` avec la commande `"%[^.]"` au lieu de `"%s"` pour permettre la saisie d'une phrase qui se termine par un point au lieu de la saisie d'un simple mot).
2. réaliser le calcul de la chaine cryptée dans un nouveau tableau
3. afficher la d'origine et la chaine cryptée.

Q-2.16.2 : Mettez la partie du programme qui réalise le calcul de la chaine uniquement dans une fonction dont le prototype sera :

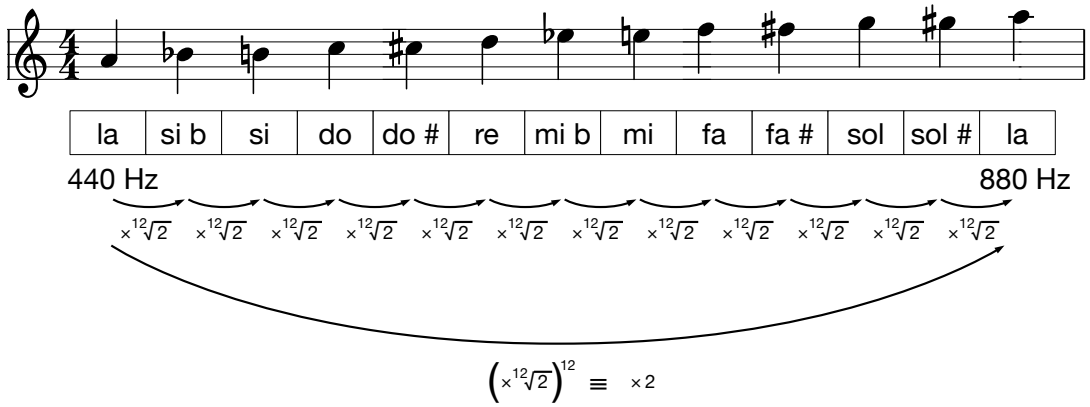
```
char* cyptage(char* s);
```

et modifiez le `main` pour qu'il utilise maintenant cette fonction à la place du code qui y réalisait le calcul avant.

Q-2.16.3 : Expliquez pourquoi le cryptage et le décryptage de la chaine consiste à utiliser le même algorythme. Proposez une fonction `main` très simple qui le montre, à partir de ce qui a été fait cia-dessus.

B.17 Notes de musique

Ce que nos percevons comme des notes de musique sont physiquement des fréquences : chaque note correspond à une fréquence. Les notes s'obtiennent par une progression géométrique (multiplicative) de raison $\sqrt[12]{2} = 2^{1/12}$). Un point de référence possible est le "la" à $440\,Hz$. On représente ci-dessous la séquence de base pour toutes les notes entre deux "la". Cet écart entre deux "la" correspond à une "octave", c'est à dire l'écart entre une fréquence et son double :



Exemple à partir du la 440 Hz

Cette séquence se répète à l'identique pour tous les "la" : on pourrait la reproduire exactement pour l'intervalle $110\,Hz - 220\,Hz$ ou encore $880\,Hz - 1760\,Hz$ (et en deça et au-dela).

Q-2.17.1 : Ecrivez une fonction qui prend en paramètre le nom d'une note et donne la valeur de sa fréquence dans l'intervalle $440\,Hz - 880\,Hz$. Essayez la dans un `main`.

Q-2.17.2 : On appelle "la0" le la à $55\,Hz$, "la1" le la à $110\,Hz$, le "la2" le la à $220\,Hz$, et ainsi de suite : c'est le "la" de base pour l'octave considérée. Ecrivez une fonction similaire à la fonction précédente, sauf qu'elle prendra en paramètre supplémentaire le "numéro" du la de base de l'octave considérée.

Q-2.17.3 : Ecrivez une fonction qui prend en paramètre la valeur d'une fréquence et renvoie le nom de la note qui correspond. La fréquence donnée pourra ne pas être exactement celle d'une note, aussi il faudra donner la note la plus proche de la fréquence considérée. Vous ajouterez une sortie à la fonction pour fournir en sortie l'erreur relative sur la fréquence par rapport à la fréquence de la note exacte.

