
L1 (2023-2024)

Éléments de programmation en C (LU1IN002)

TME

Semaines 1 à 11

Semaine 1 - TME

Objectifs

- Prise en main de l'environnement
 - Terminal
 - Arborescence de fichiers Unix
 - Éditeur
- Compilateur C
- Fonctions
- Alternatives

Exercices

Exercice 1 – Prise en main de l'environnement

Pour réaliser les TME de l'ue LU1IN002, vous devrez utiliser un éditeur de texte et le terminal.

Editeur

Nous vous conseillons `gedit`, pour lequel nous assurerons le support, mais vous avez la possibilité de choisir un autre éditeur dont vous maîtrisez l'utilisation.

Tous les fichiers contenant du code C que vous écrirez devront être nommés avec l'extension `.c`. Outre le fait que c'est une bonne pratique qui vous permet de repérer facilement le contenu d'un fichier, c'est aussi cette règle qui permet à l'éditeur d'identifier la coloration syntaxique (i.e., l'utilisation de couleurs particulières pour les mots réservés du langage) à appliquer.

Lancez `gedit` à partir du menu **Activités**. Comme le fait apparaître la Figure 1, l'utilisation peut être paramétrée à partir de différents endroits.

La partie **Configuration** permet de choisir le langage qui va définir la coloration syntaxique (`gedit` ne peut pour l'instant pas savoir que nous voulons écrire du code C), de définir le nombre de caractères associés à une tabulation (4 est une bonne valeur, profitez-en pour cocher l'indentation automatique) et de préciser que nous souhaitons afficher les numéros de lignes (indispensables pour traiter les messages du compilateur).

Les **Préférences**, accessibles à partir du menu `gedit`, permettent en particulier de configurer les greffons (plug-ins). Nous vous conseillons de vérifier que les greffons ci-dessous sont activés, ou de le faire si nécessaire :

- Commentateur de code
- Panneau de l'explorateur de fichiers
- Terminal intégré

L'entrée *Affichage* dans la partie **Menu** permet ensuite d'ajouter à la fenêtre un panneau latéral et un panneau inférieur. Le panneau inférieur permettra de lancer la compilation directement à partir de `gedit`. Positionné sur *Navigateur de fichiers*, le panneau latéral permettra de naviguer facilement dans l'arborescence.

En dehors du choix du langage, qui est dépendant du fichier ouvert, ces différents réglages sont mémorisés par l'éditeur.

Terminal

Le terminal vous permet de compiler et d'exécuter vos programmes, il vous permet aussi de vous déplacer dans l'arborescence de fichiers.

Ce terminal peut être ouvert :

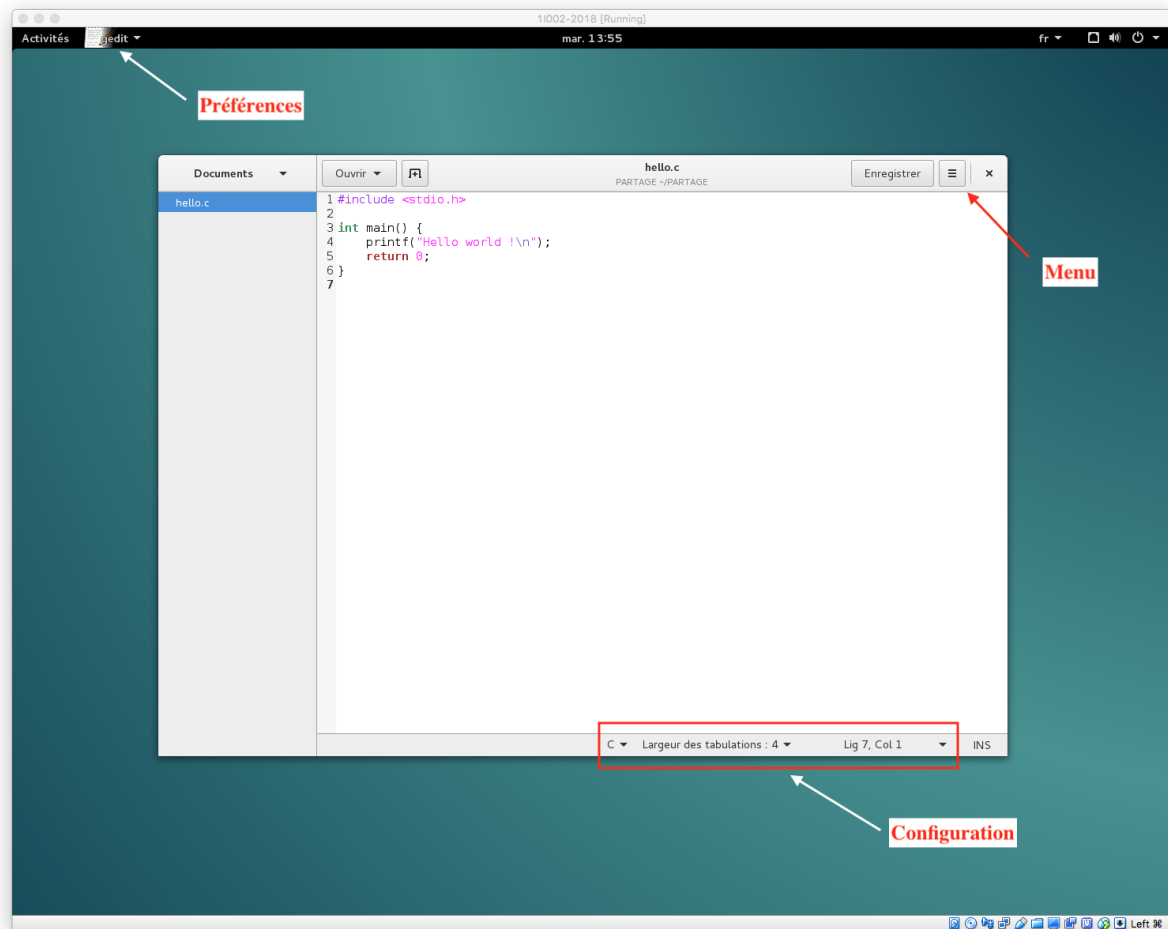


FIGURE 1 – L'éditeur gedit

- soit dans le panneau inférieur de gedit
- soit en utilisant le menu *Activités* : une zone de recherche s'ouvre dans laquelle vous devez taper (au moins les premières lettres de) `terminal`. Lorsque l'icône de l'application apparaît, vous avez la possibilité de l'ajouter aux favoris (les icônes qui s'affichent lorsque vous cliquez sur *Activités*) en cliquant dessus avec le bouton droit de la souris.

À l'ouverture du terminal, une chaîne de caractères, du type `login@ppti-14-302-01$`, appelée *prompt* ou *invite de commande* s'affiche au début de la ligne.

Cet affichage indique que l'interprète de commandes est prêt à exécuter les commandes que vous allez saisir dans le terminal (en tapant la commande, suivie d'un retour à la ligne).

Arborescence de fichiers

Le système de fichiers est organisé sous forme d'un arbre, dont le nœud racine sous Linux s'appelle `/`. Cette racine apparaît sous le nom *Ordinateur* lorsqu'on utilise une interface graphique pour naviguer dans l'arborescence.

Chaque utilisateur dispose dans cette arborescence d'un répertoire personnel, appelé *Home Directory*. C'est le répertoire dans lequel il est placé à la connexion (répertoire courant) et dans lequel il peut organiser ses données. Ce répertoire apparaît sous le nom *Dossier personnel* lorsqu'on utilise une interface graphique pour naviguer dans l'arborescence. La *Home Directory* de chaque étudiant est un répertoire dont le nom est son numéro d'étudiant. Le répertoire courant est le répertoire pris comme référence lors de l'exécution des commandes. Un répertoire ne peut pas contenir deux éléments de même nom, mais deux éléments de même nom peuvent se trouver dans deux répertoires différents, il est alors possible de les différencier.

La commande `pwd` affiche le nom du répertoire courant (celui pris en référence).

Dans un nom de répertoire (ou de fichier) :

- le `.` fait référence au répertoire courant,
- le `..` fait référence au répertoire père,
- le `~` fait référence à la (ou au) `HomeDirectory`.

La commande `cd` permet de changer le répertoire courant :

- `cd` fait de la `HomeDirectory` le répertoire courant,
- `cd .` permet de rester dans le répertoire courant (elle est donc rarement utilisée),
- `cd ..` permet de remonter au répertoire père du répertoire courant,
- `cd chemin` permet de se déplacer dans le répertoire spécifié par `chemin`. `chemin` est la suite de répertoires parcourus pour atteindre le répertoire destination, séparés par des `/`. `chemin` commence dans le répertoire courant ou à la racine de l'arborescence (le nom commence alors par le caractère `/`). Le changement de répertoire courant n'est effectué que si le répertoire cible existe,
- `cd ~/nom_rep` fait du répertoire `nom_rep` de votre `HomeDirectory` le répertoire courant (s'il existe bien sûr),
- `cd ~` est synonyme de `cd`.

La commande `ls` affiche le contenu du répertoire passé en paramètre (ou du répertoire courant s'il n'y a pas de paramètre).

La commande `mkdir nom_rep` crée le répertoire `nom_rep` dans le répertoire courant.

Question 1

Exécutez la commande `pwd` dans votre terminal. Quelle information vous donne-t-elle ?

Question 2

Donnez le chemin absolu (i.e., depuis la racine de l'arborescence) de votre *Home Directory*.

Question 3

A partir de votre `HomeDirectory` créez un répertoire `LU1IN002`, dans lequel vous créerez les répertoires `semaine1`, `semaine2` jusqu'à `semaine11`. Chaque répertoire contiendra l'ensemble des fichiers que vous aurez écrits pendant la semaine concernée et correspond au répertoire que vous devez soumettre.

Question 4

A partir de l'éditeur de votre choix créez un fichier `hello.c` contenant le code visible dans la Figure 1 (une fonction `main` qui affiche le texte `Hello world !`) que vous sauvegarderez dans le répertoire `HomeDirectory/LU1IN002/semaine1`.

Exercice 2 – Compilation et exécution dans un terminal

Contrairement à Python qui est un langage interprété, C est un langage compilé. Le code source que vous avez écrit en utilisant un éditeur de texte est analysé et traduit par un compilateur pour produire un fichier binaire dont le contenu est directement utilisable par le processeur. On parle de fichier *exécutable*. Cet exécutable n'est produit que si le compilateur n'a pas détecté d'erreur de syntaxe dans le code source.

Nous utiliserons dans cette UE le compilateur `gcc`. Pour pouvoir lancer la compilation, il est nécessaire de disposer d'un terminal par l'intermédiaire duquel nous pourrions envoyer des commandes au système.

Pour compiler notre code source, nous allons d'abord nous déplacer dans le répertoire où se trouve le fichier dans lequel nous l'avons enregistré.

Question 1

En utilisant la commande `cd`, et en vous aidant si nécessaire de la commande `ls`, placez-vous dans le répertoire où vous avez enregistré votre fichier `hello.c`.

La commande `gcc` prend en paramètre le fichier contenant le code source à compiler.

Si la compilation ne produit pas d'erreur (le *prompt* est réaffiché directement), un fichier exécutable est créé. Si la compilation produit des erreurs, celles-ci sont affichées dans le terminal et aucun exécutable n'est créé.

Question 2

Compilez votre programme, en répétant si nécessaire l'opération jusqu'à ce que toutes les erreurs aient été corrigées. Listez le contenu du répertoire courant et déduisez-en le nom de l'exécutable produit.

Puisque le système utilise un nom par défaut, tous les fichiers exécutables vont porter le même nom... C'est non seulement peu pratique, mais cela empêche aussi d'avoir plusieurs exécutables dans le même répertoire.

On peut bien sûr renommer le fichier créé (`mv ancien_nom nouveau_nom`), mais il est plus pratique de choisir le nom de l'exécutable à la compilation.

La commande `gcc` accepte un certain nombre d'options. Une option est une chaîne de caractères commençant par `-` et qui permet d'affiner le comportement de la commande. L'option `-o` prend en paramètre une chaîne de caractères qui sera utilisée pour nommer le fichier exécutable. Par exemple,

```
gcc -o mon_exec source.c
```

crée, à partir du code contenu dans `source.c`, un programme exécutable stocké dans le fichier `mon_exec`. Une bonne pratique (qui n'est pas respectée ici...) consiste à lier le nom de l'exécutable à celui du fichier contenant le code source. Par exemple, on va appeler `hello` l'exécutable produit par la compilation du fichier `hello.c`.

Question 3

Supprimez le fichier `a.out` avec la commande `rm a.out`. Recompilez votre programme en choisissant judicieusement le nom de l'exécutable.

Pour exécuter un programme, on utilise *un chemin d'accès* au fichier exécutable. Par exemple, si on est toujours dans le répertoire contenant l'exécutable, pour exécuter le programme produit par la commande `gcc` de l'exemple, on utilise la commande `./mon_exec`.

Question 4

Exécutez votre programme.

En plus de l'option `-o`, nous utiliserons l'option `-Wall` qui permet d'afficher tous les avertissements générés par le compilateur. Un avertissement cache généralement une erreur de programmation qui n'empêche pas la création de l'exécutable mais à cause de laquelle le programme ne produira pas les résultats attendus. Vous devez donc considérer que la compilation est satisfaisante lorsqu'elle n'affiche aucun avertissement ni erreur.

La commande complète à exécuter pour compiler un programme `hello.c` est donc :

```
gcc -Wall -o hello hello.c
```

Note : Lorsque votre programme utilisera les fonctions de certaines bibliothèques, il sera nécessaire d'ajouter des options de compilation :

-lm lorsque le programme inclut la bibliothèque `math.h`

-lcini lorsque le programme inclut la bibliothèque `cini.h`

Question 5

Compilez à nouveau votre programme avec l'option `-Wall`, en vérifiant qu'aucun avertissement n'apparaît.

Exercice 3 – Jeu de test

La commande `cp` permet de recopier un fichier ou tout le contenu d'un répertoire.

- `cp nom_rep/fichier1 .` recopie, dans le répertoire courant, le fichier `fichier1` se trouvant dans le répertoire `nom_rep`,
- `cp nom_rep1/fichier1 nom_rep2` recopie, dans le répertoire `nom_rep2`, le fichier `fichier1` se trouvant dans le répertoire `nom_rep1`,
- `cp nom_rep/*.c .` recopie, dans le répertoire courant, tous les fichiers, dont le nom est suffixé par `.c`, se trouvant dans le répertoire `nom_rep` et `.`

Question 1

Recopiez dans votre répertoire `LU1IN002/semaine1` le programme `jeuTest.c` suivant que vous trouverez sur la page moodle de l'UE.

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

```
int alternative(int n1, int n2, int n3) {
    int res ;

    if (n1 > 8) {
        res = 3;
    } else {
        if (n3 == 20) {
            res = 2;
        } else {
            if ((n2 >= 10) && (n3 >= 10)) {
                res = 1;
            } else {
                res = 0;
            }
        }
    }


    return res;
}


int main(){
    // A compléter
    return 0;
}
```

Question 2

Complétez la fonction `main` par un jeu de test vous permettant de tester toutes les branches de la fonction `alternative`. Vous indiquerez en commentaire, pour chaque test, quel est le cas traité .

Exercice 4 – Soumission des TP

Les exercices identifiés CodeRunner sont à soumettre sur la page moodle de l'UE. Pour chaque exercice vous devez au préalable avoir testé les fonctions et programmes hors de moodle,  **le nombre de soumissions par question étant limité à 3.**

 **Chaque étudiant doit soumettre l'ensemble des exercices sur moodle, même si vous travaillez en binôme en TP.**

Les exercices non identifiés CodeRunner ne font a priori ni l'objet d'une soumission ni celui d'une correction automatique. Vos enseignants peuvent décider de les corriger et de les noter. Dans tous les cas, ces exercices ne sont pas à négliger.

Exercice 5 – (CodeRunner) Calcul du discriminant

Etant donné qu'il s'agit du premier exercice de TP que vous devez soumettre via CodeRunner, le nombre de soumissions est illimité pour cet exercice uniquement.

Dans cet exercice, vous devez écrire plusieurs fonctions et le jeu de tests adapté à chacune d'elles. Pour vous aider dans vos tests voici quelques polynômes du second degré à coefficients entiers et leurs racines :

- $4 * x^2 + 4 * x + 1$ admet une racine double, $-0,5$
- $4 * x^2 + 6 * x + 1$ admet deux racines, $-0,191$ et $-1,309$
- $-7 * x^2 + -5 * x - 1$ n'admet pas de racine réelle.

Nous vous rappelons que :

- le discriminant (Δ) du polynôme est égal à $b^2 - 4 * a * c$,
- si $\Delta < 0$, le polynôme n'a pas de racine réelle,
- si $\Delta = 0$, le polynôme a une racine double égale à $\frac{-b}{2*a}$,
- si $\Delta > 0$, le polynôme a deux racines $\frac{-b-\sqrt{\Delta}}{2*a}$ et $\frac{-b+\sqrt{\Delta}}{2*a}$

Question 1

Écrivez une fonction `discriminant` qui renvoie la valeur du discriminant du polynôme du second degré $ax^2 + bx + c$. Les valeurs entières de `a`, `b` et `c` seront passées en paramètre de la fonction. Écrivez une fonction `main` pour tester votre fonction `discriminant`.

Question 2

Ajoutez à votre programme une fonction `afficheRacines` qui

- prend en paramètre les coefficients entiers `a`, `b` et `c` du polynôme du second degré $ax^2 + bx + c$
- affiche les racines (ou le message une racine double suivi de la racine ou pas de racine réelle suivant les cas),
- votre fonction doit obligatoirement faire appel à la fonction `discriminant`.

