Université de Liège

INFO0946

Introduction à la Programmation

Syllabus d'Exercices

Benoit Donnet, Simon Liénardy, Thomas Leuther,
David Lupien St-Pierre, Firas Safadi, Tasnim Safadi

13 août 2020

Table des matières

Table des figures

Liste des tableaux

Introduction

Ce syllabus se veut être un recueil d'exercices permettant à l'étudiant de mettre en pratique les aspects théoriques et techniques vus lors des séances *ex cathedra* du cours INFO0946 (Introduction à la Programmation).

L'objectif principal du cours INFO0946 est l'acquisition des notions fondamentales de l'information et, en particulier, ceux liés à la programmation. Plus précisement, le cours s'articulera autour des notions de syntaxe et sémantique du langage C et de la méthodologie de la programmation. Nous étudierons donc successivement les notions suivantes :

- Variable, bloc, instruction simple (Chap. 1).
- Les structures de contrôle, i.e., conditions et boucles (Chap. 2).
- La mise en place d'une solution à un problème en suivant une méthodologie en quatre étapes : définition du problème, analyse du problème (approche systémique et architecture du code), construction du code autour de l'invariant de boucle et tests [1] (Chap. 3).
- L'évaluation des performances théoriques d'un programme au travers de la notion de complexité (Chap. 4).
- La mise en place de structures de données de base, i.e., tableaux, chaînes de caractères, enregistrements, énumérations, et fichiers (Chap. 5).
- La découpe du programme en modules, i.e., fonctions et procédures, permettant de structurer et réutiliser le code ainsi que la documentation de ces modules, (i.e., spécifications (Chap. 6).
- Les notions de mémoire, pointeurs, et passage de paramètres (Chap. 7).
- L'allocation dynamique de structures de données (Chap. 8).

Les niveaux d'apprentissage attendus chez les étudiants suivent la classification de Bloom :

1. *Connaissance* : mémorisation et restitution d'informations dans les mêmes termes. Les notions théoriques vues dans le cadre du cours ne sont pas très nombreuses. Il est dès demandé de les maîtriser au mieux.
2. *Compréhension* : restitution du sens de l'information dans d'autres termes.
3. *Application* : utilisation de la méthodologie de développement et des algorithmes bien connus pour résoudre un problème.
4. *Analyse* : utilisation de la méthodologie de développement pour identifier les parties constituantes d'un tout.
5. *Synthèse* : utilisation de la méthodologie de développement pour exprimer comment les différentes parties (cfr. Analyse) doivent être combinées pour former un tout.
6. *Evaluation* : formulation de jugements qualitatifs ou quantitatifs sur la solution proposée.

1. La partie "tests" sera développée au Q2, dans le cadre du cours INFO0030.

Afin de mieux comprendre l'organisation pédagogique du cours, nous allons emprunter une métaphore musicale inspirée de Henri-Pierre Charles [2] :

Pédagogie par Objectifs : le solfège est l'étude des principes élémentaires de la musique et de sa notation. Le musicien "fait ses gammes" et chaque exercice a un objectif précis pour évaluer l'apprentissage du "langage musical". Il en va de même pour l'informaticien débutant confronté à l'apprentissage d'un langage de programmation (le C pour nous).

Pédagogie par l'Exemple : l'apprentissage des grands classiques permet au musicien de s'approprier les bases du solfège en les appliquant à ces partitions connues et en les (re)jouant lui-même. L'informaticien débutant, en (re)codant lui-même les algorithmes bien connus (et en refaisant à la maison les problème résolus ensemble en séance théorique) se constituera ainsi une base de réflexes de programmation en "imitant" ces algorithmes.

Pédagogie de l'Erreur : les bogues (ou *bugs* en anglais) sont à l'informaticien ce que les fausses notes sont aux musiciens : des erreurs. Ces erreurs sont nécessaires dans l'acquisition de la connaissance. Un élève a progressé si, après s'être trompé, il peut reconnaître qu'il s'est trompé, dire où et pourquoi il s'est trompé, et comment il recommencerait sans produire les mêmes erreurs.

Pédagogie par Problèmes : connaissant ses "classiques" et les bases du solfège, le musicien devenu plus autonome peut envisager sereinement la création de ses propres morceaux. Le développement d'un projet informatique ambitieux sera "mis en musique" au cours du Q2 dans le cadre du cours INFO0030.

Dans le cours INFO0946, nous adopterons ces différentes stratégies pédagogiques sans oublier qu'en informatique, on apprend toujours en faisant mieux par soi-même.

A travers cette pédagogie, le cours cherche à développer trois "qualités" comportementales chez l'informaticien débutant : la rigueur, la persévérance, et l'autonomie.

Rigueur : un ordinateur est une machine qui exécute vite et bien les instructions qu'on lui a données. Mais cette machine ne sait pas interpréter autre chose : même mineure, une erreur provoque le dysfonctionnement de la machine. Par exemple, pour un `;` oublié dans un programme C, le code source n'est pas compilable et le compilateur nous avertit par ce type de message :

```
fichier.c : In function ‘main’ :
fichier.c :7 : error : syntax error before "printf"
```

Même si un humain peut transiger sur le `;`, la machine ne le peut pas : l'ordinateur ne se contente pas de "l'à peu près". Le respect des consignes, la précision et l'exactitude sont donc de rigueur en informatique !

Persévérance : face à l'intransigeance de la machine, le débutant est confronté à ses nombreuses erreurs (les siennes, pas celles de la machine ! !) et sa tendance naturelle est de passer à autre chose. Mais le papillonnage (ou zapping) est une très mauvaise stratégie en informatique : pour s'exécuter correctement, un programme doit être finalisé. L'informatique nécessite "d'aller au bout des choses".

Autonomie : programmer soi-même des algorithmes qu'on a définis est sans doute le meilleur moyen pour mieux assimiler les principales structures algorithmiques et pour mieux comprendre ses erreurs en se confrontant à l'intransigeante impartialité de l'ordinateur, véritable "juge de paix" des informaticiens. Par ailleurs, l'évolution continuelle et soutenue des langages de programme et des ordinateurs nécessitent de se tenir à jour régulièrement et seule l'autoformation systématique permet de "ne pas perdre pied".

2. Henri-Pierre Charles. *Initiation à l'Informatique*. Ed. Eyrolles. 2000.

Pratique personnelle et autoformation constituent ainsi deux piliers de l'automatisation nécessaire de l'apprenti informaticien. C'est dans cette optique que ce syllabus a été rédigé.

La partie "pratique" du cours INFO0946 est organisée de deux manières :

1. *Répétition.* Les séances de répétition sont organisées, pour chaque étudiant, de manière hebdomadaire. Chaque étudiant disposera de 15 séances de répétition durant le quadrimestre. Les séances de répétition sont encadrées par un Assistant, plusieurs Elèves-Moniteurs et le Professeur. Les séances de répétition ont lieu juste après le cours théorique. L'objectif de ces séances est de résoudre, sur papier, divers problèmes faisant directement référence à la matière vue au cours. Chacune des séances de répétition commence par un rappel théorique
2. *Laboratoire.* Les séances de laboratoire sont organisées, pour chaque étudiant, cinq fois sur le quadrimestre. Les séances de laboratoire sont organisées en parallèle des séances de répétition. L'objectif du laboratoire est de résoudre des exercices sur machine. Cela permet à l'étudiant de se confronter directement à la machine, aux problèmes inhérents (erreurs de compilation, debuggage, ...) et de rendre plus concrète la matière vue au cours.

Pour suivre cette organisation, le présent document est divisé en deux parties : la Partie I fait référence aux séances de répétition, tandis que la Partie II concerne les séances de laboratoire.

Chaque étudiant est supposé travailler le syllabus d'exercices de la façon suivante [3] :

- **Répétition**. La page Web du cours (voir eCampus, Section Calendrier) contient la liste des exercices devant **idéalement** être préparés par chaque étudiant avant chaque séance de répétition. Typiquement, lors de la séance, un exercice sera résolu complètement par l'Assistant. Les autres exercices seront à réaliser, si possible seul, en séance. C'est probablement le bon moment pour poser des questions (les Elèves-Moniteurs circulent dans les rangs pendant la séance pour répondre aux questions ponctuelles) sur des points incompris de la matière. Il ne faut surtout pas attendre que le retard s'accumule, sous peine d'être noyé par la matière et de ne pas pouvoir appréhender la suite du cours et du cursus.
- **Laboratoire**. L'accès au laboratoire nécessite un PMLI (login + mot de passe). Ce PMLI vous est distribué en début d'année académique, lors d'un autre cours (INFO2056 – Premier Projet d'Informatique). Ne le perdez surtout pas, il vous sera utile durant toute votre année. Avant une séance de laboratoire, chaque étudiant est supposé avoir pris connaissance des énoncés et avoir commencé à réfléchir à une solution potentielle. L'idée des laboratoires n'est surtout pas de venir s'asseoir dans la salle, découvrir les exercices et taper frénétiquement du code sur le clavier.

Le présent document contient bien plus d'exercices que ce qui pourra être fait, soit pendant les séances de répétition, soit pendant les séances de laboratoire. L'idée alors est de permettre à l'étudiant de s'exercer par lui-même. Ce travail solitaire est vivement conseillé en vue de réussir l'examen. Les étudiants peuvent toujours contacter l'équipe pédagogique (Professeur ou Assistant) pour discuter des exercices et de leurs solutions. Un calendrier des disponibilités est donné en début d'année académique.

Un correctif de la plupart des exercices est donné dès le début de l'année sur la page Web du cours. Attention, il s'agit là d'un exemple de code correct pour les exercices. En aucun cas, ce code ne correspond à la seule et unique solution. N'hésitez donc pas à discuter de votre propre solution avec l'équipe pédagogique.

Attention, ne soyez pas attentiste vis-à-vis de ces correctifs. Faites avant tout des exercices par vous-même sans regarder les solutions ! A l'examen, vous serez, seul, face à des problèmes

3. Note préalable : chaque étudiant est supposé disposer du syllabus d'exercices dès la première séance de répétition

inédits. Si vous n'avez pas suffisamment pratiqué (seul) pendant le quadrimestre, le résultat de l'examen risque d'être catastrophique.