Q-2.17.4 : (un peu difficile) Sachant que les sons émis par les instruments de musique sont des fonction périodiques (ou presque) en fonction du temps, déduisez-en les fréquences des 10 premières harmoniques. Ecrire une fonction qui renvoie un tableau de pointeurs sur les chaines les notes associées aux 10 premières harmoniques d'une note dont on donne le nom en paramètre.

B.18 Scrabble

On veut écrire un programme qui calcule automatiquement le score d'un mot au Scrabble. Il s'agit ici d'une version simplifiée, on ne considèrera pas l'effet des bonus de la grille tels "compte double".

La liste des scores du scrabble est donnée dans le tableau des scores ci-dessous.

```
int main()
{int scoresFr[]={1,3,3,2,1,4,2,4,1,8,10,1,2,1,1,3,8,1,1,1,1,4,10,10,10,10};

}
```

Ce code est disponible ici :

`scoreScrabble.c` (attaché au PDF) ou `scoreScrabble.c` (via internet)

Ils sont simplement dans l'ordre des lettres : la case 0 contient le score du `'A'`, la case 1 celui du `'B'`, et ainsi de suite. Le score d'un mot est simplement calculé en effectuant la somme des scores des lettres qui le composent.

Q-2.18.1 : Ecrivez une fonction qui affiche la liste des scores sous la forme `lettre : score`, puis un `main` qui pemet de la tester.

Q-2.18.2 : Ecrivez une fonction qui renvoie l'équivalent majuscule d'une lettre minuscule lorsqu'elle en reçoit une. Dans les autres cas elle renverra simplement la lettre passée en paramètre. Par exemple, elle renverra `'B'` si on lui donne `'b'`, mais renverra `'M'` si on lui donne `'M'`.

Q-2.18.3 : Ecrivez une fonction qui prend une lettre en paramètre et renvoie le score associé. La lettre doit pouvoir être majuscule ou minuscule, ce qui vous conduia à utiliser la fonction écrite ci-dessus.

Remarque : Si `c` *est une variable contenant le code ASCII d'une lettre majuscule, alors le numero de la case associée dans le tableau des scores est simplement* `c-'A'`.

Q-2.18.4 : Ecrivez alors, en utilisant les fonctions ci-dessus, une fonction qui calcule le score d'un mot au scrabble. Utilisez la sur un exemple dans le `main`.

Q-2.18.5 : En plus des scores, on donne maintenant le nombre de lettres disponible dans le jeu de scrabble, dans le meme ordre que les scores :

```
int main()
{int scoresFr[]={1,3,3,2,1 ,4,2,4,1,8,10,1,2,1,1,3,8,1,1,1,1,4,10,10,10,10};
 int nombreFr[]={9,2,2,3,15,2,2,2,8,1,1 ,5,3,6,6,2,1,6,6,6,6,1,1 ,1 ,1 ,1 };
}
```

Ce code est disponible ici :

`scoreNbScrabble.c` (attaché au PDF) ou `scoreNbScrabble.c` (via internet)

Ainsi, il iy a 9 `'A'`, 2 `'B'`, et ainsi de suite.

Documentez vous sur internet au sujet de la fonction `rand` et écrivez une fonction qui tire au sort un nombrde lettres n au hasard et les renvoie sous forme de chaine de caractères. Elle mettra aussi à jour le tableau `nombreFr`du main. Son prototype sera donc :

`char* tireLettres(int n, int*nombreFr);`

Elle ne devra pas aboutir à un nombre négatif de lettres dans le tableau `nombreFr` du `main`.

Q-2.18.6 : Proposez maintenant un programme mettant en jeu une grille de Scrabble de taille mxm arbitraire (sans bonus). Il tirera les lettres automatiquement pour les joueurs, et leur permettra de placer les mots dans la grille. On ne cherchera pas à vérifier que le placement proposé par le joueur pour son mot est correct.