Pour calculer la racine carrée d'un nombre vous utiliserez la fonction `sqrt`, pour que la compilation ne pose pas de problème, vous devrez ajouter la ligne `#include <math.h>` juste sous la ligne `#include <stdio.h>`. N'oubliez pas d'ajouter l'option `-lm` à votre commande de compilation.

Question 3

Complétez la fonction `main` pour tester votre fonction `afficheRacines` en définissant un jeu de tests.

Exercice 6 – Finir le TD

Faites les exercices non terminés lors de la séance de TD.

Exercice 7 – (CodeRunner) Signe d'un produit

Question 1

Écrivez la fonction `signeProduit` qui prend en paramètres deux entiers et qui, sans calculer le produit, renvoie 0 si le produit est nul, -1 s'il est négatif et 1 sinon.

Question 2

Écrivez une fonction `main` qui permet de tester la fonction `signeProduit` en utilisant la fonction `assert`.

Exercice 8 – (CodeRunner) Visite de la Tour de Londres

Si vous achetez vos billets en ligne, les tarifs pour visiter la *Tour de Londres* sont les suivants (tarifs du 12 juin 2018) :

- adulte : 22,7 £
- enfant (entre 5 et 15 ans) : 10,75 £
- enfant de moins de 5 ans : gratuit
- famille (2 adultes et 3 enfants au maximum) : 57,80 £

Voici quelques résultats attendus qui vous permettront de tester votre programme :

- 2 adultes et 3 enfants d'au moins 5 ans paient 57,80 £
- 2 adultes et 2 enfants d'au moins 5 ans paient 57,80 £
- 2 adultes et 1 enfant d'au moins 5 ans paient 56,15 £(le tarif famille n'est pas intéressant)
- 1 adulte et 3 enfants d'au moins 5 ans paient 54,95 £(le tarif famille n'est pas intéressant)

Question 1

Écrivez une fonction `prixEntree` qui prend en paramètre le nombre d'adultes, d'enfants d'au moins 5 ans et qui renvoie la somme à payer. Pour simplifier, une seule entrée *famille* est possible (même si le nombre d'adultes et enfants pourrait en permettre plus). Les personnes non comprises dans l'entrée famille paient en fonction de leur âge.

Pour savoir s'il est intéressant d'appliquer le tarif famille votre fonction devra calculer le prix à payer sans prendre en compte le tarif famille (chacun paie en fonction de son âge) et celui à payer si on prend en compte une entrée famille. La fonction renvoie la plus petite des deux valeurs.

Question 2

Écrivez la fonction `main` qui permet de tester la fonction `prixEntree` en affichant la valeur renvoyée.

Semaine 2 - TME

Objectifs

- Prise en main de l'environnement graphique
- Boucles

Exercices

Exercice 9 – Initiation à la bibliothèque graphique

Dans cet exercice et dans les suivants, vous allez utiliser la bibliothèque graphique `cini.h` pour écrire des programmes de dessin. La bibliothèque graphique définit un ensemble de fonctions pour créer une fenêtre graphique et dessiner dedans. Dans ce TP, nous utiliserons *uniquement les fonctions suivantes* :

- `void CINI_open_window(int width, int height, char* title);`
crée une fenêtre de fond noir, de largeur `width` et de hauteur `height`, ayant pour titre `title`. **Attention**, la fenêtre graphique doit être créée une seule fois, avant tout affichage.
- `void CINI_fill_window(char* color);`
remplit la fenêtre précédemment créée de la couleur passée en paramètre.
- `void CINI_draw_pixel(int x, int y, char* color);`
affiche le point de coordonnées `(x, y)` de couleur `color`. **Attention**, le point de coordonnées `(0, 0)` correspond au coin supérieur gauche de la fenêtre et celui de coordonnées `(width-1, height-1)` au coin inférieur droit de la fenêtre.
- `void CINI_loop();`
met le programme en pause jusqu'à la fermeture de la fenêtre (ou la frappe de la touche ESC). Sans cette fonction, la fenêtre graphique ayant été créée par le programme, elle disparaît avec la terminaison (quasi-instantanée) de celui-ci. **Attention**, les instructions se trouvant **après** l'instruction `CINI_loop();` ne seront exécutées qu'une fois la fenêtre graphique fermée.

Question 1

Recopiez le fichier `exemple_graphique.c`. Ce fichier correspond à un programme faisant appel aux fonctions de la bibliothèque `cini`. Vous remarquerez que la directive `#include <cini.h>` est nécessaire pour accéder à ces fonctions et que les couleurs d'affichage sont données en anglais.

Compilez le fichier avec l'option `-lcini` et exécutez le programme obtenu. Vous devez voir une fenêtre de 400 pixels de large sur 300 de haut s'ouvrir et trois points blancs s'y afficher.

Attention Pour les questions suivantes, vous n'avez pas le droit d'utiliser d'autres fonctions que celles présentées dans ce sujet. Pour afficher une ligne, vous devrez donc afficher chacun de ses points.

Question 2

Écrivez la fonction `diagonale` qui prend en paramètre une coordonnée `x` et qui affiche la diagonale reliant le point de coordonnées `(0, 0)` au point de coordonnées `(x, x)`. Nous supposons qu'une fenêtre graphique a bien été créée et que le point de coordonnées `(x, x)` appartient à la fenêtre.

Question 3

Écrivez la fonction `main` permettant de tester la fonction `diagonale`.

Exercice 10 – Carrément graphique

Question 1

Écrivez une fonction `carre` qui prend une longueur entière en paramètre et qui dessine un carré de coin supérieur gauche le point $(0, 0)$ et dont le côté est la longueur passée en paramètre. Les côtés du carré sont parallèles aux côtés de la fenêtre graphique. Le côté supérieur doit être tracé en bleu (`blue`), le côté inférieur en vert (`green`), le côté gauche en rouge (`red`) et le côté droit en noir (`black`). Nous ferons l'hypothèse que la fenêtre graphique est déjà créée et que le carré peut être dessiné dans cette fenêtre. Ecrivez la fonction `main` permettant de tester votre fonction, faites attention à la couleur de remplissage de la fenêtre !

Question 2

Si ce n'est pas déjà le cas, modifiez votre fonction `carre` pour qu'elle ne contienne qu'une boucle. Pour vous aider, considérez un point de coordonnées (x, y) appartenant au côté supérieur du carré et déterminez les coordonnées d'un point de chacun des autres côtés en fonction de x, y et la longueur.

Question 3

Nous souhaitons maintenant que le carré puisse être placé n'importe où dans la fenêtre graphique (ses côtés étant toujours parallèles à ceux de la fenêtre). Modifiez la fonction `carre` pour qu'en plus de la longueur du carré elle prenne en paramètres les coordonnées de son coin supérieur gauche. Modifiez la fonction `main` pour pouvoir tester la nouvelle version de cette fonction.

Question 4

Écrivez la fonction `carres_remontant` qui prend en paramètres une longueur et les coordonnées d'un point et qui affiche :

- le carré dont la longueur et les coordonnées du coin en haut à gauche sont passées en paramètre,
- tous les carrés de même dimension obtenus par une translation de 20 pixels vers la gauche et de 20 pixels vers le haut, comme sur la figure 2 (le premier carré dessiné est celui du centre de la fenêtre). Les carrés dont un des points est en dehors de la fenêtre ne seront pas dessinés.

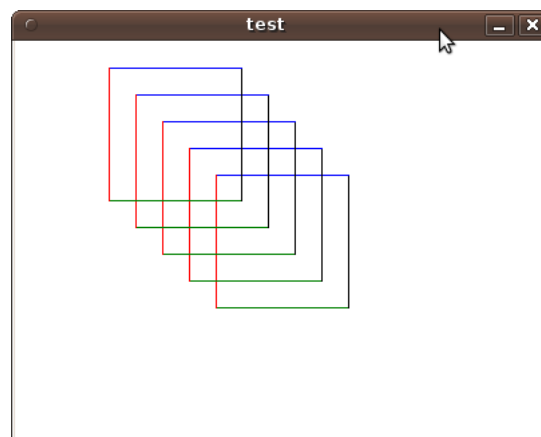


FIGURE 2 – Translation carrés

Cette nouvelle fonction doit faire appel à la fonction `carre`. Modifiez la fonction `main` pour pouvoir tester cette nouvelle fonction.

Exercice 11 – (CodeRunner) Propagation épidémie

Des chercheurs s'intéressent à la vitesse de propagation d'une épidémie dans une ville alors qu'un seul individu est initialement malade. Sachant qu'une seule personne contamine x nouvelles personnes chaque jour, nous voulons savoir en combien de jours un certain pourcentage de la population de la ville sera contaminée. Pour simplifier les calculs, une personne contaminée le reste et continue donc à contaminer d'autres personnes tous les jours suivant sa contamination.

Question 1

Ecrivez et testez la fonction `jours` qui prend en paramètre deux entiers, le nombre de personnes contaminées par une même personne chaque jour et la population totale de la ville (on suppose qu'un nombre entier est suffisant) ainsi qu'un réel, compris entre 0.0 et 100.0, correspondant au pourcentage de la population qui « doit » être infecté. La fonction renvoie le nombre de jours au bout duquel le pourcentage, passé en paramètre, de la population de la ville est contaminé. N'oubliez pas qu'initialement une seule personne est contaminée.

Voici les résultats attendus lorsque la population totale est de 10000 habitants, le nombre de personnes contaminées par une même personne est de 5 :

- nombre de jours pour que 100.00 pourcent de la population soit contaminée = 6
- nombre de jours pour que 50.00 pourcent de la population soit contaminée = 5
- nombre de jours pour que 25.00 pourcent de la population soit contaminée = 5
- nombre de jours pour que 10.00 pourcent de la population soit contaminée = 4

Question 2

Les chercheurs souhaitent maintenant savoir quel pourcentage de la population sera contaminé au bout d'un certain nombre de jours. Ecrivez et testez la fonction `pourcentage` qui prend trois entiers en paramètre, le nombre de personnes contaminées par une même personne chaque jour, la population totale de la ville et le nombre de jours étudiés. La fonction renvoie le pourcentage de la population contaminée au bout du nombre de jours donné.

Voici les résultats attendus lorsque la population totale est de 10000 habitants, le nombre de personnes contaminées par une même personne est de 5 :

- pourcentage population contaminée au bout de 2 jours = 0.36
- pourcentage population contaminée au bout de 3 jours = 2.16
- pourcentage population contaminée au bout de 4 jours = 12.96
- pourcentage population contaminée au bout de 5 jours = 77.76
- pourcentage population contaminée au bout de 6 jours = 100.00

Exercice 12 – Droite et points

Question 1

Ecrivez une fonction `position` qui prend en paramètres les entiers a et b qui caractérisent la droite d'équation $y=ax+b$, et les coordonnées entières d'un point. La fonction renvoie -1 si le point est en-dessous de la droite, 0 s'il appartient à la droite et 1 s'il est au-dessus de la droite.

La figure 3 vous rappelle quel est le repère associé à une fenêtre graphique et vous montre où se trouvent les points au-dessus et en-dessous de la droite.

Question 2

Ecrivez une fonction `affiche` qui prend en paramètres les entiers a et b qui caractérisent la droite d'équation $y=ax+b$, et la hauteur et la largeur d'une fenêtre graphique. La fonction doit afficher tous les points de la fenêtre en respectant les couleurs suivantes :

- noire pour les points se trouvant sur la droite $y=ax+b$,

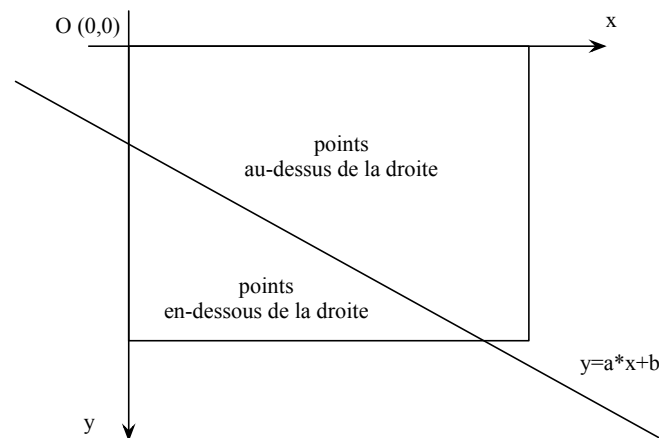


FIGURE 3 – Repère associé à la fenêtre graphique

- rouge pour les points se trouvant au-dessus de la droite $y=ax+b$,
- bleue pour les points se trouvant en-dessous de la droite $y=ax+b$.

Question 3

Ecrivez le programme complet pour pouvoir tester vos fonctions. N'oubliez pas de créer la fenêtre graphique et d'attendre sa fermeture à la fin du programme.

Exercice 13 – Finir le TD

Faites les exercices non terminés lors de la séance de TD.

Exercice 14 – (CodeRunner) Visite de la *Tour de Londres*

Nous vous rappelons les tarifs pour la visite de la tour de Londres si vous achetez vos billets en ligne (tarifs du 12 juin 2018) :

- adulte : 22,7 £
- enfant (entre 5 et 15 ans) : 10,75 £
- enfant de moins de 5 ans : gratuit
- famille (2 adultes et 3 enfants au maximum) : 57,80 £

Voici quelques résultats attendus :

- 2 adultes et 1 enfant d'au moins 5 ans paient 56,15 £(le tarif famille n'est pas intéressant)
- 2 adultes et 2 enfants d'au moins 5 ans paient 57,80 £
- 2 adultes et 3 enfants d'au moins 5 ans paient 57,80 £
- 6 adultes et 3 enfants d'au moins 5 ans paient 148,60 £(1 tarif famille + 4 adultes)
- 1 adulte et 3 enfants d'au moins 5 ans paient 54,95 £(le tarif famille n'est pas intéressant)
- 5 adultes et 7 enfants d'au moins 5 ans paient 149,05 £(2 tarifs famille + 1 adulte + 1 enfant)
- 6 adultes et 8 enfants d'au moins 5 ans paient 173,4 £(3 tarifs famille)
- 10 adultes paient 227 £
- 4 enfants paient 43 £

Question 1

Nous souhaitons maintenant appliquer le tarif *famille* autant de fois que possible. Écrivez une nouvelle fonction

`prixEntree` qui prend en paramètres le nombre d'adultes et d'enfants de plus de cinq ans et qui renvoie la somme à payer. Les personnes non comprises dans les entrées *famille* paient en fonction de leur âge.

Question 2

Écrivez la fonction `main` qui permet de tester votre fonction `prixEntree`.

Exercice 15 – Triangles en spirale

A partir de cet exercice vous pouvez utiliser la fonction

```
void CINI_draw_line(int x_1, int y_1, int x_2, int y_2, char* color);
```

qui trace, dans la couleur passée en paramètre, le segment de droite reliant les deux points de coordonnées (x_1, y_1) et (x_2, y_2) .

Question 1

Écrivez une fonction `triangles` qui prend en paramètres 2 entiers correspondant à la largeur et la hauteur d'une fenêtre graphique (fenêtre supposée déjà ouverte) et qui y trace un triangle qui remplit l'intégralité de la fenêtre graphique (comme illustré par la figure 4). Vous utiliserez une couleur différente pour chacun des trois côtés du triangle. Vous écrirez le programme permettant de tester votre fonction.

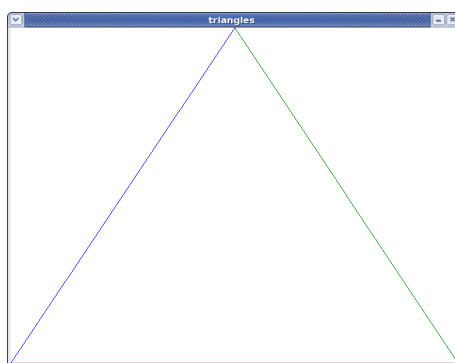


FIGURE 4 – Premier triangle

Question 2

Modifiez la fonction `triangles` pour qu'elle affiche un deuxième triangle décalé par rapport au premier comme illustré par la figure 5.

Indications : les coordonnées des sommets du second triangle peuvent être calculées facilement à partir des coordonnées des sommets du premier triangle.

Si le premier triangle a pour sommets les points A , B et C de coordonnées respectives (x_A, y_A) , (x_B, y_B) et (x_C, y_C) , alors le sommet A' du second triangle, qui est entre les points A et B a pour coordonnées $x_{A'} = \frac{x_B + 9x_A}{10}$ et, de la même manière, $y_{A'} = \frac{y_B + 9y_A}{10}$. Le calcul des coordonnées des points B' et C' se fait de façon similaire.

Faites le calcul sur papier pour vous en persuader, en considérant séparément les abscisses et les ordonnées !

Pensez à utiliser des variables temporaires pour calculer ces nouvelles coordonnées sans écraser les précédentes et n'oubliez pas de tracer les segments $[A'B']$ (resp. $[B'C']$ et $[C'A']$) de la même couleur que les segments $[AB]$ (resp. $[BC]$ et $[CA]$).

Question 3

Modifiez votre fonction `triangles` pour qu'elle affiche 10 triangles. Entre le tracé de deux triangles vous appellerez la fonction `CINI_loop_until_keyup` qui bloquera le programme tant que vous n'aurez pas tapé sur une touche.