Enfin, un examen d'une session antérieure (Janvier 2014) est disponible sur la page Web du cours. En fin de quadrimestre, il est vivement conseillé à chaque étudiant de faire, à la maison, cet examen en se mettant en "situation". Le correctif est, lui aussi, disponible sur la page Web du cours. Dans tous les cas, l'équipe pédagogique reste à votre disposition pour discuter de vos solutions à cet examen.

Première partie

Séances de Répétition

Chapitre 1

Blocs, Variables, Instructions Simples

L'objectif de ce premier chapitre d'exercices est de passer en revue les divers éléments vus au cours théorique. En particulier, les éléments de base vus dans l'Introduction et ceux portant sur les bases du langage C. Les points abordés dans ce chapitre sont :

- la manipulation d'algorithmes (Sec. 1.1).
- la représentation binaire des entiers (Sec. 1.2).
- La manipulation des divers opérateurs C (Sec. 1.3).
- Les types (Sec. 1.4).
- Le bloc d'instructions (Sec. 1.5).
- Les premiers pas en écriture de code C (Sec. 1.6).

1.1 Manipulation d'Algorithmes

Dans ces exercices, nous allons manipuler des algorithmes. Pour chacun des algorithmes à appliquer, vous devez indiquer clairement toutes les étapes de votre raisonnement.

1.1.1 Calcul écrit

À l'occasion de l'apprentissage des opérations élémentaires en mathématique, les élèves de l'enseignement primaire apprennent non seulement à effectuer des opérations simples (mettant en jeu de petits nombres) mentalement mais aussi à décomposer des calculs impliquant de grands nombres à l'aide du *calcul écrit*.

Les règles d'un calcul écrit impliquent notamment :

- de découper l'opération à réaliser en opérations plus simples ;
- de retenir le résultat de toutes les opérations simples possibles [1] ;
- de combiner les résultats des opérations simples. La manière de le faire est la plupart du temps aidée par un *agencement particulier* [2] des chiffres qui composent les nombres à manipuler.

Ces règles consistent donc à répéter plusieurs fois des opérations simples et constituent, dès lors, des algorithmes. L'ordinateur sur lequel est encodé l'algorithme n'est autre que l'élève de primaire lui-même. D'ailleurs, tout comme un ordinateur, on attend d'un bon élève qu'il applique le mieux possible les règles du calcul écrit, pas qu'il puisse expliquer *pourquoi* la méthode qu'on lui a demandé d'appliquer est correcte.

1. Souvent à l'aide de moyens mnémotechniques, comme les tables de multiplications.
2. C'est-à-dire que ces chiffres sont placés d'une manière particulière qu'il convient de respecter.

1.1.1.1 Addition écrite

Soit un nombre A, composé des chiffres $(a_{n-1}a_{n-2}\dots a_1a_0)_{10}$ et un nombre B, composé des chiffres $(b_{m-1}b_{m-2}\dots b_1b_0)_{10}$. On demande que $A \geq B$ (si ce n'était pas le cas, il suffit d'inverser les rôles entre A et B).

Disposition des chiffres des nombres Il faut dessiner $n+1$ colonnes numérotées de n à 0 de la gauche vers la droite. Sur une première ligne, on encode le nombre A de la manière suivante : le chiffre a_0 est placé dans la colonne 0, le chiffre a_1 est placé dans la colonne 1 et ainsi de suite jusqu'au chiffre a_{n-1}. Le nombre B est encodé juste en dessous du nombre A, d'une manière similaire (chiffre b_i dans la colonne i). On trace ensuite une ligne horizontale en dessous du nombre B. À la droite du nombre B, on inscrit dans la colonne n le symbole « + » qui représente l'addition.

Opérations à répéter Cette opération requiert de connaitre tous les résultats possibles des additions du type $a+b+c$, avec $a, b \in [0\dots9]$ et $c \in \{0,1\}$.

Pour chaque colonne, en partant de la colonne 0 jusque la colonne $n-1$, il faut répéter cette opération :

> Additionner les chiffres qui sont présents dans la colonne au dessus de la ligne horizontale.
> Si le résultat est représentable sur un seul chiffre, inscrire ce chiffre sous la ligne horizontale, toujours dans la même colonne.
> Si le résultat n'est pas représentable sur un chiffre (il tient donc sur deux chiffres), inscrire le chiffre de droite dans la présente colonne, sous la barre horizontale et inscrire le chiffre de gauche (normalement, c'est un « 1 ») au dessus du chiffre constituant le nombre A dans la première colonne qui se trouve à gauche de la colonne dont les chiffres viennent d'être additionnés. On dit qu'il se produit un *report*.
> Si un report à lieu à la colonne $n-1$, le chiffre de gauche doit directement être écrit sous la barre horizontale, dans la colonne n.

Lecture du résultat Les chiffres inscrits sous la ligne horizontale sont lus de la gauche vers la droite pour former le résultat de l'addition appelé *somme*.

▷ **Exercice 1** Appliquer l'algorithme d'addition écrite pour calculer $78594 + 456$.

1.1.1.2 Soustraction écrite

Voici un exemple de la soustraction $3496 - 2987$ ($= 509$) :

	3	4^{+10}	9	6^{+10}
-	2_{+1}	9	8_{+1}	7
	0	5	0	9

En vous basant des exemples précédents, proposez une description de l'algorithme de soustraction écrite. Utilisez des termes précis. Une personne qui n'a jamais vu de soustraction écrite doit pouvoir comprendre, sur base de votre description, comment fonctionne une soustraction écrite. Tenez compte que :

- Tous les résultats des soustractions $a-b$, avec $a \geq b$, $a \in [0,19]$, $b \in [0,10]$ sont supposés connus ;
- Les résultats des opérations $x+1$ et $x+10$, avec $x \in [0,9]$ sont supposés connus ;

- Il est inutile d'expliquer *pourquoi* l'emprunt (écrire « +10 » au dessus d'une colonne et « +1 » en dessous dans la colonne qui est juste à sa gauche) fonctionne et permet d'obtenir un résultat correct. Par contre, il convient de décrire précisément l'opération.

1.1.2 Multiplication à la Russe

La technique de multiplication dite *à la russe* consiste à diviser par deux le *multiplicateur* (et ensuite les quotients obtenus) jusqu'à un quotient nul, à noter les restes et à multiplier parallèlement le multiplicande par deux. On additionne alors les multiples obtenus du [3] correspondant aux restes non nuls.

Dans le produit de **6** par **4** (i.e., 6×4), on dit que

- **6** est le *multiplicande*, car c'est lui qui est répété (multiplier 6 par 4 revient à calculer $6 + 6 + 6 + 6$) ;
- **4** est le *multiplicateur*, car il indique combien de fois 6 doit être répété.

Par exemple, 123×68 :

multiplicande $M \times 2$	**multiplicateur** $m \div 2$	**reste** $m \bmod 2$	**somme partielle**
123	68	0	$(0 \times 123) + 0$
246	34	0	$(0 \times 246) + 0$
492	17	1	$(1 \times 492) + 0$
984	8	0	$(0 \times 984) + 492$
1968	4	0	$(0 \times 1968) + 492$
3936	2	0	$(0 \times 3936) + 492$
7872	1	1	$(1 \times 7872) + 492$
		$68 \times 123 =$	8364

▷ **Exercice 1** Effectuer la multiplication suivante selon la technique "à la russe" : 64×96

▷ **Exercice 2** Effectuer la multiplication suivante selon la technique "à la russe" : 45×239

1.1.3 Multiplication Arabe

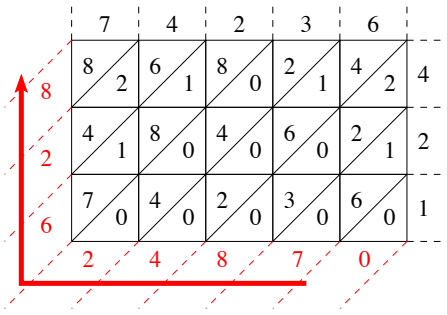

FIGURE 1.1 – Exemple de multiplication arabe : $63247 \times 124 = 7842628$.

3. multiplicande

On considère ici le texte d'Ibn al-Banna concernant la multiplication à l'aide de tableaux [4]

> Tu construis un quadrilatère que tu subdivises verticalement et horizontalement en autant de bandes qu'il y a de positions dans les deux nombres multipliés. Tu divises diagonalement les carrés obtenus à l'aide de diagonales allant du coin inférieur gauche au coin supérieur droit.
> Tu places le multiplicande au-dessus du quadrilatère, en faisant correspondre chacune de ses position à une colonne. [a] Puis, tu places le multiplicateur à gauche ou à droite du quadrilatère, de telle sorte qu'il descende avec lui en faisant correspondre égaglement chacune de ses position à une ligne. [b] Puis, tu multiplies, l'une après l'autre, chacune des positions du multiplicande du carré par toutes les positions du multiplicateur et tu poses le résultat partiel correspondant à chaque position dans le carré où se coupent respectivement leur colonne et leur ligne, en plaçant les unités au-dessus de la diagonale et les dizaines en dessous. Puis, tu commences à additionner, en partant du coin supérieur gauche : tu additionnes ce qui est entre les diagnoles, sans effacer, en plaçant chaque nombre dans sa position, en transférant les dizaines de chaque somme partielle à la diagonale suivante et en les ajoutant à ce qui y figure.
>
> *a.* Attention, l'écriture du nombre se fait de droite à gauche (e.g., 352 s'écrira donc 253).
> *b.* Attention, l'écriture du nombre s'effectue de bas en haut (e.g., $\begin{matrix}3\\5\\2\end{matrix}$) s'écrira donc $\begin{matrix}2\\5\\3\end{matrix}$.

La Fig. 1.1.3 illustre un exemple de multiplication arabe.

▷ **Exercice 1** En utilisant la méthode du tableau d'Ibn al-Banna, calculer 35617×1029

1.2 Représentation Binaire des Nombres

Dans ces exercices, nous allons manipuler l'algorithme de conversion d'un entier, en base décimale, en un entier en base binaire (et vice versa). Pour chacune des questions, vous veillerez à indiquer clairement toutes les étapes de votre raisonnement.

