



TD Semaine 1 Variables, expressions et alternatives

Version du 18 décembre 2014

Objectif(s)

- ★ Variable : type, valeur, initialisation, affectation (opérande gauche et opérande droite)
- ★ Expression : type, valeur, opérateurs
- ★ Type de base : **int**, **float**
- ★ instructions conditionnelles : **if** et **if-else**
- ★ Fonctions de sorties
- ★ Directive **#include**

Exercice(s)

Exercice 1 (*base*) – Gestion d'un compte bancaire

Nous souhaitons écrire un programme C de gestion des mouvements d'argent sur un compte bancaire. Chaque compte est identifié par un *numéro* à 6 chiffres. Il héberge une somme d'argent, appelée *solde*. Le solde ne peut être négatif que si le compte a une autorisation de *découvert*. Toute transaction correspond au transfert d'une *somme*. On distingue les *dépôts* et les *retraits*.

1. Donnez l'ensemble des variables utilisées dans ce programme, correspondant aux concepts en *italique* dans le texte, en précisant leur type et écrivez l'instruction C qui permet de les déclarer.
2. Écrivez un programme C complet dans lequel les variables ont été déclarées et initialisées avec les valeurs suivantes :
 - Le compte présente un solde positif de 123,5 €.
 - Le transfert est un *retrait* de 12,0 €.Le programme affichera le solde du compte.
3. Ajoutez les instructions modifiant le type d'opération en dépôt et le montant de la transaction à 35,5 €.
4. Le compte est rémunéré avec un intérêt de 1,5%. Écrivez une instruction qui modifie la valeur du solde.
5. Finalement, l'impôt prélève les 2/9^{èmes} du compte. Écrivez une instruction qui calcule le nouveau solde.
6. Donnez les instructions permettant d'effectuer l'opération souhaitée en fonction des valeurs des variables *somme*, *solde* et *b_retrait*.
7. Complétez les instructions précédentes de manière à garantir que le solde du compte ne devient jamais négatif. Le programme doit signaler à l'utilisateur si une opération ne peut pas être effectuée.
8. On souhaite ajouter une autorisation de découvert. Si la banque vous autorise un découvert, vous pouvez retirer plus d'argent que ce que vous avez sur votre compte. Le découvert autorisé est défini par une variable *somme_decouvert* positive ou nulle.
Déterminez la condition pour effectuer un retrait, en tenant compte de l'autorisation de découvert.
9. Il se peut que certaines personnes n'aient pas de limite de découvert. Dans ce cas, la variable *somme_decouvert* prend la valeur -1.
Si nécessaire modifiez le programme précédent pour qu'il prenne en compte les personnes sans limite de découvert.

Exercice 2 (renforcement) – Retour sur la division

Considérons le programme suivant :

```
#include <stdio.h>

int main() {
    int a = 3;
    int b = 7;
    int c = 4;
    int d;

    a = a + c - b;
    d = 1/2;
    c = b/a;
    b = c;

    printf("a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);
    return 0;
}
```

Quelles sont les valeurs des variables a, b, c et d à la fin du programme ? Justifiez.

Exercice 3 (renforcement) – Affichages, conversions, types

1. Écrivez un programme qui déclare un entier p initialisé à un poids en kilogrammes et une variable réelle t initialisée à une taille en mètres. Le programme affichera l'indice de masse corporelle (IMC) correspondant, calculé par la formule : $IMC = \frac{p}{t^2}$.
2. Modifiez le programme pour que la taille soit maintenant initialisée par un réel en centimètres.
3. Écrivez un programme qui cette fois stocke le poids (en kg) et la taille (en cm) dans des variables de type entier, initialise une variable IMC de type réel en fonction du poids et de la taille et l'affiche.
4. Enrichissez votre programme pour qu'il affiche :
 - vous êtes de corpulence moyenne si $18,5 \leq IMC < 25$
 - vous êtes en dessous de la corpulence moyenne si $IMC < 18,5$
 - vous êtes au dessus de la corpulence moyenne si $IMC \geq 25$Définissez un jeu de test.

Exercice 4 (entraînement) – Prix d'une place de cinéma

Un cinéma pratique les tarifs suivants :

- tarif normal : 11 €
- séance débutant avant 11 heures : 6,70 €
- moins de 14 ans : 4 €
- moins de 26 ans du lundi au vendredi : 4,90 €
- moins de 26 ans samedi et dimanche : 6,90 €.

Écrivez un programme qui, étant donnés un age (age, variable entière), un jour (jour, variable entière (lundi=1, dimanche=7)) et une heure qui correspond à l'heure de début de la séance (heure, variable réelle), affiche le tarif à payer. Vous devez bien sûr calculer le tarif le plus avantageux pour le spectateur.

Exercice 5 (entraînement) – Variables et Expressions arithmétiques

Écrivez un programme qui, étant donnée une somme d'argent (entière), l'affiche en un nombre minimal de pièces-billets de 5, 2 et 1 €.

Indice : vous devez utiliser l'opérateur % qui donne le reste de la division euclidienne du premier opérande par le second. Par exemple, $13\%5$ prend la valeur 3. Si vous avez 13 €, il vous faudra $13/5 = 2$ billets de 5 € et il reste $13\%5 = 3$ € à répartir en pièces.

Savoir-faire acquis

- ★ Écrire un programme permettant de résoudre un problème simple (identification des variables nécessaires, écriture des instructions,...) ;
- ★ Connaître et utiliser les instructions pour afficher des nombres ;
- ★ Connaître et utiliser les opérateurs arithmétiques (addition, soustraction, multiplication, division entière, reste de la division entière) et les opérateurs booléens ;
- ★ Connaître et savoir utiliser les deux principales instructions conditionnelles (**if** et **if-else**)
- ★ Identifier et savoir éviter des erreurs classiques sur les opérations numériques : division par zéro, confusion division entière/réelle.



TME semaine 1

Environnement de travail et premiers programmes

Version du 18 décembre 2014

Objectif(s)

- ★ Se familiariser avec l'environnement de programmation en C.
- ★ Édition, compilation et exécution d'un programme C.
- ★ Types de bases, instructions conditionnelles, fonctions de sortie.

Exercice(s)

Exercice 1 (*base*) – Démarrer

1. Connectez-vous au système Linux à l'aide de votre login et de votre mot de passe. L'ensemble des TME se fera sous l'environnement pédagogique Geany adapté spécialement pour l'UE 1I002.

Dans ce TME, nous allons mettre en place les répertoires de travail et apprendre à utiliser Geany.

2. Il est important de ranger son travail dans des dossiers séparés. C'est pourquoi nous allons créer des dossiers correspondant aux différentes séances de TME.

Ouvrez votre dossier personnel ; (menu "Raccourcis", "Dossier Personnel" dans les salles de TME)

Créez-y un dossier 1I002. Dans ce dossier, créez les dossiers TME1, TME2, ... ,TME12.

3. Ouvrez l'application Geany dans le menu "Applications/Programmation".

Attention : Il est possible que les plugins 1I002 pour Geany ne soient pas activés automatiquement. En particulier, si vous voyez plus de 8 boutons sur la barre de raccourcis, c'est probablement le cas.

Allez, dans le menu "Outils" et cliquez sur "Gestionnaire de plugins". Cochez les plugins suivants : "Hide widget", "plugin indent" et "Simple debug plugin" puis cliquez sur "Valider".

Un autre problème fréquent, surtout pour les étudiants redoublant, est que d'anciens fichiers de configuration perturbent le bon fonctionnement de votre installation Geany. Pour éviter tout problème, supprimez les fichiers de configuration de Geany, situés dans `~/.config` en tapant la commande :

```
rm -rf ~/.config/geany
```

dans un terminal ou en cliquant sur "voir les fichiers cachés" et en supprimant le répertoire `geany` dans le répertoire caché `.config`.

Exercice 2 (*base*) – Prise en main de l'environnement

L'éditeur Geany pour 1I002 (figure 1) se présente sous la forme d'une fenêtre comprenant quatre parties :

- Tout en haut, les menus et les boutons de raccourcis. Ils permettent d'ouvrir et de sauvegarder vos programmes dans des fichiers, de compiler votre programme et de l'exécuter, de faire les opérations d'édition habituelles (copier, coller, rechercher...).

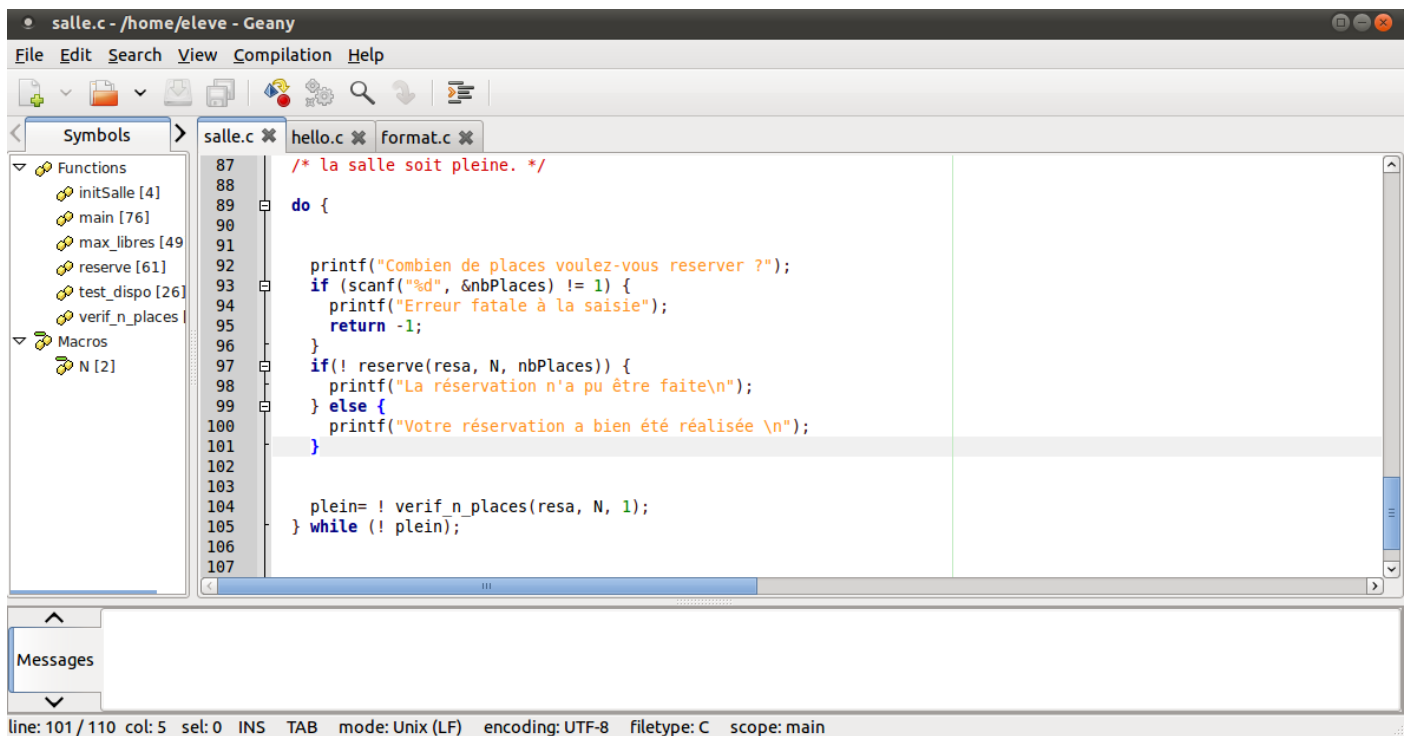


FIGURE 1 – L’environnement de travail en 1I002

- Au milieu à gauche, la liste des symboles et fonctions de votre programme (très pratique pour naviguer dans un programme un peu long).
- Au milieu à droite, la zone d’édition dans laquelle vous saisissez vos programmes.
- En bas, la zone de compilation qui affiche le résultat de la compilation. En cliquant sur une erreur de compilation, Geany positionne le curseur directement sur la ligne incriminée.

Important : Chaque semaine, vous devrez récupérer des fichiers en utilisant le menu *Fichier/Ouvrir* de Geany, puis enregistrer une copie dans votre répertoire local (menu *Fichier/Enregistrer sous* de Geany) pour pouvoir travailler sur le programme.

Les fichiers à récupérer sont dans le répertoire :

/Infos/lmd/2014/licence/ue/1I002-2015fev

rangés dans des sous-répertoires (un pour chaque semaine de TME).

Astuce : Vous pouvez "mémoriser" vos répertoires dans le menu d’ouverture de fichier, pour revenir plus vite à votre dossier 1I002 ou à celui des enseignants.

1. Ouvrez le fichier `tp1-ex1.c` et enregistrez-le dans votre répertoire 1I002/TME1.
2. Compilez et exécutez ce programme.

Attention : Vous devez enregistrer votre programme dans votre propre répertoire pour le compiler.

3. Modifiez la ligne 5 pour initialiser `a` à 14, et exécutez le programme. Que remarquez-vous pour rapport au résultat de la question précédente ? Expliquez.
4. Modifiez ce programme pour qu’il affiche la valeur de $a \times 3$ (au lieu de $a \times 2$).

Attention : Vous devez enregistrer votre programme dans votre propre répertoire pour le recompiler.

Exercice 3 (renforcement) – Mise au point (Debug)

1. Chargez le fichier `tp1-debug.c` et compilez-le. Que constatez-vous ?
2. Corrigez les trois erreurs et recompilez votre programme.
3. En C comme dans tous les langages de programmation, il ne faut pas avoir aveuglément confiance dans le compilateur. Un programme, c'est vrai, ne doit **jamais** produire d'erreur à la compilation (c'est le minimum qui sera exigé pour toutes vos soumissions de TME). Mais ça n'est pas parce que la compilation n'a pas produit de message d'erreur ou d'avertissement que le programme est correct et ne contient plus aucune erreur. En effet :
 - Il existe des erreurs dites "d'exécution", qui n'apparaissent qu'à l'exécution. Par exemple, des boucles infinies (nous en verrons dans les prochaines semaines).
 - Le compilateur C effectue certaines opérations de manière implicite, sans aucun message d'avertissement lors de la compilation (par exemple, la conversion implicite de valeurs flottantes en entiers ou la division entière). Le programme va alors s'exécuter avec des valeurs souvent fausses. Il est parfois très difficile de détecter ces erreurs à l'exécution (contrairement aux boucles infinies ou aux divisions par zéro qui se voient immédiatement).

Remplacez la ligne 4 par :

```
int a = 7.6;
```

Que constatez-vous ?

Exercice 4 (renforcement) – Types et affichage

1. Écrivez un programme qui déclare un entier `x` initialisé à la valeur 65. Puis affichez cette valeur sous trois formats différents : un entier, un réel et un caractère. Expliquez ce qui s'affiche.
2. Écrivez un programme qui déclare un réel `y` initialisé à la valeur 65,9. Puis affichez cette valeur sous trois formats différents : un entier, un réel et un caractère. Expliquez ce qui s'affiche.

Exercice 5 (renforcement) – L'outil de mise au point de Geany

Geany permet de déboguer ses programmes une fois qu'ils ont pu être compilés sans erreur. Nous allons, dans cet exercice, apprendre à nous en servir. Il sera très important que vous testiez systématiquement vos programmes avec cet outil pendant le reste du semestre : les enseignants ne répondront à vos questions que si vous avez cherché les éventuelles erreurs avec.

1. Chargez le fichier `tp1-debug2.c` et compilez-le. Ce programme n'affiche rien, il est donc inutile de l'exécuter directement pour étudier son comportement.
2. Deux boutons, *Debug* et *Step*, sont insérés à la droite de la barre d'outils de Geany (figures 2 et 3). Le bouton *Debug* est en forme de loupe initialement et prend la forme d'une croix blanche sur un disque rouge une fois l'exécution débutée. Le bouton *Step* est une flèche grisée lorsque le mode de mise au point n'est pas lancé, et orange lorsqu'il est lancé. Si ces boutons ne sont pas visibles, il faut les activer une bonne fois pour toute en allant dans le gestionnaire de *plugin* (chercher dans les menus) puis en cochant l'option *Simple debug plugin*.



FIGURE 2 – Avant la mise au point



FIGURE 3 – Pendant la mise au point

Pour utiliser le mode de mise au point, vous devez :

1. Cliquer sur le bouton *Debug*.
2. Sélectionner une ligne de départ en appuyant sur la touche "shift" et en cliquant sur la marge de l'éditeur de Geany à la ligne désirée. Cette étape n'est pas obligatoire : par défaut, l'exécution du programme commence à partir du début de la fonction `main`.

3. La flèche du bouton “Step” apparaît alors, et permet d’exécuter pas à pas (i.e. ligne par ligne) le programme. À chaque fois que vous cliquez sur ce bouton, la ligne courante, qui est colorée, s’exécute et la valeur des variables s’affiche dans le panneau en bas de l’éditeur.

Exécutez pas à pas le programme que vous venez de taper, et interprétez les résultats (i.e. le contenu de chaque variable, ligne par ligne)

Exercice 6 (*renforcement*) – Navigateur Web

Pour communiquer avec l’Université, vous utiliserez :

- Le site Web du module qui contient les ressources pédagogiques (notes de cours, viatique, etc) et sur lequel vous trouverez régulièrement les informations concernant l’organisation du module (nouvelles fraîches).
- Le courrier électronique.

Si vous utilisez un gestionnaire de fenêtres *multi-bureaux*, comme GNOME¹, nous vous recommandons d’ouvrir votre navigateur Web sur un autre bureau. Séparer travail en cours et communications est une bonne habitude d’organisation visuelle de travail.

L’accès à des sites web hors de la PPTI (à commencer par le site de soumission des TME...) nécessite la configuration d’un serveur mandataire (proxy). Allez dans le menu *Système/Préférences/Serveur Mandataire* pour accéder à la fenêtre de configuration.

Vous pouvez retrouver les informations de configuration, ainsi que beaucoup d’autres informations utiles sur l’utilisation des machines de la PPTI sur le site <http://www-ari.ufr-info-p6.jussieu.fr/>, dans le menu OUTILS, rubrique “FAQ et StarterKit”.

Il existe de nombreux navigateurs (*browsers* en anglais) sous Linux pour accéder au Web (*mozilla*, *firefox*, *chromium*, *konqueror*, *galeon*, *epiphany*, *lynx*, etc.).

Nous vous proposons d’utiliser Iceweasel qui est une version Debian de Firefox. On y accède par le menu *Applications/Internet/Iceweasel*.

Les sites qui vont vous intéresser et vous être utiles sont au premier chef :

- le site du module :
<http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2014/ue/1I002-2015fev>
à partir duquel vous avez accès à la page de soumission de compte-rendus, à votre emploi du temps, aux polycopiés de cours et de TD, aux outils pour l’installation de Geany sur votre ordinateur personnel et, enfin, à l’ensemble de vos notes :
https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/public/espace_etudiant/protege/mesNotes.php
- le webmail de l’UPMC :
<https://webmail.etu.upmc.fr>
que vous devrez, soit consulter directement, soit configurer pour qu’il redirige vos emails sur votre adresse email habituelle.
- le site de la licence :
<https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/public/>
avec ses nouvelles fraîches :
https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/public/espace_etudiant/nouvelles/

Ces pages ont des adresses longues et vous y aurez souvent recours. Pour vous simplifier la vie, utilisez les marque-pages (menu Bookmarks ou bien directement `Ctrl+D`).

Votre adresse électronique sur le webmail de l’UPMC est (normalement) `Prenom.Nom@etu.upmc.fr`. Vous DEVEZ lire régulièrement votre courrier : par exemple la convocation aux examens vous parvient par cette voie ainsi

1. La plupart des gestionnaires de fenêtres modernes le sont.

que les convocations pour règlement de problèmes administratifs. Si vous avez une adresse favorite chez un fournisseur d'accès, la page d'accueil du webmail vous indique comment rediriger votre courrier pour lire votre courrier de l'UPMC avec vos autres courriers.

1. Explorez la page Web du module et posez un marque-page dessus pour pouvoir vous y reporter aisément.
2. Si ça n'est pas déjà fait, configurez votre *proxy* pour pouvoir consulter des sites web en dehors de la PPTI d'informatique.
3. Lisez votre courrier électronique avec le webmail et ajoutez un marque-page dessus aussi.

Exercice 7 (base) – Rendre un compte rendu

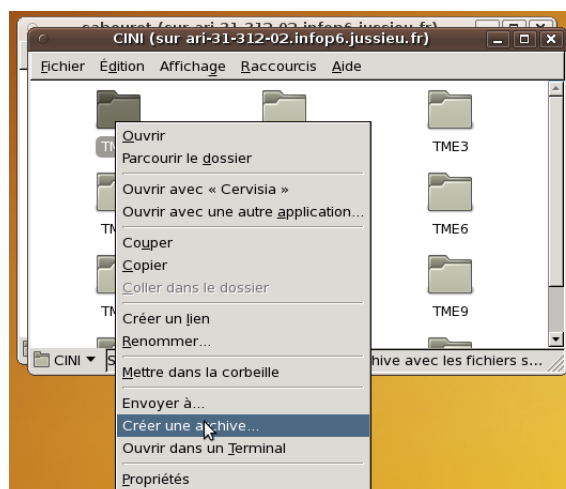
À la fin de chaque TME, vous devez envoyer à votre enseignant un compte-rendu de TME en utilisant le service de soumission en ligne disponible à partir du site web de l'UE :

<http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2014/ue/1I002-2015fev>
en cliquant sur le lien dans la section “Soumission de TME”, comme indiqué sur la figure suivante :

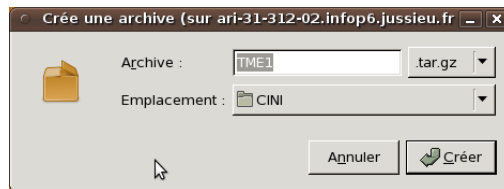


Cette procédure nécessite plusieurs étapes que nous allons détailler ci-après. Prêtez bien attention à cette manipulation : vous devrez la répéter à la fin de chaque séance de TME !

1. Ouvrez votre répertoire personnel et allez dans le répertoire 1I002.
2. Sélectionnez le répertoire de la semaine courante (aujourd'hui, en l'occurrence TME1) et créez une archive (clic droit sur le répertoire), comme illustré sur la figure suivante :



Vous n'avez pas besoin de modifier le nom et l'extension de votre archive :



3. Ouvrez votre navigateur web et allez sur la page de soumission de TME, dont vous trouverez une copie ci-dessous :

Module 1I002 : Éléments de Programmation 2 Last update: 17-08-2011

Soumission des TME

Dans le formulaire ci-dessous, les champs précédés par des **astérisques** doivent obligatoirement être renseignés.
Si vous renseignez les champs «adresse email», un mail de confirmation vous sera envoyé en cas de succès de la soumission.

* Numéro du TME :

Étudiant 1 :

* Nom :

* Numéro de carte d'étudiant :

Adresse email :

Étudiant 2 :

Nom :

Numéro de carte d'étudiant :

Adresse email :

* Fichier à soumettre aucun fichier sél.

4. Choisissez votre semaine (cette semaine, TME1), votre nom dans la liste et indiquez manuellement votre numéro d'étudiant (cela permet de vérifier que vous ne vous êtes pas trompé de nom). Vous pouvez aussi indiquer une adresse email ; dans ce cas, vous recevrez un accusé de réception avec l'archive en pièce-jointe, ce qui vous permet (par exemple) de continuer à travailler chez vous.

Attention ! Si vous avez travaillé seul, vous devez laisser vide les champs suivants. Si vous travaillez à deux, remplissez la deuxième partie de la page avec le nom et le numéro d'étudiant de votre binôme.

5. Cliquez sur le bouton "Parcourir" et sélectionnez l'archive (TME1.tar.gz) que vous venez de créer.

6. Cliquez sur le bouton "Transmettre le compte-rendu". Une page de confirmation s'affiche.

Exercice 8 (base) – Configuration du compte du binôme

Si vous travaillez en binôme, déconnectez vous et laissez votre binôme se connecter pour qu'il puisse :

- configurer l'application Geany,
- configurer son navigateur Web,
- faire une soumission depuis son compte (vous pouvez soumettre une archive vide, l'essentiel est de vérifier que vous pouvez soumettre).

IMPORTANT, dès le premier contrôle de TME en semaine 4, chaque étudiant doit savoir se connecter sur son compte, récupérer des fichiers et soumettre seul. Si votre compte n'est pas configuré d'ici là ou si vous ne savez pas faire une des ces opérations, vous risquez d'avoir un 0.

Exercice 9 (*entraînement*) – Jeu de test

Compilez et exécutez le programme `jeuTest.c` suivant que vous trouverez dans le répertoire habituel `/Infos/lmd/2014/licence/ue/1I002-2015fev/Semaine1`.

```
1 #include <stdio.h>
2
3 int main() {
4     int n1, n2, n3, res;
5
6     printf("Veuillez saisir 3 valeurs entieres : ");
7     scanf("%d", &n1);
8     scanf("%d", &n2);
9     scanf("%d", &n3);
10
11     if (n1 > 8) {
12         res = 3;
13     }
14     else {
15         if (n3 == 20) {
16             res = 2;
17         }
18         else {
19             if ((n2 >= 10) && (n3 >= 10)) {
20                 res = 1;
21             }
22             else {
23                 res = 0;
24             }
25         }
26     }
27     printf("%d\n", res);
28     return 0;
29 }
```

Déterminez un "jeu de test" vous permettant de tester toutes les branches du programme. Pour cela, vous devez identifier quatre triplets $(n1, n2, n3)$, chacun permettant de passer dans une branche différente du programme. Vous pouvez utiliser le mode debug de `geany` pour vérifier la pertinence de vos triplets.

Vous devez mettre les différents triplets choisis en commentaire dans votre programme et indiquer pour chacun d'eux quelle branche du programme il a permis de tester.

Exercice 10 (*renforcement*) – Calcul du discriminant

1. Écrivez un programme qui calcule et affiche la valeur du discriminant de polynôme du second degré $ax^2 + bx + c$. Les variables `a`, `b` et `c` seront déclarées comme des entiers et la valeur du discriminant sera affichée comme un entier par un `printf`. Vous sauvegarderez votre programme dans le fichier `tp1-discriminant.c`, vous le compilerez et vous l'exécuterez.
2. Complétez votre programme pour qu'il affiche les racines réelles du polynôme.
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>`.
3. Déterminez un jeu de test pour votre programme. Ajoutez le en commentaire à la fin de votre programme.

Exercice 11 (renforcement) – Visite de la Tour de Londres

Les tarifs pour visiter la *Tour de Londres* sont les suivants :

- adulte : 22 £
- enfant (entre 5 et 15 ans) : 11 £
- enfant de moins de 5 ans : gratuit
- senior (plus de 60 ans) : 18,70 £
- famille (2 adultes ou seniors et 3 enfants au maximum) : 59 £

Ecrivez un programme qui calcule la somme à payer en fonction du nombre d'adultes, d'enfants et de seniors. Pour simplifier, une seule entrée famille est possible (même si le nombre d'adultes, seniors et enfants pourrait en permettre plus), les personnes non comprises dans l'entrée famille paient en fonction de leur âge.

Le tarif famille est intéressant s'il y a au moins deux adultes-seniors et 2 enfants. S'il n'y a qu'un seul adulte-senior ou un seul enfant, il est plus intéressant de prendre des billets individuels.