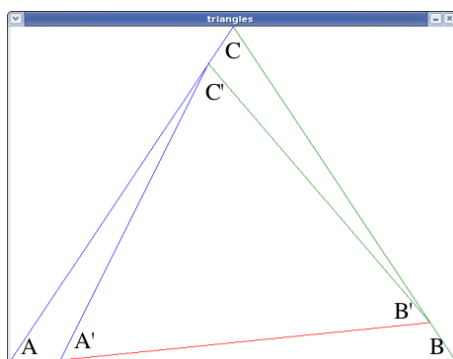


FIGURE 5 – Second triangle

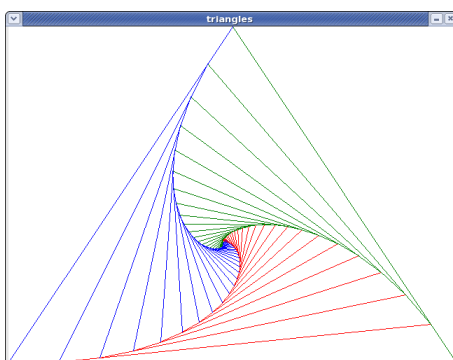


FIGURE 6 – Résultat final

Question 4

Modifiez votre fonction `triangles` pour qu'elle continue à tracer des triangles jusqu'à ce que la distance entre les points A et B soit inférieure à une valeur `epsilon` passée en paramètre de la fonction. Vous allez alors obtenir un ensemble de triangles inscrits les uns dans les autres, jusqu'à donner l'impression d'une spirale comme sur la figure 6.

Indication : Nous vous rappelons que le carré de la distance entre deux points A et B de coordonnées respectives (x_A, y_A) et (x_B, y_B) est égal à $(x_A - x_B)^2 + (y_A - y_B)^2$.

Semaine 3 - TME

Objectifs

- Adresse
- Pointeur
- Saisie d'une valeur au clavier (fonction `scanf`)
- Générateur de nombres aléatoires

Exercices

Exercice 16 – Qu'est-ce qu'un pointeur ?

Recopiez le fichier `pointeurs.c` contenant les instructions suivantes :

```
#include <stdio.h>

void ma_fonction(int v1, int v2) {
    int a;
    int b;

    a = v1;
    b = a + v2;
    a = 2 * b;
    printf("a=%d, b=%d\n", a, b);
}

int main() {
    ma_fonction(10, 20);
}
```

Question 1

Lors de l'exécution du programme, l'affichage obtenu est `a=60, b=30`. Modifiez les instructions pour que les modifications de valeurs effectuées à partir des variables `a` et `b` soient effectuées sur le même emplacement mémoire (on devra alors obtenir l'affichage `a=60, b=60`). Votre solution doit bien sûr utiliser un pointeur ! Vous utiliserez le débogueur `ddd` pour vérifier que vos deux variables mènent bien au même emplacement mémoire.

Exercice 17 – (CodeRunner) Compter les valeurs positives, négatives et les zéros

Cet exercice utilise le générateur de nombres pseudo-aléatoires du langage C. La génération de nombres pseudo-aléatoires se fait de la façon suivante :

- inclusion des bibliothèques `stdlib.h` et `time.h`, la première pour avoir accès au générateur et la seconde pour l'initialisation de ce dernier,
- initialisation du générateur pour qu'il ne produise pas la même suite de nombres à chaque exécution (instruction `srand(time(NULL));`). Cette initialisation est à faire une seule fois dans le programme, en général en début de la fonction `main`. Lors de la mise au point de votre programme il peut être judicieux de mettre cette instruction en commentaire, ceci vous permettra de tester votre programme sur la même suite de valeurs à chaque exécution.

- récupérer un nombre généré aléatoirement par l'appel `rand()` qui renvoie un entier compris entre 0 et `RAND_MAX` (constante définie dans `stdlib.h`).

Recopiez le fichier `pnz.c`

Question 1

Dans le fichier que vous avez récupéré, la fonction `valeur_aleatoire` renvoie un entier compris entre 0 et `RAND_MAX`, modifiez cette fonction pour que l'entier renvoyé soit compris entre les valeurs des paramètres `min` et `max` comprises (nous faisons l'hypothèse que $\min \leq \max$). Complétez la fonction `main` pour pouvoir tester votre fonction `valeur_aleatoire` en tirant et affichant `NB_VALEURS` entiers choisis aléatoirement entre `VMIN` et `VMAX`.

Nous souhaitons maintenant écrire un programme qui, en plus de tirer aléatoirement `NB_VALEURS` entiers compris entre `VMIN` et `VMAX`, affiche le nombre de valeurs négatives, le nombre de valeurs positives et le nombre de zéro choisis aléatoirement. La fonction `pos_neg_zero` appelée après le tirage de chaque valeur doit effectuer la mise à jour des compteurs.

Question 2

Écrivez une fonction `pos_neg_zero`.

Question 3

Complétez la fonction `main` qui fait les tirages aléatoires et les appels à `pos_neg_zero`. Vous afficherez les compteurs obtenus.

Exercice 18 – (CodeRunner) Tri de trois valeurs

Dans cet exercice, nous allons ranger par ordre croissant trois valeurs entières.

Question 1

Écrivez et testez la fonction `echange` qui permet d'échanger la valeur de deux variables entières. Soient 2 variables `a` et `b`, si leur valeur initiale est respectivement 1 et 2, après l'appel à la fonction `echange`, la valeur de `a` doit être 2 et celle de `b` 1.

Question 2

Écrivez et testez la fonction `tri` qui prend deux pointeurs sur entier en paramètre et qui "met" dans le premier paramètre le plus petit des deux entiers et le plus grand dans la deuxième. Cette fonction doit faire appel à la fonction `echange`.

Question 3

Écrivez et testez la fonction `tri_3` qui prend trois pointeurs sur entier en paramètre et qui "met" dans le premier paramètre le plus petit des trois entiers, le plus grand dans le dernier et le troisième entier dans le deuxième paramètre. Cette fonction doit faire appel à la fonction `tri`.

Exercice 19 – (CodeRunner) Calcul de la moyenne, du minimum et du maximum

Recopiez le fichier `min_max_moy.c`. Les primitives `#define` définissent des valeurs vous permettant de tester votre programme. Vous devez changer ces valeurs pour faire d'autres tests. Par contre, sauf indication contraire, vous ne devez pas modifier les instructions qui vous sont données (en particulier vous ne devez pas modifier les types de retour des fonctions). Vous devez juste compléter le fichier.

Question 1

Écrivez et testez la fonction `min_max` qui prend trois paramètres dont un entier. Les deux autres paramètres doivent

permettre d'accéder à deux valeurs entières représentant un maximum et un minimum. La fonction met à jour ces deux valeurs en fonction de celle du paramètre entier (si ce dernier est plus petit que le minimum, il faut changer la valeur du minimum, on a un raisonnement similaire si l'entier est plus grand que le maximum).

Question 2

Nous souhaitons calculer le plus grand, le plus petit et la moyenne de quatre entiers en ne prenant en compte les valeurs que tant qu'elles sont strictement positives. Écrivez et testez la fonction `stats` qui prend sept paramètres dont quatre entiers. La fonction doit déterminer le plus grand, le plus petit et la moyenne des quatre entiers passés en paramètres. Attention, si un entier est négatif ou nul, lui-même et les entiers suivants ne sont pas à prendre en compte dans les calculs. Si le premier entier est négatif ou nul, le minimum, le maximum et la moyenne seront égaux à -1. Votre fonction doit faire appel à la fonction `min_max` dès que nécessaire.

Voici quelques exemples de résultats attendus (vi correspond au ième paramètre entier) :

`v1=2, v2=7, v3=5, v4=9, maximum = 9, minimum = 2, moyenne = 5.75`

`v1=2, v2=7, v3=-5, v4=-9, maximum = 7, minimum = 2, moyenne = 4.5` (seules les valeurs 2 et 7 sont prises en compte)

`v1=2, v2=7, v3=-5, v4 =9, maximum = 7, minimum = 2, moyenne = 4.5` (seules les valeurs 2 et 7 sont prises en compte)

`v1=2, v2=-7, v3=-5, v4 =9, maximum = 2, minimum = 2, moyenne = 2.0` (seule la valeur 2 est prise en compte)

`v1=-2, v2=-7, v3=-5, v4 =9, maximum = -1, minimum = -1, moyenne = -1.0` (aucune valeur prise en compte)

Question 3

Modifiez la fonction `main` pour que le programme affiche le plus grand, le plus petit et la moyenne des valeurs définies par `VAL1`, `VAL2`, `VAL3` et `VAL4`.

Exercice 20 – (CodeRunner) Calcul des Racines d'un polynôme du second degré

Dans cet exercice, vous devez écrire plusieurs fonctions et le jeu de tests adapté à chacune d'elles. Pour vous aider dans vos tests voici quelques polynômes du second degré à coefficients entiers et leurs racines :

- $4 * x^2 + 4 * x + 1$ admet une racine double, $-0, 5$
- $4 * x^2 + 6 * x + 1$ admet deux racines, $-0, 191$ et $-1, 309$
- $-7 * x^2 + -5 * x - 1$ n'admet pas de racine réelle.

Nous vous rappelons que :

- le discriminant (Δ) du polynôme est égal à $b^2 - 4 * a * c$,
- si $\Delta < 0$, le polynôme n'a pas de racine réelle,
- si $\Delta = 0$, le polynôme a une racine double égale à $\frac{-b}{2*a}$,
- si $\Delta > 0$, le polynôme a deux racines $\frac{-b-\sqrt{\Delta}}{2*a}$ et $\frac{-b+\sqrt{\Delta}}{2*a}$

Recopiez le fichier `racines_poly.c`

Question 1

Écrivez et testez la fonction `nb_racines` qui renvoie le nombre de racines réelles du polynôme du second degré $a * x^2 + b * x + c$ à coefficients entiers (a est bien sûr non nul). La fonction prend en paramètres les 3 coefficients (entiers) du polynôme.

Question 2

Écrivez et testez la fonction `nb_racines_delta` qui en plus de renvoyer le nombre de racines réelles, permet de récupérer la valeur du discriminant. Cette nouvelle fonction s'inspirera bien sûr de la fonction `nb_racines`.

Question 3

Écrivez et testez la fonction `racines` qui renvoie le nombre de racines réelles du polynôme du second degré $a * x^2 + b * x + c$.

$x^2 + b * x + c$ et permet de récupérer la valeur des racines si elles existent. La fonction prend en paramètres les 3 coefficients (entiers) du polynôme et doit faire appel à la fonction `nb_racines_delta`. Vous devez bien sûr ajouter d'autres paramètres à la fonction.

Exercice 21 – Pierre-Feuille-Ciseaux

L'objectif de cet exercice est de programmer le célèbre jeu Pierre-Feuille-Ciseaux (ou chifoumi). Il s'agit d'un jeu à deux joueurs, chacun des deux choisit soit pierre, soit feuille, soit ciseaux. Les deux joueurs annoncent simultanément leur choix, un ensemble de règles permet de déterminer qui a gagné :

- Les ciseaux coupent la feuille (les ciseaux gagnent).
- La pierre émousse les ciseaux (la pierre gagne).
- La feuille enveloppe la pierre (la feuille gagne).

Chaque objet en bat un autre, fait match nul contre lui-même et est battu par le troisième.

Le programme que vous allez écrire va permettre à un joueur humain d'affronter l'ordinateur. Le choix de l'ordinateur sera réalisé en utilisant le générateur pseudo-aléatoire, le choix du joueur sera saisi au clavier.

La pierre, la feuille et les ciseaux seront respectivement représentés par des entiers définis en utilisant la directive **#define**) en respectant les propriétés suivantes :

- `FEUILLE = PIERRE + 1`
- `CISEAUX = FEUILLE + 1`

Les valeurs associées à `PIERRE`, `FEUILLE` et `CISEAUX` sont donc trois entiers consécutifs.

Pour réaliser la saisie, il faut utiliser la fonction `scanf` qui prend en paramètres une chaîne de caractères qui indique le type de la variable à initialiser (format donné de façon similaire à celui de la fonction `printf`) et l'adresse de la variable qui sera initialisée par les caractères lus. Pour pouvoir utiliser cette fonction, vous devez inclure la librairie `stdio.h`.

L'instruction suivante permet d'initialiser la variable `i` (déclarée par l'instruction `int i;`) avec la valeur entière saisie au clavier :

```
scanf("%d", &i);
```

Attention, dans cette UE, la chaîne de caractères, indiquant le type de la variable, ne doit contenir **rien d'autre** que le format.

Recopiez le fichier `pierre_feuille_ciseaux.c` que vous complèterez.

Question 1

Complétez la fonction `choix_ordinateur` qui ne prend pas de paramètre et qui renvoie le choix de l'ordinateur tiré aléatoirement. Votre fonction doit donc renvoyer un entier compris entre 1 et 3 choisi aléatoirement.

Question 2

Complétez la fonction `choix_joueur` qui demande au joueur de saisir au clavier une valeur entière comprise entre 1 et 3 et la renvoie. Si au bout de 3 essais, le joueur n'a pas saisi de valeur correcte, cette dernière sera tirée au sort (valeur choisie aléatoirement).

Question 3

Complétez la fonction `score` qui ne renvoie rien mais qui prend en paramètres les choix du joueur et de l'ordinateur et met à jour leurs scores respectifs (la fonction doit donc avoir 4 paramètres). Le score du joueur (resp. ordinateur) est augmenté de 1 si son choix bat celui de l'ordinateur (resp. joueur). Aucun score n'est modifié si le joueur et l'ordinateur ont fait le même choix.

Question 4

Complétez la fonction `jeu` qui représente une partie entre le joueur et l'ordinateur. Le gagnant est celui qui arrive le

premier à trois points. Votre fonction devra faire des affichages pour pouvoir suivre le déroulement de la partie et identifier le gagnant.

Exercice 22 – Bowling

Dans cet exercice nous allons calculer le score d'un joueur de bowling. Voici les règles que nous appliquons :

- une partie se joue en 10 tours,
- à chaque début de tour 10 quilles sont placées sur la piste,
- à chaque tour le joueur a 2 lancers pour renverser les 10 quilles,
- le nombre de points marqués par lancer est le nombre de quilles renversées,
- si les 10 quilles tombent au premier lancer, le joueur réalise un strike, il marque 10 points (les 10 quilles renversées) + les points des 2 lancers suivants (le nombre de points marqués pour le tour dépend donc des deux lancers suivants),
- si les 10 quilles tombent en deux lancers, le joueur réalise un spare, il marque 10 points (les 10 quilles renversées) + les points du lancer suivant (le nombre de points marqués pour le tour dépend donc du lancer suivant),
- si le joueur réalise un spare (resp. un strike) lors du dixième tour, il bénéficie de 1 (resp. 2) lancer(s) supplémentaire(s). Lors de ces tours supplémentaires, on ne fait que mettre à jour le score en fonction des spare ou strikes déjà réalisés.

Dans l'exemple suivant, nous nous limitons à une partie en 5 tours.

TOUR 1

Quilles renversees : 6

Quilles renversees : 3

Score apres tour 1 : 9

TOUR 2

Quilles renversees : 5

Quilles renversees : 5

Score apres tour 2 : 19

score incomplet : spare en cours vrai score = $19 + 7 = 26$ (7 quilles renversées au lancer suivant)

TOUR 3

Quilles renversees : 7

Quilles renversees : 3

Score apres tour 3 : 36

score incomplet : spare en cours vrai score = $36 + 0 = 36$ (aucune quille renversée au lancer suivant)

TOUR 4

Quilles renversees : 0

Quilles renversees : 2

Score apres tour 4 : 38

TOUR 5

Quilles renversees : 10

Score apres tour 5 : 48

score incomplet : strike en cours vrai score = $48 + 4 + 6 = 58$ (4 et 6 quilles renversées lors du tour supp.)

TOUR SUPPLEMENTAIRE

Quilles renversees : 4

Quilles renversees : 6

Score apres tour supplementaire : 58

Le score maximum est de 300 points. Dans ce cas, le joueur ne fait que des strikes, il marque 10 points par tour plus les 20 points des deux lancers suivants, ce qui fait 10 tours à 30 points, donc 300 points.

Si un joueur renverse 5 quilles à chaque lancer, son score sera de 150 points. Il réalise alors 10 spares. Aux 10 quilles renversées à chaque tour s'ajoutent les 5 quilles renversées au premier lancer du tour suivant. Ce qui fait 10 tours à 15 points, donc 150 points.

Et bien sûr, un joueur qui ne renverse aucune quille à chaque lancer aura un score de 0 point.

Même si ce n'est pas explicitement demandé, vous devrez bien sûr tester vos fonctions indépendamment les unes des autres.

Recopiez le fichier `bowling.c` que vous complèterez.

Question 1

Complétez la fonction `lancer` qui prend en paramètre un entier et qui renvoie la première valeur entière saisie par l'utilisateur supérieure ou égale à 0 et inférieure ou égale au paramètre. L'entier passé en paramètre représente le nombre de quilles encore debout sur la piste (10 pour le premier lancer d'un tour), la fonction renvoie le nombre de quilles renversées lors d'un lancer.

Question 2

Nous nous intéressons aux points rapportés par un lancer. Complétez la fonction `score` qui permet de mettre à jour le score en fonction du nombre de quilles renversées lors d'un lancer et des informations suivantes :

- un spare a été réalisé au lancer précédent,
- un strike a été réalisé au lancer précédent,
- un strike a été réalisé deux lancers plus tôt.

Chacune de ces informations peut-être représentée par un entier qui vaudra 0 (l'événement n'a pas eu lieu) ou 1 (l'événement a eu lieu).

Une fois le score mis à jour, les informations sur les spare et strike doivent aussi être mises à jour pour pouvoir être prises en compte lors du calcul du score du lancer suivant.

Question 3

Nous nous intéressons aux points rapportés par un tour (donc deux lancers au maximum). Complétez la fonction `tour` qui représente un tour de jeu. Cette fonction devra faire appel à la fonction `score` pour mettre à jour le score après chaque lancer. Pour chaque lancer la fonction devra :

- récupérer le lancer du joueur,
- mettre à jour le score,
- mettre à jour les variables associées aux événements spare et strike si nécessaire,
- mettre à jour le nombre de quilles encore debout pour le lancer suivant.