1.2.1 Du Décimal vers le Binaire

▷ **Exercice 1** Donnez, sur 8 bits, la représentation binaire des entiers non signés

1. 15
2. 20
3. 71
4. 100
5. 110
6. 133
7. 154

▷ **Exercice 2** Donnez la représentation binaire, sur 8 bits des entiers signés

1. -5
2. -2
3. -83
4. 42
5. -13

4. J.-L. Chabert, E. Barbin, M. Guillemot, A. Michel-Pajus, J. Borowczyk, A. Djebbar, J.-C. Martzloff. *Histoire d'Algorithmes : du Caillou à la Puce*. Ed. Belin. 1994.

1.2.2 Du Binaire vers le Décimal

▷ **Exercice 1** Représentez en base 10 les entiers non signés dont la représentation binaire est

1. 00010010
2. 00101100
3. 00100000
4. 00001001
5. 00110001

▷ **Exercice 2** Représentez en base 10 les entiers signés dont la représentation binaire est

1. 10000010
2. 10000001
3. 10111101
4. 00011001
5. 00001001

1.2.3 Réflexion

▷ **Exercice 1** "There are 10 kinds of people. Those who understand binary notation, and those who don't." Expliquez.

1.3 Manipulation d'Opérateurs

1.3.1 Table de Vérité

▷ **Exercice 1** Écrivez les tables de vérité pour les expressions suivantes :

1. A || (!A && B)
2. A && (A || B)
3. !A &&!B

Vous veillerez à indiquer toutes les étapes de votre raisonnement.

1.3.2 Affectation

▷ **Exercice 1** Écrivez de la manière la plus compacte possible les expressions suivantes :

1. `y = x + y;`
2. `x = x + 1;`
3. `y = y + 3;`
4. `b1 = b1 && b2;`
5. `b1 = b1 == b3;`
6. `b2 = b2 + 1;`

1.3.3 Priorité

▷ **Exercice 1** Déterminez le résultat des expressions suivantes, avec `x = 1` et `y = 2` :

1. `2 + 3 * 4 + 12 % 3`
2. `1 != (x++ == --y)`

▷ **Exercice 2** Soit un programme contenant les déclarations suivantes :

```
int i = 8;
int j = 5;
float x = 0.005;
float y = -0.01;
char c = 'c';
char d = 'd';
```

Pour rappel, la constante entière associée au caractère `'a'` dans la table ASCII vaut 97.

Déterminez la valeur de chacune des expressions suivantes :

1. `(3 * i - 2 * j) % (2 * d - c)`
2. `2 * ((i / 5) + (4 * (j - 3)) % (i + j - 2))`
3. `i <= j`
4. `j != 6`
5. `c == 99`
6. `5 * (i + j) > 'c'`
7. `(i > 0) && (j < 5)`
8. `(i > 0) || (j < 5)`
9. `(x > y) && (i > 0) || (j < 5)`
10. `(x > y) && (i > 0) && (j < 5)`

▷ **Exercice 3** Évaluez les expressions suivantes en supposant `a = 20`, `b = 5`, `c = -10`, `d = 2`, `x = 12` et `y = 15`. Notez chaque fois la valeur rendue comme résultat de l'expression et les valeurs des variables dont le contenu a changé.

1. `(5 * x + 2 * ((3 * b) + 4))`
2. `(5 * (x + 2) * 3) * (b + 4)`
3. `a == (b = 5)`
4. `a += (x + 5)`
5. `a != (c *= (-d))`
6. `a *= c + (x - d)`
7. `a %= d++`
8. `a %= ++d`
9. `x++ * (a + c)`
10. `a = x * (b < c) + y * !(b < c)`
11. `!(x - d + c) || d`
12. `a && b || !0 && c && !d`
13. `((a && b) || (!0 && c)) && !d`
14. `((a && b) || !0) && (c && (!d))`

1.4 Les Types

1.4.1 Déclaration

▷ **Exercice 1** Quel type de variable utiliseriez-vous pour stocker :

- le nombre d'étudiant en premier Bloc ?
- le PIB (Produit Intérieur Brut) [5] de l'état Belge en euros ?
- π ?
- les votes exprimés pour un candidat à une élection ?
- le nombre de fûts dans les caves de votre cercle préféré ?

1.4.2 Conversion de Type

▷ **Exercice 1** Soit les déclarations suivantes :

```
double money = 210.7;
int euro = 25;
double d;
int i;
```

Décrivez avec précision l'effet de chacune des instructions suivantes :

```
d = euro;
d = (double) euro;
i = money;
i = (int) money;
```

▷ **Exercice 2** Comparez l'effet des instructions suivantes sur la valeur de la variable `franc` :

```
int franc;
franc = 125 / 50;
franc = 125 / 50.0;
franc = 125.0 / 50.0;
```

1.5 Bloc d'Instructions

▷ **Exercice 1** Soit le code suivant :

```
int main(){
  int a;
  int b;
  int c;
  a = 1;
  b = a + 1;
  c = a + b;
  a = b;
}//fin programme
```

Pour chaque ligne de code, indiquez quelles sont les variables qui sont modifiées et les valeurs qu'elles prennent.

5. Le PIB vise à quantifier (pour un pays et une année donnés) la valeur totale de la « production de richesse » effectuée par les agents économiques (ménages, entreprises, administrations publiques) résidants à l'intérieur de ce territoire. Pour plus d'infos, cf. http://fr.wikipedia.org/wiki/Produit_interieur_brut

1.6 Écriture de Code

▷ **Exercice 1** Lors d'une opération de promotion, un magasin de composants *hardware* applique une réduction de 10% sur tous les composants. Ecrivez un programme qui lit le prix d'un composant au clavier et affiche le prix calculé en tenant compte de la réduction.

▷ **Exercice 2** Une bille de plomb est lâchée du haut d'un immeuble et tombe en chute libre. Au bout d'un temps t (exprimé en secondes), la bille est descendue d'une hauteur (en mètres) :

$$h = \frac{1}{2} g \times t^2. \tag{1.1}$$

avec $g = 9.81$ (exprimé en $m.s^{-2}$).

Dans un premier temps, écrivez un programme qui calcule la hauteur descendue au bout d'un temps t saisi au clavier.

Dans un second temps, écrivez un programme qui calcule la durée totale de la chute connaissant la hauteur totale h de l'immeuble saisi au clavier. On pourra, ici, utiliser la bibliothèque `math.h` qui contient un module, `sqrt`, permettant de calculer la racine carrée d'un nombre. L'utilisation de `sqrt` se fait comme suit (par exemple) :

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

int main(){
  int x = 16;
  double racine;

  racine = sqrt(x);
  printf("La racine carree de %d est %f\n", x, racine);
}//fin programme
```

▷ **Exercice 3** Sachant que le premier avril 2004 était un jeudi, écrivez un programme qui détermine le jour de la semaine correspondant au 4 mai de la même année. On pourra représenter les jours de la semaine par des nombres allant de 0 à 6.

▷ **Exercice 4** Écrivez un programme qui affiche le chiffre des dizaines d'un nombre saisi au clavier. Même question pour les centaines.

▷ **Exercice 5** Écrivez un programme qui lit un nombre r au clavier et calcule le périmètre et l'aire d'un disque de rayon r.

Chapitre 2

Structures de Contrôle

L'objectif de ce chapitre est de se concentrer sur les structures de contrôle. Ces structures nous permettent, soit d'effectuer des tests sur les valeurs des variables (*condition* – Sec. 2.1), soit de répéter un certain nombre de fois un bloc d'instructions particulier (*itération* – Sec. 2.2).

2.1 Condition

2.1.1 Lecture de Code

L'objectif de cette section est de vous apprendre à lire et comprendre du code écrit par un autre programmeur. Soyez clair et précis dans vos réponses. Inutile d'exprimer les choses en paraphrasant le code C (i.e., *Si la variable* `i` *est plus petite que ...*), ce qui correspondrait à une traduction *littérale* du code. Pensez plutôt en terme de *transcodage* (ou traduction *littéraire*) en indiquant l'objectif final du bout de code.

La meilleure façon de résoudre les exercices qui suivent est de travailler avec des valeurs pour chacune des variables, d'exécuter le code avec ces valeurs et, ensuite, essayer d'inférer une relation entre les différentes variables.

▷ **Exercice 1** Soit le morceau de code suivant :

```
1 if(i < j)
2   k = j;
3 else
4   k = i;
```

Que calcule ce code (i.e., que contient la variable `k` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez).

▷ **Exercice 2** Soit le morceau de code suivant :

```
1 if(i > j)
2   k = i;
3 else
4   k = j;
```

Que calcule ce code (i.e., que contient la variable `k` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez).

▷ **Exercice 3** Soit le morceau de code suivant :

```
1 int k;
2 if(i != j){
3   k = i;
4   i = j;
5   j = k;
```

```
}
```

Que calcule ce code (i.e., que contiennent les variables `k`, `i` et `j` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez). En particulier, quelle est l'utilité de la variable `k` ?

Pour répondre à cette question, en plus de tester différentes valeurs pour `i` et `j`, il est recommandé de faire un schéma représentant les variables et leur contenu à chaque étape du programme.

▷ **Exercice 4** Soit le morceau de code suivant :