Exercice 12 (entraînement) – Conversion temps

En vous inspirant du dernier exercice de TD (décomposition d'une somme en euros en un nombre minimal de pièces-billets), écrivez un programme qui, étant donné un nombre entier exprimant un temps en secondes, l'affiche en heures, minutes et secondes (avec bien sûr le nombre des minutes et des secondes inférieur strictement à 60).

Savoir-faire acquis

- ★ Système d'exploitation Linux :
 - Savoir créer des répertoires et naviguer dans l'arborescence des répertoires
 - Savoir utiliser le webmail de l'université
 - Savoir trouver les informations importantes concernant le module
 - Environnement de programmation C-1I002(Geany)
 - Pour s'entraîner en dehors de la PPTI : l'installer sur sa machine
 - Savoir créer ou modifier le code d'un programme avec l'éditeur
 - Savoir le compiler et corriger les erreurs détectées par le compilateur
 - Savoir rendre un compte-rendu de TME
- ★ Programmation
 - Connaître et savoir utiliser les deux principales instructions conditionnelles (**if** et **if-else**)
 - Savoir écrire un jeu de tests



TD semaine 2 Alternatives et boucles I

Version du 18 décembre 2014

Objectif(s)

- ★ saisie de valeurs
- ★ instruction d'itération : répéter un traitement tant qu'une condition est vérifiée (**while** et **do-while**)
- ★ directive **#define**

Exercice(s)

Exercice 1 (*base*) – Saisie de valeurs

1. Écrivez un programme qui demande de saisir deux nombres entiers, qui les stocke chacun dans une variable, puis qui affiche leur somme.
2. Modifiez le programme pour qu'il demande à l'utilisateur de saisir N valeurs et qu'il affiche la somme des valeurs saisies. Vous utiliserez une seule variable pour stocker successivement les valeurs saisies. Vous devez donc calculer le somme au fur et à mesure. La valeur de N sera définie par une primitive **#define**.

Exercice 2 (*base*) – Syntaxe des alternatives et boucles

1. Ecrivez un programme qui affiche le maximum de trois nombres a, b et c saisis par l'utilisateur.
2. Donnez un jeu de test pertinent.
3. Modifiez le programme pour qu'il demande à l'utilisateur de saisir N valeurs et qu'il affiche la plus grande des valeurs saisies. Inspirez-vous de ce qui a été fait dans l'exercice précédent.

Exercice 3 (*renforcement*) – Saisies répétées

Nous souhaitons écrire un programme qui demande à l'utilisateur de saisir des valeurs entières. La saisie continuera tant que l'utilisateur aura saisi moins de NB_MIN valeurs et que la somme des valeurs saisies sera inférieure ou égale à SOMME_MIN. Le programme devra afficher la somme des valeurs saisies. La valeur de NB_MIN et SOMME_MIN sera définie par une primitive **#define**.

1. — Quel doit être le comportement du programme si la première valeur saisie est supérieure à SOMME_MIN ?
— Quel doit être le comportement du programme si NB_MIN valeurs ont été saisies mais que leur somme est strictement inférieure à SOMME_MIN ?
— Complétez la phrase suivante qui doit décrire un comportement identique à celui de l'énoncé : *La saisie s'arrêtera dès que*
2. Ecrivez le programme réalisant ce qui est souhaité.
3. Ecrivez un programme qui demande à l'utilisateur de saisir des valeurs entières. La saisie s'arrêtera dès que l'utilisateur aura saisi au moins MIN valeurs avec au moins autant de valeurs strictement négatives que de valeurs strictement positives. Le programme devra afficher la moyenne des valeurs saisies. La valeur de MIN sera définie par une primitive **#define**.

Exercice 4 (entraînement) – Calcul du $N^{\text{ième}}$ terme d’une suite

1. Écrivez un programme qui calcule le terme de rang N de la suite $U_n = a.U_{n-1}$ avec $U_0 = 1$ et $a = 5$, les valeurs de N et de a étant définies à l’aide de primitives **#define**.
2. Nous considérons une nouvelle suite telle que :
 $U_0 = 2$
 $U_n = U_{n-1} + 1$ si U_{n-1} est pair
 $U_n = U_{n-1} + n$ si U_{n-1} est impair
Écrivez un programme qui permet de calculer et d’afficher le terme U_n , la valeur de n étant saisie par l’utilisateur.
3. Modifiez le programme de la question précédente pour qu’il affiche un message d’erreur si l’entier saisi est négatif et se comporte comme précédemment sinon.
4. Modifiez le programme de la question précédente pour qu’il affiche le premier terme de la suite supérieur ou égal à une valeur saisie par l’utilisateur. Le programme devra aussi afficher le rang de ce terme.

Exercice 5 (entraînement) – Tour de Londres : plusieurs entrées famille

Reprenez l’exercice de TP de la semaine dernière sur les tarifs d’entrée à la *Tour de Londres* en autorisant cette fois autant d’entrées *famille* que possible. Votre programme doit bien sûr donner le tarif le plus avantageux. Les nombres d’adultes, seniors et enfants seront saisis par l’utilisateur. Nous vous rappelons les tarifs :

- adulte : 22 £
- enfant (entre 5 et 15 ans) : 11 £
- enfant de moins de 5 ans : gratuit
- senior (plus de 60 ans) : 18,70 £
- famille (2 adultes ou seniors et 3 enfants au maximum) : 59 £

Exercice 6 (approfondissement) – Pseudo Blackjack

Le but du *Blackjack* est de tirer aléatoirement au maximum 3 cartes pour que la somme de leur valeur se rapproche le plus possible de 21 sans jamais le dépasser. Dans la version que nous voulons programmer, la valeur maximum à ne pas dépasser n’est pas connue mais le joueur choisit les valeurs dont il va faire la somme.

Le joueur peut choisir au maximum `NB_VAL` valeurs. Le maximum à ne pas dépasser est `MAX`. Ces deux valeurs sont définies par des primitives **#define**. Vous verrez la semaine prochaine comment donner à `MAX` une valeur choisie aléatoirement.

1. Écrivez le programme qui :
 - permet au joueur de choisir au maximum `NB_VAL` valeurs,
 - vérifie que chaque valeur saisie est dans l’intervalle $[1, 11]$ (la saisie sera redemandée si la valeur choisie par l’utilisateur n’est pas dans l’intervalle),
 - demande à l’utilisateur s’il veut choisir ou non une nouvelle valeur lorsqu’il a encore la possibilité de le faire (vous pouvez utiliser une variable entière qui vaudra 1 si l’utilisateur souhaite choisir une nouvelle valeur et 0 sinon),
 - affiche à la fin de la partie si elle est gagnée ou perdue (si la somme des valeurs choisies par l’utilisateur dépasse `MAX` il a perdu, sinon il a gagné). Dans les deux cas, le programme doit aussi afficher la différence entre la somme des valeurs choisies par l’utilisateur et `MAX`.

Écrivez le programme qui réalise notre version du *Blackjack*.

2. Définissez un jeu de test pertinent. Justifiez vos choix.

Savoir-faire acquis

- ★ Savoir généraliser un programme pour faciliter sa maintenance en utilisant la directive **#define**
- ★ Savoir écrire une boucle **while** ou `do_while` : vérifier l'initialisation, la condition d'entrée-continuation, l'évolution de la condition d'entrée-continuation.
- ★ Savoir écrire des conditions de boucle complexes.

Applications classiques à connaître

- ★ Saisie répétée de valeurs
- ★ Calcul itératif des valeurs successives d'une suite
- ★ Calcul d'un tarif en respectant les conditions tarifaires



TME semaine 2 Alternatives et boucles I

Version du 18 décembre 2014

Objectif(s)

- ★ expressions booléennes
- ★ instructions conditionnelles : **if** et **if-else**
- ★ instruction de répétition : répéter un traitement tant qu'une condition est vérifiée (**while** et **do-while**)
- ★ directive **#define**
- ★ première utilisation de la bibliothèque graphique.

Exercice(s)

Exercice 1 (*base*) – Préprocesseur

Nous considérons le programme `define.c` suivant :

```
#include <stdio.h>

#define A = 3;
#define B 7

int main() {
    int a = A;
    printf("%d\n", 2*a);
    B = B + 1;
    printf("%d\n", B);

    return 0;
}
```

Ce fichier est fourni dans le répertoire habituel :

`/Infos/lmd/2014/licence/ue/1I002-2015fev/Semaine2`

1. Ouvrez ce fichier et enregistrez une copie dans votre répertoire personnel `1I002/TME2`.
2. Visualisez le programme produit par le préprocesseur en cliquant sur le menu *Compilation/Pre process* sous Geany. Cela a pour effet d'ouvrir dans un nouvel onglet le résultat du préprocesseur. La partie qui nous intéresse se trouve *à la fin de l'affichage*. Quelles différences voyez-vous entre le programme que vous aviez au départ et le programme obtenu après passage par le préprocesseur ?
3. Fermez l'onglet du préprocesseur (ce fichier n'est pas exploitable : vous travaillerez toujours uniquement sur le fichier source `.c`). Compilez le programme source et corrigez les erreurs de compilation (ce que vous avez visualisé après l'action du préprocesseur devrait vous aider). Le programme corrigé doit encore contenir deux lignes **#define** (une pour A et une pour B). Les deux valeurs ainsi définies doivent obligatoirement être utilisées dans la suite du programme.

Exercice 2 (*base*) – Multiples de M inférieurs à N

Écrivez un programme C qui affiche les multiples non nuls et strictement inférieurs à N d'un entier positif M. Par exemple, pour $N = 17$ et $M = 3$, le programme doit afficher les valeurs :

3 6 9 12 15

Les valeurs associées à M et N seront définies en utilisant la directive **#define**.

Attention, la boucle que vous devez écrire doit exécuter un nombre d'itérations identique au nombre de multiples à afficher. Dans l'exemple précédent, votre boucle doit donc faire exactement 5 itérations.

Exercice 3 (*renforcement*) – Factorielle

La factorielle du nombre n , notée $n!$, est définie par $n! = 1 \times 2 \times \dots \times (n - 1) \times n$. Par convention, $0! = 1$.

Écrivez un programme qui demande à l'utilisateur de saisir une valeur et qui calcule et affiche la plus petite factorielle supérieure ou égale à cette valeur. La valeur de la factorielle $n!$ trouvée et la valeur de n seront affichées.

Initiation au graphisme

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, string 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(string color);
```

remplit la fenêtre précédemment créée de la couleur passée en paramètre.

```
void CINI_draw_pixel(int x, int y, string color);
```

affiche le point de coordonnées (x, y) de couleur `color`.

```
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.

Pour appeler une de ces fonctions dans un programme C, il suffit de taper le nom de la fonction avec, entre parenthèses et séparées par des virgules, les valeurs affectées à chacun des paramètres. Il faut autant de valeurs qu'il y a de paramètres dans la signature. L'instruction d'appel se termine par un point-virgule.

Par exemple, le programme suivant fait appel aux fonctions de la bibliothèque `cini` (il ne faut donc pas oublier la directive `#include` !) pour créer une fenêtre de 400 pixels de large sur 300 de haut et y afficher trois points blancs.

```
#include <cini.h>

int main() {

    CINI_open_window(400, 300, "test");

    CINI_draw_pixel(199, 200, "white"); /* affiche pt coord x=199, y=200*/
    CINI_draw_pixel(200, 200, "white"); /* affiche pt coord x=200, y=200*/
    CINI_draw_pixel(201, 200, "white"); /* affiche pt coord x=201, y=200*/

    CINI_loop();
    return 0;
}
```

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.

Exercice 4 (*base*) – Pour commencer avec la bibliothèque graphique

1. En utilisant les fonctions de la bibliothèque graphique, créez une fenêtre de 400×400 points, puis tracez-y une ligne droite suivant la diagonale. Cette ligne doit relier le point de coordonnées $(0, 0)$ au point de coordonnées $(399, 399)$. Vous devrez afficher la droite point par point.

Remarque : dans une fenêtre, le point de coordonnées $(0, 0)$ correspond au coin supérieur gauche et celui de coordonnées $(\text{maxX}, \text{maxY})$ correspond au coin inférieur droit.

Exercice 5 (*renforcement*) – Droites, points et croix

Dans tout cet exercice, on travaillera dans une fenêtre de taille 400×400 .

1. Écrivez un programme qui dessine dans une fenêtre une croix de couleur bleu, définie par son centre et la longueur de ses branches (on les déclarera par un `#define`). Ici, une longueur l signifie que l pixels à gauche, à droite, au-dessus et en-dessous du centre de la croix seront dessinés en bleu (cela veut donc dire qu’on dessine autour du centre de la croix un trait horizontal et un trait vertical de longueur $2l + 1$). Vous testerez votre programme pour différents jeux de paramètres.
2. Écrivez un programme qui dessine une droite d’équation $y = a * x + b$ définie dans le repère cartésien. Vous ne devez bien sûr dessiner que la partie de droite qui se trouve dans la fenêtre graphique. Vous devrez afficher un message adapté si aucun point de la droite ne s’y trouve.

Attention ! N’oubliez pas que l’axe des ordonnées dans la fenêtre graphique est inversé par rapport au repère cartésien classique. Le point de coordonnées (x, y) dans le repère cartésien aura les coordonnées $(x, YMax - y)$ dans la fenêtre graphique.

Exercice 6 (*renforcement*) – Carrément graphique

1. Écrivez un programme qui dessine un carré avec des côtés rouges, de longueur c et de coin supérieur gauche le point $(0, 0)$. La valeur de c sera saisie par l’utilisateur.
2. Écrivez un programme qui dessine un carré avec des côtés rouges, de longueur c et dont les coordonnées du coin supérieur gauche sont (x, y) . Les valeurs de c , x et y seront saisies par l’utilisateur.

- Comment peut-on tracer les quatre côtés d'un carré avec une seule boucle (les côtés du carré sont parallèles à ceux de la fenêtre graphique) ? 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 que vous pouvez tracer en même temps.
- Créez une fenêtre de 400×400 points, remplissez-la de blanc, puis, en utilisant une seule boucle, tracez-y un carré de COTE points de côté dont le coin supérieur gauche se trouve aux coordonnées X_HG, Y_HG. Les valeurs de COTE, X_HG et Y_HG seront définies chacune avec une primitive **#define**. Si vous donnez les valeurs 200 à COTE et 100 à X_HG et Y_HG, le carré doit se dessiner au milieu de la fenêtre graphique.
Le côté supérieur sera tracé en bleu ("blue"), le côté gauche en rouge ("red"), le côté inférieur en vert ("green") et le côté droit en noir ("black").
- Complétez ce programme pour qu'il dessine aussi 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 1. Les carrés dont un des points est en dehors de la fenêtre ne seront pas dessinés.

A partir de cette question vous pouvez utiliser la fonction

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

qui trace la droite reliant les deux points de coordonnées (x_1, y_1) et (x_2, y_2) .

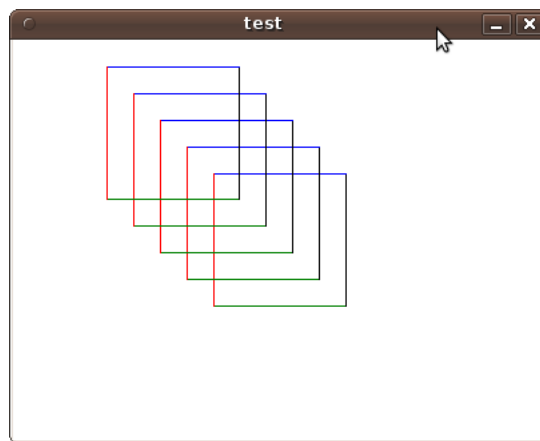


FIGURE 1 – Translation carrés

Exercice 7 (entraînement) – Triangles en spirale

- Écrivez un programme qui crée une fenêtre de 640×480 points, puis y trace un triangle qui remplit l'intégralité de la fenêtre graphique (comme illustré par la figure 2). Vous utiliserez une couleur différente pour chacun des trois côtés du triangle.
- Complétez votre programme pour qu'il affiche un deuxième triangle décalé par rapport au premier comme illustré par la figure 3.

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 le segments $[AB]$ (resp. $[BC]$ et $[CA]$).

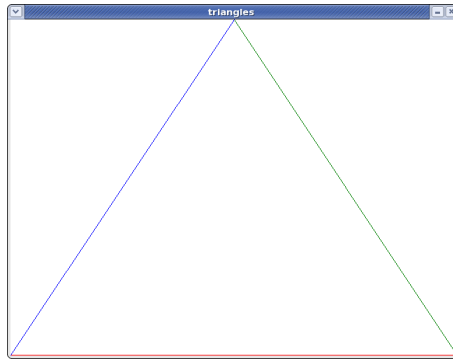


FIGURE 2 – Premier triangle

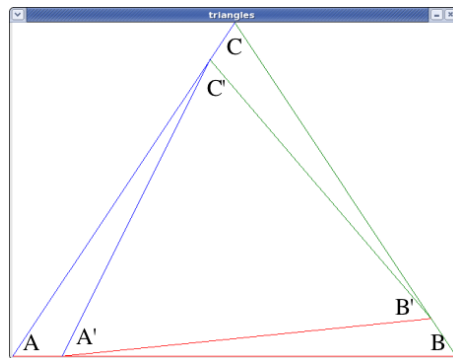


FIGURE 3 – Second triangle

3. Modifiez votre programme pour qu'il 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. L'instruction d'appel à cette fonction est `CINI_loop_until_keyup()` ;
4. Modifiez le programme pour qu'il continue à tracer des triangles jusqu'à ce que la distance entre les points A et B soit inférieure à une valeur `EPSILON` définie par une directive **#define**. 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 4.

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

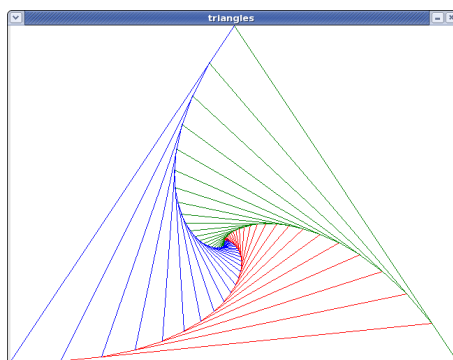


FIGURE 4 – Résultat final

Exercice 8 (entraînement) – Série harmonique

Écrivez un programme qui demande à l'utilisateur de saisir une valeur \max ($1 < \max \leq 16$) qui calcule et affiche la plus grande somme des termes de la série harmonique définie par : $1 + 1/2 + 1/3 + \dots + 1/n$ inférieure strictement à \max , la valeur de n sera aussi affichée. On demandera à l'utilisateur de saisir à nouveau une valeur si cette dernière n'est pas dans l'intervalle souhaité.

Savoir-faire acquis

- ★ Connaître et savoir utiliser les deux principales instructions conditionnelles (**if** et **if-else**)
- ★ Savoir généraliser un programme pour faciliter sa maintenance en utilisant la directive **#define**
- ★ Savoir écrire une boucle : vérifier l'initialisation, la condition d'itération et son évolution
- ★ Savoir utiliser les fonctions graphiques

Applications classiques à connaître

- ★ Savoir calculer la factorielle d'un nombre
- ★ Afficher les multiples d'un nombre m inférieurs à un nombre n
- ★ Graphisme : affichage d'une diagonale, de carrés, de triangles



TD semaine 3 Les Boucles, suite

Version du 18 décembre 2014

Objectif(s)

- ★ Une autre façon d'écrire les boucles : la boucle **for**
- ★ Boucles imbriquées
- ★ Tirages de nombres pseudo-aléatoires

Exercice(s)

Exercice 1 (*base*) – Une autre façon d'écrire les boucles

1. Ecrivez un programme qui affiche la somme des n premiers entiers strictement positifs. La valeur de n sera saisie par l'utilisateur (la saisie sera redemandée tant qu'elle n'est pas valable). L'affichage devra être réalisé par une boucle **for**.
2. Si l'utilisateur saisit une valeur négative, le programme doit maintenant afficher les entiers allant de -1 à la valeur saisie. Les affichages doivent toujours être réalisés avec une boucle **for**. Apportez au programme les modifications nécessaires.

Exercice 2 (*base*) – Affichage en lignes

1. Écrivez un programme qui affiche tous les entiers strictement positifs, strictement inférieurs à un entier **MAX**, à raison de **LONG** entiers par ligne (la dernière ligne peut bien sûr contenir moins d'entiers). Les valeurs associées à **MAX** et **LONG** seront définies par une directive **#define**.

L'affichage obtenu lorsque **MAX=12** et **LONG=3** est :

```
1    2    3
4    5    6
7    8    9
10   11
```

Aide : trouvez le lien entre la dernière valeur affichée sur chaque ligne complète et **LONG**.

L'affichage en colonnes est obtenu par affichage d'une tabulation (chaîne de caractères "`\t`") après chaque nombre.

Exercice 3 (*base*) – Tables de multiplication

1. Écrivez un programme qui affiche toutes les tables de multiplications de 1 à 10 sous la forme :

```
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
...
10 20 30 40 50 60 70 80 90
```

Exercice 4 (*base*) – Tirages aléatoires

La bibliothèque `stdlib.h` offre une fonction permettant d'utiliser le générateur pseudo-aléatoire :

`int rand()` : retourne une valeur entière.

Pour utiliser cette fonction sans avoir toujours la même liste de nombres "aléatoires", il faut initialiser le générateur avec l'instruction `srand(time(NULL))` ; que vous devez ajouter au tout début de la fonction `main`. Cette instruction utilise la bibliothèque `time.h`

1. Écrivez un programme qui tire aléatoirement `N` entiers compris entre `MIN` et `MAX` (inclus) et les affiche. Les valeurs associées à `N`, `MIN` et `MAX` seront définies par une directive **#define**.
2. Écrivez un programme qui tire des entiers compris entre `MIN` et `MAX` jusqu'à ce que la valeur `BUT` soit tirée (la valeur associée à `BUT` sera définie par une directive **#define**). Votre programme devra afficher les valeurs tirées et le nombre de tirages effectués *avant* le tirage de la valeur `BUT`.
3. Nous souhaitons maintenant limiter le nombre de tirages effectués par le programme. Le nombre maximum de tirages possibles est représenté par la valeur `NBCOUPS` définie par une directive **#define**. Modifiez votre programme pour qu'au maximum `NBCOUPS` tirages soient effectués et que s'affiche :
 - GAGNE si la valeur `BUT` a été trouvée en `NBCOUPS` tirages au maximum ;
 - PERDU si la valeur `BUT` n'a pas été trouvée après `NBCOUPS` tirages.

Exercice 5 (*renforcement*) – Nombres premiers

Nous souhaitons afficher les nombres premiers inférieurs ou égaux à `MAX`. On arrêtera le traitement d'un nombre dès qu'on est sûr qu'il n'est pas premier (i.e., dès que nous avons trouvé un diviseur différent de 1 et de lui même).

Pour décider si un nombre `nb` est premier, nous allons déclarer une variable entière `premier` initialisée à 1 que nous utiliserons comme un booléen (1 signifie vrai et 0 signifie faux). Nous allons parcourir tous les nombres à partir de 2 jusqu'à ce que l'on trouve parmi ces nombres un diviseur de `nb` ou que l'on atteigne la valeur `nb/2`. Dans le premier cas, la variable `premier` est mise à 0, dans le deuxième cas, `nb` est premier et le parcours se termine avec la valeur 1 pour `premier`.

1. Écrivez un programme qui permet de déterminer si un entier, tiré aléatoirement dans l'intervalle `[0, 99]` est premier. Votre programme affichera à l'écran "nb est premier" ou "nb n'est pas premier" selon la valeur de la variable `premier` à l'issue de la boucle (et en remplaçant `nb` par sa valeur!).
2. Modifiez votre programme pour qu'il affiche la liste de tous les nombres premiers inférieurs ou égaux à `MAX`. La valeur de `MAX` sera définie en utilisant la directive **#define**.

Exercice 6 (*approfondissement*) – Suite récurrente d'ordre 2

1. Écrivez un programme qui calcule le terme de rang `N` de la suite $U_n = a.U_{n-1} + b.U_{n-2}$ avec $U_0 = 1$, $U_1 = 2$, $a = 5$ et $b = 10$.

Exercice 7 (*approfondissement*) – Remplissage d'une salle

1. Nous souhaitons écrire un programme qui permet de placer des spectateurs dans une salle contenant `NB_RANG` rangées et `NB_PLACES` places par rangée.

Les spectateurs se présentent par groupes d'au plus `MAX_GROUPE` personnes. Le nombre de personnes composant un groupe est choisi aléatoirement. Toutes les personnes du groupe doivent être placées avant de passer au groupe suivant. Les rangées sont remplies les unes à la suite des autres. Les groupes de spectateurs sont acceptés tant qu'il y a de la place.

Ecrivez le programme qui permet de suivre l'évolution du remplissage de la salle. Voici un exemple d'exécution dans lequel `NB_RANG=5`, `NB_PLACES=5` et `MAX_GROUPE=12`

```
Il y a 25 places disponibles
7 personne(s) a placer
5 personne(s) placee(s) dans la rangee 1
2 personne(s) placee(s) dans la rangee 2
Il reste 18 place(s)
8 personne(s) a placer
3 personne(s) placee(s) dans la rangee 2
5 personne(s) placee(s) dans la rangee 3
Il reste 10 place(s)
2 personne(s) a placer
2 personne(s) placee(s) dans la rangee 4
Il reste 8 place(s)
3 personne(s) a placer
3 personne(s) placee(s) dans la rangee 4
Il reste 5 place(s)
7 personne(s) a placer
5 personne(s) placee(s) dans la rangee 5
Il reste 0 place(s)
2 personne(s) non placee(s)
```

Savoir-faire acquis

- ★ Savoir écrire des boucles **for**
- ★ Savoir quand et comment construire des boucles imbriquées
- ★ Utiliser le tirage pseudo-aléatoire de nombres

Applications classiques à connaître

- ★ Déterminer si un nombre est premier
- ★ Tirer un nombre pseudo-aléatoire dans un intervalle donné
- ★ Calculer les termes d'une suite récurrente d'ordre 2



TME Semaine 3 Les boucles, suite

Version du 18 décembre 2014

Objectif(s)

- ★ Manipulation des boucles imbriquées et des alternatives
- ★ Tirages de nombres pseudo-aléatoires

Exercice(s)

Exercice 1 (*base*) – Factorielle

1. Le programme suivant permet de calculer et d'afficher les factorielles des entiers inférieurs ou égaux à `max`. Ecrivez un programme plus efficace tirant profit du fait qu'au cours du calcul de $n!$, on calcule les factorielles de tous les entiers inférieurs à n .

```
#include <stdio.h>

int main() {
    int i, n, resu, max;

    printf("Saisie de max : ");
    scanf("%d", &max);

    for (n = 0; n <= max; n++) {
        /* calcul de n! */
        resu = 1;
        for (i = 2; i <= n ; i++) {
            resu = i * resu;
        }
        printf("%d! = %d\n", n, resu);
    }
    return 0;
}
```

2. Testez le fonctionnement de votre programme pour calculer les 20 premières factorielles. Que constatez-vous ? A partir de quelle valeur les résultats vous semblent-ils étranges (et pourquoi) ?

Exercice 2 (*base*) – Figures géométriques

1. En affichant uniquement des caractères *, tracez un rectangle plein de M lignes et N colonnes. Les valeurs de M et N seront définies par une primitive **#define**. Si M vaut 4 et N vaut 7, vous devez obtenir l’affichage suivant :

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

2. Reprenez l’affichage du rectangle et modifiez votre programme pour n’afficher que le contour du rectangle. Si M vaut 4 et N vaut 7, vous devez obtenir l’affichage suivant :

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

Exercice 3 (*base*) – Graphisme

Dans cet exercice, pour les affichages, vous ne pouvez utiliser que la fonction `CINI_draw_pixel`.