Renouveler ces opérations pour le deuxième lancer si nécessaire.

Question 4

Complétez maintenant la fonction `jeu` qui représente les 10 tours d'une partie et renvoie le score final. N'oubliez pas de prendre en compte les lancers supplémentaires lorsque nécessaire.

Question 5

Remplacez maintenant la fonction `lancer` par la fonction `lancer_aleatoire` qui prend aussi un paramètre entier et qui renvoie un entier choisi aléatoirement entre 0 et la paramètre compris. Testez votre programme avec cette nouvelle fonction.

Semaine 4 - TME

Objectifs

- Tableaux : opérations de base
- Tableau en valeur de retour
- Utilisation de l'outil de debug DDD

Une de deux séances de TP sera consacrée à un contrôle individuel de 45 minutes. L'évaluation sera réalisée sur moodle en utilisant CodeRunner.

Exercices

Exercice 23 – (CodeRunner) On prend la température

Nous considérons un tableau `tab` de 31 cases stockant les températures moyennes de chaque jour du mois écoulé (supposé faire 31 jours). Pour simuler la collecte des données, chaque case est initialisée avec un flottant, obtenu par tirage aléatoire d'une valeur entière entre -200 et 300 qui est ensuite divisée par 10.

Question 1

Écrivez une fonction `init_temp` qui initialise le tableau de températures.

Écrivez une fonction d'affichage et une fonction `main` pour tester votre initialisation.

Question 2

Écrivez une fonction `moy_temp` qui calcule et renvoie la température moyenne sur le mois. Peut-on facilement vérifier que la valeur calculée est correcte ?

Question 3

Écrivez une fonction qui compte le nombre de jours dans le mois où la température a été strictement négative et qui renvoie la température moyenne sur ces journées.

Lorsque la température est toujours restée positive, le programme de test devra afficher un message : *"Aucune temperature au-dessous de zero."*

Exercice 24 – (CodeRunner) Insertion d'un élément dans un tableau trié

L'objectif de cet exercice est d'écrire un programme qui insère des éléments dans un tableau tout en les triant.

Nous distinguerons la taille `taille` du tableau (c'est-à-dire le nombre maximum d'éléments qu'il peut contenir) du nombre `nbEl` d'éléments déjà insérés dans le tableau. Au début, le tableau est vide (`nbEl = 0`).

Avant d'insérer une valeur dans le tableau, il faut d'abord s'assurer que celui-ci n'est pas plein (`nbEl < taille`). Dans ce cas, le programme va commencer par rechercher la position où insérer la valeur dans le tableau : avant le premier élément qui lui est supérieur, ou à la fin s'il n'y en a pas.

Question 1

Écrivez une fonction `indiceInsert` qui, étant donnés un tableau d'entiers `tab` de taille `taille` contenant `nbEl` éléments triés par ordre croissant et un entier `el`, renvoie l'indice auquel `el` doit être inséré pour que le tableau reste trié.

Si le tableau est plein ou si l'élément est déjà présent, la fonction renverra la valeur -1.

Une fois l'emplacement connu, pour pouvoir réaliser l'insertion, il faut d'abord "libérer" la case qui va recevoir l'élément. Cela consiste à décaler d'un cran vers la droite le contenu de la case à libérer et de toutes les suivantes.

Question 2

Écrivez une fonction `insertElt` qui réalise l'insertion de `el` dans `tab` si l'élément n'est pas déjà présent et si le tableau n'est pas plein. Cette fonction renverra 1 si l'insertion a été réalisée, 0 sinon.

Question 3

Écrivez un programme complet qui permet de tester les fonctions précédentes. Pour vérifier l'évolution du tableau lors de l'exécution du programme, vous utiliserez l'outil de debug DDD pour lequel une notice d'utilisation est mise en ligne sur la page de l'UE.

Exercice 25 – Le tri par insertion

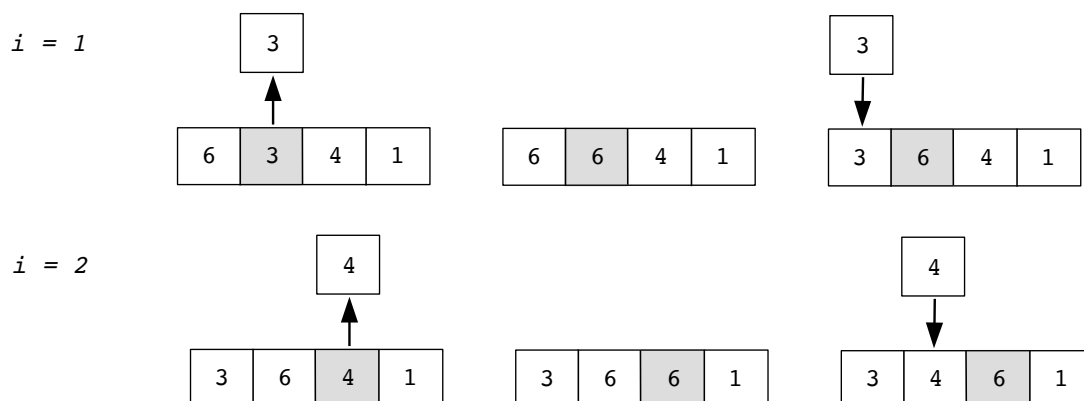
Dans l'exercice précédent, nous avons utilisé un mécanisme d'insertion permettant de garantir qu'un tableau reste trié lors de l'ajout de nouveaux éléments. Dans l'exercice, nous allons utiliser le mécanisme d'insertion pour écrire un programme qui trie par ordre croissant un tableau dont les éléments sont initialement dans un ordre quelconque. Le principe est décrit ci-dessous.

Supposons que les i premiers éléments du tableau soient triés. On extrait alors du tableau `tab` l'élément `tab[i]`, puis on insère cet élément à sa place parmi ceux qui le précèdent dans le tableau, ce qui implique de décaler d'une position vers la droite toutes les valeurs supérieures à `tab[i]` qui figurent dans les cases d'indice inférieur à i (ces valeurs sont déjà ordonnées).

En faisant varier i de 1 jusqu'à `taille-1`, on s'assure à chaque étape que tous les éléments qui précèdent `tab[i]` dans le tableau sont ordonnés. En effet,

- le tableau `tab` réduit à son premier élément (`tab[0]`) est trié;
- lorsqu'on extrait `tab[1]` du tableau, la case d'indice 1 devient libre. Si `tab[0]` est plus grand que `tab[1]`, on le copie dans la case 1 (décalage à droite), la case 0 est maintenant libre et on peut y copier `tab[1]` (insertion). Sinon, on ne fait rien. Après cette étape, les deux premières cases du tableau sont triées;
- on répète ce mécanisme jusqu'à avoir inséré à sa place le dernier élément du tableau.

Un exemple d'exécution de cet algorithme est illustré par la figure 7.



Question 1

En vous inspirant de ce qui a été fait pour l'insertion d'un élément dans un tableau trié, écrivez une fonction

```
void placeElt(float tab[], int i)
```

qui positionne l'élément d'indice i à sa place parmi ses prédécesseurs.

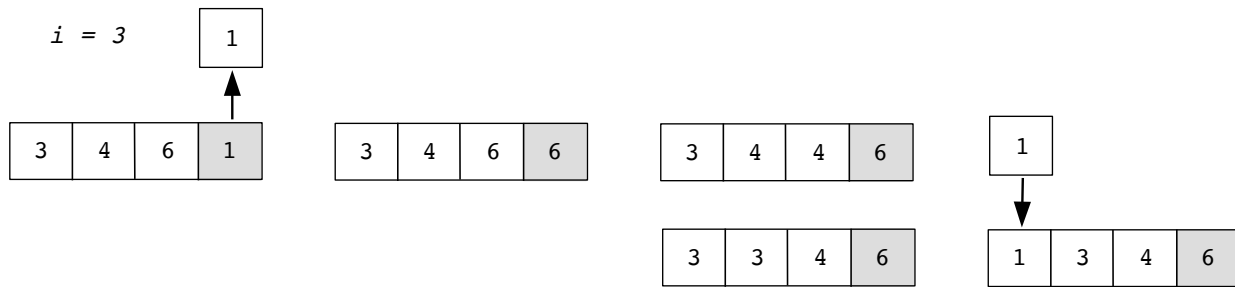


FIGURE 7 – Tri par insertion d'un tableau de 4 cases

Question 2

Écrivez une fonction `main` qui effectue le tri d'un tableau. Testez-la sur plusieurs exemples. Contrôlez son exécution en affichant l'évolution du tableau à l'aide de l'outil DDD.

Exercice 26 – (CodeRunner) Fusion de tableaux triés

Nous voulons écrire une fonction qui prend en paramètres deux tableaux d'entiers *triés* de tailles respectives `taille1` et `taille2` et qui renvoie le tableau *trié* obtenu en fusionnant les deux tableaux. Cette fonction ne supprime pas les doublons : la taille du tableau résultat est la somme des tailles des deux tableaux.

Question 1

Donnez le prototype de la fonction.

Le principe de fusion est le suivant : on gère un indice de parcours dans chacun des deux tableaux de données, et un troisième indice pour le tableau résultat.

Tant qu'on n'a parcouru entièrement aucun des deux tableaux, on compare les valeurs de `tab1[i1]` et `tab2[i2]`. On recopie la plus petite à l'indice `i` du tableau résultat et on incrémente `i` et l'indice du tableau d'où provient la donnée.

Lorsqu'on arrive au bout d'un des deux tableaux, il faut recopier dans le tableau résultat les éventuelles valeurs restant dans le tableau qui n'a pas été entièrement parcouru.

Question 2

Écrivez le corps de la fonction, ainsi qu'un programme permettant de la tester.

Semaine 5 - TME

Objectifs

- Chaînes de caractères
- Tableaux à deux dimensions

Exercices

Exercice 27 – (CodeRunner) Comptage des mots d’une chaîne

Question 1

Écrivez une fonction qui prend en paramètre une chaîne de caractères et qui renvoie le nombre de mots constituant cette chaîne. L’unique séparateur considéré ici est l’espace, il peut y avoir plusieurs espaces entre deux mots, il peut y avoir des espaces en début ou en fin de chaîne.

Toute séquence d’un ou plusieurs caractères autres que l’espace forme un mot.

Question 2

Écrivez une fonction `main` pour tester votre programme.

Exercice 28 – (CodeRunner) Participation aux frais

Un groupe de `NB_AMIS` amis, partant en vacances pour un séjour de `NB_JOURS` jours, souhaite gérer dans un tableau à deux dimensions la participation financière de chacun aux frais du séjour. Chaque colonne du tableau représente, pour une journée, le solde des dépenses de cette journée pour chaque membre du groupe (il y a une ligne par membre du groupe).

Question 1

Écrivez une fonction qui initialise à 0 le contenu de chacune des cases du tableau. Écrivez un programme qui déclare un tableau permettant de stocker l’ensemble des soldes et qui initialise à zéro le contenu de chacune de ses cases.

Pour faciliter les comptes en fin de séjour, il est décidé que :

- les dépenses d’une journée seront payées par une seule personne,
- les dépenses de la journée seront réparties immédiatement. Le tableau contiendra pour chaque personne et chaque jour son avoir (s’il a réglé les dépenses de la journée) ou sa dette (s’il n’a pas réglé les dépenses de la journée). Par exemple, si l’on considère un groupe de 3 personnes dans lequel l’un des membres dépense 45 euros, la participation de chacun s’élève à 15 euros. Les valeurs inscrites dans le tableau pour cette dépense seront de 30 (= 45 – 15) euros pour celui qui a payé et de –15 euros pour les deux autres. La somme des valeurs dans une colonne est donc toujours nulle.

Question 2

Programmez une fonction qui écrit dans le tableau les comptes du jour `j` : la fonction tire aléatoirement un montant entier de dépenses entre 30 et 50 euros et tire aussi aléatoirement l’identité du payeur.

Question 3

Complétez le programme pour afficher, pour chaque journée, le solde de chacun sous une forme similaire à celle-ci (dans le cas d’un groupe de 4 voyageant 7 jours) :

Jour 1 : 1 paye 37
 Jour 2 : 2 paye 32
 Jour 3 : 0 paye 43
 Jour 4 : 2 paye 39
 Jour 5 : 1 paye 38
 Jour 6 : 1 paye 36
 Jour 7 : 2 paye 30

	1	2	3	4	5	6	7
0	-9.25	-8.00	32.25	-9.75	-9.50	-9.00	-7.50
1	27.75	-8.00	-10.75	-9.75	28.50	27.00	-7.50
2	-9.25	24.00	-10.75	29.25	-9.50	-9.00	22.50
3	-9.25	-8.00	-10.75	-9.75	-9.50	-9.00	-7.50

NB : vous n'accorderez pas une importance excessive à la mise en page du tableau.

Question 4

Écrivez une fonction qui calcule et renvoie le solde du voyage pour un membre du groupe. Complétez la fonction `main` pour afficher le solde total de chacun des membres du groupe.

Exercice 29 – Filtrage d'une chaîne de caractères

Nous souhaitons écrire un programme qui parcourt une chaîne de caractères et qui la filtre en ne conservant que les caractères correspondant à des lettres.

Question 1

Écrivez une fonction qui prend en paramètre une chaîne de caractères et qui *affiche* uniquement les lettres de la chaîne. Écrivez une fonction `main` permettant de tester votre fonction.

Question 2

Écrivez maintenant une fonction qui prend en paramètre une chaîne de caractères et qui renvoie la chaîne construite en conservant uniquement les lettres de la chaîne en paramètre (cette dernière n'est pas modifiée).

Question 3

Peut-on écrire une fonction qui effectue un filtrage "en place" d'une chaîne de caractères, c'est-à-dire qui modifie directement la chaîne passée en paramètre ?

Exercice 30 – (CodeRunner) Compression d'un tableau de bits

Nous souhaitons écrire un programme qui compresse sans perte et décompresse des séquences de 0 et de 1. Les données brutes (décompressées) sont une suite de 0 et de 1 d'au plus `MAX` éléments, terminée **systématiquement** par la valeur `-1` (qui indique la fin des données pertinentes). Par exemple :

0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 -1

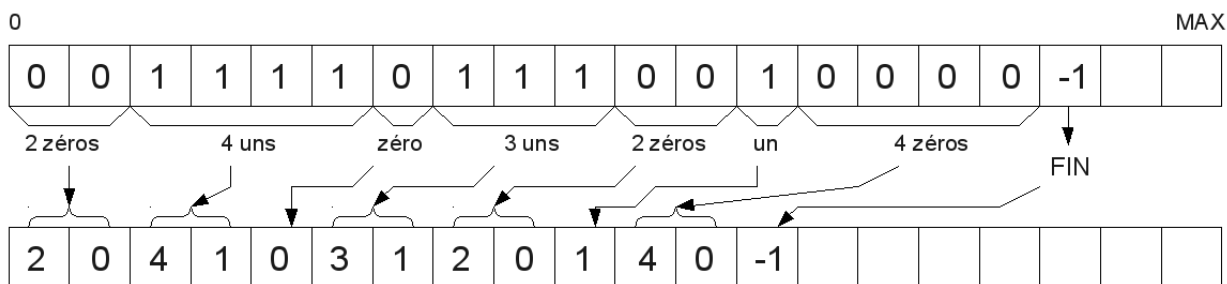
Dans notre programme C, les données brutes et les données compressées seront stockées dans deux tableaux d'entiers de taille `(MAX+1)` appelés respectivement `brut` et `compress`.

Attention : la valeur de fin (`-1`) n'est pas forcément dans la dernière case du tableau. Nous pouvons ainsi stocker des suites contenant moins de `MAX` éléments.

L'algorithme de compression utilisé est très simple¹ : on recherche les groupes d'éléments successifs identiques (par exemple, 3 fois de suite la valeur "1"). Lorsqu'un élément apparaît une seule fois, il est conservé tel quel. Mais s'il apparaît n fois de suite avec $n \geq 2$, il est stocké sous la forme " n suivi de l'élément répété". Par exemple, la séquence ci-dessus sera compressée en :

2 0 4 1 0 3 1 2 0 1 4 0 -1

car il y a 2 fois l'élément 0 puis 4 fois l'élément 1 puis une seule fois l'élément 0 puis 3 fois l'élément 1, *etc*, comme l'illustre le dessin ci-dessous.



Nous allons écrire un programme qui :

- permet de générer aléatoirement un tableau de données brutes ;
- compresse ce tableau ;
- effectue ensuite l'opération inverse pour afficher le contenu du tableau initial en parcourant le tableau compressé.

Dans la réalité, nous aurions trois programmes différents : le programme de décompression ne connaît pas le tableau brut de départ. Cela explique l'interdiction, dans l'énoncé, de réutiliser certaines variables.

Question 1

Écrivez une fonction qui initialise un tableau de données brutes de manière aléatoire de sorte que :

1. `taille`, le nombre d'éléments de la séquence, soit choisi aléatoirement entre `MIN` et `MAX` (ces deux valeurs sont définies par des primitives `#define`) ;
2. la valeur de chaque élément de la séquence soit tirée aléatoirement entre 0 et 1 ;
3. la fin du tableau soit indiquée par la valeur -1.

La fonction renvoie le nombre de cases du tableau auxquelles une valeur a été affectée.

Écrivez une fonction d'affichage de tableau et une fonction `main` qui initialise et affiche un tableau brut.

Question 2

Écrivez une fonction

```
int compress_tab(int tab_brut[], int tab_compress[])
```

qui, à partir d'un tableau `tab_brut` initialisé avec des données brutes, remplit le tableau `tab_compress` avec les données compressées. La valeur -1 est stockée dans le tableau à la fin de la séquence de données compressées.