```
int l;
if(i > j){
  l = i;
  i = j;
  j = l;
}
if(i > k){
  l = i;
  i = k;
  k = l;
}
if(j > k){
  l = j;
  j = k;
  k = l;
}
```

Que calcule ce code (i.e., que contiennent les variables `k`, `i` et `j` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez). En particulier, quelle est l'utilité de la variable `k` ?

Pour répondre à cette question, en plus de tester différentes valeurs pour `i` et `j`, il est recommandé de faire un schéma représentant les variables et leur contenu à chaque étape du programme.

2.1.2 Écriture de Code

▷ **Exercice 1** Une entreprise X vend deux types de produits. Les produits de type A qui donnent lieu à une TVA à 5,5% [1] et les produits de type B qui donnent lieu à une TVA à 19,6%. Écrivez un programme qui lit au clavier le prix hors taxe d'un produit, saisit au clavier le type du produit et affiche le taux de TVA et le prix TTC (i.e., Toutes Taxes Comprises) du produit. Appliquer une taxe de X% revient à additionner, au prix du produit, X% du prix de ce produit. En d'autres termes,

$$\text{prix}_{\text{TTC}} = \text{prix} + \text{prix} \times \text{taux\_TVA}$$

▷ **Exercice 2** Écrivez un programme qui demande à l'utilisateur de saisir un entier relatif x ($x \in \mathbb{Z}$) et qui indique, d'une part, si ce nombre est positif ou négatif et, d'autre part, si ce nombre est pair ou impair.

▷ **Exercice 3** Écrivez un programme qui demande à l'utilisateur de saisir trois nombres réels (`double`) au clavier et les affiche à l'écran par ordre croissant.

▷ **Exercice 4** Écrivez un programme qui résout l'équation $ax + b = 0$ (a et b sont entrés au clavier par l'utilisateur). Bien évidemment, on n'oubliera pas tous les cas particuliers (notamment les cas « tout x est solution » et « pas de solution »).

1. $5.5\% = 0.055$

▷ **Exercice 5** Écrivez un programme qui résout l'équation $ax^2 + bx + c = 0$ (a, b, c sont entrés au clavier par l'utilisateur) en envisageant tous les cas particuliers.

▷ **Exercice 6** Écrivez un programme qui lit un entier au clavier représentant le numéro du mois et qui affiche, ensuite, à l'écran le mois en toutes lettres (janvier=1, février=2, ...). Pensez à utiliser l'instruction conditionnelle multiple (i.e., `switch`).

▷ **Exercice 7** Pour rappel, un nombre naturel correspond à une année bissextile s'il est multiple de 4 ; mais, parmi les multiples de 100, seuls les multiples de 400 correspondent à une année bissextile.

Écrivez un programme qui détermine si un nombre naturel entré par l'utilisateur correspond à une année bissextile ou non.

▷ **Exercice 8** Écrivez un programme qui, étant donné une heure représentée sous la forme de trois variables pour les heures, `h`, minutes, `m`, et secondes, `s`, affiche l'heure qu'il sera une seconde plus tard. Il faudra envisager tous les cas possibles pour le changement d'heure. Deux exemples de sortie (à l'écran) sont :

```
L heure actuelle est 23h12m12s
Dans une seconde il sera exactement: 23h12m13s

L heure actuelle est 23h59m59s
Dans une seconde il sera exactement: 00h00m00s
```

2.2 Itération

2.2.1 Lecture de Code

A l'instar de la Sec. 2.1.1, l'objectif de cette section est de vous apprendre à lire et comprendre du code écrit par un autre programmeur. Soyez clair et précis dans vos réponses. Inutile d'exprimer les choses en paraphrasant le code C (i.e., *la boucle s'exécute tant que la variable* `i` *est plus petite que ...*)), ce qui correspondrait à une traduction *littérale* du code. Pensez plutôt en terme de *transcodage* (ou traduction *littéraire*) en indiquant l'objectif final du bout de code.

La meilleure façon de résoudre les exercices qui suivent est de travailler avec des valeurs pour chacune des variables, exécuter le code avec ces valeurs et, ensuite, essayer d'inférer une relation entre les différentes variables.

▷ **Exercice 1** Soit le morceau de code suivant :

```
1 i=1;
2 k=0;
3 while(i < j){
4   k = k + i;
5   i = i + 1;
6 }//fin while
```

Que calcule ce code (i.e., que contient la variable `k` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez).

▷ **Exercice 2** Soit le morceau de code suivant :

```
1 i=1;
2 k=1;
3 while(i < j){
4   k = k * i;
5   i = i + 1;
6 }//fin while
```

Que calcule ce code (i.e., que contient la variable `k` à la fin de l'exécution de ce code) ? Soyez le plus précis possible (notamment dans le vocabulaire que vous utilisez).

2.2.2 Manipulation de Boucles

▷ **Exercice 1** Soit le code suivant :

```c
#include <stdio.h>

int main(){
  int n = 5;
  int f = 1;
  int c = n;

  while(c > 1){
    f = f * c;
    c = c - 1;
  }//fin while

  printf("%d\n", f);
}//fin programme
```

- Que calcule ce code (i.e., que vaut `f` à la fin) ?
- Réécrivez ce code avec un compteur qui est incrémenté d'une unité à chaque itération plutôt que d'être décrémenté.

▷ **Exercice 2** Soit le code suivant :

```c
#include <stdio.h>

int main(){
  int i;

  for(......; ......; ......)
    printf("It's␣a␣long␣way␣to␣the␣top␣if␣you␣wanna␣Rock␣'n'␣Roll!\n");
}//fin programme
```

Complétez ce bout de code pour que le programme affiche **cinq** fois à l'écran la phrase "It's a long way to the top if you wanna Rock 'n' Roll !".

▷ **Exercice 3** Réécrivez le code de l'Exercice 2 en utilisant, cette fois, une boucle `while` plutôt que `for`.

▷ **Exercice 4** Réécrivez le code de l'Exercice 2 en utilisant, cette fois, une boucle `do ... while` plutôt que `for`.

▷ **Exercice 5** Réécrivez la boucle de l'Exercice 1 en utilisant l'instruction `for` plutôt que `while`.

▷ **Exercice 6** Réécrivez la boucle de l'Exercice 1 en utilisant l'instruction `do ... while` plutôt que `while`.

2.2.3 Écriture de Code

Pour chacun de ces exercices, vous veillerez à bien suivre la méthodologie vue au cours pour la construction d'une boucle. **En particulier, il vous est demandé de stipuler explicitement les quatre étapes de rédaction d'une boucle avant de rédiger la moindre ligne de code.** Faites valider les quatre étapes par un membre de l'équipe pédagogique avant de commencer la rédaction de votre code C. Pour rappel, les 4 étapes sont décrites dans le Chapitre 2, Slide 34.

▷ **Exercice 1** Écrivez un programme qui affiche, à l'écran, la table de multiplication pour un entier lu au clavier.

▷ **Exercice 2** Écrivez un programme qui calcule la somme :

$$s = 1 + 2^3 + 3^3 + \ldots + n^3. \tag{2.1}$$

en fonction de n saisi au clavier.

▷ **Exercice 3** Considérons que a, b et c sont des variables entières lues au clavier. Écrivez un programme permettant de calculer :

- la somme des entiers dans l'intervalle $[a, b[$
- la somme des entiers positifs dans l'intervalle $]a, b[$
- la somme des entiers pairs strictement inférieur à b
- la somme des diviseurs de c dans l'intervalle $[a, b]$
- le nombre d'entiers qui sont strictement inférieurs à c et à la fois multiple de a et b.

▷ **Exercice 4** Écrivez un programme qui calcule la factorielle $n!$ d'un entier n saisi au clavier.

▷ **Exercice 5** Écrivez un programme qui calcule le nombre de chiffres en base 10 d'un nombre n saisi au clavier.

▷ **Exercice 6** Écrivez un programme qui affiche la plus grande puissance de 2 inférieure à la constante $C = 2.426.555.645$.

▷ **Exercice 7** Écrivez un programme qui permet facilement d'afficher à l'écran un rectangle comme le suivant :

```
+-----------------------+
|                       |
|                       |
|                       |
+-----------------------+
```

La largeur et la longueur de ce rectangle seront saisies au clavier.

▷ **Exercice 8** En mathématiques, l'intégrale permet, entre autres, de calculer la surface de l'espace délimité par la représentation graphique d'une fonction. L'intégrale de la fonction $f(x)$ sur l'intervalle $[a, b]$ est représentée par

$$\int_a^b f(x)dx.$$

Il est possible d'approximer l'intégrale d'une fonction à l'aide de la méthode des *rectangles*. Cette méthode fonctionne comme suit : l'intervalle $[a, b]$ sur lequel la fonction $f(x)$ doit être intégrée est divisé en N sous-intervalles égaux de longueurs $h = \frac{(b-a)}{N}$. Les rectangles sont alors dessinés de sorte que le coin supérieur gauche, droit ou l'entièreté du côté supérieur touche le graphe de la fonction, tandis que la base se tient sur l'axe des X. L'approximation de l'intégrale est alors calucée en ajoutant les surfaces (base multipliée par hauteur) des rectangles. Ce qui donne la formule :

$$I = h \times \sum_{i=0}^{N-1} f(a + i \times h).$$

Écrivez un programme qui permette d'approximer, par la méthode des rectangles, l'intégrale de $cos(x)$. Les valeurs de N, a, b et x seront lues au clavier. Le module `cos()` se trouve dans la librairie `math.h`.

Chapitre 3

Méthodologie de Développement

La matière vue dans les deux premiers chapitres du cours vous permet, déjà, de résoudre pas mal de problèmes. L'objectif de ce chapitre est donc d'augmenter vos compétences en algorithmique et en écriture de code. En particulier, nous allons nous attarder sur la façon dont on peut construire efficacement et correctement une solution à un problème. Tout d'abord, les exercices porteront sur la notion de sous-problèmes (Sec. 3.1), soit comment découper un problème complexe en plusieurs sous-problèmes plus simples que l'on peut résoudre indépendamment les uns et des autres et, ensuite, les enchaîner pour résoudre le problème principal. Ensuite, nous aborderons la notion de construction d'une boucle (i.e., instruction de répétition) en appliquant la méthode de construction par *Invariant* (Sec. 3.2). Pour rappel, l'existence d'un Invariant permet de garantir qu'une boucle est correcte à condition qu'elle se termine. Nous allons donc, naturellement, ensuite nous focaliser sur la terminaison des programmes (Sec. 3.3).

3.1 Sous-Problèmes

▷ **Exercice 1** Soit le code suivant :