B.19 Opérations supplémentaires sur les matrices

Complétez l'exercice 1.6 en écrivant des fonctions qui calculent :

1. La transposée,
2. Le déterminant,
3. La matrice inverse (un peu difficile)

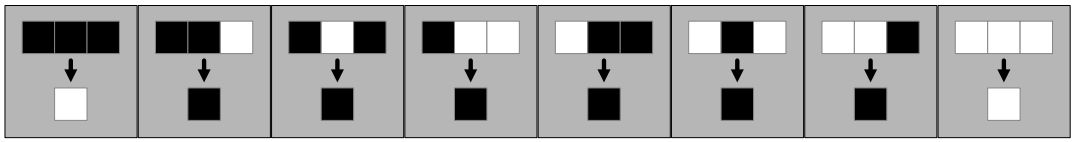
Structures

B.20 Automate cellulaire 1D

Note : Je vous encourage à aller regarder une video youtube sur ce sujet : Sciences étonantes #49 : le jeu de la vie.

On veut programmer un automate cellulaire. Il s'agit d'un programme qui part d'un tableau de valeurs contenant uniquement des 0 et des 1, et qui va, en utilisant des règles d'évolution, définir un nouvel état du tableau. A partir de ce nouveau état, on pourra appliquer à nouveau les règles, et ainsi de suite. C'est ce que l'on propose de programmer.

Dans ce que l'on propose ici, l'état de chaque case de l'étape suivante sera toujours définie par une règle sur l'état de 3 cases connexes du tableau. Voici par exemple une série de règles (appelées "règle 126") que l'on peut appliquer :



Avec ces règles et un tableau de 80 cases contenant uniquement des 0 et un seul 1 au milieu, et en calculant l'évolution du tableau sur 24 étapes, on obtient (les 0 sont des espaces et les 1 sont des *) :

```
01                                *
02                               ***
03                              ** **
04                             *******
05                            **     **
06                           ****   ****
07                          **  ** **  **
08                         ***************
09                        **             **
10                       ****           ****
11                      **  **         **  **
12                     ********       ********
13                    **      **     **      **
14                   ****    ****   ****    ****
15                  **  **  **  ** **  **  **  **
16                 ********************************
17                **                              **
18               ****                            ****
19              **  **                          **  **
20             ********                        ********
21            **      **                      **      **
22           ****    ****                    ****    ****
23          **  **  **  **                  **  **  **  **
24         ****************                ****************
```

Q-2.20.1 : Expliquez pourquoi la structure ci-dessous modélise correctement une règle :

```
typedef struct {
char tab[3];
char resultat;
} regle;
```

Q-2.20.2 : Ecrivez une fonction qui permet de remplir une structure règle :

```
regle init(char* t, char r);
```

Q-2.20.3 : Ecrivez dans le main un tableau qui contient toutes les règles énoncées dans le dessin proposé plus haut.

Q-2.20.4 : Ecrivez enfin le programme qui permet de générer la figure ci-dessus.

B.21 Matrices 2×2 de nombres complexes

On veut gérer des matrices 2× 2 de nombres complexes.

Q-2.21.1 : Reprenez le type complexe défini à l'exercice 2.1. Ecrivez des fonctions pour réaliser la saisie, l'affichage, la somme et le produit de nombres complexes :

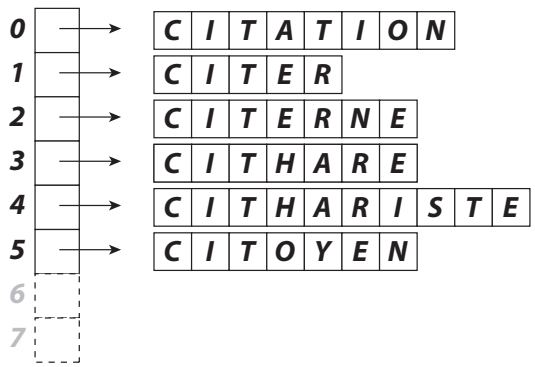
```
void afficheCplx(complexe* z);
void saisieCplx(complexe* z);
complexe sommeCplx(complexe g, complexe d);
complexe produitCplx(complexe g, complexe d);
```

Réécrivez complètement l'exercice 2.2 où les nombres de la matrice sont des nombres complexes définis par le type `complexe` évoqué ci-dessus.