La fonction renvoie le nombre de cases du tableau `tab_compress` auxquelles une valeur a été affectée.

Pour écrire cette fonction, vous pourrez utiliser un compteur lors du parcours du tableau `tab_brut`. Ce compteur est réinitialisé chaque fois que la nouvelle valeur lue diffère de la précédente.

1. Ce type de compression est utilisé, par exemple, pour les images en noir et blanc, bien que l'algorithme ne soit pas exactement celui décrit ici. Chaque pixel prend la valeur 0 ou 1 selon qu'il est blanc ou noir. L'algorithme de compression consiste alors à compter les pixels blancs ou noirs consécutifs.

Question 3

Écrivez une fonction

```
int decompress_tab(int tab_brut[], int tab_compress[])
```

qui, à partir d'un tableau `tab_compress` initialisé avec des données compressées, reconstruit le tableau `tab_brut` de données brutes. La valeur `-1` est stockée dans le tableau à la fin de la séquence de données brutes.

La fonction renvoie le nombre de cases du tableau `tab_brut` auxquelles une valeur a été affectée.

Question 4

Écrivez une fonction permettant de comparer deux tableaux dont les contenus sont des séquences terminées par `-1`. La fonction renvoie `1` si les contenus des deux tableaux sont identiques, `0` sinon.

Testez votre fonction en comparant le tableau `tab_brut` initial et le tableau obtenu par décompression du tableau compressé.

Exercice 31 – L'image mystère

La fonction `rand()` permet de tirer des valeurs dans un intervalle de manière équiprobable. On peut le vérifier en effectuant un nombre de tirages assez important et en comparant le nombre de fois où chaque valeur est tirée. Pour 10 000 tirages dans l'intervalle `0 . . 3`, on obtient ce type de résultat :

```
Valeur 0 tiree 2501 fois.
Valeur 1 tiree 2467 fois.
Valeur 2 tiree 2515 fois.
Valeur 3 tiree 2517 fois.
```

Nous voulons maintenant un tirage qui ne soit pas équiprobable. Par exemple, la probabilité de tirer la valeur `0` doit être de 17%, 28% pour la valeur `1`, 50% pour la valeur `2`, et 5% pour la valeur `3`.

Pour cela, nous allons effectuer un tirage aléatoire parmi les valeurs de `0` à `99`. Si la valeur tirée est comprise entre `0` et `16` alors la valeur affectée au résultat est `0`, si la valeur tirée est comprise entre `17` et `44` (ce qui représente un intervalle de 28 valeurs) alors la valeur affectée au résultat est `1`, si la valeur tirée est comprise entre `45` et `94` alors la valeur affectée au résultat est `2` et si la valeur est comprise entre `95` et `99` alors la valeur affectée au résultat est `3`.

Question 1

Écrivez une fonction `calcule_bornes_sup` qui prend en paramètres un tableau de pourcentages et qui remplace chaque pourcentage par la borne supérieure de l'intervalle de tirage correspondant.

Dans l'exemple, le tableau en paramètre contient les valeurs `[17, 28, 50, 5]`. Après l'exécution de la fonction, il doit contenir les valeurs `[16, 44, 94, 99]`.

Question 2

Écrivez une fonction `tire_non_equi` qui prend en paramètres un tableau de bornes et qui renvoie le résultat du tirage non équiprobable. Pour cela, la fonction tire une valeur dans l'intervalle `0 . . 99` et compare la valeur tirée aux bornes des différents intervalles.

Dans l'exemple, si la valeur tirée est `15`, la fonction renvoie `0`, si la valeur tirée est `95`, la fonction renvoie `3`.

Question 3

Écrivez une fonction `main` qui effectue 10 000 tirages non équiprobables, qui stocke dans un tableau le nombre d'occurrences de chacun des résultats et qui affiche le contenu de ce tableau.

Pour l'exemple, le type de résultat attendu est : `[1676 2730 5127 467]`.

Nous allons utiliser le tirage non équiprobable pour appliquer des transformations affines sur les points d'une image. L'application d'une transformation affine à un point (x_n, y_n) s'écrit de la manière suivante :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} \cdot \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix}$$

ou encore :

$$\begin{aligned}x_{n+1} &= a_i.x_n + b_i.y_n + e_i \\ y_{n+1} &= c_i.x_n + d_i.y_n + f_i\end{aligned}$$

Le programme que l'on va construire calcule un ensemble de points par l'application répétitive d'une transformation affine sur le dernier point calculé : on part d'un point (x_0, y_0) à partir duquel on calcule le point (x_1, y_1) , puis on calcule le point (x_2, y_2) à partir de (x_1, y_1) , etc.

On dispose de 4 transformations affines ta_0, ta_1, ta_2 et ta_3 . Le choix de la transformation à appliquer à une étape se fait de manière aléatoire, mais avec un tirage qui n'est pas équiprobable : la probabilité d'appliquer la transformation ta_0 est de 1%, chacune des transformations ta_1 et ta_2 a une probabilité de 7% et la transformation ta_3 a une probabilité de 85%.

Question 4

Recopiez le fichier `feuille_enonce.c`. Complétez le programme pour calculer un ensemble de points à partir du point $(0, 0)$ en appliquant, avec les probabilités données ci-dessus, les transformations dont les coefficients figurent dans les tableaux. La couleur d'affichage du point dépend de la transformation qui a été appliquée. Les valeurs `dX` et `dY` permettent de positionner le dessin dans la fenêtre, les valeurs `coefX` et `coefY` fixent l'échelle du dessin.

Vous ajouterez les variables qui vous sont nécessaires.

Semaine 6 - TME

Objectifs

— Récursivité sur les tableaux

Exercices

Exercice 32 – (CodeRunner) Recherche d'un élément dans un tableau

La recherche dichotomique est une technique efficace pour rechercher un élément dans un tableau *trié*. Nous considérons dans cet exercice un tableau de taille N , *non trié*, contenant des entiers.

Question 1

Écrivez une fonction *itérative* permettant de tester la présence d'un élément dans un tableau. La fonction renvoie 1 si l'élément est présent, 0 sinon.

Écrivez une fonction `main` permettant de tester votre fonction.

Nous souhaitons maintenant effectuer la recherche au moyen d'une fonction récursive.

Question 2

Quels sont les cas d'arrêt ?

Question 3

Écrivez une version *récursive* de la fonction de recherche.

Question 4

Quelle modification faut-il apporter à la fonction de recherche (dans le cas itératif et dans le cas récursif) lorsque le tableau est trié dans l'ordre croissant ? Modifiez vos fonctions en conséquence.

Exercice 33 – (CodeRunner) Inclusion de chaînes de caractères

Question 1

Écrivez une fonction *récursive* `est_deb` qui prend en paramètres deux chaînes de caractères. La fonction renvoie 1 si la première chaîne est le début de la deuxième, 0 sinon.

Par exemple, "alpha" est le début d'"alphabet", dans ce cas la fonction renvoie 1. Mais "alpaga" n'est pas le début d'"alphabet", dans ce cas la fonction renvoie 0.

Écrivez une fonction `main` permettant de tester votre fonction.

Nous souhaitons maintenant pouvoir déterminer si une chaîne `chaine1` est incluse dans une chaîne `chaine2`.

Question 2

Écrivez une fonction *récursive* `est_incluse` qui prend en paramètres deux chaînes de caractères. La fonction renvoie 1 si la première chaîne est incluse dans la deuxième, 0 sinon.

Par exemple, "abe" est incluse dans "alphabet", dans ce cas la fonction renvoie 1. Mais "beta" n'est pas incluse dans "alphabet", dans ce cas la fonction renvoie 0.

Vous pourrez utiliser la fonction `est_deb` pour programmer la fonction `est_incluse`. Complétez votre fonction `main` pour tester l'inclusion.

La solution précédente n'est pas optimale : si `chaine1` est plus longue que `chaine2`, on va malgré tout rechercher `chaine1` à partir de toutes les positions de `chaine2` avant de détecter qu'il n'y a pas inclusion.

À défaut d'une solution optimale qui nécessiterait de connaître la longueur des chaînes, on peut améliorer notre solution en stoppant la recherche dès que `est_deb` échoue parce qu'on est arrivé au bout de `chaine2` mais pas de `chaine1`.

Question 3

Proposez une nouvelle version des fonctions `est_deb` (en fait, il est sans doute judicieux d'écrire plutôt une fonction `n_est_pas_deb`) et `est_incluse`.

Exercice 34 – Jeu du morpion

L'objectif de cet exercice est de réaliser un jeu de morpion à deux joueurs. Sur un plateau de jeu de dimensions $N \times N$, 2 joueurs posent un pion à tour de rôle. Si l'un des joueurs parvient à aligner N pions, il gagne la partie. Si le plateau est rempli sans que cette condition soit réalisée, il y a match nul.

Le plateau de jeu sera représenté par un tableau de caractères à deux dimensions. Une case vide sera représentée par le caractère ' _ ', les pions du joueur 0 par le caractère ' X ' et ceux du joueur 1 par le caractère ' O '.

Question 1

Écrivez une fonction `void init(char plateau[N][N])` qui initialise une partie (toutes les cases du plateau sont vides). Écrivez une fonction `void afficher(char plateau[N][N])` qui affiche le plateau de jeu et écrivez une fonction `main` qui teste vos deux fonctions.

Question 2

Avant de commencer une partie, le programme affiche un menu permettant au joueur de choisir entre trois options :

1. partie à 2 joueurs
2. partie contre l'ordinateur
3. quitter

Écrivez une fonction `int choisir_menu2q()` qui affiche le menu et renvoie l'option choisie par le joueur. La fonction doit contrôler la validité de la saisie du joueur.

Question 3

Écrivez une fonction `void jouer(char plateau[N][N], int joueur)` qui permet au joueur passé en paramètre de la fonction de poser un pion sur le plateau.

La fonction demande au joueur de saisir les deux coordonnées de la case dans laquelle il veut jouer, vérifie que ces coordonnées correspondent à une case du plateau et que la case est vide. Lorsque toutes ces conditions sont remplies, la case du plateau est remplie avec le jeton du joueur. Si ce n'est pas le cas, le joueur doit saisir de nouvelles coordonnées.

Question 4

Écrivez une fonction `void jouerOrdinateur(char plateau[N][N])` qui pose le pion de l'ordinateur lorsque le joueur a choisi l'option 2 du menu.

La fonction fait l'hypothèse que l'ordinateur est toujours le joueur 1. La stratégie de l'ordinateur est de choisir aléatoirement une case vide.

Question 5

Écrivez une fonction `int partie_gagnée(char plateau[N][N])` qui détermine si la configuration courante du plateau est gagnante pour l'un des deux joueurs.

La fonction doit donc vérifier s'il existe sur le plateau une ligne, une colonne ou une diagonale sur laquelle tous les jetons sont identiques. L'idée est de regarder le contenu de la première case de la ligne (resp. colonne, diagonale), qui doit être non vide, et de parcourir la ligne (resp. colonne, diagonale) jusqu'à trouver un jeton différent ou atteindre la fin de la ligne (resp. colonne, diagonale).

Dans le deuxième cas, la partie est gagnée, la fonction renvoie 1. La fonction ne peut conclure qu'il n'y a pas de gagnant qu'une fois toutes les lignes, colonnes, diagonales parcourues sans succès. Dans ce cas, la fonction renvoie 0.

Question 6

Nous nous intéressons maintenant au déroulement d'une partie. La fonction

```
void jouer_a(char plateau[N][N], int nb_joueurs)
```

exécute l'enchaînement des coups, jusqu'à ce que la partie se termine. Le joueur 0 joue les coups pairs, l'ordinateur ou le joueur 1 les coups impairs. Chaque coup joué est comptabilisé, et le plateau de jeu affiché à chaque fois. La partie se termine dès qu'un joueur (ou l'ordinateur) aligne N pions, ou lorsque le plateau est rempli.

Écrivez le code de la fonction, qui doit afficher le résultat de la partie.

Question 7

Complétez la fonction `main` pour que le joueur puisse, par l'intermédiaire du menu, choisir le type de partie ou quitter le programme.

Semaine 7 - TME

Objectifs

- Structures
- Tableaux de structures

Exercices

Exercice 35 – (CodeRunner) Système solaire

Nous souhaitons représenter les planètes du système solaire. Chaque planète est caractérisée par :

- son nom (10 caractères au maximum),
- sa densité (un réel),
- sa distance au soleil en millions de kilomètres (un réel),
- le nombre de ses satellites (un entier).

Vous complèterez le fichier `planete.c` qui vous est fourni et testerez vos fonctions au fur et à mesure.

Question 1

Définissez le type `planete` qui permet de déclarer une planète. Pour pouvoir utiliser le fichier qui vous est fourni faites attention à l'ordre dans lequel vous déclarez les différents éléments, il doit correspondre à l'ordre de l'énoncé.

Question 2

Écrivez la fonction `affichePlanete` qui prend en paramètre une planète et qui affiche ses caractéristiques.

Question 3

Écrivez la fonction `afficheToutesPlanetes` qui prend en paramètre un tableau de planètes, sa taille et qui affiche les caractéristiques de toutes les planètes du tableau. Cette fonction doit faire appel à la fonction `affichePlanete`.

Question 4

Nous faisons l'hypothèse que des nouvelles découvertes peuvent remettre en cause les densités actuellement connues. Pour anticiper une telle possibilité, écrivez la fonction `modifieDensite` qui permet de modifier la densité d'une planète, son type de retour doit être `void`.

Exercice 36 – Poursuite du TP

Une fois fait le premier exercice de ce sujet, vous avez deux possibilités :

- **Vous n'avez pas fini les exercices des semaines précédentes**, vous devez les terminer, ces exercices abordent des notions de base que vous devez maîtriser.
- **Vous avez fini les exercices des semaines précédentes**, vous pouvez donc continuer avec les exercices de cette semaine (programmation d'un Tetris) qui reviennent sur les différentes notions vues dans les semaines précédentes et les structures.

Exercice 37 – Tetris : structures de données

Cette présentation est en grande partie extraite de Wikipedia (<http://fr.wikipedia.org/wiki/Tetris>).

Le principe du jeu de Tetris est simple : des pièces de couleur et de formes différentes descendent du haut de la fenêtre de jeu. Le joueur ne peut pas agir sur cette chute mais peut décider à quel angle de rotation (0° , 90° , 180° , 270°) et à quel emplacement latéral l'objet peut atterrir. Lorsqu'une ligne horizontale est complétée sans vide, elle disparaît et les blocs supérieurs tombent. Si le joueur ne parvient pas à faire disparaître les lignes assez vite et que la fenêtre se remplit jusqu'en haut, la partie est finie.

Les pièces de Tetris sont appelées "tétrominos" (du grec *tetra* : quatre) car elles sont toutes basées sur un assemblage de quatre carrés. Il en existe sept formes différentes, elles sont représentées dans la figure 8. A chacune d'elles est associée une lettre de l'alphabet.

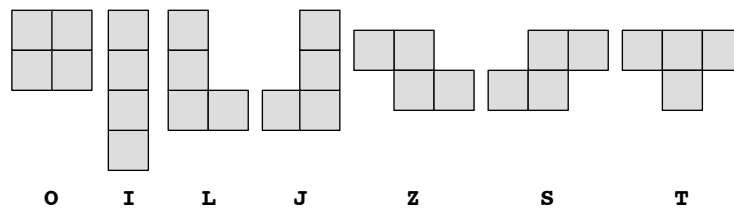


FIGURE 8 – Représentation des sept tétrominos et de la lettre associée

Le plateau de jeu est une grille formée de cases de même taille que les carrés qui composent les pièces. La largeur du plateau est de 10 cases, la hauteur de 22 cases.

Récupérez le fichier `Tetris1.c` qui contient une boucle de jeu qui va vous permettre de tester vos fonctions. Dans cette version du programme, contrairement au vrai jeu de Tetris, aucune action n'est temporisée. Il s'agit en fait de disposer d'un programme dans lequel l'utilisateur maîtrise tous les événements, ce qui rend la mise au point plus facile.

Les pièces de Tetris sont colorées. Le tableau `color` contient 7 couleurs (une couleur est une chaîne de caractères, en anglais et en minuscules), correspondant aux 7 tétrominos.

```
char* color[] = {"light salmon", "fuchsia", "lime green",
                "white", "yellow", "cyan", "grey"};
```

Un tétromino est un assemblage de quatre cases. Chacune de ces cases est repérée par deux coordonnées `colonne` et `ligne` qui permettent de la positionner sur le plateau de jeu. La structure `une_case` suivante permet de représenter une case.

```
struct une_case {
    int colonne;
    int ligne;
};
```

Lors de la création d'une nouvelle pièce sur le plateau de jeu, il faut initialiser (entre autres) les coordonnées de ses quatre cases. Pour faire cette initialisation facilement, nous choisissons tout d'abord de construire un tableau, nommé `tab_pieces`, contenant la description des 7 tétrominos. Nous allons donc déclarer un tableau à deux dimensions : une ligne par tétromino et pour chaque ligne, quatre colonnes correspondant aux 4 cases de chaque tétromino. Pour chaque tétromino, une des cases est en position (0,0), la position des autres cases est donnée relativement à la case en position (0,0). Nous expliquerons un peu plus loin, comment ce tableau est construit.

Pour déterminer la position à un instant donné d'un tétromino, nous choisissons de mémoriser la position sur le plateau de jeu de la case (0,0), c'est-à-dire, la *colonne* et la *ligne* auxquelles elle se trouve. Les emplacements occupés par les autres cases seront calculés en ajoutant les coordonnées relatives de celles-ci à (*colonne*, *ligne*).

Enfin, un tétromino a une couleur. Pour limiter les manipulations de chaînes de caractères, nous allons simplement indiquer dans la définition du tétromino la position de sa couleur dans le tableau de couleurs.

