UFR d'Ingénierie

13 janvier 2020

L1 (2019-2020)

Éléments de programmation en C (LU1IN002)

TME

Semaines 1 à 6

Semaine 1 - TME

Objectifs

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

Exercices

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine1`.

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 :

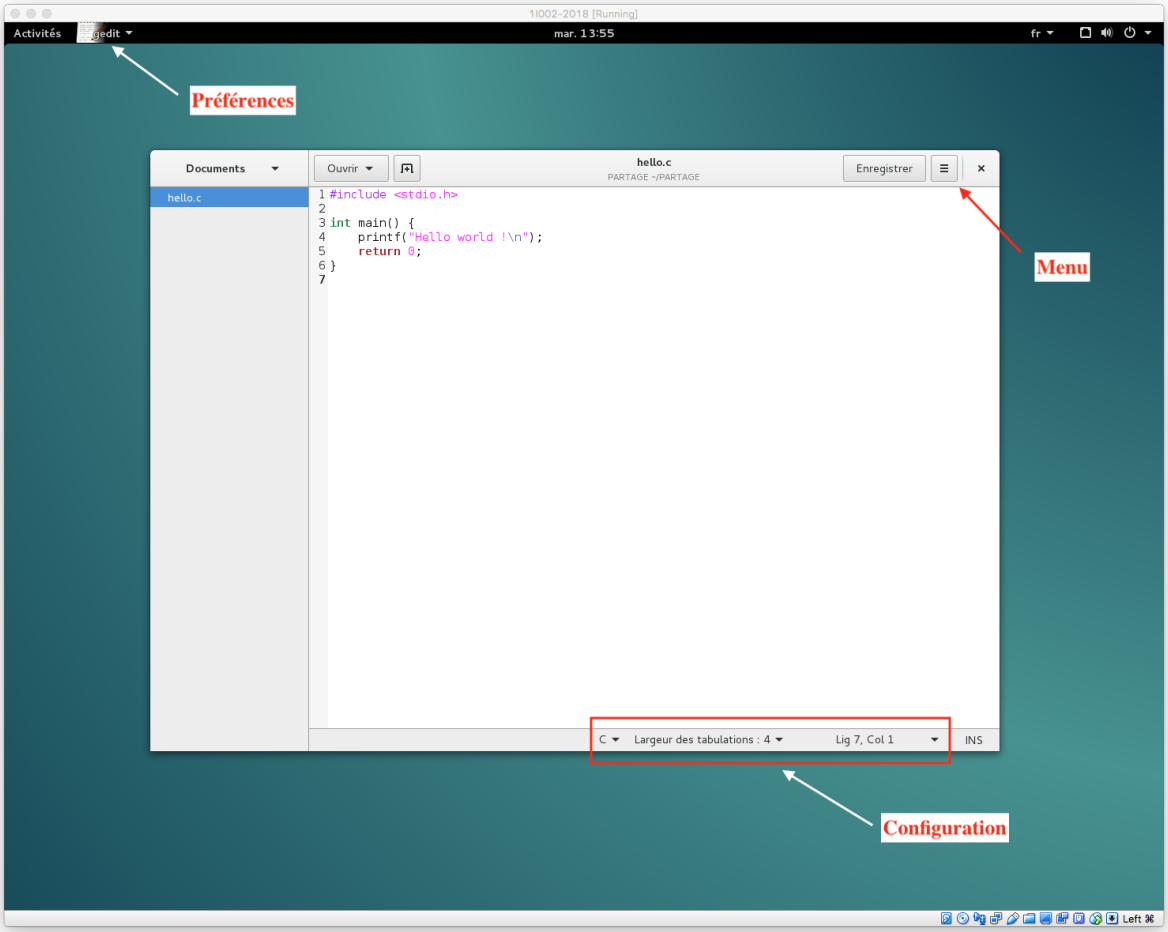


FIGURE 1 – L'éditeur `gedit`

- 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 alora 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 pourrons 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'affichage 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'aucune avertissement n'apparaît.

Exercice 3 – Soumission des TP

La soumission des exercices de TP se fait à partir du script `remcptcini`, qui prend en argument le numéro de carte d'étudiant du binôme (celui de la personne qui ne s'est pas connectée, le numéro de carte d'étudiant de la personne

connectée étant automatiquement trouvé) ou 0 si vous êtes seul, le numéro de semaine, ainsi que le répertoire contenant vos fichiers source.

Pour soumettre seul (sans binôme) le compte rendu de la semaine 1 qui est dans le répertoire `semaine1`, vous devrez exécuter, dans un terminal, la commande suivante depuis le répertoire père du répertoire `semaine1` :

`remcptcini 0 1 semaine1`

Si vous soumettez alros que le répertoire courant est le répertoire `semaine1`, la commande à exécuter est la suivante (le . indique le répertoire courant) :

`remcptcini 0 1 .`

Remarque : ce script a besoin que les listes définitives des inscrits aux groupes soient connues. Il est donc possible qu'il ne fonctionne pas pour tous la première semaine.