B.22 Dictionnaire

Note Préliminaire : Pour bien se rendre compte de ce qui est mis en cause dans cet exercice, n'utilisez pas la fonction de comparaison de chaînes `strcmp` appartenant à la bibliothèque `string.h`. En revanche, vous pourrez utiliser toutes les autres fonctions de cette bibliothèque.

Partie I : De nombreux logiciels de traitement de texte disposent d'un dictionnaire pour la correction orthographique. Une façon simple de stocker tous les mots d'un tel dictionnaire est de créer un tableau de chaînes de caractères contenant les mots les uns à la suite des autres, dans l'ordre alphabétique pour accélérer la recherche, comme celà est représenté sur la figure suivante :



Q-2.22.1 : Proposez une structure pour gérer correctement ce dictionnaire.

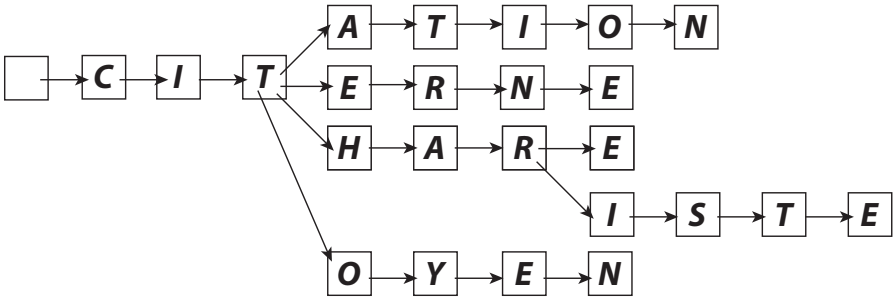
Q-2.22.2 : Ecrivez une fonction qui permet d'ajouter un mot au dictionnaire. On supposera que le dictionnaire contient 10 mots maximum, et on ne cherchera pas à maintenir l'ordre alphabatique lors de l'ajout d'un mot au dictionnaire.

Q-2.22.3 : Ecrivez une fonction qui permet d'afficher tous les mots contenus dans le dictionnaire.

Q-2.22.4 : Ecrivez une fonction qui permet de dire si un mot est dans le dictionnaire.

Partie II : Cette structure de dictionnaire a l'avantage d'être simple, mais elle a un gros défaut : si l'on recherche, sur l'exemple donné, le mot "CITERNE", on voit bien que l'on a comparé le mot demandé deux fois à la sous-chaine de caractères "CIT" pour en quelque sorte rien puisque la comparaison a au total été un échec, et une fois à "CITER" pour rien : çà n'est qu'à la comparaison suivante que l'on a trouvé le "bon" mot.

On peut réaliser quelque chose de bien plus efficace, en adoptant une structure arborescente, c'est à dire en respectant le schéma suivant :



Q-2.22.5 : Proposez un type structuré permettant de gérer correctement l'information du dictionnaire sous cette forme.

Q-2.22.6 : (facultatif : long et un peu difficile) Proposez une fonction qui permette d'ajouter un mot au dictionnaire. Modifiez éventuellement la structure proposée pour permettre de maintenir facilement l'ordre alphabétique.

Q-2.22.7 : (facultatif) Ecrivez une fonction qui permet d'afficher tous les mots contenus dans le dictionnaire.

Q-2.22.8 : (facultatif) Ecrivez une fonction qui permet de dire si un mot est dans le dictionnaire. Si le mot n'est pas trouvé, proposez comme solution le mot le plus proche.

Fichiers

B.23 Analyse de Fichier

Q-2.23.1 : Ecrivez un programme qui permette de récupérer les valeurs du fichier de sortie en mode texte créé lors de l'exercice 4.1. On souhaite que ces valeurs soient stockées dans un tableau de `float`.