La structure `piece` suivante permet de représenter un tétromino : les coordonnées de la case (0,0) sur le plateau de jeu, les coordonnées relatives de ses 4 cases et sa couleur.

```

struct piece {
    int pos_ligne, pos_colonne;
    struct une_case la_piece[4];
    int type;
};

```

Question 1

Nous revenons sur la construction du tableau `tab_pieces` contenant la description des 7 tétramino.

Nous choisissons de représenter l'assemblage des quatre cases d'un tétramino par les coordonnées relatives des cases les unes par rapport aux autres : l'une des cases a toujours pour coordonnées (0, 0) et les coordonnées des autres cases sont définies par rapport à celle-ci. Par exemple, les coordonnées des quatre cases du tétramino 'O' (carré) sont données dans la Figure 9.

Supposons que le tableau `tab_pieces` regroupe la description des sept tétramino et la pièce 0 (carré) se trouve à l'indice 0. La représentation de cette pièce est donnée dans la Figure 9. L'initialisation d'une pièce se fera ensuite en recopiant dans le champ adéquat de la pièce créée la ligne correspondante du tableau `tab_pieces`. L'ordre des 4 cases dans chaque ligne du tableau `tab_pieces` n'a pas d'importance.

(0, 0)	(1, 0)	<code>tab_pieces[0] = { {0, 0}, {0, 1}, {1, 0}, {1, 1} }</code>
(0, 1)	(1, 1)	<code>tab_pieces[0][2] = {1, 0}</code>
		<code>tab_pieces[0][2].colonne = 1</code>
		<code>tab_pieces[0][2].ligne = 0</code>

FIGURE 9 – Représentation du tétramino 'O'

Insérez, dans la fonction `main` la déclaration et l'initialisation du tableau `tab_pieces` permettant de stocker les sept tétramino. Pour simplifier l'insertion d'une nouvelle pièce dans le tableau de jeu, il est judicieux de choisir les coordonnées de sorte qu'aucune coordonnée *ligne* ne soit négative.

Dans la suite, lorsque nous parlons du *type* d'une pièce il s'agit de son indice dans le tableau que vous devez déclarer. Dans l'exemple de la Figure 9, la pièce de type 0 est la pièce associée à la lettre O.

Dans la fonction `main`, l'appel à la fonction `afficher_toutes_pieces`, affiche les sept tétramino. L'exécution du programme vous permet alors de vérifier que vous avez bien déclaré le tableau `tab_pieces`. Une fois cette vérification faite, vous pouvez mettre l'appel à `afficher_toutes_pieces` en commentaire.

Pour compiler le programme vous devez utiliser l'option `-lncini` pour pouvoir accéder aux fonctions de la bibliothèque graphique. La touche `escape` vous permet de fermer la fenêtre graphique et donc de terminer l'exécution du programme.

Exercice 38 – Tetris : le jeu

Le principe général du jeu consiste à gérer les déplacements d'une pièce sur le plateau de jeu. Celui-ci est représenté par un tableau de `HAUTEUR` lignes et `LARGEUR` colonnes. La valeur stockée dans une case du tableau est la couleur de l'élément qui l'occupe si elle n'est pas vide (sous forme d'un indice référençant la case correspondante dans le tableau de couleurs), une valeur particulière représentée par la constante `VIDE` si la case n'est pas occupée.

Le plateau de jeu

Le plateau de jeu est affiché dans une fenêtre graphique de même taille. Chaque case du plateau (qui a la même taille qu'une case de tétramino) sera représentée par un carré de `TAILLE_CASE` pixels de côté.

Question 1

Complétez la fonction `main` en déclarant et initialisant le plateau de jeu.

Question 2

Écrivez la fonction `afficher_plateau` qui prend en paramètres un plateau de jeu et un tableau de couleurs et affiche chaque case du plateau en fonction de la couleur correspondant à son contenu. Toute case vide sera affichée de la couleur de la fenêtre. Dans la fonction `main`, "décommentez" l'appel à la fonction `afficher_plateau` et complétez le.

Attention :

- les colonnes correspondent à l'axe des abscisses de la fenêtre graphique et les lignes à celui des ordonnées,
- lorsqu'une fonction a en paramètre un tableau à deux dimensions, il faut que sa signature contienne la valeur de chacune des deux dimensions.

NB : Pour que les différentes étapes du jeu ne se superposent pas il faut "effacer" le tableau avant de l'afficher à nouveau. Ceci peut se faire facilement en remplissant la fenêtre de sa couleur de fond (utilisation de la fonction `CINI_fill_window`).

La création d'une pièce

Question 3

Pour choisir le type de la prochaine pièce créée, la boucle de jeu tire aléatoirement une valeur entre 0 et 6 (inclus). L'initialisation consiste ensuite à aller lire le tableau des pièces pour recopier dans la pièce créée les informations correspondant au type tiré. La pièce créée est positionnée en haut et au milieu de la fenêtre.

NB : nous avons fait le choix, pour les fonctions qui modifient une pièce, de passer l'adresse d'une pièce en paramètre, plutôt que d'avoir un résultat de type pièce. En effet, dans ce dernier cas, il faudrait systématiquement recopier *tous* les champs de la pièce modifiée dans la variable résultat, alors qu'en travaillant directement sur la pièce, on peut n'affecter que les champs modifiés. Nous avons fait ce choix aussi pour la fonction d'initialisation, par souci d'homogénéité.

Vous disposez de la fonction `initialiser` qui prend en paramètre l'adresse d'une pièce, les caractéristiques de la pièce créée (la bonne ligne du tableau `tab_pieces`), le type de la pièce créée. Dans la fonction `main`, "décommentez" l'appel à la fonction `initialiser` et complétez le.

Question 4

Pour afficher la pièce créée, nous allons modifier le contenu de la fenêtre graphique, sans modifier le contenu du plateau de jeu. Celui-ci ne sera mis à jour qu'une fois que la pièce aura atteint sa position définitive.

Vous disposez de la fonction `afficher_piece` qui prend en paramètre une pièce et une couleur et qui affiche la pièce dans la fenêtre graphique. Dans la fonction `main`, "décommentez" les trois appels à la fonction `afficher_piece` et complétez les.

Les déplacements

La seule action que nous prenons en compte est la frappe d'une touche. Toute frappe entraîne la descente de la pièce. En fonction de la touche choisie, un mouvement peut être ajouté :

- flèche gauche : décalage de la pièce sur la gauche ;
- flèche droite : décalage de la pièce sur la droite ;
- touche `d` : rotation à gauche ;
- touche `g` : rotation à droite ;

Vous pouvez inverser l'effet des touches `d` et `g` en remplaçant, dans le `switch` de la fonction `main`, `SDLK_d` par `SDLK_g` et inversement.

La flèche vers le bas provoque la création d'une nouvelle pièce et la touche `escape` la fin de la partie.

Après chaque mouvement, il faut réafficher le plateau de jeu.

Question 5

Le mouvement de base d'une pièce est la descente. Avant de faire descendre la pièce, il faut s'assurer que ce

déplacement est possible, donc qu'aucune partie de la pièce ne sort de la fenêtre et que toutes les cases du plateau que la pièce occupera dans sa nouvelle position sont vides.

Si la descente est possible, on modifie les coordonnées de la pièce, sinon celle-ci a atteint sa position définitive et ne pourra donc plus descendre, il faut alors mettre à jour le plateau de jeu en marquant les emplacements occupés par cette pièce.

Écrivez et testez la fonction `descendre` qui prend en paramètre le plateau de jeu et l'adresse d'une pièce, qui modifie la pièce si celle-ci s'est déplacée, et qui renvoie un 0 si la pièce est bloquée et 1 sinon (ce résultat sera utilisé dans la suite, en particulier dans la version temporisée du jeu). Dans la fonction `main`, "décommentez" l'appel à la fonction `descendre` et complétez le.

N.B. : cette fonction réalise un mouvement élémentaire : la pièce ne descend que d'un niveau. Elle sera normalement appelée en boucle jusqu'au blocage de la pièce. Le plateau n'est mis à jour que lorsque la pièce est bloquée. Si vous faites apparaître une nouvelle pièce alors que la précédente n'était pas arrivée à sa position définitive, la pièce précédente disparaîtra (elle n'aura pas été enregistrée sur le plateau).

Question 6

Une pièce peut aussi se déplacer latéralement. Écrivez les fonctions `decaler_gauche` et `decaler_droite` qui modifient les coordonnées de la pièce en fonction du mouvement demandé lorsque celui-ci est possible. Si le mouvement est impossible, les fonctions sont sans effet. Ces fonctions prennent les mêmes paramètres que la fonction `descendre`. Dans la fonction `main`, "décommentez" les appels à ces fonctions et complétez les.

Question 7

Enfin, une pièce peut tourner sur elle-même d'un angle de 90° , vers la droite ou vers la gauche. Compte tenu de l'orientation du repère dans la fenêtre graphique, une rotation vers la droite modifie les coordonnées relatives du point (x, y) en (x', y') avec $x' = -y$ et $y' = x$ alors qu'une rotation vers la gauche aboutit à $x' = y$ et $y' = -x$ (voir Figure 10).

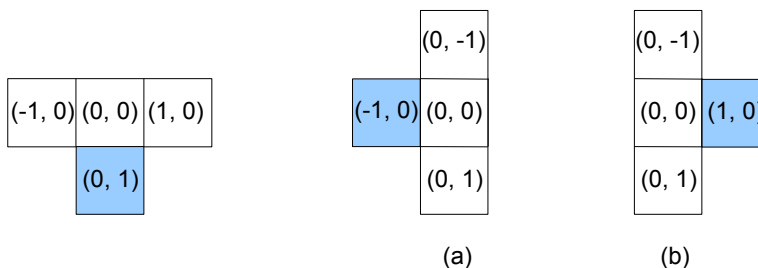


FIGURE 10 – Exemple de rotation droite (a) et gauche (b) : le tétramino 'T'

Écrivez les fonctions `rotation_gauche` et `rotation_droite` qui modifient les coordonnées de la pièce en fonction de la rotation demandée lorsque celle-ci est possible. Ces fonctions prennent les mêmes paramètres que la fonction `descendre`. Dans la fonction `main`, "décommentez" les appels à ces fonctions et complétez les.

Une fois que la pièce est bloquée

Lorsqu'une pièce ne peut plus être déplacée, il faut vérifier si son insertion permet de compléter des lignes. Si c'est le cas, toutes les lignes complètes doivent être supprimées et les lignes situées au-dessus doivent être décalées vers le bas.

Question 8

Écrivez une fonction `supprimer_lignes` qui prend en paramètre le plateau de jeu et qui efface dans le plateau de jeu toutes les lignes complètes. Dans la fonction `main`, "décommentez" l'appel à cette fonction et complétez le.

Question 9

La partie se termine lorsque l'ensemble des pièces empilées atteint le haut de la fenêtre. Écrivez une fonction

`partie_perdue` qui prend en paramètre le plateau de jeu et qui détermine si la partie est terminée ou non. Modifiez la boucle de jeu pour prendre en compte cette nouvelle condition de terminaison.

Déplacements rapides

Question 10

L'action `hard_drop` consiste à faire tomber directement une pièce lorsque sa position latérale a été choisie.

Ecrivez la fonction `hard_drop` qui prend les mêmes paramètres que la fonction `descendre`. Dans le `switch` de la fonction `main`, "décommentez" le cas dans lequel l'appel à la fonction `hard_drop` a lieu et complétez l'appel (l'appel à la fonction est associé à une action sur la touche `espace`, cas `SDLK_SPACE`).

Exercice 39 – Tetris : scores et niveaux

Nous allons, dans cet exercice, ajouter des fonctionnalités supplémentaires au jeu liées à la gestion d'une horloge pour faire descendre automatiquement les pièces à une vitesse dépendant du niveau du joueur.

Tetris ne se termine jamais par la victoire du joueur. Avant de perdre, le joueur doit tenter de compléter un maximum de lignes pour obtenir le score le plus élevé possible. Au niveau initial (le niveau 0), les points sont comptés de la manière suivante :

- une seule ligne complétée rapporte 40 points ;
- deux lignes complétées rapportent 100 points ;
- trois lignes complétées rapportent 300 points ;
- quatre lignes complétées rapportent 1200 points ;

Le niveau détermine la vitesse de la chute des pièces. Plus le niveau est élevé, plus les pièces tombent vite. Le nombre de points est augmenté à chaque niveau selon l'équation $nb_pts(n) = (n + 1).p$, où p est le nombre de points au niveau 0 et n le niveau.

Le niveau augmente de 1 toutes les `CHANGEMENT_NIVEAU` lignes supprimées.

Question 1

Récupérez le fichier `Tetris2.c` qui inclut une gestion d'horloge pour la descente des pièces, insérez-y vos déclarations de types, de variables, vos fonctions, complétez les appels et la condition de la boucle externe du jeu (tout ce que vous avez fait jusqu'à présent). Testez alors le fonctionnement de votre programme dans ce nouvel environnement. Si vous rencontrez des problèmes lors de la compilation ajoutez l'option `-lSDL` à votre commande.

Question 2

Modifiez la fonction `supprimer_lignes` pour prendre en compte le comptage des points. La fonction devra renvoyer le nombre de lignes supprimées et mettre à jour le score. La valeur de retour sera utilisée pour mettre à jour la variable `cpt` qui compte le nombre de lignes supprimées, vous devez donc aussi modifier l'appel à `supprimer_lignes` et déclarer la variable stockant le score.

Pour gérer les changements de niveau, vous pouvez "décommenter" les lignes de code agissant sur la variable `cpt`. Ces instructions diminuent la période de l'horloge de `DIMINUTION_PERIODE` ms et augmentent le niveau lorsque c'est nécessaire (la variable `niveau` est déjà déclarée et initialisée). Vous pouvez changer les valeurs associées à `CHANGEMENT_NIVEAU` et `DIMINUTION_PERIODE`.

Question 3

Pour pouvoir afficher l'évolution du score, nous allons augmenter la largeur de la fenêtre graphique, afin de réserver une partie dans laquelle se fera cet affichage.

Modifiez votre programme pour que :

- la fenêtre affichée soit deux fois plus large,
- une ligne verticale sépare la fenêtre en deux (une moitié sera le plateau de jeu, l'autre sera consacrée à l'affichage),
- modifiez la fonction `afficher_plateau` pour que seule la partie de la fenêtre représentant le plateau de jeu soit "effacée".

Question 4

Nous devons maintenant afficher le score dans la partie de la fenêtre prévue à cet effet.

Comme nous ne disposons pas d'une fonction permettant directement d'afficher graphiquement un entier, nous allons utiliser la fonction `CINI_draw_int_table` et le fait qu'un tableau d'entiers est un pointeur sur un entier.

Ecrivez une fonction `afficher_score` qui affiche le score au bon endroit dans la fenêtre graphique. Dans la fonction `main`, faites en sorte que le score soit à nouveau affiché chaque fois qu'il est modifié.

Semaine 8 - TME

Objectifs

- Bibliothèque : programme découpé en plusieurs fichiers
- Représentation des listes
- Les différents parcours de liste

Exercices

Comme pour les exercices de TD, les exercices de TME s'appuieront sur la structure de données suivante :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
    int donnee;
    cellule_t *suivant;
};
```

Exercice 40 – Plusieurs fichiers .c pour un programme

Les types et fonctions définis dans un programme peuvent être utiles à différents programmes, il est alors intéressant de les regrouper dans un même fichier qui sera ensuite inclus par les programmes en ayant besoin. Un tel fichier est appelé “bibliothèque”, cela permet d’éviter de dupliquer du code, ce qui est une très bonne pratique.

Recopiez les fichiers `liste_entiers.h`, `liste_entiers.c` et `test_liste.c`. La bibliothèque `liste_entiers` est utilisée par le programme `test_liste`.

- Le fichier `liste_entiers.h` contient la définition du type `cellule_t` et la signature de la fonction `creerListe` (dont vous n’avez pas besoin de comprendre le fonctionnement pour cette séance). Ce fichier .h est appelé *header*.
- Le fichier `liste_entiers.c` contient la définition de la fonction `creerListe` et inclut le fichier `liste_entiers.h` pour avoir accès au type `cellule_t`.
- Le fichier `test_liste.c` contient la fonction `main` permettant de tester la fonction définie dans le fichier `liste_entiers.c`. Le fichier `liste_entiers.h` est donc inclus dans le fichier `test_liste.c`.

Pour compiler un programme découpé en plusieurs fichiers, la manière la plus simple est de procéder comme habituellement mais en incluant dans la commande de compilation *tous les fichiers source* utilisés par le programme.

Question 1

Compilez le programme précédent en exécutant la commande suivante et utilisez `ddd` pour visualiser la liste construite. La compilation signale un avertissement que vous ignorerez pour cette question.