Exercice 4 – 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 dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine1`.

```
#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 5 – Calcul du discriminant

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 – 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 – 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 £

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 age.

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 le 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

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine2`.

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 programe 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.

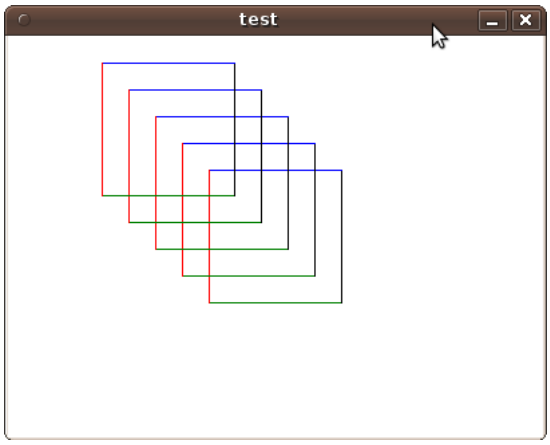


FIGURE 2 – Translation carrés

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

Exercice 11 – 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 personnne contamine `x` nouvelles personnes chaque jour, nous voulons savoir en combien de jours un certain pourcentage de la population de la ville sera contaminée.

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 qui représente à un pourcentage (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.

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.

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=a*x+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.

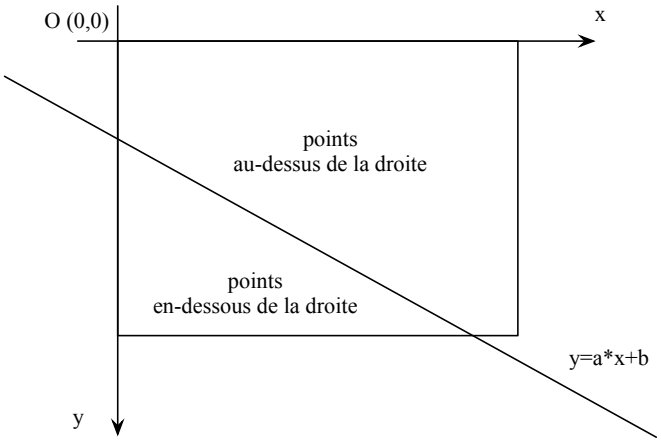


FIGURE 3 – Repère associé à la fenêtre graphique

Question 2

Ecrivez une fonction `affiche` qui prend en paramètres les entiers `a` et `b` qui caractérisent la droite d'équation

`y=a*x+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`,
- 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 – 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 £

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.

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.

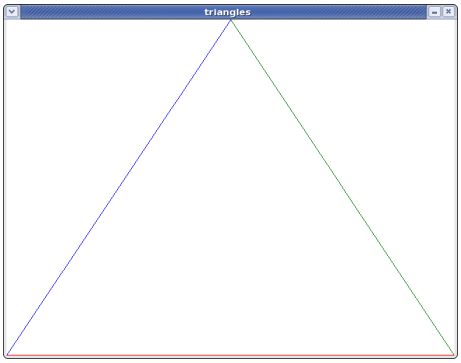


FIGURE 4 – Premier triangle

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 !

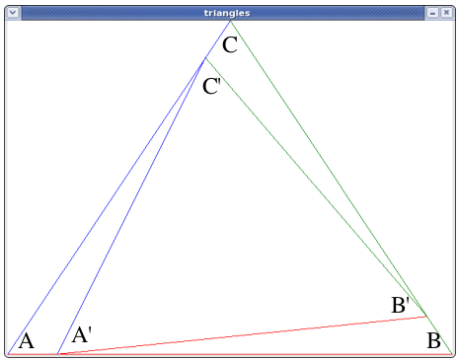


FIGURE 5 – Second triangle

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

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 $(x_A - x_B)^2 + (y_A - y_B)^2$.

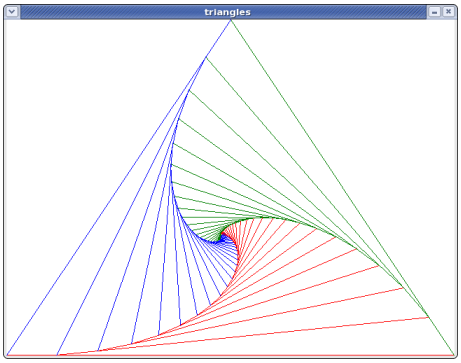


FIGURE 6 – Résultat final

Semaine 3 - TME

Objectifs

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

Exercices

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine3`.

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 debogueur `ddd` pour vérifier que vos deux variables mènent bien au même emplacement mémoire.

Exercice 17 – 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'intialisation 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 nombres 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 – 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 – Calcul de la moyenne, du minimum et du maximum

Le format de cet exercice (fichier à compléter, utilisation de primitives **`#define`**, ...) correspond au format des exercices que vous devrez résoudre lors du contrôle de TME.

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 – 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 $a * x^2 + b * x + c$ 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 un 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 indépendemment 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 foncion `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 une contrôle individuel de 45 minutes. 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 ;
— soumettre le contenu d'un répertoire.

Exercices

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine4`.

Exercice 23 – 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 – 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

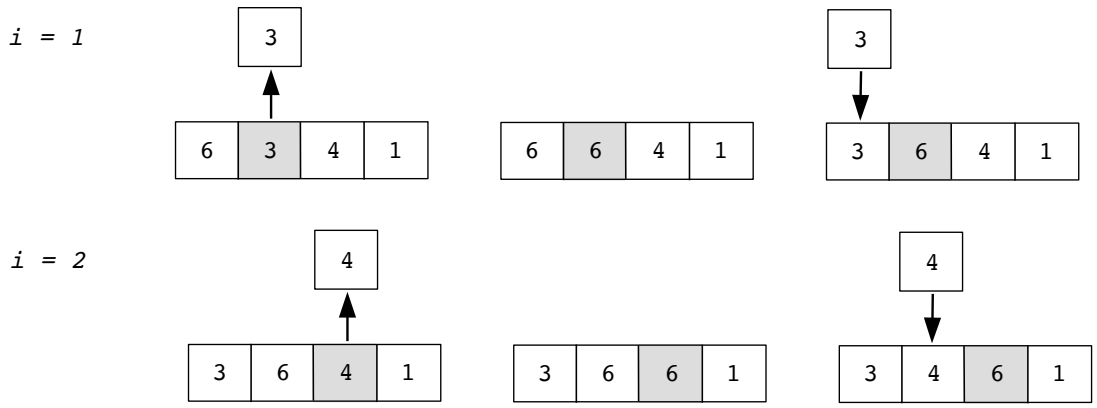
Dans l'exercice précédent, nous avons utilisé un mécanisme d'insertion permettant de garantir qu'un tableau reste trié lors de l'ajout de nouveaux éléments. Dans exercice, nous allons utiliser le mécanisme d'insertion pour écrire un programme qui trie par ordre croissant un tableau dont les éléments 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.



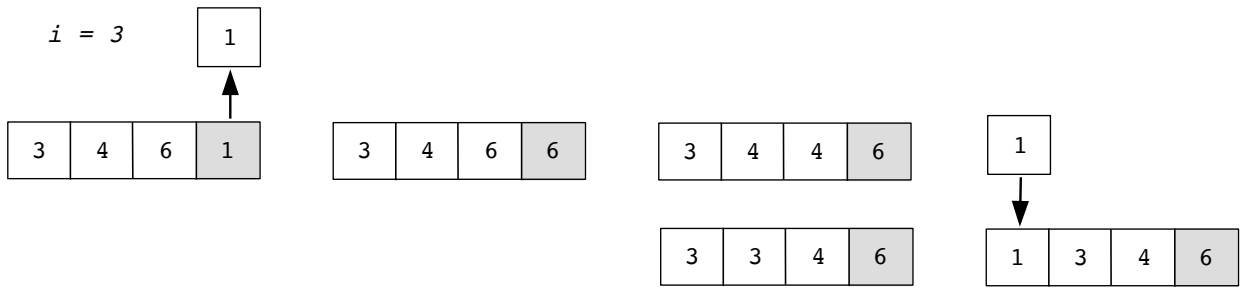


FIGURE 7 – Tri par insertion d'un tableau de 4 cases

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.

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 – 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

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine5`.

Exercice 27 – 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 – 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 ce 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 – 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
```