1. Ecrivez un programme qui, dans une fenêtre graphique carrée (sa dimension sera définie par une primitive **#define**), affiche les points de la diagonale en jaune, les points au-dessus de la diagonale en bleu et ceux en-dessous en rouge.
2. Modifiez votre programme pour que l’épaisseur de la diagonale soit un nombre impair de pixels (défini par une primitive **#define**).

Exercice 4 (*renforcement*) – Minimum, maximum et moyenne d’une série de notes

1. Écrivez un programme qui permet la saisie de notes. Une note négative mettra fin à la saisie. Cette valeur négative ne sera bien sûr pas considérée comme une note. Le programme doit aussi calculer et afficher la moyenne des notes saisies (assurez-vous de ne pas faire une division par zéro et de bien calculer une moyenne réelle).
Le programme utilisera une unique variable pour stocker les valeurs saisies (donc chaque valeur saisie “écrase” la précédente).
2. Enrichissez le programme précédent pour qu’il vérifie que les notes saisies sont toutes inférieures ou égales à 20. Si ce n’est pas le cas, la saisie sera redemandée.
3. Enrichissez le programme précédent pour que l’utilisateur ne saisisse que 10 notes au maximum.
4. Enrichissez le programme précédent pour qu’en plus de la moyenne, le minimum et le maximum saisis soient affichés.

Exercice 5 (*renforcement*) – Nombres premiers

1. Ecrivez le programme répondant à la question 2 de l’exercice 5 de TD.
2. Modifiez le programme pour qu’il demande à l’utilisateur de saisir une valeur entière m . Le programme affiche ensuite tous les entiers de 2 à m en les séparant par des tabulations (`'\t'`) et en passant à la ligne après chaque nombre premier.
Par exemple, pour $m = 25$, on doit obtenir :

```
Saisissez un entier 25
2
3
4      5
6      7
8      9      10      11
12     13
14     15      16      17
18     19
20     21      22      23
24     25
```

Exercice 6 (*renforcement*) – Suites

1. Soit U_n la suite définie comme suit : $U_1 = 42$ et

$$U_n = \begin{cases} \frac{U_{n-1}}{2} & \text{si } U_{n-1} \text{ est pair} \\ 3U_{n-1} + 1 & \text{sinon} \end{cases}$$

Écrivez un programme qui demande à l'utilisateur de saisir une valeur entière k et qui calcule et affiche le terme U_k de rang k de la suite.

2. Soit U_n la suite définie comme suit : $U_1 = c$ et

$$U_n = \begin{cases} \frac{U_{n-1}}{2} & \text{si } U_{n-1} \text{ est pair} \\ 3U_{n-1} + 1 & \text{sinon} \end{cases}$$

Écrivez un programme qui demande à l'utilisateur de saisir une valeur initiale c et qui affiche, pour chacune des 70 premières valeurs de la suite, un rectangle de 10 pixels de large et $U_n \cdot 10$ pixels de haut. Les rectangles seront affichés de gauche à droite de la fenêtre et alignés sur le haut de celle-ci. Vous pouvez utiliser la fonction `CINI_draw_line`.

Pour $c = 9$, vous devez obtenir l’affichage suivant de la figure 1.

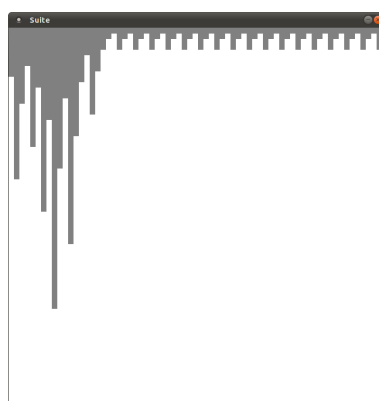


FIGURE 1 – Affichage des 70 premières valeurs de la suite pour $c = 9$

Faites tourner votre programme pour différentes valeurs de c . Attention, pour certaines valeurs, des points seront affichés hors de la fenêtre graphique.

Exercice 7 (entraînement) – Coefficients binomiaux et triangle de Pascal

1. Écrivez un programme qui demande à l'utilisateur de saisir deux valeurs entières n et k . Les valeurs de n et k doivent être telles que $n \geq k$ et que $k \geq 0$. Si ce n'est pas le cas, les valeurs sont saisies à nouveau. Une fois les valeurs correctes, le programme calcule et affiche le coefficient binomial

$$C_n^k = \binom{n}{k} = \frac{\prod_{i=n-k+1}^n i}{k!}.$$

Comme le coefficient binomial C_n^k est un entier, on peut le calculer sur une variable entière. En revanche, il est nécessaire de calculer séparément le numérateur et le dénominateur de la formule ci-dessus et de ne faire la division qu'à la toute fin.

2. Le triangle de Pascal est une représentation graphique des coefficients binomiaux. La ligne n du triangle contient, séparés par des blancs, les C_n^k pour k allant de 0 à n . L'affichage ci-dessous est le triangle de Pascal de hauteur 7.

```

          1
        1 1
      1 2 1
    1 3 3 1
  1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

```

Écrivez un programme qui demande à l'utilisateur de saisir une valeur entière h et qui affiche le triangle de Pascal de hauteur h , c'est-à-dire pour n allant de 0 à h .

Pour l'affichage, vous pouvez « incliner » le triangle et l'afficher justifié à gauche comme ci-dessous :

Saisissez un entier h 7

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

```

Exercice 8 (*entraînement*) – La quadrature du cercle

1. Ecrivez un programme qui initialise aléatoirement une variable r à une valeur entière comprise entre MIN et MAX (valeurs définies par une primitive **#define**), qui affiche une fenêtre graphique carrée de côté $2*r$ ainsi que le disque de rayon r inscrit dans la fenêtre graphique.

Votre programme doit fournir un affichage similaire à celui de la figure 2.



FIGURE 2 – Disque inscrit dans la fenêtre graphique

2. Modifiez maintenant votre programme pour compter le nombre n de pixels affichés (il s'agit du nombre de pixels affichés en gris dans l'exemple). Rajoutez un affichage pour indiquer le rapport $p = \frac{n}{r^2}$ (un nombre avec des décimales).

Ce rapport vous semble-t-il connu ? Pourquoi tend-il vers cette valeur si connue quand r tend vers l'infini ?

Exercice 9 (*entraînement*) – Montagne d'étoiles

1. Écrivez un programme qui permet de choisir aléatoirement un entier n (compris entre 0 et 15) et de tracer une ligne composée d'une étoile (*), suivie de $(2n-1)$ espaces puis d'une autre étoile si $n > 0$. Lorsque $n=0$, le programme affiche une ligne réduite à une étoile.
2. Modifiez votre programme pour qu'il trace "une montagne". À la place de n votre programme doit choisir aléatoirement un entier h (compris entre 0 et 15) et tracer une montagne de hauteur h . Pour $h=5$, on doit obtenir

```

      *
     * *
    *  *
   *   *
  *    *
 *     *

```

La première ligne contient 4 espaces suivis d'une étoile. La deuxième ligne contient 3 espaces suivis d'une étoile, un espace et une étoile. La troisième ligne contient 2 espaces suivis d'une étoile, trois espaces et une étoile. La quatrième ligne contient 1 espace suivi d'une étoile, 5 espaces et une étoile. La cinquième et dernière ligne contient une étoile, 7 espaces et une étoile.

Votre solution doit bien sûr être valable pour toute valeur de la variable h .

3. Modifiez votre programme pour qu'il marque l'enneigement de la montagne. En plus de la hauteur de la montagne, votre programme doit choisir aléatoirement la hauteur de l'enneigement he (entier compris entre 0 et $h-1$). Attention, l'enneigement n'a de sens que si $h > 1$. La montagne doit être tracée comme précédemment et les deux étoiles de la $(he + 1)^{\text{ème}}$ ligne doivent encadrer des étoiles si $he < h$. Pour $h = 5$ et $he = 3$, on doit obtenir :

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

Savoir-faire acquis

- ★ Savoir construire des boucles **for** imbriquées
- ★ Savoir utiliser des alternatives dans des boucles **for**
- ★ Réfléchir à l'exactitude des résultats obtenus
- ★ Utiliser la bibliothèque graphique

Applications classiques à connaître

- ★ Calculer la somme, la moyenne d'une suite de nombres saisis au clavier
- ★ Calculer la factorielle d'une suite d'entiers inférieurs à une borne
- ★ Calculer les coefficients binomiaux et afficher le triangle de Pascal



TD Semaine 4 Les boucles et les tableaux

Version du 18 décembre 2014

Objectif(s)

- ★ Primitive **#define**
- ★ Tableaux de données à une dimension, taille d'un tableau

Exercice(s)

Exercice 1 (*base*) – Calcul de PGCD

On souhaite déterminer le PGCD de deux nombres en appliquant l'algorithme suivant :

```
tant que (nb1 différent de nb2)
faire
    si nb1 > nb2 alors
        nb1 = nb1 - nb2
    sinon
        nb2 = nb2 - nb1
    fin si
fin tant que
```

1. Écrivez un programme C calculant le PGCD de deux nombres N1 et N2 définis par la directive **#define**.

Exercice 2 (*base*) – Pour commencer avec les tableaux

1. (**Utilisation de tableau**) Que fait le programme suivant (volontairement sans commentaire) ?

```
#include <stdio.h>

#define TAILLE 10
#define VAL 3.14159

int main() {
    float tab[TAILLE];
    int i;

    tab[0] = 0;
    for(i = 1; i < TAILLE; i++) {
        tab[i] = VAL/i;
    }

    printf("truc = %f\n", tab[4]);

    return 0;
}
```

Modifiez-le et commentez-le de manière à rendre plus explicite son fonctionnement.

Que se passe-t-il si l'on ajoute l'instruction suivante dans la fonction `main`, juste avant l'instruction `return` ?

```
printf("tab[12 ] = %f\n", tab[12]);
```

Exercice 3 (*renforcement*) – Tableau de nombres aléatoires

1. Écrivez un programme qui déclare un tableau de 30 entiers et qui l'initialise avec des valeurs tirées aléatoirement entre -49 et 49 (inclus).
2. Complétez votre programme pour qu'il affiche les éléments du tableau à raison de `p` éléments par ligne. Par exemple :

```
27    -15    28    34    36    31    -25    -5     4    -12
23    -32    38    22    14    -20    -15    -33    48    -12
-43    24    -4     7   -45    -7     29    -22    22    39
```

La valeur de `p` (ici 10) doit être fournie par l'utilisateur lors de l'exécution du programme.

3. Complétez le programme pour qu'il affiche la plus grande valeur contenue dans le tableau, ainsi que sa position.
4. Complétez le programme pour qu'il calcule l'élément qui apparaît le plus grand nombre de fois dans le tableau, et qu'il affiche cet élément, ainsi que son nombre d'occurrences. Si plusieurs éléments apparaissent le même nombre de fois dans le tableau, c'est le premier rencontré qui sera affiché.

Exercice 4 (*approfondissement*) – Tirage aléatoire non équiprobable

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.
```

Mais comment faire si le tirage ne doit pas être équiprobable ? On veut par exemple que la valeur 0 représente 17% des cas, la valeur 1, 28%, la valeur 2, 50% et la valeur 3, 5%.

L'idée est assez simple : on considère un intervalle de 100 valeurs. Si la valeur tirée est comprise entre 1 et 17 alors la valeur affectée au résultat est 0, si la valeur tirée est comprise entre 18 et 45 (ce qui représente un intervalle de 28 valeurs) alors la valeur affectée au résultat est 1, si la valeur tirée est comprise entre 46 et 95 alors la valeur affectée au résultat est 2 et si la valeur est comprise entre 96 et 100 alors la valeur affectée au résultat est 3.

1. Écrivez un programme qui, à partir d'un tableau de `N` valeurs représentant les pourcentages de tirages d'un ensemble de `N` valeurs, effectue un tirage aléatoire selon ces pourcentages.

Vous devrez utiliser deux tableaux :

- `tab_pro` contient les pourcentages des différents tirages (`tab_pro[i]`=pourcentage de tirages de la valeur `i`).
- `tab_res` contient le nombre de fois où chaque valeur est tirée (`tab_res[i]`=nombre de fois où la valeur `i` est tirée).

Testez votre programme sur 10 000 tirages avec un ensemble de valeurs = $\{0, \dots, N - 1\}$.

Savoir-faire acquis

★ Déclarer et initialiser un tableau

- ★ Consulter ou modifier les valeurs d'un tableau
- ★ Parcourir un tableau du début à la fin

Applications classiques à connaître

- ★ Algorithmes classiques sur les tableaux
 - Initialiser un tableau à la déclaration, initialiser un tableau avec des valeurs aléatoires
 - Afficher les valeurs d'un tableau sur une ligne ou avec p éléments par ligne
 - Calculer le plus petit, le plus grand des éléments d'un tableau d'entiers
 - Calculer le nombre d'occurrences d'un élément dans un tableau
- ★ Tirages aléatoires non équiprobables



TME Semaine 4 Les tableaux et TME Solo

Version du 18 décembre 2014

Objectif(s)

- ★ Création, utilisation et parcours simple (boucle **for**) de tableaux à une dimension
- ★ Tirages non-équiprobables de nombres pseudo-aléatoires
- ★ Retour sur les boucles **while** et **do-while**

Exercice(s)

Exercice 1 (*base*) – Bonnes pratiques d'écriture d'un programme

Nous considérons le programme `ex01.c` suivant :

```
#include <stdio.h>
#define M 10

int main() {
    int var1, var2, var3;

    for (var1 = 0; var1 <= M; var1++) {
        /* on fait varier var1 de 0 à M */
        var3 = 1;
        for (var2 = 2; var2 <= var1 ; var2++) {
            var3 = var2 * var3;
        }
        printf("%d! = %d\n", var1, var3);
    }
    return 0;
}
```

Ce fichier est fourni dans le répertoire habituel :

`/Infos/lmd/2014/licence/ue/1I002-2015fev/Semaine5`

Ouvrez ce fichier et enregistrez une copie dans votre répertoire personnel `1I002/Semaine5`. Compilez le programme (il n'y a pas d'erreur : le programme compile et affiche bien le résultat souhaité).

1. Qu'affiche ce programme ? Que pouvez-vous constater sur l'écriture de ce programme ?
2. À partir des constatations faites à la question précédente, corrigez le programme.

Exercice 2 (base) – Affichage d'un tableau de taille fixe

Dans cet exercice, vous n'utiliserez pas les fonctions graphiques d'affichage d'un tableau.

1. Écrivez un programme qui déclare un tableau d'entiers dont la taille est fixée à l'aide d'une primitive **#define** et dont le contenu est défini à la déclaration. Le programme doit afficher ce contenu.
2. Complétez votre programme pour qu'il copie le contenu du tableau dans un deuxième tableau de même taille et qu'il affiche ce deuxième tableau.

Exercice 3 (base) – Échange (graphique) du contenu de deux cases

Dans cet exercice, vous utiliserez les fonctions graphiques d'affichage d'un tableau.

1. Écrivez un programme qui
 - déclare un tableau d'entiers dont la taille est fixée à l'aide d'une primitive **#define** et dont le contenu est défini à la déclaration ;
 - échange le contenu de la première et de la dernière case.
 Votre programme devra "dessiner" le contenu du tableau, initialement et **à chaque fois que celui-ci est modifié**. Pour visualiser correctement ces affichages, vous devrez insérer un appel à une fonction d'attente après chaque affichage.

Exercice 4 (entraînement) – Moyenne pondérée de notes

1. On souhaite calculer la note obtenue par un élève de terminale au baccalauréat. On suppose que l'élève a passé N épreuves pour lesquelles il a donc obtenu N notes. On utilise donc un tableau contenant ses notes ($tabN$) et un tableau contenant le coefficient associé à chaque note ($tabC$) : la note finale est obtenue en parcourant les deux tableaux pour calculer l'expression ci-dessous. Les notes seront représentées par des flottants et les coefficients seront des entiers.

La note finale est donc obtenue en calculant l'expression :

$$\text{note finale} = \frac{\sum_{i=0}^{N-1} tabN[i] \times tabC[i]}{\sum_{i=0}^{N-1} tabC[i]}$$

Écrivez le programme qui calcule la note de l'élève.

Exercice 5 (approfondissement) – Pour le plaisir...

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 :

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

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%.

1. Complétez le programme suivant 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. Le programme peut être récupéré dans le répertoire de l'UE (`feuille_enonce.c`).

```
#include <cini.h>

#define N      4
#define COEFX  50    /* coefficients d'echelle */
#define COEFY  38
#define DX     250    /* position dans la fenetre */
#define DY     420

int main() {

    float tab_A[N] = {0.5,  0.2, -0.15,  0.85};    /* les coefficients a_i */
    float tab_B[N] = {0,    -0.26, 0.28,  0.04};    /* les coefficients b_i */
    float tab_C[N] = {0,    0.23, 0.26, -0.04};    /* les coefficients c_i */
    float tab_D[N] = {0.16, 0.22, 0.24,  0.85};    /* les coefficients d_i */
    float tab_E[N] = {0,    0,    0,    0};        /* les coefficients e_i */
    float tab_F[N] = {0,    1.6, 0.44,  1.6};        /* les coefficients f_i */

    int tab_P[N] = {1, 7, 7, 85};    /* pourcentages de chaque transformation */

    string couleurs[N] = {"lime green", "fuchsia", "yellow", "turquoise"};

    /* Calcul des probabilites cumulees tab_P doit contenir les probabilites cumulees */
    /* dans l'exemple on obtient tab_P = {1, 8, 15, 100} */

    /* A COMPLETER */

    CINI_open_window(500, 500, "feuille");

    do {

        /* Choix du numero noT de la transformation a appliquer */

        /* A COMPLETER */

        /* Calcul du point a tracer (coordonnees x et y) */

        /* A COMPLETER */

        CINI_draw_pixel(DX + x*COEFX, DY - y*COEFY, couleurs[noT]);

    } while (! CINI_key_down());

    return 0;
}
```

}

Exercice 6 (*entraînement*) – Retour sur les boucles : Ciseaux-Puits-Feuille

L'objectif de cet exercice est de programmer le célèbre jeu Ciseaux-Puits-Feuille. Il s'agit d'un jeu à deux joueurs, chacun des deux choisit soit ciseaux, soit puits, soit feuille. 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).
- Les ciseaux tombent dans le puits (le puits gagne).
- La feuille recouvre le puits (la feuille gagne).

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.

1. Écrivez un programme qui tire aléatoirement une valeur parmi 1 (Ciseaux), 2 (Puits) et 3 (Feuille) et affiche la valeur choisie.
2. Écrivez un programme complet qui :
 - demande au joueur humain de choisir entre ciseaux, puits ou feuille, ou encore d'arrêter le jeu (fin du programme),
 - si le joueur n'a pas choisi d'arrêter le jeu
 - affiche le choix de l'ordinateur et indique qui a gagné le coup,
 - affiche le score après chaque tour,
 - répète ces opérations tant que le joueur n'a pas choisi d'arrêter.

Exercice 7 (*approfondissement*) – Retour sur les boucles : Multiplication à la russe

Le but de la multiplication à la russe est de réaliser n'importe quelle multiplication d'entiers avec uniquement des multiplications/divisions (entières) par 2 et des additions (ce qui est très utile pour faire des multiplications en binaire dans un microprocesseur). Elle repose sur la formule suivante :

$$a * b = (a/2) * (b * 2) + (a\%2) * b$$

Une variable `cumul` est utilisée afin de calculer le résultat de la multiplication. Si le reste de la division entière (r) de a par 2 vaut 1, alors b est ajouté à `cumul` (ne pas oublier d'initialiser `cumul` à 0). Sinon, `cumul` est inchangé. La nouvelle valeur de a est le résultat de la division entière (q) de a par 2. Avant de réitérer, il ne faut pas oublier de remplacer b par $b * 2$.

La multiplication est terminée lorsque le quotient de la division de a par 2 est nul. Le résultat est alors dans `cumul`.

Voici un exemple de multiplication à la russe sur $218 * 15$:

a	b	q=a/2	r = a%2	cumul
218	15	109	0	0
109	30	54	1	30
54	60	27	0	30
27	120	13	1	30+120=150
13	240	6	1	150+240=390
6	480	3	0	390
3	960	1	1	390+960=1350
1	1920	0	1	1350+1920=3270

donc $218 * 15 = 3270$

1. Écrivez un programme principal qui réalise la multiplication à la russe après avoir demandé à l'utilisateur de saisir des nombres entiers à multiplier. Vous afficherez la multiplication réalisée ainsi que le résultat obtenu.

TME Semaine 4 - 2^{ème} partie

TME solitaire

Version du 18 décembre 2014

Objectif(s)

★ Contrôle sur machine

Introduction

Le second créneau de TME de cette semaine est consacré à un exercice pratique sur machine, appelé “TME solitaire”. Par demi-groupes de 15 étudiants, vous allez tous composer pendant 45 minutes seul devant votre machine.

Le but de ce TME solitaire est d'évaluer votre capacité à écrire des programmes C syntaxiquement corrects et à résoudre des problèmes simples en C. Le sujet se découpe en 4 exercices.

Le premier exercice est un exercice de debug : il s'agit simplement de corriger les erreurs de compilation et d'exécution dans le programme qui vous est fourni.

Les trois exercices suivants consistent à compléter un fichier `.c` pour obtenir un programme précis.

Pour ces 3 exercices, vous perdrez un tiers des points si votre programme ne compile pas (syntaxe non correcte), même s'il est correct à 95%. **Vous devrez systématiquement vous assurer que ce que vous avez écrit compile sans erreur.**

Le reste des points est attribué en fonction :

- de l'adéquation entre votre code et la résolution du problème de l'énoncé (votre code résout le problème posé) ; attention à considérer les différents cas de tests possibles lors de l'écriture de votre code
- de la qualité du code : utilisation des instructions appropriées, clarté du code, indentation, présence de commentaires, etc..

Il est très facile d'avoir 10/20 à ce TME solitaire (commencez par vous assurer que tous vos programmes compilent).

Pour commencer

Créez un répertoire `TMEsol01` dans votre répertoire personnel. Comme lors des TME habituels, vous devrez récupérer des fichiers `.c` dans le répertoire que vous indiquera votre enseignant puis les enregistrer dans le répertoire `TMEsol01` pour pouvoir travailler dessus. Vous avez en tout 4 fichiers à récupérer : `ex1.c`, `ex2.c`, `ex3.c` et `ex4.c`.

À la fin de la séance, vous ferez une archive du répertoire `TMEsol01` que vous enverrez via la page de soumission habituelle.

Exercice 1 (4 points)

Le fichier `ex1.c` contient des erreurs qui empêchent sa compilation ou son exécution correcte. Corrigez ces erreurs.

Attention : votre programme doit non seulement compiler, mais en plus s'exécuter correctement par rapport à ce qui est indiqué en commentaire.

L'exercice est noté sur 4 points. Chaque erreur de compilation ou d'exécution restante coûte 1 point.

Exercice 2 - Facile (5 points)

Complétez le fichier `ex2.c` pour que le programme soit conforme à ce qui est indiqué en commentaire.

Certaines parties du programme ne doivent pas être modifiées. En particulier, vous ne devez pas supprimer les constantes définies à l'aide de la primitive `#define`.

Exercice 3 - Facile (5 points)

Complétez le fichier `ex3.c` (mêmes remarques que précédemment).

Exercice 4 - Moyen (6 points)

Complétez le fichier `ex4.c` (mêmes remarques que précédemment).

Savoir-faire acquis

- ★ Savoir choisir une instruction de répétition
- ★ Savoir répéter des instructions tant qu'une condition est vérifiée
- ★ Utiliser le tirage aléatoire de nombres
- ★ Afficher un tableau
- ★ Échanger le contenu de deux cases d'un tableau

Applications classiques à connaître

- ★ Calculer une moyenne pondérée
- ★ Utiliser les tirages aléatoires pour programmer un jeu simple
- ★ Multiplication à la russe



TD Semaine 5 Les tableaux II

Version du 18 décembre 2014

Objectif(s)

- ★ Parcours de tableau à une dimension avec une condition d'arrêt autre que la taille du tableau
- ★ Manipulation d'un tableau à deux dimensions
- ★ Initialisation incomplète d'un tableau
- ★ Chaînes de caractères

Exercice(s)

Exercice 1 (*base*) – Parcours (potentiellement) incomplet d'un tableau

1. Écrivez un programme dans lequel sont déclarés deux tableaux d'entiers de même taille (cette taille est définie au moyen de la directive `#define`). Le programme affiche *vrai* si le contenu des deux tableaux est identique, *faux* sinon. Si le contenu des deux tableaux n'est pas identique, votre programme doit se terminer dès qu'une différence est trouvée.

Exercice 2 (*base*) – Tableaux de caractères

Un tableau de caractères peut être initialisé avec une chaîne de caractères. Si la chaîne de caractères contient n caractères alors le tableau contiendra $(n + 1)$ cases : chacune des n premières cases contient le caractère correspondant dans la chaîne et la $(n + 1)^{\text{ème}}$ case contient le caractère de contrôle noté `'\0'` qui signale la fin de la chaîne.

1. Déclarez un tableau de caractères `chaîne` initialisé à la valeur "bonjour".
2. Écrivez un programme qui indique la longueur de la chaîne de caractères associée à un tableau de caractères. Par exemple, la longueur de la chaîne "bonjour" est de 7, la longueur de la chaîne "" est 0.

Exercice 3 (*base*) – Manipulation de tableau à deux dimensions

Un groupe de `NB_AMIS` amis, partant en vacances pour un séjour de `NB_JOURS` jours, souhaite gérer dans un tableau la participation financière de chacun aux frais du séjour. Chaque colonne du tableau représente les dépenses d'une journée, il y a une ligne par membre du groupe.

1. Écrivez un programme qui déclare un tableau permettant de stocker l'ensemble des dépenses et qui initialise à 0 le contenu de chacune des cases du tableau.
2. 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 soldées 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.

Écrivez un ensemble d'instructions qui tire aléatoirement pour chaque journée un montant de dépenses entre 30 et 50 euros et qui modifie le tableau en conséquence, sachant que l'identité du payeur est aussi obtenue par un tirage aléatoire.

Exercice 4 (*renforcement*) – Recherche d'un élément dans un tableau

On souhaite comparer l'efficacité de différentes méthodes de recherche d'un élément dans un tableau à une dimension.

1. On considère, dans un premier temps, un tableau d'entiers de taille N non trié. Soit un tableau `tab` d'entiers non trié et une variable `elem` de type entier.

Écrivez un programme qui détermine si la valeur de `elem` est présente dans `tab`.

2. On suppose maintenant que le tableau `tab` est un tableau d'entiers triés par ordre croissant. L'élément le plus petit est donc dans la première case du tableau et le plus grand dans la dernière case.
3. Comparez le nombre de fois où on passe dans la boucle dans chacun des programmes. Déduisez-en la méthode la plus efficace.