```
gcc -Wall -g -o test_liste liste_entiers.c test_liste.c
```

Question 2

Vous allez maintenant enrichir la bibliothèque `liste_entiers` en y ajoutant une fonction `void AfficherListeInt(cellule_t *liste)` qui affiche le champs `donnee` de tous les éléments de `liste`. Vous devez alors :

- ajouter la signature de la fonction au fichier `liste_entiers.h`;
- ajouter la définition de la fonction au fichier `liste_entiers.c`;

- modifier le fichier `test_liste.c` pour tester la fonction `AfficherListeInt`.

Exercice 41 – (CodeRunner) Parcours de listes

Les fonctions suivantes sont à ajouter à la bibliothèque `liste_entiers`. Vous devez donc pour chacune d'elles,

- ajouter la signature de la fonction au fichier `liste_entiers.h`;
- ajouter la définition de la fonction au fichier `liste_entiers.c`;
- modifier le fichier `test_liste.c` pour tester la nouvelle fonction.

Question 1

Écrivez une fonction `int nb_occurences(int val, cellule_t *liste)` qui renvoie le nombre de fois où la valeur `val` apparaît dans la liste `liste`.

Question 2

Écrivez une fonction `int tous_plus_grand(int val, cellule_t *liste)` qui renvoie 1 (vrai) si tous les éléments de la liste sont supérieurs ou égaux à la valeur `val`, 0 sinon.

Question 3

Écrivez une fonction `cellule_t* Maximum(cellule_t *liste)` qui renvoie un pointeur vers la cellule dont le champ `donnee` a la valeur maximum dans la liste. Si plusieurs cellules vérifient cette condition, la fonction renvoie un pointeur vers la première rencontrée.

Question 4

Écrivez une fonction `int Renvoyer_val_element_pos(int pos, cellule_t* liste)` qui renvoie la valeur du champ `donnee` de l'élément en position `pos` de la liste (le premier élément est en position 0).

Nous ferons l'hypothèse que la fonction est toujours appelée avec une valeur de `pos` inférieure au nombre d'éléments de la liste et supérieure ou égale à 0.

Question 5

Question non notée

Écrivez une fonction itérative `cellule_t *Concatener_it(cellule_t *liste1, cellule_t *liste2)` qui renvoie la liste obtenue en ajoutant les éléments de `liste2` à la fin de `liste1`. Si l'une des deux listes est vide, la fonction renvoie l'autre liste. Si les deux listes sont vides, la fonction renvoie `NULL`. Si aucune des deux listes n'est vide, la fonction modifie `liste1` et renvoie la tête de la liste après ajout des éléments de `liste2`.

Question 6

Question non notée

Écrivez une fonction `int nb_maximum(cellule_t *liste)` qui calcule et renvoie le nombre de fois où la valeur maximale est présente dans la liste. La liste ne devra être parcourue qu'une seule fois.

Exercice 42 – Finir le TD

Faites les exercices non terminés lors de la séance de TD. Ajoutez les fonctions demandées à la bibliothèque `liste_entiers`

Semaine 9 - TME

Objectifs

- Ajout d'un élément dans une liste
- Suppression d'un élément d'une liste
- Parcours simultané de plusieurs listes

Exercices

Nous allons considérer des listes représentant des multi-ensembles. Un multi-ensemble est un ensemble dans lequel chaque élément peut apparaître plusieurs fois. Nous utilisons les types suivants pour représenter un élément de la liste. Le champ `frequence` correspond au nombre de fois où `valeur` apparaît dans l'ensemble. Deux éléments de la liste ne doivent donc jamais avoir la même `valeur`.

```
typedef struct _element_t element_t;
struct _element_t{
    int valeur;
    int frequence;
    element_t *suivant;
};
```

Recopiez les fichiers `multi_ensembles.h`, `multi_ensembles.c` et `test_multi_ensembles.c`. La bibliothèque `multi_ensembles` est utilisée par le programme `test_multi_ensembles`.

Comme en TD, les fonctions que vous allez écrire cette semaine modifient la liste sur laquelle elles s'appliquent soit en y ajoutant un élément soit en en supprimant un. Si l'ajout se fait en tête de liste ou si le premier élément de la liste est supprimé, la tête de liste est modifiée, il faut donc récupérer sa nouvelle valeur. Ces fonctions vont donc prendre en paramètre un pointeur sur un élément de liste (le premier élément de la liste avant modification) et renvoyer un pointeur sur un élément de liste (le premier élément de la liste modifiée).

Attention, ces fonctions modifient la liste initiale et renvoient le pointeur sur le premier élément de la liste après modification. La liste initiale n'existe plus après appel à l'une de ces fonctions, elle a été modifiée.

Exercice 43 – (CodeRunner) Création d'un multi-ensemble

Question 1

Complétez la fonction `Recherche_val` pour qu'elle renvoie un pointeur sur le premier élément de valeur `val` du multi-ensemble. La fonction renvoie `NULL` si la valeur ne se trouve pas dans le multi-ensemble.

Question 2

Complétez la fonction `Ajout_tete_ensemble` pour qu'elle ajoute au multi-ensemble passé en paramètre un élément de valeur `val` et de fréquence `freq`. Si la valeur apparaît déjà dans le multi-ensemble, sa fréquence sera augmentée de `freq`, sinon un nouvel élément sera créé avec la fréquence `freq` et ajouté en tête de liste. La fonction renvoie la tête de la nouvelle liste. Vous pouvez bien-sûr faire appel à la fonction `Recherche_val` pour déterminer si la valeur est présente dans le multi-ensemble ou non.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Exercice 44 – (CodeRunner) Suppression d'un élément d'un multi-ensemble

Question 1

Ajoutez à la bibliothèque une fonction `Supprime_total_element_ensemble` qui prend en paramètre un multi-ensemble et une valeur et qui supprime du multi-ensemble l'élément de la valeur passée en paramètre. L'élément ne doit plus apparaître dans l'ensemble même si initialement sa fréquence était supérieure à 1. Si la valeur ne se trouve pas dans le multi-ensemble, ce dernier n'est pas modifié. La fonction doit renvoyer le pointeur sur le premier élément du multi-ensemble après suppression.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Question 2

Ajoutez à la bibliothèque une fonction `Supprime_element_ensemble` qui cette fois va mettre à jour la fréquence associée à la valeur à supprimer. Si la fréquence est supérieure à 1, elle est simplement diminuée de 1. Si la fréquence est égale à 1, la valeur est supprimée du multi-ensemble. Comme pour la fonction précédente, si la valeur ne se trouve pas dans le multi-ensemble, ce dernier n'est pas modifié. La fonction doit renvoyer le pointeur sur le premier élément du multi-ensemble après suppression.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Exercice 45 – (CodeRunner) Ajout et suppression dans un multi-ensemble trié

Nous allons maintenant créer un multi-ensemble en triant ses éléments par valeur croissante et supprimer des éléments de cet ensemble en prenant en compte son caractère trié.

Question 1

Ajoutez à la bibliothèque la fonction `Ajout_ensemble_trie` pour qu'elle ajoute au multi-ensemble passé en paramètre un élément de valeur `val` et de fréquence `freq` en respectant l'ordre, après ajout l'ensemble est toujours trié en ordre croissant des valeurs. Si la valeur apparaît déjà dans la liste, sa fréquence sera augmentée de `freq`, sinon un nouvel élément sera créé et ajouté à la bonne place dans la liste, avec la fréquence `freq`. La fonction renvoie la tête de la nouvelle liste.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Question 2

Ajoutez à la bibliothèque les fonctions `Supprime_element_ensemble_trie` et `Supprime_total_element_ensemble_trie` qui tiennent compte du caractère trié du multi-ensemble pour supprimer un élément du multi-ensemble (diminuer sa fréquence de 1 ou l'enlever complètement de l'ensemble).

Modifiez le fichier `test_multi_ensembles.c` pour tester vos fonctions.

Exercice 46 – (CodeRunner : non noté) Parcours simultané de plusieurs multi-ensembles

Question 1

Ajoutez à la bibliothèque la fonction `Inclus` qui prend en paramètre deux multi-ensembles et renvoie 1 si le premier est inclus dans le deuxième, 0 sinon. Un multi-ensemble `e1` est inclus dans un multi-ensemble `e2` si tout élément de `e1` apparaît dans `e2` avec une fréquence supérieure ou égale. Nous ferons l'hypothèse que les multi-ensembles sont triés en ordre croissant des valeurs et qu'une même valeur n'apparaît qu'une fois dans un multi-ensemble.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Question 2

Ajoutez à la bibliothèque la fonction `Intersection_vide` qui prend en paramètre deux multi-ensembles et renvoie

1 si leur intersection est vide, 0 sinon. Nous ferons l'hypothèse que les multi-ensembles sont triés en ordre croissant des valeurs et qu'une même valeur n'apparaît qu'une fois dans un multi-ensemble.

Modifiez le fichier `test_multi_ensembles.c` pour tester votre fonction.

Semaine 10 - TME

Objectifs

- Récursivité
- Extraction d'une sous-liste
- Parcours simultané de plusieurs listes

Exercices

Dans ce TP vous allez continuer à enrichir la bibliothèque `multi_ensembles`. Vous devez donc compléter les fichiers `multi_ensembles.h`, `multi_ensembles.c` et `test_multi_ensembles.c` que vous avez écrits la semaine dernière. Même si ce n'est pas explicitement demandé, vous devez bien sûr modifier le fichier `test_multi_ensembles.c` pour tester chaque nouvelle fonction.

Exercice 47 – (CodeRunner) Parcours récursif "simple" et extraction d'une sous-liste

Question 1

Ajoutez à la bibliothèque la fonction **récursive** `taille` qui prend en paramètre un multi-ensemble et qui renvoie le nombre d'éléments qu'il contient (un élément sera compté autant de fois que sa fréquence).

Question 2

Ajoutez à la bibliothèque la fonction `Supprime_frequence_inf_seuil` qui prend en paramètres un multi-ensemble et un entier et qui supprime du multi-ensemble tous les éléments dont la fréquence est inférieure à la valeur passée en paramètre. La liste initiale est donc modifiée. La fonction renvoie la nouvelle tête de liste

Exercice 48 – (CodeRunner - non noté) Opérations ensemblistes - Parcours simultané de plusieurs listes

Dans cet exercice vous allez programmer les principales opérations ensemblistes (inclusion, intersection, différence). Une fois l'une de ces opérations réalisée, ne recommencez pas tout à zéro pour les opérations suivantes, demandez-vous quelles sont les différences entre les opérations déjà réalisées et celle que vous voulez ajouter. Vous vous rendrez alors compte que vous n'avez plus grand chose à faire !

Question 1

Ajoutez à la bibliothèque la fonction `Inclus_rec` qui est la version récursive de la fonction `Inclus` demandée la semaine dernière. La fonction prend en paramètre deux multi-ensembles et renvoie 1 si le premier multi-ensemble est inclus dans le second, 0 sinon. Nous ferons l'hypothèse que les multi-ensembles sont triés par ordre croissant des valeurs et qu'une même valeur n'apparaît qu'une fois dans un multi-ensemble.

Question 2

Ajoutez à la bibliothèque la fonction itérative `Union` qui prend en paramètres deux multi-ensembles triés, crée le multi-ensemble égal à l'union des deux multi-ensembles passés en paramètre et renvoie un pointeur sur le premier élément du nouveau multi-ensemble (les deux multi-ensembles passés en paramètres ne sont pas modifiés). Dans le nouveau multi-ensemble, chaque valeur ne doit apparaître qu'une seule fois avec la bonne fréquence. Pour ajouter un

nouvel élément au multi-ensemble résultat, vous ferez appel à la fonction `Ajout_tete_ensemble` (la liste sera alors triée en ordre décroissant des valeurs).

Question 3

Nous souhaitons maintenant que le multi-ensemble résultat de l'union de deux multi-ensemble soit trié en ordre croissant sur les valeurs des éléments. Nous ne souhaitons pas remplacer les appels à la fonction `Ajout_tete_ensemble` par des appels à la fonction `Ajout_ensemble_trie` car dans ce cas, la liste serait parcourue entièrement à chaque ajout. Nous allons écrire une fonction qui évite ce parcours.

Ajoutez à la bibliothèque la fonction `Ajout_suivant` qui prend en paramètres un pointeur sur un élément d'un multi-ensemble, une fréquence et une valeur et qui ajoute à la suite de l'élément passé en paramètre un nouvel élément dont la valeur et la fréquence correspondent aux paramètres. Le suivant du nouvel élément sera le suivant de l'élément passé en paramètre. La fonction renvoie l'adresse du nouvel élément créé.

Question 4

Ajoutez à la bibliothèque la fonction `Union_triee` qui agit comme la fonction `Union` mais qui fait appel à la fonction `Ajout_suivant` pour que le multi-ensemble résultat soit trié en ordre croissant des valeurs. N'oubliez pas de modifier la tête de liste lorsque c'est nécessaire.

Question 5

Ajoutez à la bibliothèque la fonction `Union_triee_rec` qui agit comme la fonction `Union_triee` mais qui est récursive. Pour obtenir un ensemble trié dans ce cas, l'ajout d'un élément se fera par appel à la fonction `Ajout_tete_ensemble`.

Question 6

Ajoutez à la bibliothèque la fonction `Intersection_triee` qui prend en paramètres deux multi-ensembles triés, crée le multi-ensemble égal à l'intersection des deux multi-ensembles passés en paramètre et renvoie un pointeur sur le premier élément du nouveau multi-ensemble (les deux multi-ensembles passés en paramètres ne sont pas modifiés). Dans le nouveau multi-ensemble, chaque valeur ne doit apparaître qu'une seule fois avec la bonne fréquence (minimum entre la fréquence de chacune des listes). Si l'intersection est vide, la fonction renvoie `NULL`. Le multi-ensemble résultat est trié par ordre croissant des valeurs. À vous de choisir si vous souhaitez écrire une fonction récursive ou non.

Question 7

Ajoutez à la bibliothèque la fonction `Difference_triee` qui prend en paramètres deux multi-ensembles triés, crée le multi-ensemble contenant les éléments présents dans le premier paramètre mais pas dans le second et renvoie un pointeur sur le premier élément du nouveau multi-ensemble (les deux multi-ensembles passés en paramètres ne sont pas modifiés). Dans le nouveau multi-ensemble, chaque valeur ne doit apparaître qu'une seule fois avec la bonne fréquence. Si le résultat ne contient aucune valeur, la fonction renvoie `NULL`. Le multi-ensemble résultat est trié par ordre croissant des valeurs. À vous de choisir si vous souhaitez écrire une fonction récursive ou non.

Les éléments du multi-ensemble résultat et leur fréquence seront calculés de la façon suivante pour chaque valeur `val`. Nous notons `f1` la fréquence de la valeur dans le premier multi-ensemble et `f2` la fréquence dans le second multi-ensemble (si la valeur n'est pas présente dans un multi-ensemble cette fréquence sera égale à 0) ;

- si $f1 \leq f2$, le multi-ensemble résultat ne contiendra pas la valeur `val` ;
- Si $f1 > f2$, le multi-ensemble résultat contiendra $f1 - f2$ fois la valeur `val`.

Question 8

Ajoutez à la bibliothèque la fonction `Xor_triee` qui prend en paramètres deux multi-ensembles triés, crée le multi-ensemble trié contenant les éléments présents dans un des deux multi-ensembles mais pas dans les deux et renvoie un pointeur sur le premier élément du nouveau multi-ensemble (les deux multi-ensembles passés en paramètres ne sont pas modifiés). Vous ferez appel aux fonctions `Union_triee` et `Difference_triee`.

Question 9

L'utilisation des fonctions `Union_triee` et `Difference_triee` a deux inconvénients :

- plusieurs parcours des mêmes listes,
- création de multi-ensembles dont l'usage n'est que temporaire, sans libération de la mémoire utilisée.

Dans cette question nous nous intéressons au problème de la gestion de la mémoire. Pour ne pas utiliser inutilement de la mémoire, il est nécessaire de supprimer les multi-ensembles dont l'usage n'est que temporaire une fois qu'ils ne sont plus utilisés.

Ajoutez à la bibliothèque la fonction `Detruire` qui prend en paramètre un multi-ensemble et qui libère toute la mémoire qu'il occupe, la fonction ne renvoie rien.

En utilisant la fonction `Detruire`, modifiez la fonction `Xor_triee` pour que les multi-ensembles dont l'usage est temporaire soient détruits. Après l'appel à la fonction seuls les multi-ensembles passés en paramètres et le résultat doivent occuper de la mémoire.

Semaine 11 - TME

Objectifs

- Évaluation

Exercices

Cette semaine est consacrée à une évaluation de TP qui durera 1h45 par étudiant. Chaque étudiant ne sera donc présent qu'à une des deux séances de la semaine. La répartition des étudiants sur les deux séances vous sera donnée par votre enseignant.

En plus de répondre aux questions de programmation, vous devrez savoir :

- créer un répertoire ;
- recopier des fichiers d'un répertoire à un autre ;
- compiler un programme ;
- produire un exécutable à partir de plusieurs fichiers .c ;
- soumettre le contenu d'un répertoire.

Semaine 12 - TME

Objectifs

— Pour aller plus loin

Exercices

Ce dernier TME n'utilise aucune nouvelle connaissance. Il permet juste de revoir et d'appliquer les notions vues jusqu'à présent à un petit problème, et de construire un programme un peu plus complexe que d'habitude. Ce dernier TME n'est à faire que si vous avez déjà fini tous les exercices des séances précédentes.

L'objectif du TME est de programmer une interface et (au moins) un joueur automatique pour le jeu Othello (ou Reversi).

Othello se joue à 2, sur un plateau unicolore de 64 cases (8 sur 8), avec des pions bicolores, noirs d'un côté et blancs de l'autre. Le but du jeu est d'avoir plus de pions de sa couleur que l'adversaire à la fin de la partie, celle-ci s'achevant lorsque aucun des deux joueurs ne peut plus jouer de coup légal, généralement lorsque les 64 cases sont occupées. Au début de la partie, la position de départ est indiquée figure 11. Les noirs commencent. Le pion en haut à gauche de la figure correspond au prochain pion à poser (donc à la couleur du joueur dont c'est le tour).

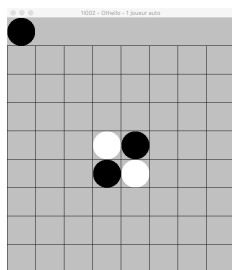


FIGURE 11 – Plateau de jeu de Othello au départ

Chacun à son tour, les joueurs vont poser un pion de leur couleur sur une case vide, adjacente à un pion adverse. Chaque pion posé doit obligatoirement encadrer un ou plusieurs pions adverses avec un autre pion de sa couleur, déjà placé. Le joueur retourne alors le ou les pions adverse(s) qu'il vient d'encadrer. Les pions ne sont ni retirés du plateau de jeu, ni déplacés d'une case à l'autre. On peut encadrer des pions adverses dans les huit directions et plusieurs pions peuvent être encadrés dans chaque direction.

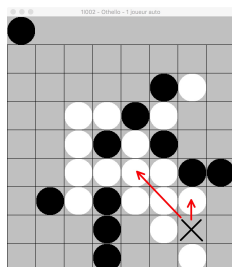


FIGURE 12 – Plateau de jeu de Othello en cours de jeu

Par exemple (cf. figure 12), si le joueur noir joue à l'endroit marqué par une croix, il retourne deux pions blancs en diagonale et un autre pion au dessus. Il n'y a pas de réaction en chaîne : les pions retournés ne peuvent pas servir à retourner d'autres lors du même tour de jeu.

Si aucune case vide ne permet le retournement de pions adverses, le joueur est bloqué et passe son tour et c'est à l'adversaire de jouer.

La partie se termine si les deux joueurs sont bloqués ou si toutes les cases sont remplies.

Le programme sera organisé en quatre fichiers .c et trois fichiers .h que vous devez recopier :

- Affichage.c et Affichage.h qui contiennent toutes les fonctions permettant l'affichage du plateau de jeu. Toutes ces fonctions vous sont fournies.
- ListePos.c et ListePos.h qui contiendront toutes les fonctions de manipulation de listes que vous écrirez.
- Othello.c et Othello.h qui contiendront toutes les fonctions concernant le jeu lui même.
- Main.c qui contiendra la fonction main.

Le plateau de jeu est un tableau à deux dimensions de taille $H \times H$, H étant définie (avec un **#define**) comme valant 8 dans Othello.h. La case en haut à gauche sera la case $(0, 0)$. Chaque case de ce tableau peut soit être vide, soit être occupée par un pion noir, soit être occupée par un pion blanc. Dans le fichier Othello.h sont aussi définies ces trois valeurs possibles :