```c
#include <stdio.h>

int main(){
  int i, j;
  printf("2␣");

  for(i = 3; i < 10000; i += 2){
    for(j=3; (j * j <= i) && (i % j); j += 2);

    if(j * j>i)
      printf("%d␣", i);
  }//fin for - i

  printf("\n");
}//fin programme
```

Quel est l'objectif de ce programme ? A l'instar des exercices de la Sec. 2.1.1 et de la Sec. 2.2.1, il vous demandé de fournir une réponse pertinente autre que *Ce code affiche des chiffres sur la sortie standard*. Travaillez sur des valeurs et exécutez le code avec ces valeurs. Ensuite, essayez de préciser quelle propriété mathématique partagent tous les nombres affichés sur la sortie standard.

De plus, expliquez le rôle de chaque boucle en indiquant les sous-problèmes.

▷ **Exercice 2** Nous sommes dans une école après la période d'examens. On désire encoder les notes pour chaque élève de la classe. La classe compte 35 élèves, chacun d'entre eux ayant présenté 10 examens. La note de chaque examen est comprise entre 0 et 20.

L'utilisateur du programme saisit au clavier, pour chaque étudiant, les notes des examens. Après l'encodage des notes, le programme affiche le pourcentage de l'élève le plus performant.

Définissez le problème (Input, Output, Objets Utilisés), identifiez et expliquez les éventuels sous-problèmes et indiquez comment ils s'emboîtent. Ensuite, écrivez le code C de votre programme. Pensez à indiquer, dans votre programme, où se trouvent les différents sous-problèmes.

▷ **Exercice 3** Dans cet exercice, nous illustrons une banque de dépôt qui permet à un client de déposer une fois par an (en début d'année) une certaine somme d'argent (exprimée en €). La banque applique, sur les dépôts, un certain taux d'intérêt x (exprimé en pourcentage). À la fin d'une année, le client qui a fait un dépôt touche $x\%$ d'intérêt qu'il remet immédiatement sur son compte. Il est demandé, dans cet exercice, d'indiquer au client (i.e., affichage à l'écran) combien d'années au minimum il devra attendre avant d'atteindre un certain montant (indiqué par lui-même).

Pour illustrer l'exercice, prenons un exemple où le client dépose 100€ et l'intérêt fournit par la banque est de 10%. Après un an, le client disposera de 100€ (montant initial) + 10€ (les intérêts) sur son compte. Après deux ans, le client dipose de 110€ (le résultat de la 1ère année) + 100€ (le dépôt de la deuxième année) + 21€ (les intérêts de la deuxième année). Pour atteindre la somme de 1000€, le client devra attendre 7 ans.

Le montant déposé sur le compte en banque, le taux d'intérêt et le montant cible sont introduits au clavier.

Définissez le problème (Input, Output, Objets Utilisés), identifiez et expliquez les éventuels sous-problèmes et indiquez comment ils s'emboîtent. Ensuite, écrivez le code C de votre programme. Pensez à indiquer, dans votre programme, où se trouvent les différents sous-problèmes.

3.2 Invariant Graphique

Cette section nous permet de travailler la notion de Invariant Graphique. C'est la notion essentielle du cours. L'Invariant Graphique sera au coeur du cours INFO0947. Il est donc crucial de maîtriser cette notion le plus rapidement possible.

Pour les exerices de cette section, n'oubliez pas que vous pouvez vous appuyez sur l'outil Gli afin de dessiner et valider vos Invariants Graphiques.

Pour faciliter votre travail, nous vous recommandons d'avoir, à côté de vous, le Vademecum de l'Invariant Graphique. Rendez-vous au Chapitre 3, Slide 100, pour consulter le Vademecum.

3.2.1 Compléter un Invariant Graphique

Dans les exercices qui suivent, nous vous fournissions des énoncés de problèmes et les Invariants Graphiques associés. Les Invariants Graphiques, à chaque fois, sont **incomplets**. Il vous est demandé de les compléter en fonction du Vademecum (cfr. Chapitre 3, Slide 100).

▷ **Exercice 1** L'utilisateur entre un nombre entier positif au clavier et le programme, à la fin, affiche la somme des entiers positifs jusqu'à ce nombre. Complétez la Fig. 3.1 afin de produire un Invariant Graphique permettant de résoudre ce problème.

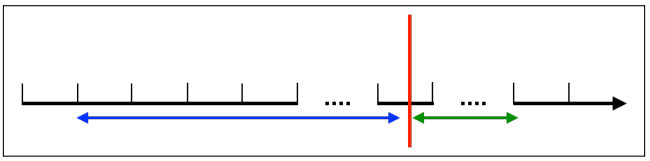

Figure 3.1 – Invariant Graphique partiel pour le calcul de la somme des entiers positifs.

▷ **Exercice 2** L'utilisateur souhaite afficher sur la sortie standard une ligne composée de n caractères. Le caractère à afficher et la longueur de la ligne sont lues au clavier. Complétez la Fig. 3.2 afin de produire un Invariant Graphique permettant de résoudre ce problème.

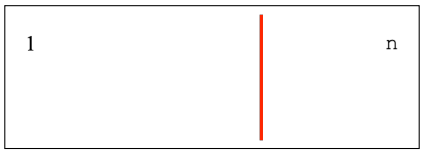

FIGURE 3.2 – Invariant Graphique partiel pour l'affichage d'une ligne de n caractères.

3.2.2 Invariant Graphique et ZONE 1

Dans les exercices qui suivent, nous vous fournissions des énoncés de problèmes et les Invariants Graphiques associés. Les Invariants Graphiques, à chaque fois, sont **complets**. Il vous est demandé de dériver, des Invariants Graphiques, les variables dont on a besoin, leurs types, ainsi que les valeurs d'intialisation (i.e., ZONE 1). Vous veillerez à justifier proprement vos choix sur base des Invariants Graphiques.

▷ **Exercice 1** L'utilisateur demande d'afficher à l'écran le résultat d'un exposant négatif[1], la base et l'exposant étant lu au clavier. L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.3.

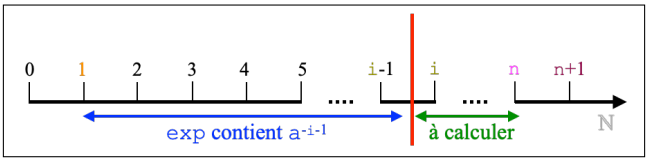

FIGURE 3.3 – Invariant Graphique pour le calcul d'un exposant négatif.

Indiquez les variables et leurs types ainsi que les valeurs d'intialisation (ZONE 1). Justifiez sur base de l'Invariant Graphique. Il n'est pas demandé d'écrire le code C des ZONES 2 et 3.

Rappel préliminaire sur les puissances :

$$a^{-n} = \left(\frac{1}{a}\right)^n$$

Par exemples :

$$3^{-2} = \left(\frac{1}{3}\right)^2$$
$$\left(\frac{2}{3}\right)^{-4} = \left(\frac{3}{2}\right)^4$$

▷ **Exercice 2** L'utilisateur désire calculer la somme des entiers dans l'intervalle $[a, b[$, avec a et b lus au clavier. L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.4.

Indiquez les variables et leurs types ainsi que les valeurs d'intialisation (ZONE 1). Justifiez sur base de l'Invariant Graphique. Il n'est pas demandé d'écrire le code C des ZONES 2 et 3.

1. Pour rappel, $3^{-5} = \left(\frac{1}{3}\right)^5$.

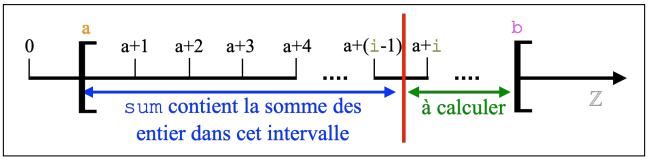

FIGURE 3.4 – Invariant Graphique pour le calcul de la somme des entiers dans un intervalle.

3.2.3 Invariant Graphique, ZONE 1 et Critère d'Arrêt

Dans les exercices qui suivent, nous vous fournissions des énoncés de problèmes et les Invariants Graphiques associés. Les Invariants Graphiques, à chaque fois, sont **complets**. Il vous est demandé de dériver, des Invariants Graphiques, les variables dont on a besoin, leurs types, les valeurs d'intialisation (i.e., ZONE *1*) ainsi que le Critère d'Arrêt de boucle. Vous penserez aussi à dériver du critère d'arrêt le Gardien de Boucle. Vous veillerez à justifier proprement vos choix sur base des Invariants Graphiques.

▷ **Exercice 1** L'utilisateur désire afficher à l'écran la suite de caractères suivante :

```
+----------------------+
```

Il s'agit donc d'une suite de `'-'` entourée (à gauche et à droite) par le caractère `'+'`. La longueur de la ligne, n, correspond au nombre de `'-'` sur la ligne. n est introduit au clavier par l'utilisateur. L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.5.

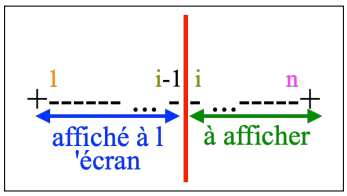

FIGURE 3.5 – Invariant Graphique pour l'affichage d'une ligne faite de `'-'` et `'+'`.

Indiquez les variables et leurs types ainsi que les valeurs d'intialisation (ZONE 1). Dérivez aussi le Critère d'Arrêt de la boucle ainsi que le Gardien de Boucle. Justifiez sur base de l'Invariant Graphique. Il n'est pas demandé d'écrire le code C des ZONES 2 et 3.

▷ **Exercice 2** La banque "MegaPognon" accepte de vous accorder un prêt si la somme de vos intérêts dépasse 1000€. Le taux d'intérêt est de 3.5% par an. Il vous est demandé d'écrire un programme permettant de déterminer, sur base d'un montant placé initialement, le nombre d'années à attendre pour pouvoir bénéficier d'un prêt. Voici un exemple d'output de votre programme :