B.24 Tableau de Mendeleev

On dispose d'un fichier contenant le tableau de Mendeleev au format CSV :

`PeriodicTable.csv`
(attaché au PDF)

ou

`PeriodicTable.csv`
(via internet)

Ouvrez ce fichier avec un éditeur de texte ou avec un tableur. La première ligne décrit le contenu de chaque colonne.

Q-2.24.1 : Définissez une structure adaptée pour stocker les informations utiles pour chaque atome.

Q-2.24.2 : Réalisez le chargement du fichier pour remplir un tableau de structures, selon le modèle de structure que vous avez défini ci-dessus.

Q-2.24.3 : Ecrivez un `main` qui demande à l'utilisateur le symbole chimique d'un atome et affiche les informations associées.

Q-2.24.4 : (un peu difficile) Ecrivez une fonction qui, à partir de la formule d'une molécule donnée sous la forme d'une chaine de caractères, calcule sa masse. Testez la dans un `main`.

OpenGL

Ces exercices supposent tous les connaissances sur structures, pointeurs, etc.

B.25 GULT/OpenGL - Jeu de la vie

Q-2.25.1 : En vous inspirant de l'exercice , programmez une version graphique des automates cellulaires 1D.

Q-2.25.2 : Programmez maintenant un jeu de la vie (Sciences étonantes #49 : le jeu de la vie.).

B.26 GLUT/OpenGL - Puissance 4

Q-2.26.1 : Programmez un Puissance 4.

B.27 GLUT/OpenGL - Tetris

Q-2.27.1 : Ecrivez un Tetris! Vous aurez besoin de gérer le callback timer pour gérer les animations, renseignez vous sur : `glutTimerFunc`.

Mixtes

B.28 Carnet d'Adresses sur Fichier

Fait travailler : Structures, Fichiers

On souhaite gérer un carnet d'adresses en s'appuyant sur un fichier dans lequel les coordonnées des différents contacts seront stockées. Pour chaque contact, on souhaite stocker :

— le Nom, sur 20 caractères maximum,
— le Prénom, sur 20 caractères maximum,
— l'age, sous forme d'un long.

Pour celà, on propose les étapes suivantes :

Q-2.28.1 : Ecrivez le type `contact` pour qu'il corresponde bien à la description ci-dessus.

Q-2.28.2 : Ecrivez une fonction `main` réalisant le travail préalable d'ouverture du fichier dans le bon mode pour cette application.

Q-2.28.3 : Ecrivez une fonction permettant de faire la saisie d'un contact et de la stocker à la fin du fichier des contacts supposé déjà ouvert (on passera le `FILE* correspondant à la fonction`.

Q-2.28.4 : Ecrivez une fonction permettant d'afficher un contact chargé à partir du fichier, supposé déjà ouvert. On supposera aussi que le curseur sur le fichier est correctement placé au début d'une fiche.

Q-2.28.5 : Ecrivez une fonction permettant d'afficher le n-ième contact contenu dans le fichier.

Q-2.28.6 : Ecrivez une fonction permettant d'afficher la fiche d'un personnage dont on connaît le nom.

Q-2.28.7 : Ecrivez une fonction `main` proposant un menu à l'écran permettant de réaliser chacune des 4 opérations implémentées ci-dessus.

B.29 Gestion des Images

Fait travailler : Structures, Fichiers, Pointeurs

Nous avons lors de l'exercice 1.6 comment gérer une mémoire en deux dimensions. Une image est bien entendu une mémoire à deux dimensions, et donc les concepts qui ont été développés précédemment restent valables. On souhaite malgré tout améliorer un peu cette gestion en utilisant les structures. En effet, lorsque l'on manipule une image, on a toujours besoin de trois informations :

1. le pointeur sur les données
2. la largeur de l'image
3. la hauteur de l'image

On propose donc le travail suivant :

Q-2.29.1 : Ecrivez la structure `image` permettant de coder une image selon la description ci-dessus. On suppose que l'on va gérer des images en niveau de gris à 8 bit par pixel.

Q-2.29.2 : Ecrivez une fonction qui initialise correctement la structure. Cette fonction aura pour prototype :

```
image initialise(long x, long y);
```

Q-2.29.3 : Ecrivez une fonction xy permettant, à partir d'un couple de valeurs x et y de trouver à quel point du tableau de données celà correspond.