```
#define VIDE 0
#define NOIR 1
#define BLANC 2
```

Exercice 49 – Règles du jeu Othello

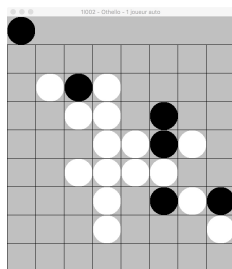


FIGURE 13 – Plateau de jeu de Othello en cours de jeu

Question 1

Le plateau de jeu d'une partie en cours est présenté figure 13. Pour chacune de ces cases, dites si la position est jouable ou non pour les noirs et si oui, combien de pions blancs vont être retournés. Dans les couples (i, j) , i correspond à la ligne et j à la colonne.

- $(2, 1)$
- $(3, 2)$
- $(5, 2)$
- $(7, 2)$

Question 2

Quel est le nombre de positions jouables pour les noirs pour le plateau présenté figure 14 ?

Exercice 50 – Affichage du plateau, prise en main de l'affichage

Les fonctions d'affichage que vous aurez à utiliser sont :

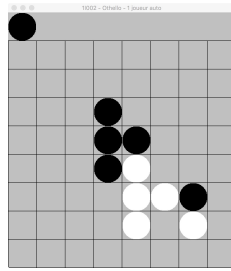


FIGURE 14 – Plateau de jeu de Othello en cours de jeu

- **void** Creer_fenetre(**char** *ModeStr); qui prend en argument le titre de la fenêtre (correspondant au mode de jeu : 2 joueurs...) et crée une fenêtre avec le plateau de jeu;
- **void** Detruire_fenetre(); qui détruit la fenêtre avec le plateau de jeu;
- **int** Loop_until_play(**int** plateau[H][H], **int** *pi, **int** *pj, **int** joueurCourant); qui prend en argument le plateau de jeu, deux pointeurs vers des entiers et le joueur courant (NOIR pour le joueur des pions noirs et BLANC pour le joueur des pions blancs). Cette fonction attend que le joueur clique dans la fenêtre et met à jour (les variables dont l'adresse est passée en paramètre) avec les indices de la case du plateau où le joueur a cliqué. Cette fonction renvoie -1 si la fenêtre a été fermée (0 sinon).

Question 1

Dans le fichier `Othello.c`, vous complèterez la fonction

```
void Initialiser_plateau(int plateau[H][H]);
```

qui met toutes les cases du plateau donné en argument à vide sauf les 4 cases du milieu qui doivent contenir des pions blancs ou noirs comme présenté précédemment.

Question 2

Dans le fichier `Othello.c`, vous complèterez la fonction **int** Autre_joueur(**int** joueur); qui prend en argument la couleur du joueur et qui renvoie BLANC (2) si le joueur est NOIR (1) et NOIR (1) sinon.

Question 3

Dans le fichier `Main.c`, vous complèterez la fonction `main` qui appellera les fonctions précédentes pour :

- initialiser le plateau;
- créer une fenêtre;
- Tant que la fenêtre n'est pas fermée, mettre alternativement un pion noir ou un pion blanc là où l'utilisateur a cliqué; pour ajouter un pion dans une case (i, j) donnée il suffit de mettre la couleur jouée dans cette case;
- détruire la fenêtre.

Pour compiler votre programme, il vous faudra compiler en même temps tous les fichiers `.c` dont vous utilisez des fonctions. Ainsi, dans ce premier exercice, vous devrez compiler à la fois `Affichage.c`, `Othello.c` et `Main.c`. Comme nous utilisons la bibliothèque SDL2, il faut inclure cette bibliothèque; vous devrez ajouter `'sdl2-config --libs'` à votre ligne de commande de compilation qui deviendra² :

```
gcc -Wall -o Othello Affichage.c Othello.c Main.c 'sdl2-config --libs'
```

Si vous préférez, nous vous avons aussi fourni un *Makefile* qui permet d'effectuer facilement la compilation. Vous apprendrez éventuellement l'année prochaine comment cela fonctionne. Pour l'utiliser, et compiler votre programme, il suffit de taper `make` dans le terminal.

Si tout se passe bien, vous pourrez mettre des pions blancs ou noirs sur le plateau de jeu, mais sans respecter les règles de placement. C'est ce que nous allons programmer dans la suite du sujet.

2. Si vous souhaitez compiler votre programme sous mac, il vous faut avoir la bibliothèque SDL2 et remplacer le `'sdl2-config --libs'` avec `-I/Library/Frameworks/SDL2.framework/Headers -framework SDL2`

Exercice 51 – Liste de positions jouables

Nous allons maintenant restreindre les positions où un joueur peut placer son pion aux positions permettant d'encadrer un ou plusieurs pions adverses avec un autre pion de sa couleur, déjà placé. Pour cela, pour une couleur donnée et un joueur donné, nous allons construire la liste des positions possibles. Les fonctions de manipulation de listes sont à écrire dans le fichier `ListePos.c`. La structure d'un élément de liste vous est donnée dans le fichier `ListePos.h`. La voici :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
    int i, j;
    int val;
    cellule_t *suivant;
};
```

Elle comporte 4 champs : les coordonnées `i` et `j` de la position sur le plateau, `val` le nombre de points que ce coup rapporterait (*ie.* le nombre de pions adverses qui seraient retournés) et `suivant`, le pointeur vers l'élément suivant de la liste.

Question 1

Dans le fichier `ListePos.c` vous complèterez les fonctions suivantes en vous inspirant des fonctions vues lors des TD et TP précédents :

- `cellule_t *Creer_cellule(int i, int j, int val)` qui prend en argument les coordonnées `i` et `j` ainsi que `val`, le nombre de points associés au coup, qui alloue un élément de liste et qui renvoie un pointeur vers cet élément;
- `void Afficher_liste(cellule_t *liste)` qui prend en argument une liste et qui affiche celle ci dans le terminale (cela vous permettra de vérifier que vos fonctions sont correctes);
- `void Detruire_liste(cellule_t *liste)` qui prend en argument une liste `liste` et qui libère toute la mémoire réservée pour la liste.
- une des deux fonctions suivantes :
 - `cellule_t *Insérer_tete(int i, int j, cellule_t *liste, int val)` qui prend en argument les coordonnées `i` et `j`, le nombre de points associés au coup `val` (qui peut être à 0 si nécessaire) et une liste `liste` (qui peut être NULL), qui alloue un nouvel élément de liste, l'ajoute en tête de list et qui renvoie un pointeur vers la liste;
 - `cellule_t *Insérer_decrois(int i, int j, cellule_t *liste, int val)` qui prend en argument les coordonnées `i` et `j`, le nombre de points associés au coup `val` (qui peut être à 0 si nécessaire) et une liste `liste` (qui peut être NULL), qui alloue un nouvel élément de liste, l'ajoute dans la liste qui renvoie un pointeur vers la liste; attention, la liste est ordonnée par ordre décroissant des valeur du champs `val`; cet ordre devra être maintenu;

Vous prendrez soin de bien tester vos fonctions en les appelant (transitoirement) dans la fonction `main`.

Exercice 52 – Une position est elle jouable ?

Pour savoir si une position est jouable nous allons compter le nombre de pions adverses qui seraient retournés si ce coup était joué. Il y a 8 directions possibles dans lesquelles les pions sont retournés : vers le haut, le bas, la gauche, la droite, mais aussi les 4 diagonales. La première fonction `Gain_dir` renverra le nombre de pions retournés pour une direction donnée tandis que `Est_jouable_gain` renverra le nombre de pions retournés dans toutes les directions. Enfin, la fonction `Trouver_liste_pos_jouables` renverra la liste de toutes les positions jouables. Ces trois fonctions sont à compléter dans le fichier `Othello.c`

Question 1

Vous allez écrire la fonction

```
int Gain_dir(int plateau[H][H], int iLigne, int iCol, int dirLigne, int dirCol, int
couleurQuiJoue)
```

qui prend en argument le plateau de jeu `plateau`, les coordonnées de la case `iLigne` et `iCol`, la direction (`dirLigne` et `dirCol`) et la couleur du joueur qui joue `couleurQuiJoue`. La direction est représentée par deux entiers valant -1, 0 ou 1. Ainsi, si par exemple `dirLigne` vaut -1 et `dirCol` vaut 0, la direction sera celle de même colonne et d'indices de ligne décroissant, c'est-à-dire vers le haut (pour rappel, la case en haut à gauche est la case d'indice (0, 0)). La fonction renverra le nombre de pions de couleur adverse potentiellement retournés. Pour rappel, s'il n'y a pas de pion de la couleur du joueur courant, aucun pion ne sera retourné.

Question 2

Vous allez maintenant écrire la fonction

```
int Est_jouable_gain(int plateau[H][H], int iLigne, int iCol, int couleurQuiJoue)
```

qui prend en argument le plateau de jeu `plateau`, les coordonnées de la case `iLigne` et `iCol`, et la couleur du joueur qui joue `couleurQuiJoue`. Cette fonction renverra le nombre de pions de couleur adverse potentiellement retournés dans toutes les directions.

Question 3

Vous allez maintenant écrire la fonction

```
cellule_t *Trouver_liste_pos_jouables(int plateau[H][H], int couleurQuiJoue)
```

qui prend en argument le plateau de jeu `plateau` et la couleur du joueur qui joue `couleurQuiJoue`. Cette fonction renvoie la liste des positions jouables, chaque position contenant aussi le nombre de pions potentiellement retournés dans le champs `val` de la structure `cellule_t`.

Exercice 53 – Limitation du jeu aux positions jouables

Nous voulons maintenant modifier la fonction `main` pour que seules les cases jouables puissent être jouées (*ie.* qu'un pion ne soit ajouté que si la case est jouable). Pour cela, nous aurons besoin de savoir si la case sur laquelle l'utilisateur a cliqué est jouable.

Question 1

Ecrivez, dans le fichier `ListePos.c` une fonction

```
int Est_dans_liste(cellule_t *liste, int i, int j)
```

qui prend en argument une liste `liste` et les coordonnées `i` et `j` d'une case et qui renvoie 1 si une case ayant ces coordonnées existe dans la liste et 0 sinon.

Question 2

Vous modifierez ensuite votre fonction `main` pour que seules les cases jouables puissent être jouées. Si l'utilisateur clique sur une case qui n'est pas jouable, le plateau n'est pas modifié et un autre clic dans une case est attendu. Attention à ne pas générer de fuite mémoire avec les listes.

Question 3

Nous allons maintenant réfléchir aux conditions de terminaison du jeu. Le jeu se termine si les deux joueurs sont bloqués, ce qui peut arriver si le plateau est plein, ou non. On sait qu'un joueur est bloqué si sa liste de positions jouables est vide. Vous modifierez votre fonction `main` pour que le jeu s'arrête si la fenêtre est fermée ou que les deux joueurs sont bloqués.

Exercice 54 – Othello à deux joueurs

Pour terminer notre implémentation du jeu Othello il nous faut encore retourner les pions, et compter le score de chaque joueur.

Question 1

Dans le fichier `Othello.c`, écrire la fonction

```
void Retourner_pions(int plateau[H][H], int iLigne, int iCol, int dirLigne, int dirCol, int couleurQuiJoue)
```

qui prend en argument un plateau (`plateau`), une case jouée de coordonnées `iLigne` et `iCol`, une direction (`dirLigne`, `dirCol`) et une couleur de joueur (`couleurQuiJoue`). Elle retournera tous les pions adverses qui seront encadrés par le pion joué et l'autre pion de même couleur le plus proche dans la direction donnée. Attention, il est possible qu'aucun pion ne doive être retourné dans cette direction.

Question 2

Vous complèterez ensuite la fonction

```
void Jouer_pion(int plateau[H][H], int iLigne, int iCol, int couleurQuiJoue)
```

qui pour un plateau (`plateau`), une case jouée de coordonnées `iLigne` et `iCol`, une couleur de joueur (`couleurQuiJoue`) retourne tous les pions adverses qui sont encadrés par le pion joué et un autre pion de même couleur dans toutes les directions.

Question 3

Vous modifierez votre fonction `main` pour que les pions soient retournés lorsqu'un pion est joué.

Question 4

Il serait plus satisfaisant maintenant de savoir qui a gagné. Vous écrirez pour cela une fonction `Nb_pions` (dont vous trouverez le prototype) qui renvoie le nombre de pions noirs et le nombre de pions blancs présents sur le plateau. Vous utiliserez cette fonction dans le `main` pour afficher dans le terminal qui a gagné et de combien.

Exercice 55 – Othello avec un joueur automatique simple

Nous allons maintenant implémenter un joueur automatique simple : il jouera à chaque tour le coup rapportant le plus de points pour ce tour. Le joueur automatique sera toujours le joueur blanc. Pour cela, il suffit de trouver dans la liste des coup possibles le meilleur coup.

Question 1

Ecrivez dans le fichier `ListePos.c` une fonction `Max_liste` - dont vous déterminerez vous même le prototype - qui Renvoie le meilleur coup pour ce tour de jeu. Il y a plusieurs possibilités :

- si votre liste est déjà triée (*ie.* votre fonction d'insertion ajoute les éléments en respectant l'ordre du champs `val`), le premier élément de la liste est le maximum,
- si votre liste n'est pas triée (*ie.* vous pouvez soit trier la liste puis renvoyer le premier élément, soit parcourir la liste pour trouver le maximum).

Question subsidiaire, quelle est, selon vous, la méthode la plus efficace ?

Question 2

Nous avons maintenant deux modes possibles pour notre jeu. Le mode sera choisi par l'utilisateur en ajout un argument (0 ou 1) sur la ligne de commande permettant de lancer le jeu. Ces arguments sont accessibles avec les arguments `argc` et `argv` de la fonction `main`. Suivant l'argument, le second joueur sera donc soit un joueur humain soit le joueur automatique. Vous modifierez votre fonction `main` pour que ce soit le cas.

Exercice 56 – Othello avec un joueur automatique à deux coups

Nous allons maintenant implémenter un joueur automatique normalement un peu plus malin. Nous voulons qu'il cherche le meilleur coup possible en regardant un coup plus loin. Ainsi, pour chaque coup possible au tour `i`, il supposera que l'adversaire au tour `i+1` jouera le meilleur coup et calculera quel est alors le meilleur coup possible au tour `i+2`.

Question 1

Vous écrirez, dans le fichier `Othello.c` la fonction

`void Joueur_Auto_2(int plateau[H][H], int *pi, int *pj)` qui met à jour les variables pointées par `pi` et `pj` avec les coordonnées du meilleur coup prédit en regardant deux tours plus loin.

Pour cela, les étapes sont :

1. Calculer la liste des positions jouables (coups) du joueur courant au tour i ;
2. Pour chaque coup possible c :
 - (a) Recopier le plateau dans un plateau temporaire ;
 - (b) Jouer le coup c sur ce plateau temporaire ;
 - (c) Calculer la liste des positions jouables du joueur adverse au tour $i+1$;
 - (d) Trouver le coup rapportant le plus de point au joueur adverse au tour $i+1$; conserver le nombre de points de ce coup ;
 - (e) Jouer ce coup sur le plateau temporaire ;
 - (f) Calculer la liste des positions jouables du joueur courant au tour $i+2$ avec ce plateau temporaire ;
 - (g) Trouver le coup rapportant le plus de point au joueur courant ; conserver le nombre de points de ce coup ;
 - (h) Si le nombre de points gagnés pour le joueur courant pour ces trois coups successifs est meilleur que ceux qui ont été vus auparavant, conserver le coup c comme étant celui à jouer.

N'oubliez pas de détruire les listes au fur et à mesure.

Question 2

Vous modifierez votre fonction `main` pour proposer un troisième mode de jeu avec ce joueur automatique.