```
Somme initiale placée: 2000 euros
1ère année: intérêt = (2000 * 3.5)/100 = 70
2ème année: intérêt = (2070 * 3.5)/100 = 72.45
...
```

L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.6.

Indiquez les variables et leurs types ainsi que les valeurs d'intialisation (ZONE 1). Dérivez aussi le Critère d'Arrêt de la boucle ainsi que le Gardien de Boucle. Justifiez sur base de l'Invariant Graphique. Il n'est pas demandé d'écrire le code C des ZONES 2 et 3.

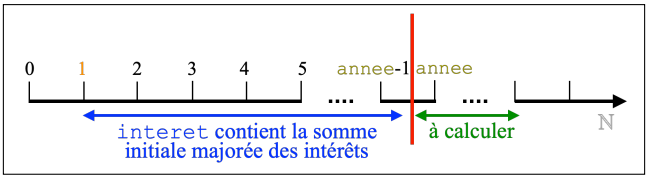

FIGURE 3.6 – Invariant Graphique pour le calcul d'intérêts.

3.2.4 Invariant Graphique et Zone 2

Dans les exercices qui suivent, nous vous fournissions des énoncés de problèmes, les Invariants Graphiques associés et une partie du code (à savoir la ZONE 1 et la Gardien de Boucle). Les Invariants Graphiques, à chaque fois, sont **complets**. Il vous est demandé de dériver, des Invariants Graphiques, la ZONE 2. Vous veillerez à justifier proprement vos choix sur base des Invariants Graphiques.

▷ **Exercice 1** L'utilisateur lit un entier positif au clavier, n, et désire afficher à l'écran le nombre de 0 qui compose n dans sa représentation binaire. L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.7.

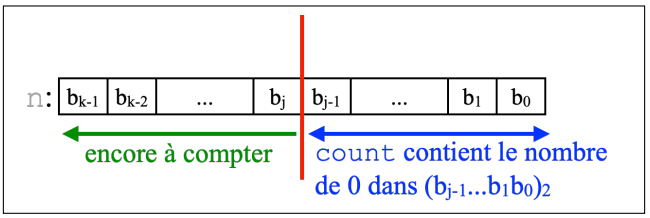

FIGURE 3.7 – Invariant Graphique pour le calcul du nombre de 0 dans la représentation binaire de n.

Complétez l'extrait de code 3.1 en remplissant le corps de la boucle (i.e., ZONE 2). Justifiez sur base de l'Invariant Graphique.

Extrait de code 3.1 – Code à compléer pour le calcul du nombre de 0 dans la représentation binaire de n.

```
#include <stdio.h>

int main(){
  unsigned int n, count;

  scanf("%u", &n);

  while(n > 0){
    //ZONE 2 -- VOTRE CODE ICI
  }//fin while

  printf("Nombre␣de␣0␣dans␣%u:␣%u\n", n, count);
}//fin programme
```

▷ **Exercice 2** L'utilisateur entre un nombre entier positif, n, au clavier et désire obtenir, à l'écran, la liste des facteurs de n.

Rappel préliminaire sur les facteurs.
Le *facteur* de tout nombre est un nombre entier qui divise exactement le nombre en un nombre entier sans laisser de reste. Par exemple, 2 est un facteur de 6 car 2 divise 6 exactement sans laisser de reste.

Voici un exemple d'éxécution du programme :

```
saisir un nombre: 20
les facteurs de 20 sont: 1 2 4 5 10 20
```

L'Invariant Graphique pour résoudre ce problème est donné à la Fig. 3.8.

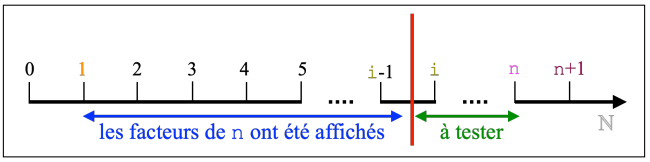

FIGURE 3.8 – Invariant Graphique pour le calcul des facteurs de n.

Complétez l'extrait de code 3.1 en remplissant le corps de la boucle (i.e., ZONE 2). Justifiez sur base de l'Invariant Graphique.

Extrait de code 3.2 – Code à compléer pour le calcul des facteurs de n.

```c
#include <stdio.h>

int main(){
  unsigned int n, i, facteur;

  printf("saisir un nombre");
  scanf("%u", &n);

  i = 1;
  printf("les facteurs de %u sont: ", n);

  while(i <= n){
    //ZONE 2 -- VOTRE CODE ICI
  }//fin while

  printf("\n");
}//fin programme
```

3.2.5 Construction par Invariant

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur l'entièreté de la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. Définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. Analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques. Pensez à suivre le vadémécum fourni lors du cours théorique (Chapitre 3, Slide 100). N'hésitez pas à vous appuyer sur le GLI pour la construction et la validation automatique de vos Invariants Graphiques.
4. Code C.

Attention, pour le code, il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., ZONE 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt. Jusitifiez sur base de l'Invariant Graphique.

Faites valider chacune de ces étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** Soient a et b, deux variables entières lues au clavier. Écrivez un programme permettant de calculer la somme des entiers pairs strictement inférieurs à b.

▷ **Exercice 2** En physique, on calcule la position s d'un mobile en mouvement au temps t par la formule $s = s_0 + v_0 \times t + 0.5 \times a \times t^2$, où s_0 est la position initiale, v_0 la vitesse initiale et a l'accélération (supposée constante). Écrivez un programme C qui affiche les positions du mobile au temps $t = 0, 5, 10, 15, 20, \ldots$ (avec t entré au clavier par l'utilisateur).

▷ **Exercice 3** La suite de Fibonacci est définie comme suit :

$$f(n) = \begin{cases} f(n-1) + f(n-2), si\ n > 1 \\ 1, si\ 0 \leq n \leq 1 \end{cases}$$

Écrivez un programme C permettant de calculer la suite de Fibonacci (la valeur de n est entrée au clavier par l'utilisateur).

▷ **Exercice 4** Le produit de deux naturels x et y consiste à sommer y fois la valeur de x.

$$x \times y = \underbrace{x + x + \ldots + x}_{y\ fois}.$$

Toutefois, on peut améliorer cet algorithme rudimentaire en multipliant le produit courant par deux et en divisant le nombre de termes restants à sommer par deux à chaque fois que ce nombre est pair. Les opérations de multiplication et division par deux sont des opérations très efficaces puisqu'elles consistent à décaler de un bit vers la gauche ou vers la droite.

Écrivez un programme C qui lit au clavier deux naturels, x et y, et qui calcule le produit $x \times y$ en suivant l'algorithme indiqué supra.

▷ **Exercice 5** Il est possible de calculer la racine carrée d'un nombre en utilisant un algorithme très particulier et sans avoir recours à la fonction mathématique racine carrée (prédéfinie, en C, dans `math.h`). Soit n, le nombre dont on souhaite extraire la racine carrée. On construit une suite de nombres X_i dont le premier terme vaut 1 et dont le terme général a pour expression :

$$X_i = \frac{\frac{n}{X_{i-1}} + X_{i-1}}{2}.$$

Aussi surprenant que cela puisse paraître, cette suite converge systématiquement vers $\sqrt{n}$.

Écrivez un programme permettant de calculer la racine carrée d'un nombre n (entré par l'utilisateur au clavier) en suivant la formule donnée supra (le nombre total d'itérations est donné, aussi, par l'utilisateur). Proposez un invariant avant d'écrire votre code. Attention, ne négligez pas le risque d'erreur (e.g., division par 0) dans votre code.

▷ **Exercice 6** Écrivez un programme affichant un nombre naturel (entier non signé) entré par l'utilisateur comme la somme de puissances de 2 correspondant à sa décomposition en base 2. Par exemple, pour le nombre 50, le programme affichera `50 = 2 + 16 + 32`.

▷ **Exercice 7** Écrivez un programme affichant la représentation binaire d'un nombre entier (signé) entré par l'utilisateur.

▷ **Exercice 8** Écrivez un programme qui calcule (à un centième près) au moyen d'une recherche dichotomique la racine carrée d'un nombre réel entré par l'utilisateur.

Pour rappel, une recherche dichotomique consiste à trouver un intervalle initial contenant la valeur cherchée, puis à ne garder de proche en proche que la moitié d'intervalle dans laquelle se trouve la valeur.

3.2.6 Code Mystère

▷ **Exercice 1** Soit la démarche méthodologique suivante :

Définition du Problème :

1. Input : n, un entier positif lu au clavier
2. Output : k, le logarithme binaire de n est affiché à l'écran [2]
3. Objets Utilisés :

   `n`, un entier positif (`unsigned int n;`)

Analyse du Problème :

SP1 : Lecture au clavier de `n`

SP2 : Calcul du logarithme binaire de `n`

SP3 : Affichage à l'écran du logarithme binaire de `n`

Enchaînement des SPs :
SP1 → SP2 → SP3

Invariant :
$k \geq n,\ p = 2^k$

Pouvez-vous écrire le programme correspondant ?

3.3 Fonction de Terminaison

▷ **Exercice 1** Pour déterminer une Fonction de Terminaison, nous avons vu deux techniques au cours (Chap. 3, Slides 105 → 111) : un raisonnement graphique basé sur l'Invariant Graphique et un raisonnement sur base du Gardien de Boucle. Dans cet exercice, nous vous demandons d'appliquer exclusivement la technique basée sur l'Invariant Graphique (Chap. 3, Slide 105 et 109).

Repartez des Invariants Graphiques suivants et déterminez, pour chacun d'eux, la Fonction de Terminaison :

1. Calcul d'un exposant négatif, Fig. 3.3.
2. Calcul de la somme des entiers dans un intervalle, Fig. 3.4.
3. Affichage d'une ligne faite de `'-'` et `'+'`, Fig. 3.5.
4. Calcul d'intérêts, Fig. 3.6.
5. Comptage du nombre de 0 dans la représentation binaire d'un nombre, Fig. 3.7.
6. Calcul des facteurs de n, Fig. 3.8.

Justifiez toujours sur base de l'Invariant Graphique.

▷ **Exercice 2** Soit le code C suivant :

```c
#include <stdio.h>

int main(){
  int a, b;

  scanf("%d", &b);

  a = b;
  while(a != 0){
```

2. On appelle *logarithme binaire entier* d'un entier strictement positif n le nombre entier k tel que $2^k \leq n < 2^{k+1}$. Par défaut, le logarithme binaire de 0 est -1.