Exercice 5 (*renforcement*) – Recherche dichotomique d'un élément dans un tableau trié

La recherche dichotomique consiste à déterminer si un élément appartient à un tableau trié en divisant par 2 l'intervalle de recherche à chaque itération. Les opérations à effectuer lors d'une itération sont donc :

- déterminer la position de l'élément qui se trouve au milieu de l'intervalle courant ; si cet élément est l'élément recherché, l'algorithme se termine avec une réponse positive ;
- sinon
 - si l'élément du milieu est supérieur à l'élément recherché, poursuivre la recherche dans l'intervalle à gauche de l'élément du milieu ;
 - si l'élément du milieu est inférieur à l'élément recherché, poursuivre la recherche dans l'intervalle à droite de l'élément du milieu ;

On note g et d les indices délimitant l'intervalle de recherche à gauche et à droite respectivement.

1. Quelle condition permet de déterminer que l'élément recherché ne figure pas dans le tableau ?
2. Que valent :
 - l'indice de l'élément au milieu de l'intervalle ?
 - les bornes du nouvel intervalle de recherche si l'élément du milieu est supérieur à l'élément recherché ?
 - les bornes du nouvel intervalle de recherche si l'élément du milieu est inférieur à l'élément recherché ?
3. Écrivez un programme qui initialise un tableau de N nombres entiers avec des valeurs triées dans l'ordre croissant, puis qui demande à l'utilisateur de choisir une valeur à rechercher dans le tableau et affiche le résultat de la recherche.
4. Quelles sont les valeurs que vous choisiriez pour tester votre programme ?
5. Étudiez le nombre de fois où on passe dans la boucle de la recherche dichotomique. Comparez l'efficacité avec les méthodes de l'exercice précédent.

Exercice 6 (*approfondissement*) – Opérations de base sur les tableaux

On souhaite 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

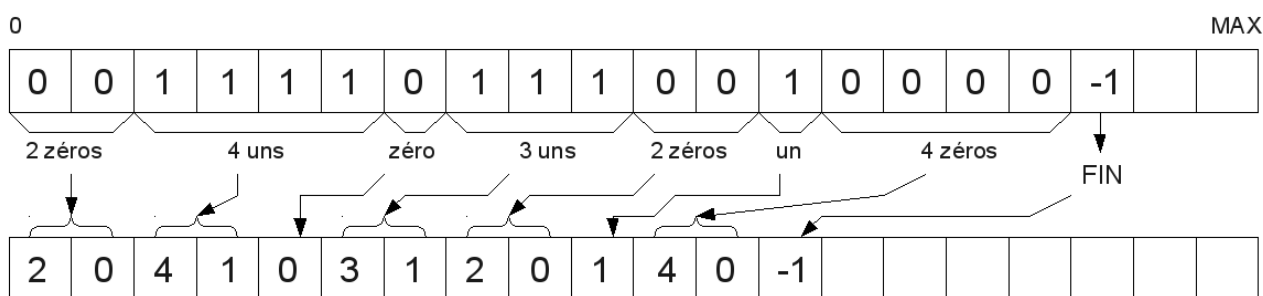
Dans notre programme C, les données brutes et les données compressées seront stockées dans deux tableaux d'entiers de taille $(MAX+1)$ appelés respectivement `brut` et `compress`.

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.

1. Écrivez un programme qui initialise le tableau `brut` des 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.

Nous vous recommandons aussi d'écrire, pour tester votre programme, une boucle d'affichage (ou bien d'afficher le tableau créé dans une fenêtre graphique assez grande).

2. Écrivez la partie du programme qui permet, à partir d'un tableau `brut` préalablement rempli (on n'a donc pas accès à la variable `taille` qui détermine le nombre de valeurs significatives dans le tableau), de stocker les données compressées dans le tableau `compress`. La séquence stockée se termine par -1.

L'idée est de parcourir le tableau `brut` en gérant une variable `compteur` que l'on réinitialise chaque fois que l'on commence une nouvelle séquence de valeurs.

Le parcours se termine évidemment lorsqu'on rencontre la valeur -1.

3. Écrivez la partie du programme qui permet, en parcourant le tableau `compress`, d'afficher les données brutes sur la console.

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.

Savoir-faire acquis

- ★ Initialiser un tableau de manière incomplète
- ★ Réaliser des opérations de base dans un tableau à deux dimensions
- ★ Parcourir un tableau tant qu'une condition n'est pas vérifiée
- ★ Parcourir un tableau de manière non séquentielle

Applications classiques à connaître

- ★ Calcul de la longueur d'une chaîne de caractères
- ★ Recherche dichotomique
- ★ Compression/décompression de séquences de 0 et de 1



TME Semaine 5 Les tableaux II

Version du 18 décembre 2014

Objectif(s)

- ★ Initialisation et parcours incomplets de tableaux à une dimension
- ★ Tableaux de caractères et chaînes de caractères
- ★ Tableaux à 2 dimensions

Exercice(s)

Exercice 1 (*base*) – Marqueur de fin de données et moyenne arithmétique

1. Écrivez un programme qui demande à l'utilisateur de saisir au plus MAX nombres flottants positifs. Si l'utilisateur souhaite saisir moins de MAX valeurs, il termine sa saisie en tapant -1. Les valeurs saisies, y compris le -1 seront stockées dans un tableau. Vous choisirez le type de boucle le mieux adapté pour une bonne lisibilité du programme.
2. Complétez le programme pour afficher (sans utiliser de fonctions graphiques) le tableau que vous venez de saisir. La valeur -1 représentant la fin des valeurs significatives ne doit pas être affichée.
3. Complétez le programme pour calculer la moyenne des nombres saisis. **Attention**, l'utilisateur peut n'avoir saisi aucune valeur significative.

Exercice 2 (*base*) – Chaînes de caractères

Un tableau de caractères peut être initialisé avec une chaîne de caractères. Si la chaîne de caractères contient n caractères alors le tableau contiendra $(n + 1)$ cases : chacune des n premières cases contient le caractère correspondant dans la chaîne et la $(n + 1)^{\text{ème}}$ case contient le caractère de contrôle noté `' \0 '` qui signale la fin de la chaîne.

1. Écrivez un programme qui demande à l'utilisateur de saisir un caractère qui vous servira à initialiser la variable `carac` et, étant donnée une chaîne de caractères initialisée lors de sa déclaration, affiche le nombre d'occurrences du caractère `carac` dans la chaîne.

Exercice 3 (*base*) – Tableau à deux dimensions

Nous souhaitons compléter le programme du TD calculant les dépenses effectuées lors d'un séjour de vacances. Il faut donc, dans un premier temps, implémenter les opérations écrites lors du TD (c'est l'occasion de vérifier que vous êtes capables de les (re)faire...).

1. Complétez le programme pour afficher les dépenses effectuées par 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

2. Complétez le programme pour afficher le solde total de chacun des membres du groupe.

Exercice 4 (*renforcement*) – Palindromes

1. Écrivez un programme qui indique si une chaîne de caractères est un palindrome. Un palindrome est un mot qui se lit de la même manière à l'endroit et à l'envers (par exemple les mots "rever" et "radar" sont des palindromes). Vous testerez votre programme avec différentes chaînes de caractères que vous devrez indiquer en commentaire dans le programme en justifiant ce que chaque chaîne choisie vous permet de tester.

Exercice 5 (*renforcement*) – Tableaux équilibrés

Un tableau `tab` d'entiers de taille `N` est équilibré si et seulement si :

il existe un indice `p`, $0 \leq p < N$ tel que $\text{tab}[0] + \text{tab}[1] + \dots + \text{tab}[p-1] = \text{tab}[p+1] + \dots + \text{tab}[N-1]$, `p` est alors nommé indice d'équilibre.

La somme des éléments d'un tableau vide est égale à 0, ceci peut être utile si `p=0` ou `p=N-1`.

Exemple : Considérons le tableau `tab[N]={-7, 1, 5, 2, -4, 3, 0}` où `N=7`, `p=3` et `p=6` sont des indices d'équilibre. Si le tableau a plusieurs indices d'équilibre, vous afficherez le premier identifié.

1. Recopiez le fichier `tableau-equilibre.c` et modifiez le pour que l'affichage obtenu soit correct (tableau équilibré ou non, indice d'équilibre si oui).

```

#include <stdio.h>

#define TAILLE 7

int main() {
    int tab[TAILLE]={-7,1,5,2,-4,3,0};

    /* A COMPLETER */

    if ( ..... ) {
        printf("indice d'equilibre : %d\n",....);
    } else {
        printf("tableau non equilibre\n");
    }
    return 0;
}

/* Valeurs pour lesquelles vous devez avoir teste votre programme

```

```

tab[TAILLE] = {-7, 1, 5, 2, -4, 3, 0};
    indice d'equilibre = 3
tab[TAILLE] = {7, 2, 3, 4, -4, -3, -2};
    indice d'equilibre = 0
tab[TAILLE] = {1, -1, 1, -1, 1, -1, 7};
    indice d'equilibre = 6
tab[TAILLE] = {1, 2, 3, 4, 5, 6, 7};
    Tableau non equilibre
*/

```

Exercice 6 (*entraînement*) – Miroir d'une chaîne de caractères

1. Écrivez un programme qui, étant donnée une chaîne de caractères stockée dans un tableau `chaine`, écrit dans le tableau `miroir` le miroir de `chaine` puis l'affiche à l'écran.

Le miroir d'une chaîne correspond à la chaîne de caractères obtenue quand on lit la chaîne initiale de droite à gauche. Par exemple, le miroir de "bonjour" est "ruojnob", et le miroir de "rever" est "rever".

Pour pouvoir déclarer le tableau `miroir` sans compter le nombre de caractères de `chaine`, vous pouvez l'initialiser avec la même valeur que `chaine`. N'oubliez pas de tester votre programme avec des chaînes de caractères de différentes tailles.

2. Recopiez le programme de la question précédente et modifiez le pour que la chaîne miroir soit écrite dans le tableau `chaine`. Si initialement le tableau `chaine` contient "bonjour", il devra contenir "ruojnob" à la fin du programme.

Contrainte : votre programme ne doit pas utiliser d'autre tableau que la variable `chaine`.

Exercice 7 (*entraînement*) – Affichage du nombre d'occurrences des valeurs d'un tableau

On repart du tableau créé dans l'exercice 1 (vous pouvez compléter ce programme, ou recopier le code correspondant).

1. Affichez le nombre d'occurrences de chaque valeur contenue dans le tableau. Si une même valeur apparaît plusieurs fois, son nombre d'occurrences sera affiché à chaque fois.

Si le tableau contient les valeurs {1.5, 2.0, 3.2, 1.5, 2.0, 2.0} votre programme doit afficher :

```

La valeur 1.500000 apparait 2 fois dans le tableau.
La valeur 2.000000 apparait 3 fois dans le tableau.
La valeur 3.200000 apparait 1 fois dans le tableau.
La valeur 1.500000 apparait 2 fois dans le tableau.
La valeur 2.000000 apparait 3 fois dans le tableau.
La valeur 2.000000 apparait 3 fois dans le tableau.

```

2. Modifiez votre programme pour qu'il n'affiche que les valeurs qui apparaissent plus d'une fois dans le tableau, ainsi que leur nombre d'occurrences. Chaque valeur ne devra être affichée qu'une seule fois.

Pour le même tableau qu'à la question précédente, vous devez obtenir l'affichage suivant :

```

La valeur 1.500000 apparait 2 fois dans le tableau.
La valeur 2.000000 apparait 3 fois dans le tableau.

```

Exercice 8 (*entraînement*) – Identification d'une sous chaîne

Le but de cet exercice est d'écrire un programme permettant de savoir si une chaîne de caractères est incluse dans une autre.

1. Ecrivez un programme qui, étant données deux chaînes de caractères `chaine1` et `chaine2`, affiche si `chaine1` débute par `chaine2`.

Par exemple, si `chaine1 = "bonjour"` et `chaine2 = "bon"`, `chaine1` débute par `chaine2` ; ce qui n'est pas le cas si `chaine2 = "on"`. Les chaînes de caractères seront initialisées lors de leur déclaration.

2. Modifiez votre programme pour qu'il détermine si `chaine2` est incluse dans `chaine1` et, dans l'affirmative, affiche l'indice de `chaine1` à partir duquel `chaine2` commence.

Le principe est de tester si `chaine1` débute par `chaine2` (i.e, `chaine2` se trouve dans `chaine1` à partir du premier caractère). Si c'est le cas, `chaine2` est incluse dans `chaine1` à partir de l'indice 0. Si ce n'est pas le cas, il faut tester si `chaine2` se trouve dans `chaine1` à partir du deuxième caractère. Si ce n'est pas le cas, on continue à partir du troisième caractère et ainsi de suite.

Savoir-faire acquis

- ★ Remplir un tableau de manière incomplète à partir de valeurs saisies par l'utilisateur.
- ★ Effectuer des opérations classiques (affichage, moyenne) sur des tableaux remplis de manière incomplète.
- ★ Parcourir un tableau de caractères.
- ★ Parcourir partiellement un tableau.
- ★ Parcourir deux tableaux en même temps mais avec des indices évoluant différemment.
- ★ Rechercher plusieurs occurrences d'une même valeur dans un tableau
- ★ Parcourir un tableau à deux dimensions

Applications classiques à connaître

- ★ Vérification de palindromes
- ★ Inversion d'une chaîne de caractères
- ★ Test de l'inclusion d'un chaîne dans une autre
- ★ Identification de tableaux équilibrés



TD Semaine 6

Les fonctions et les structures

Version du 18 décembre 2014

Objectif(s)

- ★ Fonctions
- ★ Déclaration d'une fonction (signature ou prototype)
- ★ Définition d'une fonction
- ★ Appel d'une fonction
- ★ Pile d'exécution
- ★ Paramètres formels et paramètres d'appel
- ★ Passage par valeur des paramètres à l'appel d'une fonction
- ★ Déclaration et utilisation des structures
- ★ Portée des variables

Rappel de cours :

Un programme est toujours structuré de la manière suivante :

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

#define NOMS VALEURS

/* Les variables globales */
int xg, yg;
float zg;

/* Fonctions annexes (dans l'ordre) */
int fonction1(...) {
    ...
}
float fonction2(...) {
    ... /* peut éventuellement utiliser la fonction 1 */
}
void fonction3(...) {
    ... /* peut éventuellement utiliser les fonctions 1 et 2 */
}

/* la fonction main, qui utilise les fonctions précédentes */
int main() {
    ...
    return 0;
}
```

Il est très important de respecter cette structure lors de l'écriture de programmes pour éviter les erreurs de compilation inutiles...

Exercice(s)**Exercice 1 (*base*) – Appels de fonction**

1. Dites ce qu’affiche le programme suivant et donnez les évolutions de la pile d’exécution.

```
#include <stdio.h>

int calcul(int a, int b) {
    int x;

    if (a > b) {
        x = a-b;
    }
    else {
        x = b-a;
    }
    return x;
}

int main() {
    int res;

    res = calcul(7,2);
    printf("Le premier resultat du calcul est %d.\n",res);
    res = calcul(-15,3);
    printf("Le deuxieme resultat du calcul est %d.\n",res);
    return 0;
}
```

2. Modifiez la fonction `calcul` pour qu’elle n’utilise plus de variable locale (elle ne doit pas non plus utiliser de variable globale).
3. Dites ce qu’affiche le programme suivant et donnez les évolutions de la pile d’exécution.

```
#include <stdio.h>

int f1(int a, int b) {
    int res1;

    res1 = a*b;
    return res1;
}

int f2(int x, int y) {
    int res2;

    if (x < y) {
        res2 = x;
    }
    else {
        res2 = x + f1(x,y);
    }
    return res2;
}

int f3(int u, int v, int w) {
    int res3;
```

```
    res3 = f2(u,w) + f2(v,w);  
    return res3;  
}  
  
int main() {  
    int res;  
  
    res = f3(4,2,3);  
    printf("%d\n", res);  
    return 0;  
}
```

Exercice 2 (base) – Structures et fonctions

L'objectif de cet exercice est de définir des structures et des fonctions, travaillant sur des structures, afin de manipuler des objets géométriques.

L'objet de base que nous considérons est le point. Il est défini par ses coordonnées x et y (de type entier) et une couleur d'affichage (du type chaîne de caractères limitée à 10 caractères, n'oubliez pas de réserver la place pour le caractère `'\0'` qui signale la fin de chaîne).

1. Expliquez pourquoi un point ne peut pas être représenté par un tableau à trois éléments. Quel type est alors le plus adapté ?
2. Donnez le code de la structure `point`. Déclarez et initialisez ensuite un point nommé `mon_point`, situé aux coordonnées $x=3$, $y=4$ et de couleur rouge.
3. On veut maintenant pouvoir décrire des rectangles dont les côtés sont horizontaux et verticaux. Un rectangle est alors déterminé par les coordonnées de deux coins opposés (en bas à gauche et en haut à droite), la couleur d'affichage des traits et celle du fond. Écrivez une structure `rectangle` en utilisant la structure `point`.
4. Déclarez et initialisez une variable `un_rectangle` de type `rectangle` dont les deux sommets opposés sont respectivement aux coordonnées (100, 200) et (300, 2), la couleur d'affichage des traits et des points est le rouge et la couleur du fond est le blanc. Vous proposerez une solution dans laquelle aucune variable de type `point` n'est déclarée et une autre dans laquelle deux variables de type `point` sont déclarées et utilisées pour définir la variable `un_rectangle`.
5. Écrivez une fonction `point_dans_rectangle` qui prend un point et un rectangle en paramètres et qui retourne 1 (vrai) si le point est dans le rectangle (les bords du rectangle sont dans le rectangle) et 0 (faux) sinon. On vous rappelle que le repère associé à une fenêtre a son origine dans le coin en haut à gauche.

Exercice 3 (base) – Afficher et retourner

1. On considère la fonction `main` suivante :

```
int main() {  
    int a, b, c;  
  
    a = 5;  
    b = 3;  
    c = min2Int(a,b);  
    printf("le minimum de %d et de %d est %d.\n", a, b, c);  
    return 0;  
}
```

Définissez la fonction `min2Int` appelée dans la fonction `main` précédente et qui **retourne** le minimum de 2 entiers.

2. En utilisant la fonction `min2Int` définie à la question précédente, écrivez une nouvelle fonction `min3Int` qui **affiche** le minimum de 3 entiers.
3. Écrivez la fonction `main` permettant de tester la fonction `min3Int`.
4. Peut-on utiliser la fonction `min3Int` (et uniquement la fonction `min3Int`) pour afficher le minimum de 5 entiers ?

Exercice 4 (*base*) – Portée des variables

Cet exercice est un classique qu'on retrouve souvent à l'examen.

1. Considérons le programme suivant :

```
#include <stdio.h>

int a, b, c;

int f(int x) {
    return a*x;
}

int g() {
    int c;

    c = a;
    a = b;
    b = c;
    return c;
}

int main() {
    int x, y;

    a = 2;
    b = 3;
    c = 4;
    x = f(a);
    y = g();

    printf("%d %d %d %d %d\n", x, y, a, b, c);
    return 0;
}
```

1. Quelles sont les variables globales ?
2. Quelles sont les variables locales à `f` ?
3. Quelles sont les variables locales à `g` ?
4. Quelles sont les variables locales à `main` ?
5. Quels sont les paramètres formels de `f` ?
6. Quels sont les paramètres formels de `g` ?
7. Quels sont les paramètres effectifs de `f` ?
8. Que vaut `c` à la fin de l'appel à la fonction `g` ?
9. Que fait la fonction `g` ?
10. Qu'affiche le programme ?
11. Que retourne le programme ?

Exercice 5 (base) – Réutiliser et imbriquer des fonctions

1. Donnez la signature, écrivez puis testez une fonction `divise` qui prend en argument deux entiers et retourne un entier. La fonction retourne la valeur 1 lorsque le premier argument est un diviseur du second argument. Elle retourne la valeur 0 sinon.
2. En utilisant la fonction `divise`, écrivez une fonction `ecrire_division` qui prend en argument deux entiers et affiche un message indiquant si le second divise le premier. Par exemple :

```
ecrire_division(4, 2) -> 2 divise 4
ecrire_division(5, 2) -> 2 ne divise pas 5
```

3. Toujours en utilisant la fonction `divise`, écrivez maintenant une fonction `nb_div` qui renvoie le nombre de diviseurs d'un entier `n` passé en paramètre.
4. Écrivez une fonction `premier` qui renvoie vrai ou faux selon que l'entier `n` passé en paramètre est premier ou non.
5. Considérons la fonction `main` suivante :

```
int main() {
    int x = 4;

    printf("%d", x);
    if (premier(x)) {
        printf(" est premier\n");
    }
    else {
        printf(" n'est pas premier\n");
    }
    return 0;
}
```

Donnez la liste des appels de fonctions effectués par ce programme.

6. Que se passe-t-il si on modifie la fonction `divise` de la manière suivante :

```
int divise(int a, int b) {
    return premier(a) && premier(b);
}
```

Exercice 6 (approfondissement) – Types et appels de fonctions

Soit le programme suivant (volontairement sans commentaire) :

```
#include <stdio.h>

int f1(int a, int b) {
    return a*b;
}

int f2(int a, int b) {
    if (b) {
        return a*a;
    }
    else {
        return -1;
    }
}
```

```
int f3(int a, char bc) {
    if (bc == 'c') {
        return a*a;
    }
    else {
        return -1;
    }
}

int main() {
    printf("%d\n", f1(3, 2) + f2(2, 1) - f3(4, 'g'));
    printf("%d\n", f1(3, 2) + f2(2, 0) - f3(4, 'c'));
    return 0;
}
```

1. Qu'affiche ce programme ? Expliquez.
2. Que se passe-t-il si on ajoute les instructions suivantes dans la fonction main ? Expliquez.

```
printf("%d\n", f1(2, 3.0));
printf("%d\n", f3(2, 3));
printf("%d\n", f3(2, z));
printf("%d\n", f3(2, "t"));
```

3. Donnez des noms plus explicites aux fonctions f1, f2 et f3.

Savoir-faire acquis

- ★ Savoir déclarer et utiliser une structure
- ★ Savoir déterminer la signature d'une fonction
- ★ Savoir écrire une fonction qui calcule un résultat, une fonction qui ne modifie pas l'environnement (e.g. affichage)
- ★ Savoir distinguer retourner un résultat et afficher un résultat
- ★ Savoir appeler une fonction
- ★ Savoir utiliser des variables locales et des paramètres
- ★ Étant donné un problème décomposé en sous-problèmes, écrire des petites fonctions qui résolvent des sous-problèmes, pour compléter le programme qui appelle ces fonctions pour résoudre le problème



TME Semaine 6

Les fonctions et les structures

Version du 18 décembre 2014

Objectif(s)

- ★ Les deux premiers exercices permettent d'écrire des fonctions simples et de les utiliser pour obtenir différents résultats.
- ★ Le troisième exercice permet d'écrire des fonctions, manipulant des entiers et des structures, qui sont ensuite utilisées pour dessiner le triangle de Sierpinski qui est le sujet de l'exercice 4.
- ★ Le cinquième exercice généralise le triangle de Sierpinski à d'autres figures.

Exercice(s)

Exercice 1 (*base*) – Polynômes

Vous allez écrire les fonctions et le programme permettant de travailler sur des polynômes du second degré ($ax^2 + bx + c$). Nous représenterons un polynôme par ses coefficients a , b et c réels avec a non nul.

1. Écrivez une fonction `discriminant` qui prend en paramètre trois réels, les coefficients d'un polynôme du second degré, et qui retourne le discriminant du polynôme.
2. En réutilisant la fonction précédente, écrivez une fonction `racine_poly_dg2` qui prend en paramètre les trois coefficients d'un polynôme du second degré et qui affiche sur la sortie standard ses racines.

Vous pourrez utiliser la fonction `sqrt` qui prend un réel en paramètre et retourne sa racine carrée. Cette fonction est définie dans la bibliothèque mathématique `math.h` que vous devrez donc inclure dans votre programme.

3. Écrivez maintenant un programme complet. Pour cela, vous devez :
 - écrire une fonction `ecrire_poly` qui prend en paramètre les trois coefficients d'un polynôme et affiche le polynôme. Seuls les monômes dont le coefficient est non nul seront affichés (*rappel* : nous faisons l'hypothèse que a est non nul). Si les coefficients sont 3, 5 et 2, l'affichage doit être :
 $3.00*x^2 + 5.00*x + 2.00$
Si les coefficients sont 3, 0 et 2, l'affichage doit être :
 $3.00*x^2 + 2.00$
Aide : pour afficher uniquement deux chiffres après la virgule, il faut utiliser le format `% .2f`.
 - écrire une fonction `main` pour tester vos fonctions. Votre programme devra contenir un jeu de tests complet et la justification de chaque test en commentaire.

Exercice 2 (base) – Affichage points

Le programme suivant permet d'afficher une ligne horizontale de points dont la couleur, bleu ou blanc, est choisie aléatoirement.