Q-2.29.4 : Utilisez tout ce qui vient d'être écrit pour générer une image en RAM de 400×400 pixels représentant un dégradé en diagonale du blanc vers le noir.

Note 1 : Si vous écrivez le code permettant de sauvegarder le fichier binaire contenant les données vues par le pointeur de données, vous obtiendrez un certain fichier. A partir de ce fichier, vous pouvez afficher ce dégradé dans notamment deux logiciels de dessin : "Adobe Photoshop"[1] ou "the Gimp"[2] en effectuant un chargement au format "données brutes"[3] et en précisant au logiciel que l'image est en niveau de gris, 8 bit, de largeur 400 pixels avec un "header" (ou en-tête) de 0 octets. Si tout se passe bien, vous devez voir le logiciel afficher le dégradé dans une fenêtre.

Note 2 : Certaines versions de "the Gimp" comportent un bug qui empêche parfois le chargement au format "données brutes". Ce bug peut etre contourné en effaçant le répertoire "`.gimp`" situé dans votre dossier utilisateur, puis en relançant "the Gimp".

B.30 Traitement de texte

Fait travailler : Pointeurs, Fichiers, structures avancées

Q-2.30.1 : Ajoutez le chargement et la sauvegarde d'un fichier texte dans le traitement de texte à listes chainées, exercice 3.3.

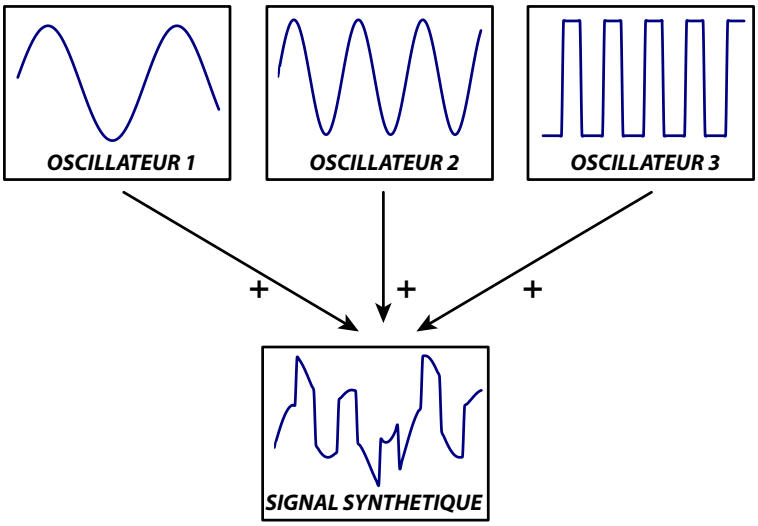
B.31 Synthétiseur

Fait travailler : Structures, Fichiers, Pointeurs, Pointeurs sur fonctions

Un synthétiseur est un instrument de musique qui génère des sons à partir d'oscillateurs de différentes formes (sinus, triangle, carré, bruit, etc ...), à des fréquences et amplitudes différentes : en faisant la somme de plusieurs oscillateurs, on obtient un son composite, "synthétique" car totalement fabriqué. Un synthétiseur ressemble donc beaucoup à un ensemble de générateurs de fonction indépendants que l'on ajoute les uns aux autres pour obtenir un nouveau signal.

On propose ici d'écrire un programme qui permettra de générer de tels sons à partir des oscillateurs de base, comme celà est représenté ci-dessous :

1. payant, Windows et Macintosh uniquement
2. gratuit, librement téléchargeable sur internet, existe pour Windows, Macintosh et Linux
3. appelé parfois "format RAW"