```
    if(a % 2)
      a = 2 * a;
    else
      a--;
  }//fin while - a

  printf("%d\n", a);
}//fin programme
```

Est-ce que ce programme s'arrête ? Si oui, donnez une Fonction de Terminaison. Sinon, justifiez.

▷ **Exercice 3** Soit le code C suivant :

```
#include <stdio.h>

int main(){
  int n, f;

  do{
    scanf("%d", &n);
  }while(n < 0);

  f = 7 * 6 * 5 * 4 * 3 * 2 * 1;
  while(n != 7){
    if(n > 7){
      f = f * n;
      n--;
    }else{
      n++;
      f = f / n;
    }
  }//fin while

  printf("%d\n", f);
}//fin programme
```

Est-ce que ce programme s'arrête (justifiez) ? Si oui, que calcule ce programme ?

▷ **Exercice 4** L'algorithme d'Euclide pour le calcul du PGCD est donné par le code suivant :

```
#include <stdio.h>

int main(){
  int a, b;
  scanf("%d␣%d", &a, &b);

  if(!a)
    printf("%d\n", b);
  else{
    while(b){
      if(a > b)
        a -= b;
      else
        b -= a;
    }//fin while

    printf("%d\n", a);
  }
}//fin programme
```

Prouvez que la boucle se termine (i.e., donnez une Fonction de Terminaison) quels que soient les naturels a et b fournis en entrée.

Pour rappel, si deux nombres sont divisibles par un autre, leur différence l'est aussi.

Chapitre 4

Introduction à la Complexité

Ce chapitre aborde un sujet relatif à l'informatique théorique. L'idée est d'introduire les notions permettant d'évaluer, de manière générale, les performances d'un programme et de comparer différentes solutions entre elles afin de s'orienter vers la plus efficace. La notion derrière tout cela est la *complexité*. Dans le cadre du cours, nous nous intéressons à la *complexité temporelle* dans le *pire des cas*. Ceci nous est indiqué par une notation spéciale, le "O" (ou notation de *Landau*). La Sec. 4.1 nous permet d'apprendre à manipuler cette notation tandis que la Sec. 4.2 nous permet d'apprendre à analyser la complexité d'un programme.

4.1 Notation "O"

4.1.1 Grandeurs

▷ **Exercice 1**

1. Comparez 10^{100} et 100^{10}.
2. Comparez $10^{(10)^{10}}$ et $(10^{10})^{10}$.

▷ **Exercice 2** Rangez les fonctions suivantes par ordre de grandeur croissant :

$$n,\ n \times \log(n),\ \sqrt{n},\ \log(n),\ \sqrt{n} \times \log^2(n)$$
$$n^2,\ (3/2)^n,\ n^{10},\ \log^2(n),\ e^{n^2},\ 1/3^n$$

▷ **Exercice 3** Complétez les tableaux suivants.

Dans le Tableau 4.1.1, il s'agit de calculer le nombre d'opérations nécessaire pour exécuter le programme en fonction de sa complexité et de la taille d'instance du problème traité. Dans le Tableau 4.2, il s'agit de calculer le temps nécessaire à cela en supposant qu'on dispose d'un ordinateur capable de réaliser 10^9 opérations/seconde. Enfin, dans le Tableau 4.3, on calcule la plus grande instance du problème traitable dans le temps imparti.

Rappels : $1\mu\text{s} = 10^{-6}\text{s}$; $1\text{ ns} = 10^{-9}\text{s}$; 10^{x^y} et $\sqrt{n} = n^{1/2}$.

4.1.2 Comportements Asymptotiques

▷ **Exercice 1** Déterminez si les affirmations suivantes sont vraies ou fausses. Justifiez votre réponse.

1. $2 \times n + 3 \in O(n)$
2. $\log^{145} n \in O(\log(n))$
3. $2^n + \log(n) \in O(n^2)$
4. $2^n \in O(n^2 + 4 \times n^2 + \log(n))$

Complexité	**n=10**	**n=100**	**n=1000**
n			
n^2			
n^3			
2^n			
$\sqrt{n}$			
$\sqrt[3]{n}$			
$log(n)$			

TABLE 4.1 – Nombre d'opérations

Complexité	**n=10**	**n=100**	**n=1000**
n			
n^2			
n^3			
2^n			
$\sqrt{n}$			
$\sqrt{3}n$			
$log(n)$			

TABLE 4.2 – Temps nécessaire à 10^9 opérations/seconde

Complexité	**1 sec.**	**1 heure**	**1 an**
n			
n^2			
n^3			
2^n			
$\sqrt{n}$			
$\sqrt{3}n$			
$log(n)$			

TABLE 4.3 – Plus grande instance faisable à 10^9 opérations/seconde

5. $2 \times n^7 + 4 \times n^6 + 5 \times n^5 + 2 \times n^3 + 2 \times n \in O(n^7)$

▷ **Exercice 2** Quelle est la classe de complexité de la fonction suivante :

$$T(n) = 2 \times n^3 + 4 \times n^2 + 2^3$$

Démontrez.

▷ **Exercice 3** Montrez que les assertions suivantes sont exactes :

- $n \times (n-1)$ est en $O(n^2)$
- $\max(n^3, 10 \times n^2)$ est en $O(n^3)$

▷ **Exercice 4** Soient $f(x)$ et $g(x)$, des fonctions positives asymptotiquement. On suppose que $S(n) \in O(f(n))$ et $T(n) \in O(g(n))$. Montrez que

$$\text{si } f(n) \in O(g(n)), \text{ alors } S(n) + T(n) \in O(g(n)).$$

▷ **Exercice 5** Soient $f(x)$ et $g(x)$, des fonctions positives asymptotiquement. On suppose que $S(n) \in O(f(n))$ et $T(n) \in O(g(n))$. Montrez que

$$\text{si } f(n) \in O(g(n)), \text{ alors } S(n) \times T(n) \in O(f(n) \times g(n))$$

▷ **Exercice 6** Peut-on écrire :

1. $2^{n+1} \in O(2^n)$?
2. $2^{2^n} \in O(2^n)$?

▷ **Exercice 7** Soient $f(x)$ et $g(x)$, des fonctions asymptotiquement strictement positives. Peut-on affirmer que $f(n) \in O(g(n))$ implique que $(g(n) \in O(f(n)))$. Justifiez !

4.2 Analyse de Complexité

Pour chacun des exercices suivants, il vous est demandé d'effectuer une analyse complète de la complexité du code. Le raisonnement que vous devez donner doit être **formel** (i.e., mathématique). Chaque étape de votre raisonnement doit être clairement indiquée, en particulier quand il

s'agit d'indiquer le nombre de tours d'une boucle. Vous devez aussi indiquer, pour chaque étape, les **règles et propriétés** que vous utilisez. [1] N'hésitez pas vous aider des numéros de ligne pour chaque bout de code.

▷ **Exercice 1**

```c
#include <stdio.h>

int main(){
  int i, n;
  scanf("%d", &n);

  for(i=0; i<n; i++)
    printf("Bonjour!");
}//fin programme
```

▷ **Exercice 2**

```c
#include <stdio.h>

int main(){
  int a, b, r, i;

  scanf("%d%d", &a, &b);

  r = 1;
  for(i=0; i<b; i++)
    r = r*a;

  printf("%d\n", r);
}//fin programme
```

Que calcule ce programme ? Donnez la complexité du programme. Justifiez votre

▷ **Exercice 3**

```c
#include <stdio.h>

int main(){
  int i, j, n;

  scanf("%d", &n);

  for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
      printf("Salut!\n");
}//fin programme
```

▷ **Exercice 4**

```c
#include <stdio.h>

int main(){
  int i, n;

  scanf("%d", &n);

  i = 1;
  while(i<=n){
    printf("Salut!\n");
    i *= 2;
```

1. cfr. Cours Théorique, Chapitre 4, Slides 37 → 43.

```
  }//fin while
}//fin programme
```

▷ Exercice 5

```
#include <stdio.h>

int main(){
  int i, j, n;

  scanf("%d", &n);

  i = 1;
  while(i<=n){
   for(j=1; j<=i; j++)
      printf("Salut\n");

   i *= 2;
  }//fin while
}//fin programme
```

▷ Exercice 6

```
#include <stdio.h>

int main(){
  int i, j, n;

  scanf("%d", &n);

  for(j=1; j<=n; j++){
    i = 1;
    while(i<=j){
      printf("Salut\n");
      i *= 2;
    }//fin while
  }//fin for
}//fin programme
```

▷ Exercice 7

```
#include <stdio.h>

int main(){
  int a, b, p, x, i;

  scanf("%d%d", &a, &b);

  p = 1;
  x = a;
  i = b;

  while(i!=0){
    if(i%2){
      p = p*x;
      i--;
    }

    x = x*x;
    i = i/2;
  }//fin while - i

```

```
22    printf("%d\n", p);
23 }//fin programme
```

Chapitre 5

Structures de Données

Ce chapitre se concentre sur la notion de structures de données et les algorithmes qui tournent autour. Jusqu'à présent, les « objets » manipulés dans nos programmes étaient simples, e.g., `int`, `char`, ou encore `double`. Nous allons aborder maintenant des « objets » plus complexes que nous devons construire nous-même (i.e., ils ne sont pas prédéfinis par le langage). Tout d'abord, les exercices porteront sur une des structures de données les plus courantes, les tableaux unidimensionnels (Sec. 5.1). Nous passerons ensuite à des variations des tableaux unidimensionnels, soit les tableaux multidimensionnels (Sec. 5.2) et les chaînes de caractères (Sec. 5.3). Nous nous exercerons ensuite à manipuler des structures de données qui permettent d'avoir, sous un même nom, plusieurs variables de types différents, les enregistrements (Sec. 5.4). Nous verrons ensuite les énumérations et les fichiers (Sec. 5.6). Enfin, nous finirons ce chapitre avec un exercice de modélisation d'un problème via des structures de données (Sec. 5.7).

5.1 Tableaux Unidimensionnels

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [1]
4. Fonction de Terminaison
5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

1. cfr. Cours théorique, Chapitre 5, Slide 22.

5.1.1 Manipulations Simples

Soit le code suivant servant de base à plusieurs exercices :

Extrait de code 5.1 – Code de base pour les exercices de manipulation des vecteurs.