```
#include <stdlib.h>
#include <time.h>
#include <cini.h>
#define MIN 0
#define MAX 300

void affiche_pixel_couleur(int x, int y, char color1[], char color2[]) {
    int color;

    color = rand()%2;
    if (color == 1) {
        CINI_draw_pixel(x, y, color1);
    }
    else {
        CINI_draw_pixel(x, y, color2);
    }
}

int main() {
    int i;

    srand(time(NULL));
    CINI_open_window(MAX, MAX, "Pixels");
    for (i = MIN; i < MAX; i++) {
        affiche_pixel_couleur(i, MAX/2, "blue", "white");
    }
    CINI_loop();
    return 0;
}
```

Pour chacune des questions suivantes, vous devez :

- recopier le programme de la question précédente (de l'énoncé pour la première question, accessible dans le répertoire `/Infos/lmd/2014/licence/ue/1I002-2015fev/Semaine7`),
- utiliser les "constantes" définies par les primitives **#define**,
- utiliser **obligatoirement** les fonctions précédemment définies lorsque c'est possible.

1. Recopiez le programme et :

- écrivez une fonction `remplir_points` qui prend quatre entiers en paramètre (`minX`, `maxX`, `minY` et `maxY`) et deux chaînes de caractères correspondant à des couleurs et qui affiche tous les pixels dont l'abscisse est comprise entre `minX` inclus et `maxX` exclus, et l'ordonnée entre `minY` inclus et `maxY` exclus (l'ensemble de ces pixels représente un rectangle dont les côtés sont parallèles aux bords de la fenêtre graphique). La couleur de chaque pixel est choisie aléatoirement entre les deux couleurs passées en paramètre.
- modifiez la fonction `main` pour que la fenêtre graphique soit remplie de pixels blancs ou bleus dont la couleur est choisie aléatoirement.

2. Recopiez le programme de la question précédente et :

- écrivez une fonction `afficher_colonnes` qui prend cinq entiers en paramètre (`minX`, `maxX`, `minY`, `maxY` et `taille`) et quatre chaînes de caractères correspondant à des couleurs. La fonction remplit le rectangle défini par les quatre premiers entiers avec des colonnes de `taille` pixels de large. Toutes les colonnes d'indice pair (on les numérote à partir de 0) sont remplies de points dont la couleur est choisie aléatoirement entre les deux premières couleurs passées en paramètre. Toutes les colonnes d'indice impair sont remplies de points dont la couleur est choisie aléatoirement entre les deux couleurs suivantes.
- modifiez la fonction `main` pour que la fenêtre graphique soit remplie de colonnes de largeur 5 pixels et de couleurs (rouge, jaune) et (bleu,vert).

Votre programme doit donner un affichage similaire à celui de la figure 1.a (mais en couleur !).

3. Recopiez le programme de la question précédente et :
 - modifiez la fonction `main` pour que l’affichage obtenu soit similaire à celui représenté par la figure 1.b.
 La fenêtre est découpée en quatre zones de même taille, et chacune d’elles est remplie par des colonnes comme dans la question précédente. Les couleurs des zones qui ont un côté commun ne doivent pas être les mêmes. Les colonnes ont 10 pixels de large.

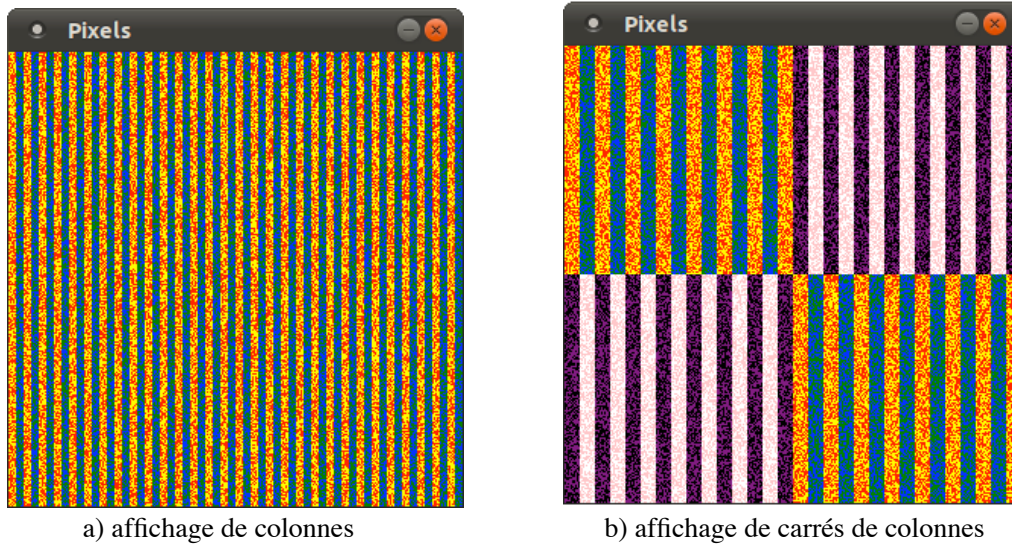


FIGURE 1 – Colonnes

4. Recopiez le programme de la question précédente et :
 - écrivez une fonction `afficher_damier` qui prend cinq entiers en paramètre (`minX`, `maxX`, `minY`, `maxY` et `taille`) et quatre chaînes de caractères correspondant à des couleurs et qui affiche un damier dont les cases font `taille` pixels sur `taille` pixels. Les pixels de chaque case sont de deux couleurs, la couleur de chaque pixel est choisie aléatoirement. Les cases ayant un côté commun ont des pixels de couleurs différentes.
 - modifiez la fonction `main` pour que l’affichage obtenu soit similaire à celui représenté par la figure 2.a.

Aide : Chaque ligne du damier est un rectangle rempli avec des colonnes.
5. Recopiez le programme de la question précédente et :
 - modifiez la fonction `main` pour que l’affichage obtenu soit similaire à celui représenté par la figure 2.b.
 La fenêtre est découpée en quatre zones de même taille, et chacune d’elles est remplie par un damier comme dans la question précédente. Les couleurs des zones qui ont un côté commun ne doivent pas être les mêmes.

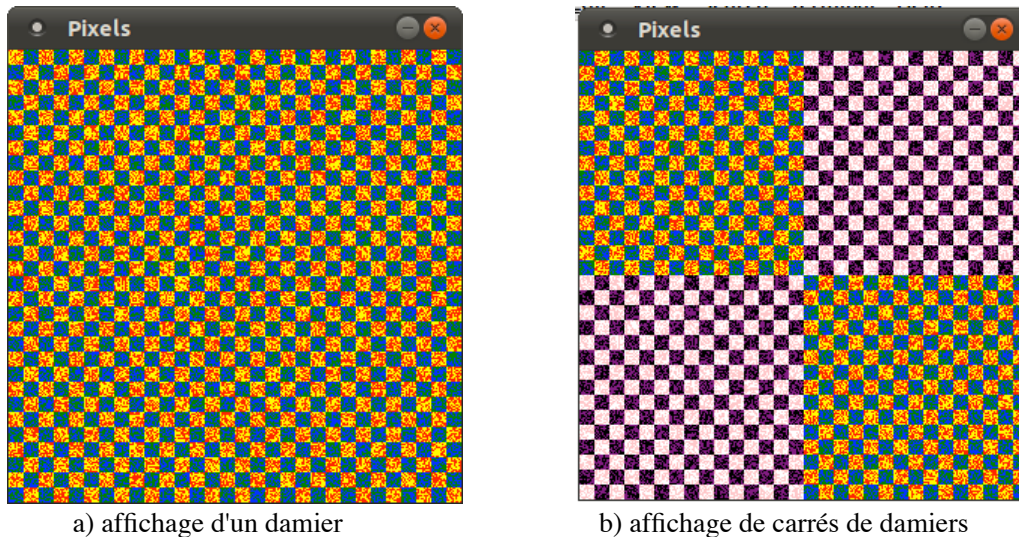


FIGURE 2 – Damiers

Exercice 3 (*renforcement*) – Fonctions et structures

Le type que vous allez définir et les fonctions que vous allez écrire dans cet exercice vont être réutilisés dans l'exercice suivant. Il est donc **impératif** que vous respectiez les contraintes énoncées (fonctionnalité, paramètres, type de retour, ...).

1. Déclarez la structure `point` qui regroupe les deux coordonnées entières d'un point.
2. Écrivez une fonction `milieu` qui, étant donnés deux points a et b , renvoie le point de coordonnées entières se trouvant au milieu du segment $[ab]$. Par exemple, si a est le point de coordonnées $(3, 3)$ et b le point de coordonnées $(5, 6)$, le milieu du segment $[ab]$ aura les coordonnées $(4, 4)$.
Écrivez une fonction `main` permettant de tester votre fonction.
3. Écrivez une fonction `choix` qui renvoie un entier de l'ensemble $S = \{0, 1, 2\}$ choisi aléatoirement.
Écrivez une fonction `main` permettant de tester votre fonction en :
 - choisissant 10000 valeurs,
 - comptant et affichant le nombre de 0, de 1 et de 2 choisis,
 - dès qu'une valeur choisie n'est pas dans l'ensemble S , le programme doit s'arrêter en affichant un message d'erreur (si ce cas se produit lors de l'exécution c'est que votre programme n'est pas correct, il faut le corriger).
4. Écrivez une fonction `coord_D` qui, étant donnés trois points, a, b, c , et un entier $s \in S = \{0, 1, 2\}$ passés en paramètre, renvoie le point a si $s = 0$, le point b si $s = 1$, et le point c si $s = 2$. Écrivez une fonction `main` permettant de tester votre fonction (vous devez tester votre fonction pour toutes les valeurs possible de s). Vous ferez l'hypothèse que la valeur de s appartient bien à l'ensemble S , ce n'est donc pas à vérifier.
5. Pour cette question (et pour la suite du TME), nous allons utiliser la bibliothèque graphique et en particulier la fonction :

```
void CINI_draw_disc(int x, int y, int rayon, char color[]);
```

définie dans notre bibliothèque CINI.

Écrivez une fonction `draw_triangle` prenant en paramètres 3 points et qui affiche dans une fenêtre graphique le triangle dont ces 3 points sont les sommets. Ceux-ci seront représentés par des disques de rayon 5. Vous devez obtenir un affichage similaire à celui de la figure 3.

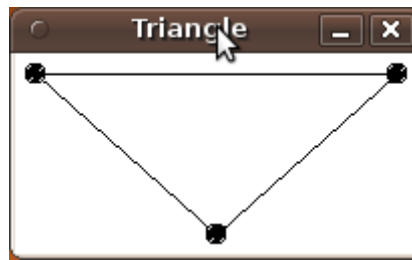


FIGURE 3 – Dessin d'un triangle

Exercice 4 (renforcement) – Triangle de Sierpinski

Les fractales sont des figures se répétant à l'infini, identiques à elles-mêmes. Ce sont des formes qu'on trouve couramment dans la nature (motifs de feuilles, de coquillages, etc.).

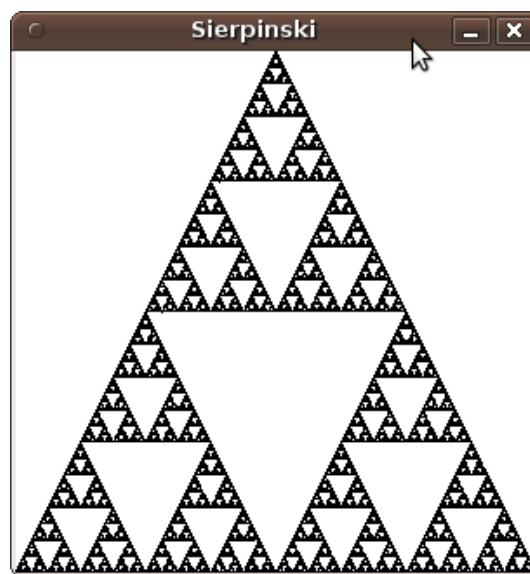


FIGURE 4 – Un triangle de Sierpinski

Le but de cet exercice est de réaliser un programme qui dessine un triangle de Sierpinski, comme celui de la figure 4. C'est l'un des exemples les plus classiques de fractale en programmation. Nous allons vous guider pas à pas pour la construire.

Pour réaliser cette figure, vous allez réutiliser les fonctions définies dans l'exercice précédent.

1. Pour commencer, recopiez dans un seul fichier `sierpinski.c` la structure `point` et toutes les fonctions définies dans l'exercice précédent. Ajoutez les primitives **#define** définissant les constantes `XA`, `YA`, `XB`, `YB`, `XC` et `YC` correspondant aux coordonnées des trois extrémités de votre triangle de Sierpinski ainsi que `R` pour le rayon de vos disques.

Écrivez ensuite une fonction `sierpinski` qui prend en paramètre les trois points correspondant aux sommets du triangle, et dessine un triangle *ABC* (en appelant la fonction `draw_triangle` avec les coordonnées des trois sommets).

Vous testerez le tout à l'aide d'une fonction `main` qui :

- déclare et initialise les trois points en utilisant les constantes,
- crée la fenêtre graphique,
- appelle la fonction `sierpinski`,
- attend que l'utilisateur ferme la fenêtre.

2. L'idée générale de l'algorithme de Sierpinski que nous utilisons dans ce TME est de tracer successivement des points au **milieu** d'un segment invisible reliant un point déjà tracé de la figure à l'un des sommets du triangle ABC . Le nouveau point sert alors de sommet pour le prochain segment.

Pour commencer, modifiez votre fonction `sierpinski` en y ajoutant la variable locale m qui représente un point qui doit être le milieu du segment $[BC]$. Après avoir tracé le triangle ABC , vous dessinerez un disque de rayon R centré en m .

Testez votre nouveau programme. Le point m sera le premier "milieu" de notre figure de Sierpinski, situé sur le segment BC .

3. Modifiez votre fonction `sierpinski` pour qu'après avoir dessiné le point m , elle choisisse un point au hasard parmi A , B et C (appelons-le D) et trace un trait **rouge** entre le point M et le point D (c'est le deuxième "segment invisible" de la figure).

La fonction `choix` définie dans l'exercice 3 nous servira à choisir au hasard un des 3 sommets A , B ou C .

En utilisant la fonction `coord_D` définie dans l'exercice précédent, on pourra calculer les coordonnées du point D .

Exécutez plusieurs fois votre programme pour vérifier qu'il choisit bien aléatoirement l'un des trois sommets possibles !

4. Complétez votre fonction `sierpinski` pour qu'après avoir tracé le segment $[MD]$, elle calcule les coordonnées du milieu de $[MD]$. Ce nouveau point correspondra au nouveau point M (deuxième "point milieu").

Pour tester votre programme, vous pouvez afficher ce nouveau point M en bleu, par exemple.

5. Dans l'algorithme de Sierpinski, on ne trace jamais de triangle, mais seulement des pixels aux points M successifs. Modifiez votre fonction `sierpinski` pour que :

1. elle n'affiche plus le triangle ABC (ni ses sommets) ;
2. après avoir calculé M , milieu de $[BC]$, elle affiche seulement un pixel au point M ;
3. après avoir choisi D , elle n'affiche plus le segments MD ;
4. elle calcule le nouveau point M milieu de $[MD]$ sans l'afficher.

Pour tracer un pixel, vous utiliserez la fonction `CINI_draw_pixel` présentée en cours.

6. En utilisant intelligemment quelques fonctions simples définies au début du TME, nous avons réussi à écrire les étapes suivantes dans notre fonction `sierpinski` :

1. Initialisation de M (choix du premier point de la figure) ;
2. Affichage de M ;
3. Choix d'un sommet parmi A , B et C ;
4. Calcul d'un nouveau point M ;

Pour dessiner le triangle de Sierpinski, il suffit de répéter les étapes 2 à 4 un grand nombre de fois (50 000 par exemple).

Modifiez votre fonction `sierpinski` en répétant 50 000 fois les étapes 2 à 4 dans une boucle **for**.

Exercice 5 (*approfondissement*) – Polyèdres de Sierpinski

L'objectif de cet exercice est de réaliser d'autres fractales en utilisant le même principe que celui du triangle de Sierpinski. Nous allons nous intéresser ici à la construction de polyèdres.

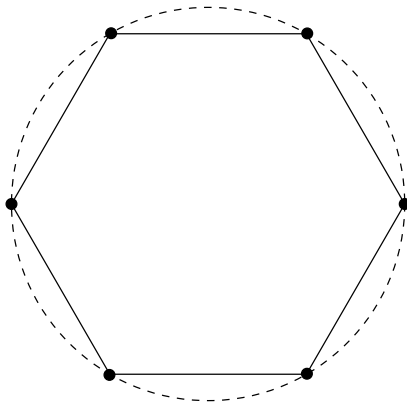
L'idée pour construire le polyèdre est la suivante : les sommets de ce dernier sont situés sur un cercle dont le centre est le centre du polyèdre. De plus, les sommets sont équi-espacés. Donc le i^{eme} sommet du polyèdre a pour coordonnées $(x_C + R \cos(i \times 2\pi/n), y_C + R \sin(i \times 2\pi/n))$, où (x_C, y_C) sont les coordonnées du centre du polyèdre, R est le rayon du cercle, et n le nombre de faces du polyèdre. Les fonctions `sin` et `cos` sont accessibles via `math.h`.

Par exemple, la fonction `main` suivante engendrera la fenêtre de la figure 5.b.

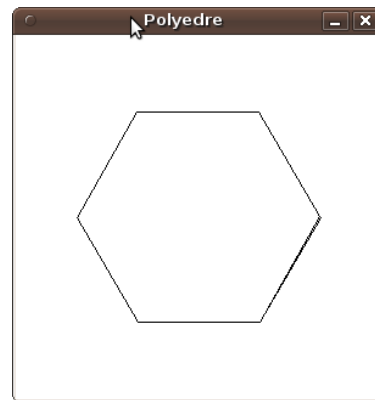
```
int main() {
    CINI_open_window(300, 300, "Polyedre");
    CINI_fill_window("white");

    polyedre (centre,      /* coordonnées du centre du polyèdre */
              6,           /* nombre de faces du polyèdre */
              100);        /* rayon du cercle contenant les sommets */

    CINI_loop();
    return 0;
}
```



a) construction du polyèdre



b) copie d'écran

FIGURE 5 – Affichage d'un polyèdre

1. Écrivez une fonction `polyedre` qui prend en argument les coordonnées du centre du polyèdre, le nombre de ses faces (6 pour un hexagone, etc.), la distance entre les sommets du polyèdre et son centre, et qui affiche le polyèdre dans une fenêtre graphique.
2. Écrivez une fonction `polyedre_Sierpinski`, prenant en paramètres le point correspondant au centre C d'un polyèdre, son nombre de faces, son rayon, et qui, pour chaque face AB du polyèdre, affiche le triangle de Sierpinski de sommets ABC (cf. la figure 6).
3. Écrivez une fonction `cone_Sierpinski`, prenant en paramètres le point correspondant au centre C d'un polyèdre, son nombre de faces, son rayon, les coordonnées d'un point D à l'intérieur du polyèdre, et qui, pour chaque face AB du polyèdre, affiche le triangle de Sierpinski de sommets ABD (cf. la figure 7).

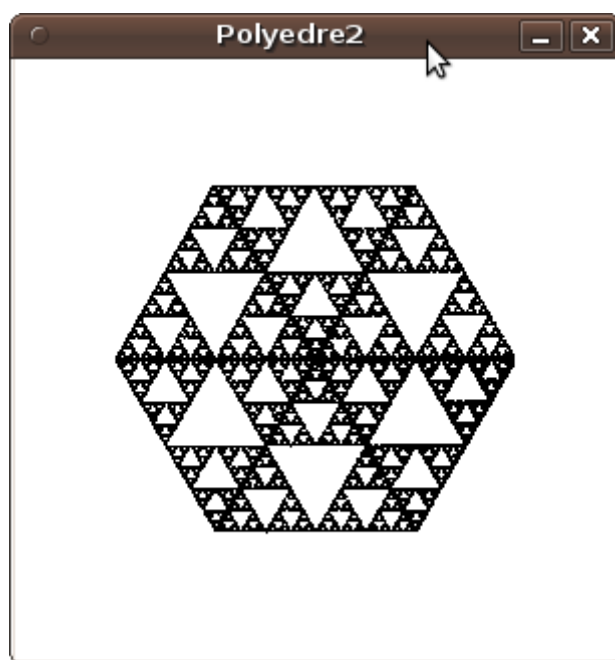


FIGURE 6 – Affichage d'un polyèdre de Sierpinski

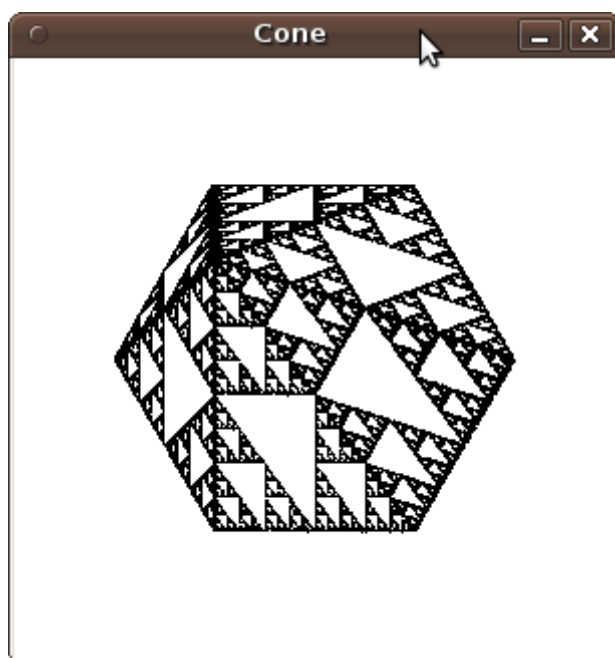


FIGURE 7 – Affichage d'un cône de Sierpinski

Savoir-faire acquis

- ★ Écrire une fonction à partir de sa spécification
- ★ Utiliser une même fonction pour obtenir des résultats différents
- ★ Écrire une fonction `main` pour tester ses fonctions
- ★ Définir et utiliser des structures

Applications classiques à connaître

- ★ Triangle de Sierpinski
- ★ Polyèdres de Sierpinski
- ★ Racines d'un polynôme



TD semaine 7 Fonctions et tableaux

Version du 18 décembre 2014

Objectif(s)

- ★ Passage de tableaux en paramètres de fonctions
- ★ Manipulation de tableaux dans des fonctions

Exercice(s)

Exercice 1 (*base*) – Passage de tableau en paramètre

1. Qu'affiche le programme suivant ? Justifiez.

```
#include <stdio.h>
#define N 5

/* Cette fonction fait certainement quelquechose... */
void transformation(int tab[], int taille) {
    int i;

    for (i = 0; i < taille; i++) {
        if (tab[i]%2 == 0) {
            tab[i] = tab[i]/2;
        }
    }
}

/* Le programme principal a pour but de montrer ce que fait la fonction. */
int main() {
    int tabf[N] = {1, 2, 6, 3, 7};
    int i;

    /* On affiche le tableau avant */
    for (i = 0; i < N; i++) {
        printf("%d ", tabf[i]);
    }
    printf("\n");

    /* On appelle la fonction */
    transformation(tabf, N);

    /* On affiche le tableau apres */
    for (i = 0; i < N; i++) {
        printf("%d ", tabf[i]);
    }
}
```

```
printf("\n");  
return 0;  
}
```

Exercice 2 (base) – Quelques fonctions sur les tableaux

Dans les questions qui suivent, on supposera que les tableaux sont complètement remplis et que leur taille est définie dans la fonction `main`.

1. Écrivez une fonction `tabAbs` qui étant donné un tableau d'entiers, remplace tous les nombres négatifs par leur valeur absolue.
2. Écrivez un programme complet permettant de tester la fonction `tabAbs`.
3. Écrivez une fonction `nbInf` qui compte et retourne le nombre de valeurs strictement inférieures à un flottant x dans un tableau de flottants.
4. Écrivez une fonction qui calcule et retourne la moyenne des valeurs supérieures à x dans un tableau d'entiers.
5. Écrivez une fonction `conjonction` qui calcule la conjonction (i.e., ET logique) des valeurs d'un tableau contenant des valeurs booléennes codées par 0 et 1.

Exercice 3 (entraînement) – Insertion dans un tableau trié

L'objectif 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 et à chaque nouvelle valeur, nous allons rechercher où l'insérer dans le tableau puis nous allons décaler les éléments suivants pour faire de la place.

Il n'est plus possible d'ajouter de nouvel élément dès que `nbEl=taille`.

1. Écrivez une fonction `indiceInsert` qui, étant donné un tableau de flottants `tab` de taille `taille` contenant `nbEl` éléments triés par ordre croissant et un flottant f , retourne l'indice auquel f doit être inséré pour que le tableau reste trié.
Si le tableau est plein ou si l'élément est déjà présent, la fonction retournera la valeur -1.
2. Écrivez une fonction `insertElt` qui réalise l'insertion de f dans `tab` si l'élément n'est pas déjà présent et si le tableau n'est pas plein. Cette fonction retournera 1 si l'insertion a été réalisée, 0 sinon.
3. Écrivez un programme complet qui permet de tester les fonctions précédentes.
4. Quel est l'inconvénient de retourner les valeurs 0 ou 1 dans la fonction `insertElt` ?

Exercice 4 (entraînement) – Normalisation d'un tableau

1. Écrivez une fonction qui calcule le maximum d'un tableau de flottants `tab` contenant n éléments.
2. Écrivez une fonction qui normalise un tableau de flottants, c'est-à-dire qui ramène toutes les valeurs sur l'intervalle $[0, 1]$ (0 pour la valeur minimum, 1 pour la valeur maximum).

Une valeur X de l'intervalle de départ sera ainsi ramenée à la nouvelle valeur :

$$\text{nouvelle valeur} = \frac{(X - \text{valeur minimum})}{(\text{valeur maximum} - \text{valeur minimum})}$$

Exercice 5 (entraînement) – Tables de multiplication

1. Écrivez une fonction qui stocke dans un tableau à une dimension la table des multiplications de 1 à 12.

Idée : dans une table de multiplication, l'élément en i^{e} ligne et j^{e} colonne vaut $i * j$. Si les lignes et les colonnes sont numérotées à partir de 0 (comme c'est le cas pour les tableaux en C), cet élément vaut $(i+1) * (j+1)$. Pour connaître sa position dans un tableau à une dimension, il suffit de compter combien d'éléments le précèdent. Il y a i lignes de 12 éléments plus les $j-1$ éléments qui le précèdent sur sa ligne, c'est donc le $(i*12+j)$ ^{ème} élément.

2. Écrivez un programme qui stocke dans un tableau à une dimension la table des multiplications de 1 à 12 puis affiche cette table en appelant une fonction `affiche_tableau` supposée connue et de prototype :