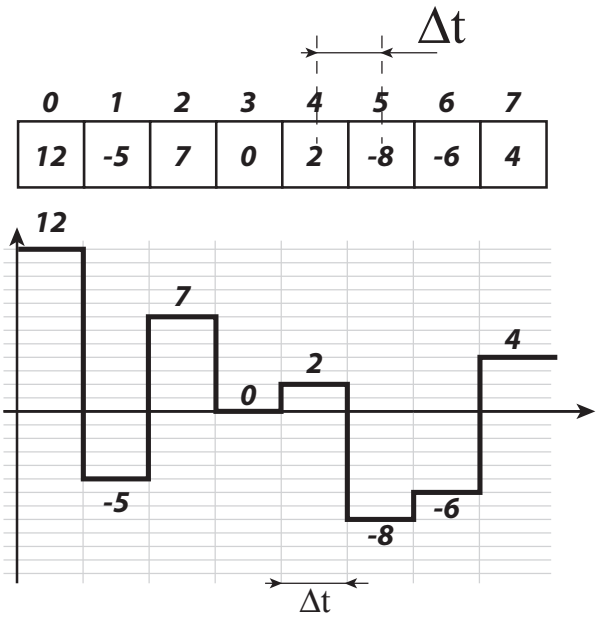
Il faut voir qu'un oscillateur est caractérisé par trois choses à la fois :

— son amplitude,
— sa fréquence,
— sa forme d'onde.

Il est facile de stocker dans la même structure l'amplitude et la fréquence : on peut choisir que chacune de ces données est de type `float` par exemple. En revanche, pour la forme d'onde, les choses sont plus complexes. Il ne s'agit pas d'une donnée simple : c'est une formule mathématique, et en langage C ce sera donc "du code". On va donc avoir besoin d'utiliser un pointeur de fonction pour gérer différentes formes d'onde. Les fonctions sur lesquelles on va être amené à pointer seront de prototype `float f(float t, float amplitude, float frequence)`, puisque pour pouvoir évaluer la fonction en chaque point, on a besoin du temps, de son amplitude et de sa fréquence. Ainsi, pour décrire chaque oscillateur, on se basera sur la structure suivante :

```
typedef struct
{
float amplitude;          /* valeur de l'amplitude */
float frequence;          /* valeur de la frequence */
float (*forme)(float t,float amplitude ,float frequence);
                          /* pointeur sur la fonction mathematique
                          de la forme d'onde */
} oscillateur;
```

De plus, le "signal synthétique" à générer sera sous la forme d'une fonction tabulée, puisqu'on ne peut en général pas la représenter sous une forme mathématique simple. On supposera que l'écart de temps Δt entre deux amplitudes successives vaut $20\mu s$, ce qui est comparable à la qualité d'un CD. Ainsi, le tableau qui contient les valeurs de ses amplitudes peut être représenté de la façon suivante :



Q-2.31.1 : Ecrivez en Langage C les fonctions mathématiques correspondant aux formes d'onde Créneau (rapport cyclique = 1), Triangle et Sinus.

Q-2.31.2 : Ecrivez un programme qui permet à un utilisateur de saisir n oscillateurs et de les placer dans un tableau.

Q-2.31.3 : Complétez ce programme pour générer le signal synthétique dans un tableau en RAM sur un temps décidé par l'utilisateur.

Note : Si le tableau contenant le signal est constitué de `short` ou de `char`, il est possible d'écouter le résultat avec un logiciel du type `SoundForge`, en précisant la fréquence d'échantillonage, qui correspondra à $1/\Delta t$, le nombre de bit par amplitude (ici respectivement 16 ou 8), et le fait que l'échantillon est mono.

Q-2.31.4 : Développez maintenant un programme pour sauvegarder les oscillateurs entrés par l'utilisateur dans un fichier. Attention aux pointeurs de fonction ...

Q-2.31.5 : Ecrivez un programme qui permet de recharger le fichier précédent en recréant le tableau d'oscillateurs, puis en affichant les données contenues dans ce tableau.

Q-2.31.6 : Ecrivez un programme qui permet de sauvegarder l'échantillon généré au format texte.

B.32 Dictionnaire

Fait travailler : Pointeurs, Structures avancées, Fichiers

Q-2.32.1 : Ajoutez le chargement et la sauvegarde du dictionnaire de l'exercice B.22 sous forme d'un fichier.