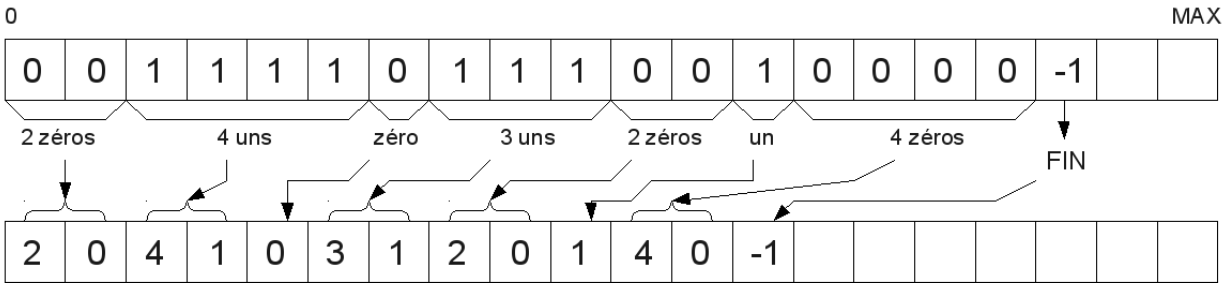
Dans notre programme C, les données brutes et les données compressées seront stockées dans deux tableaux d'entiers de taille `(MAX+1)` appelés respectivement `brut` et `compress`.

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 simple [1] : 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

Si vous devez recopier des fichiers pour réaliser ce TP, ces derniers se trouvent dans le répertoire `/Infos/lmd/2019/licence/ue/LU1IN002-2020fev/Semaine6`.

Exercice 32 – 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 – 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_menu12q()` 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_gagnee(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.