```
void affiche_tableau(int tab[], int nb_elt_total, int nb_elt_par_ligne);
```

où `nb_elt_par_ligne` représente le nombre d'éléments sur chaque ligne de la table et `nb_elt_total` est le nombre total de cases de la table.

Savoir-faire acquis

- ★ Utiliser un tableau comme paramètre d'une fonction
- ★ Écrire des fonctions qui utilisent des tableaux
- ★ Déterminer le type de retour d'une fonction utilisant un tableau

Applications classiques à connaître

- ★ Écrire des fonctions pour des opérations classiques sur les tableaux (moyenne, nombre d'occurrences, etc.)
- ★ Définir un tableau contenant les tables de multiplications de 1 à 12
- ★ Normaliser les valeurs d'un tableau



TME Semaine 7 Les fonctions et les tableaux

Version du 18 décembre 2014

Objectif(s)

- ★ Décomposition d'un problème en fonctions
- ★ Manipulation de tableaux à une dimension dans des fonctions

Exercice(s)

Exercice 1 (*base*) – Gestion de comptes bancaires

On souhaite écrire un programme permettant de gérer l'ensemble des comptes d'une banque. Pour ce faire, on stocke le montant disponible sur chaque compte dans un tableau de flottants `comptes` de taille `MAXCOMPTES`. La valeur de `MAXCOMPTES` correspond au nombre maximum de comptes pouvant être gérés par la banque et sera définie par une directive `#define`.

Le programme que vous allez écrire doit permettre de réaliser des opérations de base sur les comptes bancaires :

- Consultation, dépôt, retrait... pour le client ;
- Moyenne, minimum, rechercher les comptes à découvert, rémunérer les comptes... pour le banquier.

Pour chacune des opérations possibles, nous allons définir une fonction réalisant cette opération.

Récupérez le fichier `comptes.c` dans le répertoire de l'UE 1I002. Ce fichier contient une fonction `main` minimale dans laquelle sont définis le tableau `comptes` ainsi que le tableau `autoris` qui sera utilisé par la suite.

La case d'indice i du tableau `comptes` contient le montant disponible sur le compte i . La variable `nbCl` qui est locale à la fonction `main` a pour valeur le nombre de comptes déjà créés et stockés dans le tableau `comptes`. La case d'indice `nbCl-1` du tableau `comptes` contient donc le montant du dernier compte géré par la banque.

On s'intéresse tout d'abord aux opérations de gestion des comptes réalisées par la banque.

1. Écrivez et testez une fonction `moyenne` qui retourne la moyenne des montants des `nbCl` comptes gérés par la banque.
2. Écrivez et testez une fonction `minPos` qui retourne l'indice du compte ayant le montant minimum sans être à découvert (minimum des valeurs positives). La fonction renvoie `-1` si tous les comptes sont à découvert. On pensera à vérifier que le premier compte n'est pas à découvert.
3. Écrivez et testez une fonction `creerCompte` qui permet d'ajouter un nouveau compte avec un montant initial m . Cette fonction aura parmi ses paramètres le nombre de comptes déjà créés et retournera le nouveau nombre de comptes gérés par la banque ($1 +$ nombre de comptes déjà créés avant l'appel de la fonction). Lors de l'appel à cette fonction, la variable `nbCl` de la fonction `main` devra alors être mise à jour.
S'il n'est pas possible de créer un nouveau compte, la fonction renvoie `-1`.
4. On souhaite que chaque compte qui n'est pas à découvert puisse être rémunéré à 5%.
Écrivez et testez une fonction `remunere` qui modifie le montant de chaque compte suivant cette rémunération.

On s'intéresse à présent aux opérations effectuées par un client sur son compte. On spécifiera, pour chaque opération, le numéro de compte concerné.

5. Écrivez une fonction `consulte` qui retourne le solde du compte i .
6. Écrivez une fonction `depotOuRetrait` qui permet de réaliser un dépôt ou un retrait d'un montant m sur le compte i . Pour savoir s'il s'agit d'un dépôt ou d'un retrait, on passera, en paramètre de la fonction, un entier valant 0 ou 1 (la valeur 1 sera utilisée s'il s'agit d'un dépôt).

On souhaite à présent gérer les autorisations de découvert de chaque client. On utilise pour ce faire un tableau d'entiers `autoris` de taille `MAXCOMPTE`. Les entiers de ce tableaux codent des booléens. La case d'indice i de ce tableau contient la valeur 1 ou 0 indiquant si le compte i a ou non une autorisation de découvert.

7. En vous aidant de l'exercice 7 du TD 2, écrivez une fonction `depotOuRetrait2` qui permet de réaliser un dépôt ou un retrait d'un montant m sur le compte i .

La fonction renvoie la valeur 1 lorsque l'opération a pu être exécutée : il s'agit soit d'un dépôt, soit d'un retrait inférieur au montant disponible sur le compte, soit d'un retrait avec une autorisation de découvert.

Si l'opération n'a pas pu être effectuée, la fonction renvoie la valeur 0.

8. Dans le répertoire `/Infos/lmd/2014/licence/ue/1I002-2015fev/Semaine9`, récupérez le fichier `comptesQuest9.c`. Vous y trouverez la définition d'une nouvelle fonction `main`.

Remplacez votre actuelle fonction `main` par cette nouvelle version. Cette nouvelle fonction `main` définit les instructions nécessaires à la mise en place d'un menu permettant à un utilisateur d'effectuer les opérations qu'il souhaite sur les comptes.

L'instruction **switch** permet de gérer une succession de tests sur la valeur d'une variable (ici la variable `choix`). Chaque **case** correspond à une valeur possible de la variable. Lorsqu'une correspondance est trouvée (par exemple **case** 3 lorsque `choix` vaut 3), l'ensemble des instructions suivant le **case** est exécuté jusqu'à rencontrer une instruction **break** ou la fin du bloc défini par le **switch**.

Complétez le code de la nouvelle fonction `main` par les appels aux fonctions précédentes. Les parties à compléter sont signalées par : `/*A COMPLETER */`. Vous devez obtenir un programme permettant à un utilisateur d'effectuer les différentes opérations que vous avez mises en place dans les questions précédentes.

Exercice 2 (approfondissement) – Histogramme

Le but de cet exercice est d'écrire et de tester un programme qui permet de saisir des notes et qui affiche leur répartition sous la forme d'un histogramme. La figure 1 donne un exemple d'histogramme qui représente l'ensemble des notes (0, 3, 6, 3, 4, 3, 6, 5, 6, 5, 4, 3, 5, 1, 1, 0, 5, 5, 1, 6). Un histogramme est un diagramme en bâtons dans lequel on représente en abscisse les notes (de 0 à la note maximale sur l'ensemble traité), et en ordonnée la fréquence d'apparition d'une note dans cet ensemble. Sur notre exemple, la note 0 apparaît deux fois, il y a donc un bâton de hauteur 2 pour la note 0. Ensuite, la note 1 apparaît 3 fois, il y a donc un bâton de hauteur 3 pour cette note, etc.



FIGURE 1 – Exemple d'histogramme

1. Écrivez et testez une fonction `init_tab` qui permet à un utilisateur d'initialiser un tableau de notes (entières). L'utilisateur termine sa saisie, soit parce que le tableau est plein, soit en saisissant une note négative. La fonction `init_tab` prend en arguments un tableau et sa taille, et retourne le nombre de notes saisies.
2. Écrivez et testez une fonction `compte` qui retourne le nombre d'occurrences d'une valeur v dans un tableau de notes.

3. Écrivez et testez une fonction `max_note` qui retourne le maximum d'un tableau de notes non vide. Si le tableau est vide la fonction retourne -1 (ceci ne pose pas de problème car les seules valeurs significatives du tableau sont positives ou nulles).
4. Écrivez et testez une fonction `histogramme` qui, à partir d'un tableau de notes et de sa taille, calcule la répartition des notes. Le prototype de la fonction est :

```
void histogramme (int tab[], int h[], int taille);
```

où `tab` est le tableau de notes, `h` contient la répartition des notes, autrement dit `h[i]` correspond au nombre d'occurrences de la note `i`, et `taille` est la taille maximale du tableau de notes.

5. Écrivez et testez une fonction `dessiner_baton` qui, à partir d'un point (tableau contenant deux coordonnées), d'une largeur et d'une hauteur ainsi que d'une couleur définie par son nom, dessine une barre de la hauteur, largeur dans la couleur choisie, et de coin inférieur gauche le point.
6. Écrivez et testez une fonction `dessiner_histogramme` qui, à partir d'un histogramme de taille connue, et d'un point de départ (coordonnées stockées dans un tableau), d'une largeur et d'une couleur définie par une chaîne de caractères en donne une représentation sous forme de diagramme en bâtons dans la couleur choisie. Chaque bâton ayant la même largeur et une hauteur correspondant à la largeur fois le nombre d'occurrences de la note. Le premier bâton sera dessiné tel que son coin inférieur gauche soit au niveau du point de départ.
7. On souhaite dessiner cet histogramme dans une fenêtre, dont nous devons déterminer la taille (largeur et hauteur), de manière à ce que cette fenêtre soit circonscrite à l'histogramme. De quelles données du problème dépendent ces deux dimensions ? Écrivez et testez la fonction `largeur_fenetre` qui, étant donnés un histogramme de taille connue et la largeur fixée pour les bâtons le représentant, renvoie la largeur nécessaire pour la fenêtre d'affichage. Écrivez et testez la fonction `hauteur_fenetre` qui, étant donnés un histogramme de taille connue et la hauteur fixée pour un nombre d'occurrences de 1, renvoie la hauteur nécessaire pour la fenêtre d'affichage.
8. Écrivez et testez le programme complet qui initialise un tableau de notes et trace son histogramme.

Exercice 3 (*approfondissement*) – Gestion des réservations d'une salle de spectacle

On souhaite écrire un programme permettant de gérer les réservations d'une salle de spectacle de N places numérotées de 0 à $N - 1$. Les réservations sont représentées par un tableau d'entiers contenant les valeurs 0 et 1. Ainsi, la case i de ce tableau `resa` vaut 1 si la place i est libre et 0 sinon.

1. Écrivez une fonction `initSalle` qui étant donné un tableau `resa` représentant les réservations d'une salle de `taille` places, initialise toutes les places comme étant libres.
2. Écrivez une fonction `verif_n_places` qui vérifie s'il reste au moins n places libres dans le tableau de réservations d'une salle de nombre de places donné. La fonction renverra la valeur 1 s'il reste au moins n places, 0 sinon.
3. Des personnes peuvent vouloir réserver nb places côte à côte. Écrivez une fonction `test_dispo` qui cherche si un bloc de nb places disponibles existe encore dans la salle. S'il n'y a plus nb places contigües disponibles, la fonction retourne -1. Sinon, elle renvoie le numéro de la première place du bloc de nb places libres.
4. Soit un groupe de nb personnes souhaitant être assises côte à côte. Dans le cas, où il n'existe plus nb places côte à côte disponibles, on souhaite permettre au maximum de ces nb personnes d'être assises ensemble. Écrivez une fonction `max_libres` qui trouve le nombre maximum de places contigües disponibles pour ces nb personnes (le plus grand bloc de places libres inférieur ou égal à nb). La fonction renverra le numéro de la première place du plus grand bloc trouvé. On supposera qu'il reste au moins une place libre dans la salle.
5. Écrivez une fonction `reserve` qui réalise la réservation de nb places dans la salle et retourne une valeur indiquant si la réservation a pu être faite. S'il existe nb places consécutives de libres, on réserve ces places. Sinon, on regroupe le plus grand nombre de personnes ensemble (on réserve le plus grand ensemble n de places consécutives inférieur à nb et on essaye ensuite de placer les $nb - n$ personnes restantes ensemble).
6. Écrivez une fonction `main` permettant la saisie par un utilisateur des réservations. Le programme attend la saisie d'un nombre de places et réalise la réservation si possible tant qu'il reste au moins une place libre dans la salle.

Exercice 4 (*entraînement*) – Loto

Dans cet exercice, nous voulons simuler un jeu de loto. Deux tableaux représentent l'un le tirage du loto, l'autre la grille du joueur. Le premier tableau sera rempli aléatoirement de 6 entiers entre 1 et 49 (comme au loto), le second sera rempli par l'utilisateur. Dans chacun des tableaux, toutes les cases doivent contenir des valeurs différentes.

Une fois les deux tableaux remplis, il faut comparer leur contenu pour déterminer si le joueur a gagné.

Pour pouvoir comparer facilement le contenu des deux tableaux, nous allons trier leur contenu au fur et à mesure du remplissage. L'algorithme d'ajout d'une valeur dans le tableau est le suivant :

Soit n le nombre de cases du tableau `tab` déjà initialisées et `val` la valeur à ajouter dans le tableau. Les valeurs contenues dans les n premières cases du tableau sont triées. Nous supposons que ces valeurs sont rangées par ordre croissant.

Pour insérer la valeur `val`, l'algorithme parcourt le tableau en s'arrêtant lorsque les n premières cases ont été visitées ou lorsqu'il tombe sur une case dont le contenu est supérieur ou égal à `val`.

Soit i l'indice de la case sur laquelle le parcours s'arrête. Si `tab[i]` contient `val`, l'algorithme n'insère pas la valeur dans le tableau puisque toutes les cases doivent avoir des contenus différents.

Sinon ($i == n$ ou `tab[i] > val`), l'algorithme décale d'une position vers la droite le contenu de toutes les cases dont les indices sont compris entre $n-1$ et i (il peut n'y en avoir aucune). Puis la valeur `val` est copiée dans la case i . De cette manière, on préserve la propriété triée du tableau.

1. Écrivez une fonction `ajouter_tab` qui ajoute la valeur `val` "à sa place" dans le tableau `tab` dont n cases sont déjà initialisées. La fonction renvoie une valeur codant un booléen et indiquant si le tableau a été modifié.
2. Écrivez une fonction `init` qui prend en argument un tableau d'entiers et sa taille, et qui initialise aléatoirement ce tableau avec des nombres entiers compris entre 1 et 49 (inclus). Testez cette fonction pour remplir un tableau de 6 entiers (49 étant défini par une directive `#define`) avec des valeurs toutes différentes et en le triant par ordre croissant.
3. Écrivez une fonction `lire_tab` qui prend en argument un tableau d'entiers et sa taille, et qui demande à l'utilisateur de remplir ce tableau avec des valeurs entre 1 et 49. Cette fonction doit assurer que toutes les cases du tableau contiennent des valeurs différentes, triées par ordre croissant.
Testez cette fonction pour remplir un deuxième tableau de 6 entiers.
4. Écrivez et testez une fonction `compare_tab` qui prend en paramètre deux tableaux d'entiers de longueur n et renvoie 1 si les deux tableaux sont identiques, 0 sinon.

Dans la fonction `main`, vous devrez afficher les deux tableaux et le résultat de la comparaison.

Savoir-faire acquis

- ★ Utiliser un tableau comme paramètre d'une fonction
- ★ Écrire des fonctions qui résolvent des sous-problèmes
- ★ Écrire le programme qui utilise des fonctions pour résoudre un problème

Applications classiques à connaître

- ★ Gérer des comptes bancaires
- ★ Dessiner un histogramme, un diagramme en bâtons
- ★ Allouer des blocs de cases dans un tableau
- ★ Remplir un tableau avec des valeurs aléatoires, sans répétition



TD semaine 8 Récursion

Version du 18 décembre 2014

Objectif(s)

- ★ Savoir exécuter une fonction récursive avec des instructions avant et/ou après l'appel récursif.
- ★ Importance de la convergence vers la condition d'arrêt de la récursivité.
- ★ Savoir écrire une fonction récursive simple.

Remarque:

Avant d'écrire le code de chaque fonction récursive, vous devrez préciser sa signature, ainsi que le cas d'arrêt et le cas général.

Exercice(s)

Exercice 1 (*base*) – Fonction récursive et affichage

1. Soit la fonction `affiche` suivante :

```
1 void affiche(int n) {  
2     if (n > 0) {  
3         printf("%d ", n);  
4         affiche(n-1);  
5     }  
6     else {  
7         printf("%d ", n);  
8     }  
9 }
```

Qu'affiche l'appel `affiche(6)` ?

2. Soit la fonction `affiche` suivante :

```
1 void affiche(int n) {  
2     if (n > 0) {  
3         affiche(n-1);  
4         printf("%d ", n);  
5     }  
6     else {  
7         printf("%d ", n);  
8     }  
9 }
```

Qu'affiche l'appel `affiche(6)` ?

3. On souhaite afficher les nombres de deux en deux. Pour cela nous utilisons la fonction `affiche` suivante :

```

1 void affiche(int n) {
2     if (n == 0) {
3         printf("%d ", n);
4     }
5     else {
6         printf("%d ", n);
7         affiche(n-2);
8     }
9 }

```

Qu'affichent les appels `affiche(8)` et `affiche(7)` ?

Exercice 2 (base) – Fonction récursive et valeur de retour

1. Le programme ci-après définit une fonction `somme` qui calcule la somme de deux entiers en utilisant uniquement l'opérateur "successeur". Cette définition récursive du groupe des entiers relatifs est au programme de L1 math. Décrivez l'exécution du programme suivant en détaillant soigneusement les valeurs des différentes variables mises en jeu lors des appels à la fonction `somme`.

```

1 #include <stdio.h>
2
3 /* La fonction somme renvoie la valeur de a+b, calculée
4  * uniquement à l'aide de l'opérateur "successeur". */
5 int somme(int a, int b) {
6     int res;
7
8     /* a+0 vaut a */
9     if (b == 0) {
10        return a;
11    }
12    /* si b est différent de 0, alors (a+b) vaut succ(a+prec(b)). */
13    else {
14        res=somme(a,b-1);
15        return res + 1;
16    }
17 }
18
19 /* Le programme principal teste deux appels à "somme" */
20 int main() {
21     printf("2 + 4 =");
22     printf("%d\n", somme(2,4));
23     printf("2 - 3 =");
24     printf("%d\n", somme(2,-3));
25     return 0;
26 }

```

2. Modifiez la fonction `somme` pour que le problème identifié dans la question précédente soit résolu.
3. Décrivez l'exécution de l'appel `somme(2, -3)` qui vient d'être écrite, en détaillant soigneusement les valeurs des différentes variables mises en jeu.
4. Ré-écrivez la fonction `somme` en supprimant la variable locale `res`.

Exercice 3 (base) – Récursivité sur les tableaux et chaînes de caractères

1. Écrivez une fonction récursive `chercheLettreTab` qui étant donnés un tableau de caractères, sa taille et un caractère, renvoie 1 si le caractère appartient au tableau, 0 sinon. Écrivez un programme pour tester cette fonction
2. Écrivez maintenant une fonction récursive `chercheLettreChaine` qui étant donnés une chaîne de caractères et un caractère, renvoie 1 si le caractère appartient au tableau, 0 sinon. Attention, ici la taille de la chaîne n'est pas connue à l'avance. Écrivez un programme pour tester cette fonction.

Exercice 4 (renforcement) – Fonction puissance

Nous souhaitons écrire une fonction récursive `puissance(float x, int n)` qui retourne la valeur x^n en utilisant la relation :

$$x^n = \begin{cases} x^{\frac{n}{2}} . x^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x . x^{\frac{n-1}{2}} . x^{\frac{n-1}{2}} & \text{si } n \text{ est impair} \end{cases}$$

1. Nous supposons dans un premier temps que n est un entier positif. Donnez le cas de base et le cas général. Justifiez la convergence de n vers la valeur du cas d'arrêt.
2. Donnez le code de la fonction `puissance` sans utiliser de variable locale.
3. Modifiez la fonction `puissance` pour réduire le nombre d'appels récursifs.

Modifiez la fonction `puissance` pour qu'elle donne un résultat correct dans le cas d'une puissance négative (rappel : $x^{-n} = 1/(x^n)$).

Exercice 5 (renforcement) – Codage de nombres

Dans cet exercice on va transformer un nombre codé en binaire en décimal. On rappelle ici qu'à partir d'un codage binaire, on peut obtenir un codage décimal et vice-versa de telle sorte que :

$\dots xyz = z \times 2^0 + y \times 2^1 + x \times 2^2 \times \dots$, où x, y et z (et tous les autres chiffres qui les précèdent) sont des 0 ou des 1. Ainsi : $01101 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 = 13$

1. Écrivez une fonction récursive `Bin2Dec` qui transforme un nombre codé par une suite de 0 et 1 en nombre décimal.
2. Écrivez une fonction récursive `Dec2Bin` qui transforme un nombre codé en décimal par une suite de 0 et 1.

Savoir-faire acquis

★ Écrivez et testez une fonction récursive, avec ou sans valeur de retour.

Applications classiques à connaître

- ★ Fonction puissance récursive
- ★ Passage du codage binaire ou décimal (et réciproquement) en récursif



TME Semaine 8 TME solitaire

Version du 18 décembre 2014

Objectif(s)

★ Contrôle sur machine

Introduction

Les deux séances de TME de cette semaine sont consacrées à un deuxième TME solitaire. Par demi-groupes de 15 étudiants, vous allez tous composer pendant **1h30** seul devant votre machine.

Le but de ce TME solitaire est d'évaluer votre capacité à écrire des programmes C faisant appel aux boucles, alternatives, tableaux, structures et fonctions. Le sujet se découpe en 6 exercices : 2 exercices étiquetés "facile", 3 exercices étiquetés "moyens" et 1 exercice étiqueté "difficile".

Pour tous ces exercices, vous perdrez un tiers des points si votre programme ne compile pas (syntaxe non correcte), même s'il est correct à 95%. **Vous devrez systématiquement vous assurer que ce que vous avez écrit compile sans erreur.**

Le reste des points est attribué en fonction :

- de l'adéquation entre votre code et la résolution du problème de l'énoncé (votre code résout le problème posé) ; attention à considérer les différents cas de tests possibles lors de l'écriture de votre code
- de la qualité du code : utilisation des instructions appropriées, clarté du code, indentation, présence de commentaires, etc..

Pour commencer

Créez un répertoire `TMEsolo2` dans votre répertoire personnel. Comme lors des TME habituels, vous devrez récupérer des fichiers `.c` dans le répertoire que vous indiquera votre enseignant puis les enregistrer dans le répertoire `TMEsolo2` pour pouvoir travailler dessus. Vous avez en tout 6 fichiers à récupérer : `ex1.c`, `ex2.c`, `ex3.c`, `ex4.c`, `ex5.c` et `ex6.c`.

Travail à réaliser

À la fin de la séance, vous ferez une archive du répertoire `TMEsolo2` que vous enverrez via la page de soumission habituelle.

Pour chaque exercice, complétez le fichier correspondant pour que le programme soit conforme à ce qui est indiqué en commentaire dans le fichier `.c`.

Certaines parties du programme ne doivent pas être modifiées. En particulier, vous ne devez pas supprimer les constantes définies à l'aide de la primitive `#define`.



TD semaine 9 Les pointeurs

Version du 18 décembre 2014

Objectif(s)

- ★ Introduction des pointeurs
- ★ Passage de paramètres par adresse

Exercice(s)

Exercice 1 (*base*) – Qu'est-ce qu'un pointeur ?

Soit le programme suivant :

```
1  #include <stdio.h>
2
3  void ma_fonction(int *param1, int param2) {
4      int var_loc = 3;
5
6      *param1 = var_loc * param2;
7      param2 = var_loc + 1;
8  }
9
10 int main() {
11     int v1, v2;
12
13     v1 = 10;
14     v2 = 3;
15     ma_fonction(&v2, v1);
16     return 0;
17 }
```

1. Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme précédent. Nous vous rappelons que les paramètres sont implantés sous la forme de variables locales à la fonction.

Exercice 2 (*base*) – Passage de paramètres par valeur et par adresse

On considère une fonction f ayant deux paramètres $i1$ et $i2$ de type entier et renvoyant un résultat de type entier. La fonction calcule la somme des deux paramètres, l'enregistre dans une variable locale $i3$ (de type entier), puis retourne la valeur de cette variable.

1. Écrivez la fonction f .
2. Les instructions suivantes d'appel de f provoquent-elles une erreur de typage ? Pourquoi ?

1. `int a; a = f(1,2);`
2. `int a,b,c; a = f(b,c);`
3. `int a,b,c; a = f(&b,c);`
3. Quel sens cela a-t-il d'ajouter l'instruction `i2 = i3` au code de la fonction `f` (juste avant l'instruction `return`) ? Expliquez votre réponse.
4. Dans le programme suivant, la fonction `f` est celle obtenue après la modification de la question précédente. Donnez :
 - l'instruction d'appel à la fonction `f` dans la fonction `main`,
 - la valeur des variables `i1`, `i2` et `res` avant l'appel à `f`, au début de `f`, à la fin de `f` et après l'appel à `f`.

Attention, pour les variables de nom identique dans les fonctions `f` et `main`, vous devrez préciser de quelle variable vous donnez la valeur.

```
#include <stdio.h>

/* fonction f*/
.... f(.....){
    .....
}

int main() {
    int i1 = 1, i2 = 2, res;

    /* res prend la valeur retournée par la fonction f
       appelée avec i1 et i2 comme paramètres */
    .....

    return 0;
}
```

5. Maintenant, nous voudrions que la modification du paramètre `i2` soit répercutée également à l'extérieur de la fonction. Modifiez le prototype et le corps de la fonction `f` en conséquence. Donnez le contenu des variables `i1` et `i2` et `res` avant l'appel à la fonction `f`, au début de `f`, à la fin de `f` et après l'appel à la fonction `f`.
6. Avec cette nouvelle version de `f`, quelles instructions d'appel provoquent des erreurs de typage ?
 1. `int a; a = f(1,2);`
 2. `int a,b,c; a = f(b,c);`
 3. `int a,b,c; a = f(b,&c);`
7. Donnez les états successifs de la mémoire (pile d'exécution) lorsque l'exécution du programme suivant atteint les lignes 17, 6, 9, 11 et 19.

```
1  #include <stdio.h>
2
3  int f(int i1, int *p_i2) {
4      int i3;
5      int i4 = *p_i2;
6
7      i3 = i1 + i4;
8      i4 = i3;
9
10     *p_i2 = i4;
11
12     return i3;
13 }
14
15 int main() {
```

```

16     int i1=1, i2=2, res=0;
17
18     res=f(i1, &i2);
19     return 0;
20 }

```

Exercice 3 (*entraînement*) – Calcul des Racines d'un polynôme du 2nd degré

Soit le programme suivant qui doit permettre d'afficher le nombre de racines d'un polynôme du second degré.

```

#include <stdio.h>
#include <math.h>

/* fonction racine */
..... racine (.....){
    .....
}

int main(){
    int a,b,c;
    int nb_racines;

    printf("Veuillez saisir a :");
    scanf("%d", &a);

    printf("Veuillez saisir b :");
    scanf("%d", &b);

    printf("Veuillez saisir c :");
    scanf("%d", &c) ;

    /* appel a la fonction racine pour determiner la valeur du nombre de racines
       et leur valeur */
    .....

    if (nb_racines==2){
        /* affichage des deux racines */
        .....
    }
    if (nb_racines==1){
        /* affichage de la racine double */
        .....
    }
    if (nb_racines==0){
        printf("Le polynome n'a pas de racine reelle.\n");
    }

    return 0;
}

```

1. Donnez le prototype de la fonction `racine` qui doit retourner le nombre de racines du polynôme $ax^2 + bx + c$ ainsi que leur valeur. Ajoutez dans la fonction `main` qui vous est donnée, l'appel à la fonction `racine`.
2. Écrivez la fonction `racine` et complétez l'affichage dans la fonction `main`.

Savoir-faire acquis

- ★ Savoir ce qu'est un pointeur - une adresse
- ★ Exécuter un programme dont une fonction a un paramètre pointeur
- ★ Écrire et appeler une fonction ayant un paramètre pointeur

Applications classiques à connaître

- ★ Calculer les racines d'un polynôme du second degré



TME Semaine 9

Compilation en ligne de commandes

Récursion

Version du 18 décembre 2014

Objectif(s)

- ★ Compiler et exécuter un programme en ligne de commandes
- ★ Décomposition d'un problème en fonctions
- ★ Manipulation de tableaux à une dimension dans des fonctions
- ★ Récursivité sur les nombres
- ★ Récursivité sur les tableaux
- ★ Appel et retour de fonctions

Exercice(s)

Exercice 1 (*base*) – Compilation et exécution en ligne de commandes

Jusqu'à présent, vous avez utilisé les boutons « compiler » et « exécuter » de Geany afin de compiler et exécuter vos programmes. Ces raccourcis sont des plugins propres à l'UE 1I002 et ne sont pas présents dans la version de base de Geany ou dans d'autres éditeurs de texte (bien que certains environnements de développement -IDE- proposent des équivalents).

Nous allons apprendre, dans cet exercice, comment compiler et exécuter un programme en langage C sans utiliser ces raccourcis de Geany. Vous continuerez à éditer votre code en utilisant Geany. En revanche, pour compiler et exécuter votre programme, vous utiliserez des commandes saisies dans un terminal.

La fenêtre du terminal peut être ouverte depuis le menu *Applications/Accessoires/Terminal*.

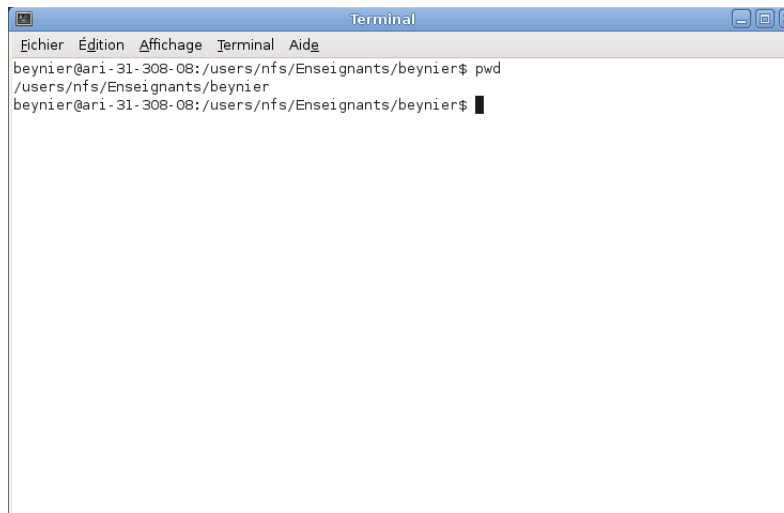
Vous pouvez aussi ajouter un raccourci pour l'ouverture d'un terminal depuis la barre des tâches :

- Soit en cliquant sur la barre des tâches avec le bouton droit de la souris et en sélectionnant *Ajouter au tableau de bord* puis *Lanceur d'application* et en suivant ensuite les menus comme dans le menu *Applications* ;
- Soit en cliquant droit sur le menu *Terminal* (dans *Applications/Accessoires/Terminal*) et en sélectionnant *Ajouter ce lanceur au tableau de bord*.

La fenêtre du terminal est identifiée par le *prompt* ou *invite de commande* au début de la ligne (par exemple, `login@ari-31-308-08:/users/login$`) qui vous indique que l'interprète de commandes est prêt à exécuter les commandes Unix que vous allez lui indiquer en terminant la commande par un retour à la ligne.

Dans la fenêtre qui suit, nous avons exécuté la commande `pwd` une commande Unix très utilisée.

C'est dans cette fenêtre que nous allons exécuter les commandes Unix sur les fichiers et les répertoires ainsi que les commandes de compilation des fichiers C (écrits sous Geany).



```
Terminal
Fichier  Édition  Affichage  Terminal  Aide
beynier@ari-31-308-08:/users/nfs/Enseignants/beynier$ pwd
/users/nfs/Enseignants/beynier
beynier@ari-31-308-08:/users/nfs/Enseignants/beynier$
```

1. Exécutez la commande `ls` puis la commande `pwd`. A quoi servent ces commandes ?

Chaque utilisateur dispose d'un répertoire personnel, appelé *Home Directory*. C'est le répertoire dans lequel il est placé à la connexion et dans lequel il peut organiser ses données.

La commande `cd` permet de changer de répertoire :

- `cd chemin` permet de se déplacer dans le répertoire spécifié par *chemin*
- `cd ..` permet de remonter au répertoire supérieur,
- `cd` permet de revenir à la racine de votre *Home Directory*.

2. A l'aide de la commande `cd`, placez-vous dans le répertoire que vous avez créé pour la semaine 9 de 1I002.
3. Sous Geany, créez un programme affichant « Hello World » et enregistrez-le avec le nom `hello.c` dans le répertoire de la semaine 9 pour 1I002. Exécutez ensuite dans le terminal, la commande `ls`. Vous devez voir le fichier du programme que vous venez de créer.

La commande `gcc` permet de compiler le programme. Si la compilation ne produit pas d'erreur, la commande crée, à partir du fichier source passé en paramètre, un fichier exécutable. Si la compilation produit des erreurs, celles-ci sont affichées dans le terminal et aucun exécutable n'est créé.

4. Compilez le programme de la question précédente. 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 ! Il est possible de renommer, avec la commande `mv`, le fichier créé, 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`.

5. Supprimez le fichier `a.out` avec la commande `rm a.out`. Recompilez le programme source en créant un exécutable `hello`.

Pour exécuter un programme, on utilise le nom du fichier exécutable. Par exemple, pour exécuter le programme de la question précédente on utilise la commande `./hello`

6. Exécutez le programme de la question précédente.

En plus de l'option `-o`, nous utiliserons les options `-O`, `-Wall` et `-Werror` :

- `-O` permet d'optimiser le temps d'exécution et la taille du code,

- `-Wall` permet d’afficher tous les avertissements générés par le compilateur,
- `-Werror` transforme tous les avertissements en erreurs, le fichier exécutable ne sera donc pas généré tant qu’il restera des avertissements à corriger.

Les deux dernières options sont vivement recommandées afin de produire des programmes respectant les bonnes pratiques de programmation et afin de limiter les erreurs d’exécution.

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

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

7. Compilez à nouveau votre programme avec cette commande. Corrigez les éventuelles erreurs de votre programme.

Lorsque vous utiliserez des fonctions de la bibliothèque `math.h`, il faudra ajouter un lien vers cette bibliothèque, avec l’option de compilation `-lm`.

Lorsque vous utiliserez des fonctions de la bibliothèque `cini.h`, il faudra ajouter un lien vers cette bibliothèque avec l’option de compilation `-lcini`.

Si vous affichez graphiquement des caractères (fonctions `CINI_draw_string()`, `CINI_draw_???_table()`, etc.), il faudra aussi ajouter la localisation du fichier de polices de caractères avec l’option `-DFONTFILE`. La commande complète sera alors :

```
gcc -DFONTFILE='/usr/share/libcini/font.ttf' -Wall -Werror -O -o hello hello.c -lm -lcini
```

A présent, nous vous demandons de **toujours** compiler vos programmes en ligne de commandes.

Exercice 2 (*base*) – Manuel Unix

Sous Unix, la commande `man` (pour *manual*) permet d’accéder à une aide en ligne sur les commandes (section 1), les appels système (section 2), les fonctions de bibliothèque (section 3), etc.

La commande `man toto` parcourt les différentes sections du manuel jusqu’à en trouver une qui contient des informations sur `toto`, puis affiche les informations trouvées. Si plusieurs sections contiennent des informations sur `toto` (en cas d’homonymie entre une commande et une fonction par exemple), seules les informations de la première section dans laquelle figure `toto` sont affichées par `man toto`.

Pour sortir de la page de manuel, il suffit d’appuyer sur la touche `q`.

On peut afficher les informations sur `toto` contenues dans une section `x` donnée en utilisant la commande :

```
Prompt> man x toto
```

(Remplacer `x` par le numéro de la section). On peut afficher les informations contenues dans toutes les sections en utilisant `man -a toto`. Dans ce cas, lorsque vous appuyez sur la touche `q`, le manuel passe à la section suivante, au lieu de réafficher le *prompt* (invite de commande).

1. On veut ici savoir quel fichier d’en-tête inclure dans un programme C pour pouvoir utiliser la fonction `printf`. Quelle(s) commande(s) permettent d’obtenir cette information ?

Exercice 3 (*base*) – `n` premiers entiers

1. Écrivez une fonction récursive `som_carres_rec` et le programme principal (fonction `main`) permettant de calculer la somme des `n` premiers entiers positifs au carré.
2. Écrivez une fonction récursive `prod_rec` et le programme principal (fonction `main`) permettant de calculer le produit des `n` premiers entiers positifs.

Exercice 4 (*base*) – Nombre, somme et produit de chiffres

1. Écrivez une fonction récursive `NombreChiffres` qui étant donné un nombre, renvoie le nombre de chiffres qui le composent.
2. Écrivez une fonction récursive `SommeDesChiffres` qui étant donné un nombre, renvoie la somme des chiffres qui le composent.
3. Écrivez une fonction récursive `ProduitDesChiffres` qui étant donné un nombre, renvoie le produit des chiffres qui le composent.
4. Écrivez une fonction récursive `niemeChiffre` qui étant donnés un nombre x et un rang n , renvoie le $n^{\text{ième}}$ chiffre (en partant de la droite) de ce nombre. La fonction renverra, cette fois, -1 si le $n^{\text{ième}}$ chiffre n'existe pas.
5. Écrivez une fonction récursive `renverseNombre` inverse les chiffres d'un nombre. Par exemple, le nombre 12345 deviendra 54321. On n'utilisera pas de tableaux ! Écrire un programme test.

Exercice 5 (*base*) – Calcul de π

Dans cet exercice nous allons coder deux algorithmes qui ont été proposés pour le calcul d'une valeur approchée de π .

1. Écrivez la fonction récursive `piWallis` qui calcule une valeur approchée de π par la méthode de Wallis., avec :

$$\frac{\pi}{2} \approx \prod_{n=1}^{+\infty} \frac{4n^2}{4n^2 - 1}$$

La fonction calculera en fait le n^{me} terme de la suite ci-dessus et sera initialement appelée pour une grande valeur de n .

2. Écrivez la fonction récursive `piLeibniz` qui calcule une valeur approchée de π par la méthode de Leibniz, avec :

$$\frac{\pi}{4} \approx \sum_{n=0}^{+\infty} \frac{(-1)^n}{2n+1}$$

On utilisera la fonction `pow(x, n)` qui calcule x^n et est définie dans la bibliothèque `math.h`.

3. On veut maintenant comparer laquelle de ces deux méthodes converge le plus vite vers une valeur $\pi = 3.14157$ à 0.01 près (ces deux valeurs seront définies à l'aide de `#define`). Dans le programme principal, calculer à partir de quelle valeur de `nL` et `nW` (respectivement pour les approches de Leibniz et Wallis) on obtient une valeur approchée de π à 0.01 près. Afficher chacune de ces deux valeurs ainsi que les valeurs de π approchées obtenues. On utilisera la fonction `fabs`, définie dans la bibliothèque `math.h`, pour calculer l'écart entre π et son estimation.
4. Que se passe-t-il si on diminue la valeur de l'écart (à 0.01 dans l'exemple ci-dessus) ?

Exercice 6 (*approfondissement*) – Nombres de Fibonacci

La suite des nombres de Fibonacci est définie par la relation de récurrence suivante :

$$F_0 = 1, \quad F_1 = 1 \quad \text{et} \quad F_n = F_{n-1} + F_{n-2}, \quad n \geq 2$$

1. Donnez le code d'une fonction **récursive** `fibonacci_rec` calculant le $n^{\text{ème}}$ nombre de Fibonacci. Faites en sorte que l'ensemble des appels récursifs s'affichent à l'écran. Écrivez un programme pour la tester. Que constate-t-on concernant les appels récursifs ?

2. Nous souhaitons maintenant écrire une fonction **itérative** `fibonacci_ite` qui calcule le $n^{\text{ème}}$ nombre de Fibonacci. Pour ne pas retrouver le même problème qu'avec la fonction récursive, vous devez trouver une solution ne calculant pas plusieurs fois le même nombre. Vous devrez écrire une boucle qui lors du premier tour calculera la valeur de F_2 , celle de F_3 le tour suivant et ainsi de suite.

Votre fonction devra utiliser uniquement trois variables nommées `a`, `b` et `c` par exemple : `a` et `b` contiennent les valeurs de F_{n-1} et F_{n-2} respectivement (au début, F_0 et F_1) et `c` contiendra la valeur de l'élément F_n calculé en fonction de `a` et `b`. Ensuite, les valeurs des variables `a` et `b` seront modifiées pour préparer le calcul du nombre suivant (il faut mettre F_{n-1} dans `b` (ce sera la valeur de F_{n-2} au prochain tour), et F_n dans `a` (ce sera la valeur de F_{n-1} au prochain tour)).

Votre fonction devra donc avoir une structure similaire à ceci :

```
int fibonacci_ite(int n) {
    /* calcule itérativement le nième nombre de Fibonacci */
    int i; /* indice de boucle */
    int a, b, c;

    initialisation des variables

    boucle de calcul du nième nombre de Fibonacci
        /* calcul du nouveau nombre en fonction de a et b */
        c = ?
        /* preparation du calcul suivant */
        a = ?
        b = ?

    return c;
}
```

Exercice 7 (renforcement) – Récursivité sur les tableaux

Dans cet exercice, on considère des tableaux dont la taille est connue.

- Écrivez une fonction récursive `verifTabEgaux` qui renvoie 1 si deux tableaux de caractères `ch1` et `ch2` sont identiques, et renvoie 0 sinon. On supposera ici que la taille des tableaux est la même. Écrivez un programme pour tester cette fonction.
- Écrivez une fonction récursive `plusDeLettres1` qui détermine si un tableau de caractères comporte plus de lettres `LETTRE1` que de lettres `LETTRE2` (les deux sont définies par des `#define`), en respectant la casse, c'est-à-dire qui différencie les majuscules et les minuscules.
- Écrivez une fonction récursive `palindrome` qui renvoie 1 si un mot stocké dans un tableau de caractères (taille connue) est un palindrome (c'est-à-dire qu'il peut se lire dans les deux sens).
- Écrivez une fonction récursive `afficheTab` qui affiche un tableau de caractères dont la taille est connue.
- Écrivez une fonction récursive `miroir` qui, étant donné un tableau de caractères, calcule son miroir. Le tableau sera modifié de sorte à contenir le miroir. Écrivez un programme test qui affiche le tableau avant et après l'appel à la fonction `miroir` (on se servira pour cela de la fonction `afficheTab`).
- Écrivez une fonction récursive `produitScalaire` qui, étant donné deux tableaux d'entiers de même taille (connue), calcule le produit scalaire de ces deux vecteurs. Rappel : le produit scalaire est la somme des produits des composantes. Ainsi, si on a $v_1 = \{1, 2\}$ et $v_2 = \{3, 4\}$, alors $v_1 \cdot v_2 = 1 \times 3 + 2 \times 4 = 11$. Écrivez également un programme test.

Exercice 8 (renforcement) – Récursivité sur les chaînes de caractères

On veut cette fois gérer des chaînes de caractères dont on ne connaît par forcément à l’avance la longueur.

1. Écrivez une fonction `compteLettres` qui étant donnée une chaîne de caractères renvoie le nombre de caractères qu’elle contient. Écrivez un programme test.
2. Écrivez une fonction récursive `verifChainesEgales` qui renvoie 1 si deux chaînes de caractères `ch1` et `ch2` sont identiques, et renvoie 0 sinon. Écrivez un programme pour tester cette fonction.

Exercice 9 (base) – Minimum d’un tableau

1. Écrivez une fonction `init` qui prend en argument un tableau d’entiers et sa taille, et qui initialise aléatoirement ce tableau avec des nombres entiers compris entre 0 et 99 (inclus). La valeur maximale 99 devra être définie par une directive `#define`.
2. Écrivez une fonction **récursive** `min_tab` qui prend en paramètre un tableau d’entiers `tab` de longueur `n` ($n \geq 1$) et retourne le minimum de `tab`.

Exercice 10 (renforcement) – Recherche d’un élément dans un tableau

1. Écrivez et testez une fonction récursive `presence_el` qui prend en paramètre un élément `el`, un tableau d’entiers `tab` de longueur `n` et renvoie 1 si l’élément `el` appartient au tableau, 0 sinon.

Exercice 11 (entraînement) – Affichage de l’inverse d’un mot

Les tableaux de caractères peuvent être utilisés pour manipuler des *chaînes de caractères*. Dans cet exercice, nous allons afficher l’inverse d’une chaîne de caractères en utilisant une fonction récursive sur le tableau de caractères. Contrairement à ce que nous avons fait sur les tableaux d’entiers, nous n’utiliserons pas directement la taille du tableau comme critère d’arrêt ou non de la récursion puisque l’information qui nous indique que la fin du tableau est atteinte n’est pas la taille mais la présence du caractère ‘\0’.

1. Écrivez et testez une fonction **récursive** `affiche_inverse` qui prend comme seul paramètre un mot décrit par un tableau de caractères `tabCar` et qui affiche le mot miroir de `tabCar`.

Par exemple, si le mot est “bonjour”, le programme doit afficher “ruojnob”.

Savoir-faire acquis

- ★ Utiliser un terminal
- ★ Connaître les commandes de base d’un terminal
- ★ Compiler et exécuter un programme en ligne de commandes
- ★ Savoir identifier le(s) cas de base et le(s) cas d’arrêt d’une récursion
- ★ Savoir écrire une fonction récursive
- ★ Savoir manipuler tableaux et chaînes de caractères

Applications classiques à connaître

- ★ Savoir décomposer un nombre

- ★ Calculer les valeurs approchées de π
- ★ Savoir coder/décoder un nombre
- ★ Calculer les termes de la suite de Fibonacci
- ★ Rechercher un élément dans un tableau
- ★ Afficher l'inverse d'une chaîne de caractères



TD semaine 10 Pointeurs et tableaux

Version du 18 décembre 2014

Objectif(s)

- ★ Pointeurs et tableaux
- ★ Passage des paramètres d'une fonction par valeur et par adresse (répétition)

Exercice(s)

Exercice 1 (*base*) – Tableaux et pointeurs

Rappel : Les tableaux peuvent être modifiés par les fonctions qui les manipulent, car la variable qui représente le tableau a pour valeur l'adresse du premier élément du tableau. C'est cette adresse qui est passée en paramètre. L'objectif de cet exercice est d'assimiler cette notion à l'aide de fonctions simples.

1. Comparez les deux signatures suivantes :

```
void affiche(int tab[], int taille);  
void affiche(int *tab, int taille);
```

Sont-elles interchangeables ?

2. Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme suivant :

```
#include <stdio.h>  
  
#define DIM 3  
  
void affiche(int tab[], int taille) {  
    int i;  
  
    for(i = 0; i < taille; i++) {  
        printf("%d\t", tab[i]);  
    }  
    printf("\n");  
}  
  
int main() {  
    int mon_tab[DIM] = {-3, 2, 7};  
  
    affiche(mon_tab, DIM);  
    return 0;  
}
```

3. Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme suivant :

```

#include <stdio.h>

#define DIM 3

void mult(int tab[], int taille) {
    int i;

    for(i = 0; i < taille; i++) {
        tab[i] = tab[i] * 2;
    }

    int main() {
        int mon_tab[DIM] = {-3, 2, 7};

        mult(mon_tab, DIM);
        return 0;
    }

```

4. Écrivez une fonction `somme_tab` qui prend en argument trois tableaux d'entiers de taille identique et la taille de ces tableaux. Cette fonction fera la somme des deux premiers tableaux et la stockera dans le troisième tableau.
5. Pourquoi ne pouvons-nous pas écrire une fonction qui prend deux tableaux en paramètres et qui renvoie la somme des deux comme résultat ?

Exercice 2 (*renforcement*) – Comment se passer des variables globales

Soit le programme suivant :

```

1  #include <stdio.h>
2
3  int a = 3, b = 15, c = 2;
4
5  int f1(int v, int n) {
6      int i;
7
8      for (i = 0; i < a; i++) {
9          v=v+n;
10     }
11     return v;
12 }
13
14 void f2(int n1, int n2) {
15     if (n1 < n2) {
16         b = n1;
17         c = n2;
18     }
19     else {
20         b = n2;
21         c = n1;
22     }
23 }
24
25 int main() {
26     int v1 = 3, v2 = 5, v3 = 1;
27
28     printf("v1 = %d, v2 = %d, v3 = %d, ", v1, v2, v3);
29     printf("a = %d, b = %d, c = %d\n", a, b, c);
30 }

```

```

31     v1 = f1(v1, 2);
32     printf("v1 = %d, v2 = %d, v3 = %d, ", v1, v2, v3);
33     printf("a = %d, b = %d, c = %d\n", a, b, c);
34
35     f2(v2, v3);
36     printf("v1 = %d, v2 = %d, v3 = %d, ", v1, v2, v3);
37     printf("a = %d, b = %d, c = %d\n", a, b, c);
38
39     return 0;
40 }

```

1. Donnez les affichages effectués par ce programme.
2. Modifiez la fonction `f1` pour qu'elle ne retourne pas de valeur mais qu'elle modifie directement la variable de la fonction appelante qui doit stocker le résultat (dans notre exemple, la variable `v1` de la fonction `main` devra directement être modifiée par les instructions de la fonction `f1`). Donnez également la nouvelle instruction d'appel à `f1` dans la fonction `main`. La valeur des variables `v1`, `v2`, `v3`, `a`, `b` et `c` lors de chaque affichage devra être la même que dans le programme précédent.
3. Modifiez le programme initial pour qu'il n'y ait plus de variables globales. Vos modifications ne doivent pas changer le comportement du programme. **Attention** : votre solution doit être indépendante de la valeur initiale des variables et vous ne devez passer un paramètre par adresse que si c'est nécessaire.

Exercice 3 (*renforcement*) – Analyse du contenu d'un tableau

1. Écrivez une fonction `nb_pos_neg` qui prend en argument un tableau d'entiers et sa taille et permet d'obtenir **en un seul appel** le nombre de valeurs strictement positives dans le tableau **et** le nombre de valeurs strictement négatives.
2. En utilisant la fonction précédente (que vous n'avez pas besoin de réécrire ici), écrivez un programme qui déclare et initialise un tableau d'entiers et qui affiche le nombre de valeurs nulles du tableau et s'il y a plus, autant ou moins de valeurs positives que négatives dans le tableau.

Exercice 4 (*entraînement*) – Analyse du contenu d'un tableau - suite

1. Écrivez une unique fonction `min_max` qui permet de calculer et de récupérer **en un seul appel** le minimum et le maximum d'un tableau d'entiers.
2. En utilisant la fonction précédente, écrivez un programme qui déclare et initialise un tableau d'entiers et qui affiche la valeur du plus petit entier du tableau et celle du plus grand.

Exercice 5 (*entraînement*) – Moyenne et nombre de notes

Nous considérons un tableau contenant les notes des étudiants ayant passé un examen (notes comprises entre 0 et 20). La taille maximale du tableau (nombre maximum d'étudiants) est définie par la directive `#define`. Il peut y avoir moins d'étudiants ayant passé l'examen que le nombre maximum d'étudiants possible. Afin de marquer la fin de la liste des notes, la dernière case significative (dernière note) du tableau est suivie par la valeur -1.

1. Écrivez une fonction `moyenne_nb` qui étant donné le tableau de notes, retourne la moyenne des notes et le nombre d'étudiants ayant la moyenne (note supérieure ou égale à 10). La fonction retournera 0 si le tableau contient au moins une note et si toutes les notes sont comprises entre 0 et 20. Sinon, elle retournera -1.
2. Complétez la fonction `main` du programme suivant afin d'appeler la fonction `moyenne_nb` et d'afficher les résultats calculés par cette fonction.

```

#include <stdio.h>

#define DIM 100

.... moyenne_nb(.....) {
....
....
....
}

int main() {
    float tab_notes[DIM] = {6, 7, 10, 20, 15, 3, 12, 5, 14, 17, -1};

    /* A compléter*/

    return 0;
}

```

Exercice 6 (*approfondissement*) – Calendrier grégorien

Rappelons qu’il y a une année bissextile tous les 4 ans à l’exception de “la dernière année de chaque siècle [qui] n’est bissextile que si son millésime est divisible par 400” (d’après le Petit Larousse 2000).

On ne s’intéresse ici qu’aux dates du calendrier grégorien, c’est-à-dire au delà du 15 octobre 1582 à partir duquel s’applique cette règle pour les années bissextiles.

1. On souhaite déterminer si une année est bissextile. Vous allez écrire pour cela la fonction `est_bis`, dont la signature est donnée ci-dessous, qui retourne 1 si l’année fournie en argument est bissextile et 0 sinon.

```
int est_bis(int annee);
```

On veut réaliser un programme nous permettant de faire des “calculs” sur des dates, par exemple : déterminer la date du lendemain ou bien encore connaître le nombre de jours séparant deux dates. Pour toutes ces manipulations, il va être très utile de convertir les dates usuelles présentées sous la forme **jour, mois, année** en dates absolues exprimées en jours depuis une date origine correspondant au début du calendrier grégorien (le 15 octobre 1582).

Ce programme reposera sur deux fonctions principales permettant de passer du mode de représentation usuel des dates à celui des dates absolues et inversement.

2. Dans un premier temps nous allons traiter la fonction permettant de passer de la représentation usuelle à la représentation absolue. Pour réaliser cette conversion en limitant le nombre de tests, nous allons utiliser une table à deux dimensions pour stocker le nombre de jours de chaque mois d’année normale (première ligne) et d’une année bissextile (seconde ligne).

31	28	31	30	31	30	31	31	30	31	30	31
31	29	31	30	31	30	31	31	30	31	30	31

Nous utiliserons également un tableau à une dimension qui stocke le nombre de jours d’une année normale (première case) et bissextile (deuxième case).

365	366
-----	-----

Pour accéder à ces deux tableaux, on utilisera la valeur de retour de la fonction `int est_bis(int annee)`.

Écrivez la première fonction de conversion `jma2abs` dont la signature est donnée ci-dessous :

```
int jma2abs(int jour, int mois, int annee)
```


Cette fonction retourne une valeur entière correspondant au nombre de jours séparant (jour, mois, année) de la date origine (le 15 octobre 1582) définie à l'aide des trois directives **#define** :

```
#define ORIGINSEM    4
#define JORIGIN      15
#define MORIGIN      10
```

Par exemple, le 7 janvier 1583 est à 84 jours de la date d'origine.

Vous utiliserez obligatoirement les tableaux définis précédemment ainsi que la fonction `est_bis`.

3. Écrivez maintenant la fonction `abs2jma` réalisant la conversion inverse et dont la signature est donnée ci-dessous :

```
void abs2jma(int *j, int *m, int *a, long nbj);
```

Vous remarquerez que les paramètres formels `j`, `m` et `a` sont des pointeurs. Ceci permet à la fonction de produire trois résultats qui seront directement stockés dans des variables de la fonction appelante.

4. Maintenant que nous disposons des fonctions précédemment écrites et sachant que le 15 octobre 1582 était un vendredi, il vous est facile d'écrire une fonction `jdelasemaine` indiquant à quel jour de la semaine correspond une date. La signature de cette fonction est donnée ci-dessous.

```
int jdelasemaine(int jour, int mois, int annee);
```

Cette fonction retourne une valeur numérique comprise entre 0 et 6. La valeur 0 correspond à lundi.

5. Écrivez maintenant un programme principal permettant de tester vos fonctions. Pour les affichages, vous pourrez utiliser les deux fonctions suivantes :

```
void affiche_jour (int no) {
    if (no == 0) { printf("lundi"); }
    if (no == 1) { printf("mardi"); }
    if (no == 2) { printf("mercredi"); }
    if (no == 3) { printf("jeudi"); }
    if (no == 4) { printf("vendredi"); }
    if (no == 5) { printf("samedi"); }
    if (no == 6) { printf("dimanche"); }
    if ( (no < 0) || (no > 6) ) {
        printf("Cette valeur ne correspond pas a un jour de la semaine\n");
    }
}
```

```
/*-----*/
```

```
void affiche_mois (int no) {
    if (no == 1) { printf("janvier"); }
    if (no == 2) { printf("fevrier"); }
    if (no == 3) { printf("mars"); }
    if (no == 4) { printf("avril"); }
    if (no == 5) { printf("mai"); }
    if (no == 6) { printf("juin"); }
    if (no == 7) { printf("juillet"); }
    if (no == 8) { printf("aout"); }
    if (no == 9) { printf("septembre"); }
    if (no == 10) { printf("octobre"); }
    if (no == 11) { printf("novembre"); }
    if (no == 12) { printf("decembre"); }
    if ( (no < 1) || (no > 12) ) {
        printf("Cette valeur ne correspond pas a un mois du calendrier\n");
    }
}
```

Savoir-faire acquis

- ★ Savoir qu'une variable de type tableau est un pointeur
- ★ Savoir écrire un programme sans variables globales
- ★ Écrire des fonctions calculant plusieurs résultats
- ★ Savoir définir des structures avec des types appropriés
- ★ Savoir manipuler des structures et les passer par adresse en paramètres d'une fonction

Applications classiques à connaître

- ★ Calculer, avec une seule fonction, le plus grand et plus petit élément d'un tableau
- ★ Calculer, avec une seule fonction, le nombre d'éléments positifs et le nombre d'éléments négatifs d'un tableau
- ★ Calculer, à partir d'une date sous la forme jj/mm/aaaa, le nombre de jours écoulés depuis le 15 octobre 1582 (et réciproquement)



TME semaine 10

Les pointeurs et les structures

Version du 18 décembre 2014

Objectif(s)

- ★ Passage de paramètres de fonctions par référence
- ★ Écriture de fonctions à résultats multiples
- ★ Définir et utiliser des structures
- ★ Appeler des fonctions prenant en paramètre des structures et des tableaux de structures

Exercice(s)

Pour tous les exercices de ce TME, sauf le premier, vous compilerez votre code en ligne de commande comme vu au TME 9.

Exercice 1 (*base*) – Accès à la mémoire

1. Ouvrez et sauvegardez dans votre répertoire TME10 le programme `tp10-debug.c` suivant que vous trouverez dans le répertoire Semaine10.

```
#include <stdio.h>

int f(int p, int n, int r) {
    int res = 0;

    r = p%n;
    if (r == 0) {
        res = 1;
    }
    printf("%d\n", r);
    return res;
}

int main() {
    int x, y, reste;
    int div;

    printf("Saisissez un entier : ");
    scanf("%d", &x);
    printf("Saisissez un deuxieme entier : ");
    scanf("%d", &y);

    div = f(x, y, reste);
    if( div ) {
        printf("%d est un diviseur de %d\n", y, x);
    }
    else {
```

```

    printf("le reste de la division de %d par %d est %d\n", x, y, reste);
}
return 0;
}

```

Visualisez son comportement à l'aide de *geany-plugin-debug*, comme vu en TME 1. Vous devrez exécuter votre programme pour différentes valeurs des variables *x* et *y*. Expliquez et corrigez les éventuelles erreurs de compilation et d'exécution.

Exercice 2 (*renforcement*) – Échange de valeurs

1. Nous souhaitons écrire une fonction `echanger` permettant d'échanger les valeurs de deux variables. Si $a=2$ et $b=4$, après appel à la fonction `echanger`, on doit avoir $a=4$ et $b=2$. La valeur initiale des variables sera saisie par l'utilisateur.

Complétez le programme suivant.

```

#include <stdio.h>

..... echanger(.....) {
    .....
}

int main () {
    int a;
    int b;

    /* Saisie de la valeur de la variable a */
    printf("Saisie de a : ");
    .....

    /* Saisie de la valeur de la variable b */
    printf("Saisie de b : ");
    .....

    printf("Avant echange : a = %d et b = %d\n", a, b);

    /* appel a la fonction echanger */
    .....

    printf("Après echange : a = %d et b = %d\n", a, b);
    return 0;
}

```

Exercice 3 (*renforcement*) – Quelques petites fonctions

Les questions suivantes vous demandent d'écrire plusieurs fonctions. Vous devez **toujours** écrire un programme minimal qui permettra de tester, **au fur et à mesure**, les fonctions que vous allez écrire. On parle de "jeu de tests".

1. On souhaite saisir une certaine quantité de nombres entiers, cette quantité devant être saisie par l'utilisateur. Ecrivez un programme permettant la saisie de la quantité d'entiers souhaitée, puis leur saisie. À chaque fois qu'on saisira un entier, on se contentera de l'afficher, sans le sauvegarder. On pourra ainsi utiliser la même variable pour stocker chacun des entiers.

2. On souhaite maintenant calculer la somme des entiers saisis. Il faut donc, pour chaque nouvel entier saisi, mettre à jour la somme obtenue jusque là. Cette mise à jour doit être faite, dans le cadre de cet exercice, par une fonction. On affichera ensuite, une fois que tous les entiers auront été saisis, leur somme totale dans la fonction `main`.

Écrivez et testez pour cela la fonction `void somme(...)` qui ne retourne pas de valeur, et n'utilise pas de variables globales.

Indication : si vous n'avez aucune idée de comment faire, ajoutez les instructions nécessaires pour le calcul de la somme dans la fonction `main` et une fois que votre programme est correct déplacez ces instructions dans la fonction `somme` en faisant bien attention à ce que toutes les informations nécessaires soient passées en paramètre.

3. On souhaite déterminer la valeur du plus grand entier saisi ainsi que celle du plus petit.

Écrivez une fonction `void min_max(...)`, qui ne retourne rien, qui est appelée à chaque fois qu'un nouvel entier est saisi et qui permet de récupérer la valeur du plus grand entier parmi les entiers saisis jusque là ainsi que celle du plus petit. Vous ne devez pas utiliser de variables globales. Modifiez votre programme pour pouvoir tester cette fonction. L'affichage de la plus petite et de la plus grande valeur doit se faire dans la fonction `main`.

Indication : comme pour la question précédente, si vous n'avez aucune idée de la manière de faire, ajoutez, dans la fonction `main`, les instructions nécessaires pour l'identification des plus grand et plus petit entiers saisis jusque là et, une fois que votre programme est correct, déplacez ces instructions dans la fonction `min_max` et faisant bien attention à ce que toutes les informations nécessaires soient passées en paramètre.

Exercice 4 (*renforcement*) – Manipulation des tableaux de structures

On cherche à comptabiliser le nombre d'étudiants inscrits dans une Unité d'Enseignement (UE). Une UE est identifiée par un code qui est un nombre entier.

À chaque étudiant sont associées deux informations :

- un numéro,
- l'ensemble des UE auxquelles il est inscrit.

Nous disposons des constantes suivantes (définies par des directives `#define`) :

- `NOMBRE_ETUDIANT`, qui détermine le nombre d'étudiants considérés,
- `NOMBRE_UE`, qui détermine le nombre d'ue suivies par un étudiant.

Dans cet exercice, on complètera le contenu du fichier `tp10-etudiants.c` que vous pouvez récupérer à l'emplacement habituel.

1. Définissez une structure qui permet de représenter un étudiant.

La fonction `main` déclare une variable `ensemble_etudiants` qui permet de stocker les informations sur l'ensemble des étudiants.

2. Complétez le code de la fonction `initialisation_tableau_etudiants`, qui permet d'initialiser aléatoirement les informations relatives à l'ensemble des étudiants
3. Complétez le code de la fonction `compte_membre`, qui retourne le nombre d'étudiants qui sont inscrits à l'ue dont le numéro est passé en paramètre. Un étudiant inscrit plusieurs fois à la même UE ne sera comptabilisé qu'une seule fois.
4. On souhaite à présent modifier le code d'une UE. Complétez le code de la fonction `modif_code` qui modifie le code `c1` d'une UE en `c2` pour tous les étudiants inscrits à `c1` (et pour chaque inscription si l'étudiant est inscrit plusieurs fois à l'ue `c1`).
5. Complétez le programme principal qui doit afficher le nombre d'étudiants inscrits à chaque ue.

Exercice 5 (*entraînement*) – Fonction qui modifie une chaîne de caractères et renvoie des résultats

Nous vous rappelons qu'un tableau de caractères peut être initialisé avec une chaîne de caractères. Dans ce cas, en plus d'être utilisable comme n'importe quel autre tableau, il a les caractéristiques des chaînes de caractères : le caractère `'\0'` est ajouté à la suite du dernier caractère, pour signaler la fin de la chaîne.

1. Modifiez le programme suivant que vous trouverez dans le fichier `tp10-espaces.c` en :

- complétant la fonction `supprime_espaces` qui doit transformer une chaîne de caractères contenant une phrase (avec espaces) en une chaîne de caractères ne contenant pas d'espace et qui compte le nombre d'espaces supprimés,
- complétant la fonction `main`

```
#include <stdio.h>

void supprime_espaces(char chaine[], .....){
    .....
}

int main () {
    char phrase[] = "Bonjour tout le monde.";
    int i, nb;

    .....

    printf("Nombre d'espaces supprimees : %d\n", nb);
    printf("Phrase modifiee : \n");

    .....

    return 0;
}
```

Exercice 6 (*entraînement*) – Codage Morse

On souhaite écrire un programme affichant le codage en morse d'un texte. Les caractères susceptibles d'être codés en morse sont les suivants :

Les 26 lettres de l'alphabet en majuscules :

A .-	B -...	C -. .	D -..	E .	F ..-	G --.
H	I ..	J .---	K -.-	L .---	M --	N -.
O ---	P .---	Q --.-	R .-	S ...	T -	U ..-
V ...-	W .--	X -.-.	Y -. --	Z --..		

Le point : .-.-.-

Les chiffres :

0 -----	1 .-----	2 ..----	3 ...---	4--	5-	6 -.....
7 --...	8 ---..	9 ----.				

Chaque couple (caractère, code) est représenté par une structure ayant deux champs :

- le caractère codé,
- le code associé au caractère (le code le plus long comporte 5 caractères mais il faut prévoir la place pour le caractère `'\0'` qui signale la fin de chaîne),

1. Définissez, en langage C, la structure précédente.
2. L'ensemble des codes morse est stocké dans un tableau de structures initialisé dans le programme principal. Le programme principal contient également la déclaration et l'initialisation de la chaîne de caractères à traduire (on suppose que celle-ci ne contient que des lettres majuscules, des points et des chiffres).

Écrivez une fonction `affiche_code` qui prend en paramètres un tableau de correspondance et la chaîne de caractères à traduire et affiche le code correspondant.

Exercice 7 (entraînement) – Tableau de structures : signes du Zodiaque

On souhaite afficher le signe du Zodiaque correspondant à une date saisie par l'utilisateur.

Nous rappelons que les périodes correspondant à chaque signe sont les suivantes :

Capricorne	23 décembre - 19 janvier
Verseau	20 janvier - 19 février
Poisson	20 février - 20 mars
Bélier	21 mars - 19 avril
Taureau	20 avril - 20 mai
Gémeau	21 mai - 20 juin
Cancer	21 juin - 21 juillet
Lion	22 juillet - 22 août
Vierge	23 août - 22 septembre
Balance	23 septembre - 22 octobre
Scorpion	23 octobre - 21 novembre
Sagittaire	22 novembre - 22 décembre

La constante suivante est définie :

```
# define NOMBRE_SIGNES 12
```

1. Définissez la structure `fin_signe` qui permet de stocker dans une même variable le nom d'un signe, le jour et le mois de fin du signe.
2. Dans la fonction `main`, définissez et initialisez le tableau `zodiaque` qui permet de regrouper l'ensemble des informations sur les douze signes du zodiaque comme suit :

```
struct fin_signe zodiaque[NOMBRE_SIGNES] = {
    {22, "decembre", "Sagittaire"},
    {19, "janvier", "Capricorne"},
    {19, "fevrier", "Verseau"},
    {20, "mars", "Poisson"},
    {19, "avril", "Belier"},
    {20, "mai", "Taureau"},
    {20, "juin", "Gemeau"},
    {21, "juillet", "Cancer"},
    {22, "aout", "Lion"},
    {22, "septembre", "Vierge"},
    {22, "octobre", "Balance"},
    {21, "novembre", "Scorpion"}
};
```

3. Définissez la fonction `signe_zodiaque` qui prend en paramètre un jour (entier) et un mois (chaîne de caractères) et qui retourne la position du signe correspondant dans le tableau `zodiaque`. Si le mois saisi n'est pas correct, la fonction retourne la valeur `NOMBRE_SIGNES` et ne fait pas de vérification sur la validité du jour.

Vous utiliserez la fonction

```
int strcmp(char *s1, char *s2);
```

qui compare les deux chaînes de caractère `s1` et `s2` et renvoie la valeur 0 si ces deux chaînes sont identiques et une valeur différente de 0 sinon.

L'utilisation de cette fonction nécessite l'ajout de la ligne `#include <string.h>` en début de fichier.

4. L'utilisateur doit saisir la date sous la forme :

jour mois

jour étant exprimé sous forme numérique et mois sous la forme d'une chaîne de caractères en minuscules et sans accent.

La saisie de la date se fera en utilisant la fonction `scanf("%d %s", &var1, var2)`, où `var2` est une variable de type tableau de caractères qui aura été précédemment déclarée. Lors de la déclaration de cette variable, on spécifiera la taille du tableau comme valant 10 (i.e. taille maximum, on doit prévoir la place pour stocker le caractère '`\0`' et aucun mois ne fait plus de 9 caractères).

Écrivez le code C permettant de réaliser ce programme.

Exercice 8 (*approfondissement*) – **Calendrier grégorien**

1. Programmez les fonctions et le programme de test présentés dans le dernier exercice de TD.

Savoir-faire acquis

- ★ Écrire et appeler des fonctions ayant des paramètres de type pointeur
- ★ Débugger un programme utilisant des pointeurs
- ★ Définir les structures adaptées à un problème
- ★ Définir et appeler des fonctions prenant des structures en paramètres
- ★ Manipuler des tableaux de structures

Applications classiques à connaître

- ★ Échanger la valeur de deux variables
- ★ Supprimer les espaces dans une chaîne de caractères
- ★ Gérer les données d'étudiants
- ★ Coder une chaîne en Morse



TD semaine 11

Fonctions, pointeurs et structures

Version du 18 décembre 2014

Objectif(s)

- ★ Établir le lien entre tableaux et pointeurs
- ★ Passage de structures par adresse

Exercice(s)

Exercice 1 (*base*) – Tableaux et pointeurs

Soit la déclaration :

```
int tab[10];
```

Nous vous rappelons que la valeur de `tab` est l'adresse du premier élément du tableau (élément `tab[0]`) et que $\forall i$ tel que $0 \leq i \leq n-1$, `tab+i` est l'adresse du $(i+1)^{\text{ème}}$ élément du tableau (élément `tab[i]`).

1. Indiquez ce que fait le programme suivant et réécrivez-le en utilisant la notation `tab+i` au lieu de `tab[i]`.

```
#include <stdio.h>

#define DIM 10

int main() {
    int tab[DIM];
    int i, aux;

    for (i = 0; i < DIM; i++) {
        printf("Contenu de la case %d ? ", i);
        scanf("%d", &tab[i]);
    }
    printf("\n");

    aux = tab[0];
    for (i = 0; i < DIM-1; i++) {
        tab[i] = 4*tab[i+1];
        printf("%d ", tab[i]);
    }
    tab[DIM-1] = 4*aux;
    printf("%d\n", tab[DIM-1]);
    return 0;
}
```

Exercice 2 (*renforcement*) – Gestion d'un carnet d'adresses

On souhaite gérer un carnet d'adresses de contacts. Un contact est composé d'un nom, d'un prénom, d'un numéro de téléphone, d'une adresse mail et d'une adresse postale. Une adresse postale se compose d'un numéro, d'un intitulé de rue, d'un code postal et d'une ville.

1. Définissez les structures `adresse_postale` et `contact`.
2. On représente le carnet d'adresses par un tableau de structures `contact`. Le nombre de contacts présents dans le tableau sera stocké dans une variable `numC` déclarée dans la fonction `main`.
Donnez le code de la fonction `main` déclarant le tableau de contacts de taille maximum `MAXC` (cette valeur sera définie par une directive `#define`) et le nombre de contacts. Vous initialiserez le tableau de contacts avec quelques contacts.
3. Définissez une fonction `affiche_contact` qui affiche tous les contacts d'un carnet d'adresses.
4. Définissez une fonction `modifie_num_tel` qui prend en paramètres un contact et un numéro de téléphone et affecte ce nouveau numéro au contact.
Faites appel à cette fonction dans la fonction `main`.

Exercice 3 (*approfondissement*) – Bataille navale

La bataille navale est un jeu à deux joueurs dans lequel chaque joueur dispose sur une grille des pièces représentant des bateaux. Pour gagner la partie, un joueur doit repérer la position des bateaux de son adversaire avant que celui-ci n'ait localisé les siens.

Pour localiser un bateau adverse, un joueur propose une position sur la grille. Une position est donnée par une ligne (une lettre de 'A' à 'J') et une colonne (un entier de 1 à 10). Son adversaire lui indique si cette position est occupée ou non.

Nous nous intéressons dans cet exercice uniquement à la représentation des pièces du jeu. Les pièces (bateaux) sont de quatre types : les croiseurs occupent deux cases de la grille, les frégates trois, les destroyers quatre et les porte-avions cinq. À chaque type est associée une couleur pour sa représentation sur la grille de jeu.

La flotte d'un joueur est composée d'un porte-avions, deux destroyers, trois frégates et quatre croiseurs.

1. Toutes les pièces de même type ont la même taille et la même couleur. Proposez une solution qui permette de regrouper cette information sans avoir à la recopier dans chaque pièce.
2. Un bateau est localisé par la position de la case la plus en haut et la plus à gauche qu'il occupe, ainsi que sa direction ('h' pour horizontale, 'v' pour verticale).
Donnez la définition du type permettant de déclarer un bateau et écrivez un programme qui déclare un bateau et affiche l'ensemble de ses caractéristiques.
3. Supposons que l'on veuille écrire des fonctions qui permettent par exemple de déplacer un bateau verticalement, ou horizontalement, ou de modifier sa direction. Pourquoi est-il plus intéressant de passer en paramètre à la fonction un pointeur sur la pièce à modifier, plutôt que d'utiliser une valeur de retour ?

Savoir-faire acquis

★ Utiliser le passage par référence pour des structures



TME semaine 11 TME solitaire

Version du 18 décembre 2014

Objectif(s)

★ Contrôle sur machine

Introduction

Les deux séances de TME de cette semaine sont consacrées à un troisième TME solitaire. Par demi-groupes de 15 étudiants, vous allez tous composer pendant **1h30** seul devant votre machine.

Le but de ce TME solitaire est d'évaluer votre capacité à écrire des programmes C faisant appel aux boucles, alternatives, tableaux, structures, fonctions, pointeurs. Le sujet se découpe en 2 exercices : 1 exercice étiqueté "difficile" et un problème constitué de plusieurs questions.

L'énoncé du problème vous sera distribué en début de séance. La feuille d'énoncé est à rendre à l'enseignant en fin de séance.

Pour l'exercice 1, vous perdrez un tiers des points si votre programme ne compile pas (syntaxe non correcte), même s'il est correct à 95%. **Vous devrez systématiquement vous assurer que ce que vous avez écrit compile sans erreur.**

Le reste des points est attribué en fonction :

- de l'adéquation entre votre code et la résolution du problème de l'énoncé (votre code résout le problème posé) ; attention à considérer les différents cas de tests possibles lors de l'écriture de votre code
- de la qualité du code : utilisation des instructions appropriées, clarté du code, indentation, présence de commentaires, etc..

Pour commencer

Créez un répertoire `TMEsolo3` dans votre répertoire personnel. Comme lors des TME habituels, vous devrez récupérer des fichiers `.c` dans le répertoire que vous indiquera votre enseignant puis les enregistrer dans le répertoire `TMEsolo3` pour pouvoir travailler dessus. Vous avez en tout 2 fichiers à récupérer : *ex1.c* et *probleme.c*.

Travail à réaliser

À la fin de la séance, vous ferez une archive du répertoire `TMEsolo3` que vous enverrez via la page de soumission habituelle.

Pour chaque exercice, complétez le fichier correspondant pour que le programme soit conforme à ce qui est indiqué en commentaire dans le fichier `.c` et dans la feuille d'énoncé pour le problème.

Certaines parties du programme ne doivent pas être modifiées. En particulier, vous ne devez pas supprimer les constantes définies à l'aide de la primitive `#define`.

TD Semaine 12 Les Tris

Version du 18 décembre 2014

Objectif(s)

- ★ être capable d'écrire un tri de base
- ★ au travers des tris, réviser quelques notions essentielles : échange de valeurs, recherche du max, ...

Exercice(s)

Exercice 1 (*base*) – Échange de deux valeurs

- Redonnez le code de la fonction qui permet d'échanger les valeurs contenues dans deux variables *a* et *b*. Vous supposerez que ces deux variables sont de type entier et vous écrirez un programme appelant cette fonction.
- Quelles sont les modifications à apporter si maintenant les valeurs à échanger sont contenues dans deux cases d'un tableau ?
- Redonnez le code de la fonction qui permet d'afficher le contenu des *n* premières cases d'un tableau, à raison de *p* valeurs par ligne.

Exercice 2 (*renforcement*) – Tri par sélection du maximum (Selection Sort)

L'une des approches classiques pour trier un tableau par ordre croissant est de commencer par placer dans la dernière case la plus grande valeur contenue dans le tableau.

- Donnez une suite d'instructions C permettant de stocker dans la variable *i_max* l'indice de la case contenant la valeur maximum d'un tableau de *i_fin* cases.
- Que faut-il faire pour placer la valeur maximum dans la dernière case du tableau sans perdre aucune des valeurs contenues dans le tableau ? Donnez le code correspondant.

Une fois qu'on a placé la plus grande valeur dans la dernière case, le contenu de cette case ne sera plus affecté par le tri. On peut donc poursuivre le tri en modifiant la valeur de *i_fin* de manière à éliminer du parcours la dernière case du tableau.

1	8	3	6	5	4	7	2
0	1	2	3	4	5	6	7
	<i>i_max</i>						<i>i_fin - 1</i>

FIGURE 1 – Tableau initial

On note alors que pour amener la deuxième plus grande valeur dans l'avant-dernière case du tableau, les opérations à effectuer sont exactement les mêmes que pour le placement du maximum : on a juste supprimé la dernière case dans le parcours. En répétant cette suite d'opérations sur un tableau dont on élimine une case à chaque fois, on obtient un tableau entièrement trié.

1	2	3	6	5	4	7	8
0	1	2	3	4	5	6	7

$i_fin - 1$

FIGURE 2 – Après placement du max

- Écrivez le code de la fonction `tri_selection` qui permet de trier entièrement un tableau en appliquant cet algorithme.

Exercice 3 (*renforcement*) – Tri à bulles (Bubble Sort)

Le tri à bulles doit son nom à un mouvement des valeurs dans le tableau qui s'apparente à la remontée des bulles à la surface d'un liquide (la surface est ici représentée par l'indice maximum du tableau).

Pour faire remonter la valeur maximum vers la “surface”, une série de duels est organisée : lors d'un duel entre la case i et la case $i-1$, le vainqueur (la plus grande valeur) est placée (reste) dans la case i , le perdant occupe alors la case $i-1$.

Le premier duel oppose la case 0 à la case 1. Le vainqueur est opposé à la case 2, etc. jusqu'au duel opposant la case $N-2$ à la case $N-1$ (où N est la taille du tableau).

- Quelle est la valeur contenue dans `tab[N-1]` après l'exécution de cette série de duels ?
- Écrivez un ensemble d'instructions C qui réalise cette série de duels dans un tableau dont le dernier indice est noté `ifin`.
- Représentez l'évolution du tableau de la Figure 3 lors de l'exécution de cette suite d'opérations.

1	5	4	3	8	6
---	---	---	---	---	---

FIGURE 3 – Tableau initial

- Complétez le code pour écrire une fonction `tri_bulle` qui trie entièrement le tableau.
- Combien faut-il effectuer de parcours pour trier entièrement le tableau ? Est-ce toujours nécessaire ? L'algorithme détecte-t-il que le tableau est trié ?
- Un tableau trié est caractérisé par la propriété suivante : $\forall i, 1 \leq i < N, \text{tab}[i-1] \leq \text{tab}[i]$.
Que peut-on dire sur l'exécution de la fonction `tri_bulle` lorsque le tableau est trié ? Déduisez-en une optimisation du programme qui permet de stopper les parcours dès que le tableau est trié.

Exercice 4 (*approfondissement*) – Un peu de complexité

Les algorithmes de tri s'appliquent généralement sur des ensembles de données de grande taille. Il est donc important d'évaluer leur efficacité en essayant de mesurer le nombre d'opérations nécessaires pour trier le tableau. Ce nombre d'opérations dépend de la taille N du tableau et de l'ordre initial des éléments dans le tableau.

Les opérations de base que nous considérons sont :

- comparaison des contenus de deux cases du tableau ;
- échange des contenus de deux cases du tableau.

- Nous considérons le cas où le tableau initial est déjà trié.

Combien l'algorithme du tri à bulles optimisé exécute-t-il d'opérations ? Combien l'algorithme du tri sélection exécute-t-il d'opérations ?

2. Quelle est la configuration de tableau la plus défavorable ? Quel est le nombre d'opérations exécutées par chacun des algorithmes dans cette configuration ?
3. Lequel de ces deux algorithmes choisiriez-vous pour trier un tableau ?

Mini-Projet - Tetris

Version du 18 décembre 2014

Objectif(s)

- ★ Révisions : ces révisions peuvent se faire en programmant le mini-projet proposé dans cet énoncé, qui revient sur les principales parties du programme. Cependant, si vous n’avez pas terminé les exercices des séances précédentes, **il est beaucoup plus important de revenir dessus que de programmer le projet.**

En effet, les notions abordées dans ces exercices sont des notions de *base* qui vous seront *indispensables pour réussir l’examen*.

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 1. A chacune d’elles est associée une lettre de l’alphabet.

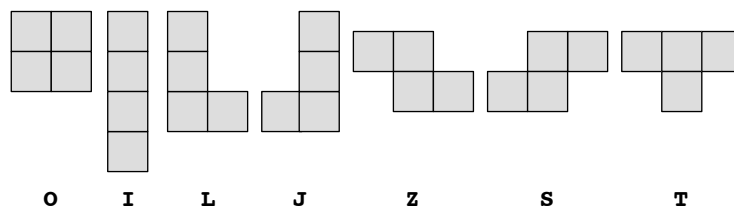


FIGURE 1 – Représentation des sept tétrominos et de la lettre associée

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.

Exercice(s)

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.

Exercice 1 (*renforcement*) – Structures de données

1. Les pièces de Tetris sont colorées. Dans la fonction `main` déclarez un tableau de 7 couleurs (une couleur est une chaîne de caractères, en anglais et en minuscules), correspondant aux 7 tétrominos.

- Un tétramino 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. Définissez une structure permettant de représenter une case.
- 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 contenant la description des 7 tétramino. Nous allons donc déclarer un tableau à deux dimensions : une ligne par tétramino et pour chaque ligne, quatre colonnes correspondant aux 4 cases de chaque 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 2.

Supposons par exemple que le tableau se nomme `tab_pieces`, que la structure utilisée pour définir une case possède deux champs nommés `colonne` et `ligne` (dans cet ordre) et que l'indice 0 soit utilisé pour référencer la pièce 0 (carré). La représentation de cette pièce est donnée dans la Figure 2. 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 2 – Représentation du tétramino 'O'

Insérez, dans la fonction `main` la déclaration et l'initialisation du tableau permettant de stocker les sept tétramino que nous nommerons dans la suite `tab_pieces`. 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 2, la pièce de type 0 est la pièce associée à la lettre O.

- Pour déterminer la position à un instant donné d'un tétramino, 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 tétramino a une couleur. Pour limiter les manipulations de chaînes de caractères, nous allons simplement indiquer dans la définition du tétramino la position de sa couleur dans le tableau de couleurs.

Définissez la structure `piece` permettant de représenter un tétramino : les coordonnées de la case (0, 0) sur le plateau de jeu, les coordonnées relatives de ses 4 cases et sa couleur.

Exercice 2 (renforcement) – 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.

1. Complétez le programme en définissant les valeurs de `LARGEUR`, `HAUTEUR` et `VIDE` par des directives `#define`.

Complétez la fonction `main` en :

- déclarant et initialisant le plateau de jeu,
- ajoutant l'instruction d'ouverture de la fenêtre graphique.

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.

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

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.

Ecrivez 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.

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.

Ecrivez 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.

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 on modifie le plateau de jeu pour marquer les emplacements occupés.

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.

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.
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 3).

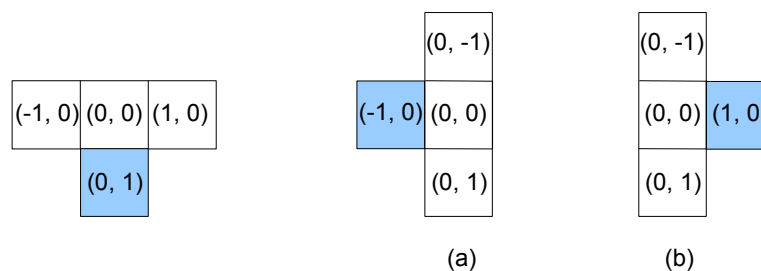


FIGURE 3 – Exemple de rotation droite (a) et gauche (b) : le tétramino 'T'

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.

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.
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

10. L'action `hard_drop` consiste à faire tomber directement une pièce lorsque sa position latérale a été choisie. Écrivez 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 3 (entraînement) – Gestion du score et des 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.

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.

2. Modifiez la fonction `supprimer_lignes` pour prendre en compte le comptage des points. La fonction devra retourner 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`.

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".

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.