```c
#include <stdio.h>

int main(){
   //Constante qui va définir la taille du tableau
   const unsigned int N = ...;

   //tableau d'entiers que nous allons manipuler
   int tab[N];

   /*
    * Code de remplissage du tableau.
    *
    * Invariant:
    *
    *        |0          |i          N-1|N
    *        +-----------+-------------+
    * tab:   |           |             |
    *        +-----------+-------------+
    *         <---------> <------------>
    *           rempli        encore
    *          au clavier    à remplir
    *
    * Fonction de terminaison: N-i
    */
   unsigned int i = 0;
   while(i < N){
      printf("Introduisez une valeur: ");
      scanf("%d", &tab[i]);
      i++;
   }//fin while -- i

   //CODE À COMPLÉTER

}//fin programme
```

▷ **Exercice 1** Complétez le code de base (cfr. Listing 5.1) de façon à calculer et afficher la somme des éléments du tableau.

▷ **Exercice 2** Complétez le code de base (cfr. Listing 5.1) de façon à trouver le maximum et le minimum du tableau. Attention, vous ne pouvez utiliser qu'une seule boucle pour la recherche du maximum et du minimum.

▷ **Exercice 3** Écrivez un programme qui calcule le produit scalaire de deux vecteurs d'entiers U et V (de mêmes dimensions, `N`). Les valeurs contenues dans U et V ont déjà été introduites au clavier par l'utilisateur (code sensiblement identique au Listing 5.1).

Pour rappel, dans un repère orthonormé, le produit scalaire de deux vecteurs se calcule comme suit : Soient $u = (x, y)$ et $v = (x', y')$, le produit scalaire de u et v se calcule comme

$$\vec{u} \times \vec{v} = x \times x' + y \times y'.$$

▷ **Exercice 4** Calculez, pour une valeur X donnée (de type `float`), la valeur numérique d'un polynôme de degré n :

$$P(X) = A_n X^n + A_{n-1} X^{n-1} + \ldots + A_1 X^1 + A_0 X^0.$$

Les valeurs des coefficients $A_n, A_{n-1}, \ldots, A_1, A_0$ ont déjà été entrées au clavier et mémorisées dans un tableau `A` de type `float` et de dimension $n+1$ (code sensiblement identique au Listing 5.1).

Pour éviter les opération d'exponentiation, vous veillerez à utiliser le schéma de Horner :

$$\overbrace{\underbrace{\underbrace{A_n}_{\times X + A_{n-1}}}_{\cdots}}^{X + A_0}$$

.

Pour rendre les choses plus simples, appliquons le schéma de Horner sur l'exemple suivant. Soit la fonction polynôme P définie par

$$P(x) = 2x^3 - 7x^2 + 4x - 1.$$

On souhaite calculer $P(a)$ pour $a = 5$. Le calcul de $P(5)$ nécessite 6 multiplications et 3 « additions-soustractions ». Soit

$$P(5) = 2\times5\times5\times5 - 7\times5\times5 + 4\times5 - 1 = 94.$$

Si on utilise l'écriture suivante :

$$P(x) = ((2x - 7)x + 4)x - 1.$$

alors le calcul de l'image de 5 ne nécessite plus que 3 multiplications et 3 « additions-soustractions ». Soit

$$P(5) = ((2\times5 - 7)\times5 + 4)\times5 - 1.$$

▷ **Exercice 5** Un tableau A de dimension $N+1$ contient N valeurs entières triées par ordre croissant, la $N+1^{\text{e}}$ valeur étant indéfinie. Écrivez un programme permettant d'insérer une valeur `x` donnée au clavier dans le tableau A de manière à obtenir un tableau de $N+1$ valeurs triées (le tableau A a déjà été pré-rempli avec les valeurs triées par ordre croissant – code sensiblement identique au Listing 5.1).

▷ **Exercice 6** Le crible d'Ératosthène permet de déterminer les nombres premiers inférieurs à une certaine valeur N. On place dans un tableau unidimensionnel `tab` les nombres entiers compris entre 1 et N. L'algorithme consiste, pour chaque élément `tab[i]` du tableau, à rechercher parmi tous les suivants (d'indices $i+1$ à N), ceux qui sont des multiples de `tab[i]` et à les éliminer (en les remplaçant, par exemple, par 0). Lorsque l'ensemble du tableau a subi ce traitement, seuls les nombres premiers du tableau n'ont pas été éliminés.

Écrivez un programme permettant de trouver, grâce au crible d'Ératosthène, les nombres premiers inférieurs à 500.

▷ **Exercice 7** Écrivez un programme qui lit au clavier les points de N élèves d'une classe pour un devoir et les mémorise dans un tableau à N éléments. Les notes saisies sont comprises entre 0 et 60.

Une fois les notes obtenues, il est demandé de rechercher et d'afficher :

- la note maximale
- la note minimale
- la moyenne des notes

▷ **Exercice 8** Cet exercice complète l'exercice précédent. Une fois le code de l'Exercice 7 écrit, il est demandé de le modifier comme suit : à partir des points des élèves, créer un tableau (appelons le `notes`) de dimension 7 qui est composé de la façon suivante :

`notes[6]` contient le nombre de notes à 60.

`notes[5]` contient le nombre de notes comprises entre 50 et 59.

`notes[4]` contient le nombre de notes comprises entre 40 et 49.

...

`notes[0]` contient le nombre de notes comprises entre 0 et 9.

Votre programme va devoir afficher à l'écran un graphique de barreaux (i.e., un *histogramme*) représentant le tableau `notes`. Utilisez les symboles `#######` pour la représentation des barreaux et affichez le domaine des notes en dessous du graphique.

Voici un exemple de ce que votre programme doit afficher :

```
La note maximale est 58
La note minimale est 13
La moyenne des notes est 37.250000


6  >                              #######
5  >                      ####### #######
4  >                ####### ####### #######
3  >                ####### ####### ####### #######
2  >          ####### ####### ####### ####### #######
1  >          ####### ####### ####### ####### #######
    +-------+-------+-------+-------+-------+-------+-------+
    I 0 - 9 I 10-19 I 20-29 I 30-39 I 40-49 I 50-59 I  60   I
```

Pour cet exercice, il est plus qu'impératif de commencer par réfléchir au problème en l'analysant et en identifiant les divers sous-problèmes. Prenez le temps de bien réfléchir avant de commencer à coder.

5.1.2 Algorithmique

▷ **Exercice 1** La *distance de Hamming* entre deux tableaux de même taille est le nombre de position où les deux tableaux diffèrent. Par exemple, la distance de Hamming entre $\boxed{0}\boxed{1}\boxed{0}\boxed{0}$ et $\boxed{1}\boxed{1}\boxed{0}\boxed{1}$ égale 2.

Écrivez un programme qui calcule la distance de Hamming entre deux tableaux de même taille (on supposera que les tableaux ont déjà été remplis par l'utilisateur – code sensiblement identique au Listing 5.1).

▷ **Exercice 2** Écrivez un programme qui renverse l'ordre des éléments d'un tableau (au préalablement rempli) dans un nouveau tableau. Affichez, ensuite, à l'écran le contenu du deuxième tableau.

Par exemple, le tableau $\boxed{5}\boxed{1}\boxed{6}\boxed{2}\boxed{7}$ devient $\boxed{7}\boxed{2}\boxed{6}\boxed{1}\boxed{5}$. L'affichage à l'écran donnera :

```
[ 7 2 6 1 5 ]
```

▷ **Exercice 3** Écrivez un programme qui affiche, à l'écran, le nombre d'occurrences de chaque élément d'un tableau préalablement rempli au clavier (code identique au Listing 5.1). La seule hypothèse qu'on puisse faire sur le tableau, c'est qu'il ne peut contenir que des entiers strictement positifs.

▷ **Exercice 4** Même question que l'Exercice 2 mais sans passer, cette fois, par un deuxième tableau. Il faut donc faire le renversement directement dans le tableau initial. Comparez la complexité théorique (notation O) obtenue dans cette exercice avec la complexité théorique de votre code à l'Exercice 2. Quelle est la version la plus performante ?

5.2 Tableaux Multidimensionnels

5.2.1 Compléter des Invariants Graphiques

Dans les exercices qui suivent, nous vous fournissions un énoncé et les Invariants Graphiques associés. Les Invariants Graphiques, à chaque fois, sont **incomplets**. Il vous est demandé de les compléter en fonction du Vademecum (cfr. Chapitre 3, Slide 100 et son adaptation pour les tableaux, Chapitre 5, Slide 22).

▷ **Exercice 1** On dispose d'une matrice (i.e, tableau à deux dimensions) `mat` de dimensions $N \times M$ (i.e., N lignes et M colonnes, où N et M peuvent être différents). La matrice `mat` a déjà été remplie par l'utilisateur avec des valeurs entières. Il est demandé d'afficher, à l'écran, le contenu de `mat`.

Cet exercie ne se concentre que sur les Invariants Graphiques. Inutile, donc, de produire du code C. Afin de faciliter le travail, nous vous fournissons la définition du problème, l'analyse et des Invariants Graphiques incomplets. L'objectif de l'exercice est donc d'illustrer l'approche globale et faire en sorte que vous appréhendiez au mieux les Invariants Graphiques pour les tableaux multidimensionnels.

Définition du Problème

Input : la matrice `mat`, de dimensions $N \times M$, préalablement remplie par l'utilisateur.

Output : le contenu de la matrice `mat` est affiché à l'écran

Objets Utilisés :

`N`, le nombre de lignes de la matrice (`const unsigned int N = ...;`)[2]

`M`, le nombre de colonnes de la matrice (`const unsigned int M = ...;`)

`mat`, une matrice d'entiers (`int mat[N][M];`)

Analyse du Problème On peut envisager deux sous-problèmes, comme illustré à la Fig. 5.1.

SP_1 : Enumération et affichage des lignes de la matrice.

SP_2 : Affichage des colonnes de la ligne courante de la matrice.

On voit clairement que le SP_2 est appliqué autant de fois qu'il y a de lignes dans la matrice. L'enchaînement des SPs est donc une inclusion : $SP_2 \subset SP_1$.

Quoiqu'il en soit, les deux SPs requièrent un traitement itératif et donc des Invariants Graphiques.

Invariant Graphique incomplet pour le SP_1 . Il s'agit ici de s'assurer de l'énumération et de l'affichage de toutes les lignes de la matrice. Complétez la Fig. 5.2 afin de produire un Invariant Graphique permettant de résoudre ce problème.

Invariant Graphique incomplet pour le SP_2 . Il s'agit ici de s'assurer de l'affichage de toutes les colonnes d'une ligne de la matrice, ligne fournie par le SP_1. Complétez la Fig. 5.3 afin de produire un Invariant Graphique permettant de résoudre ce problème.

2. Peu importe la valeur de N.

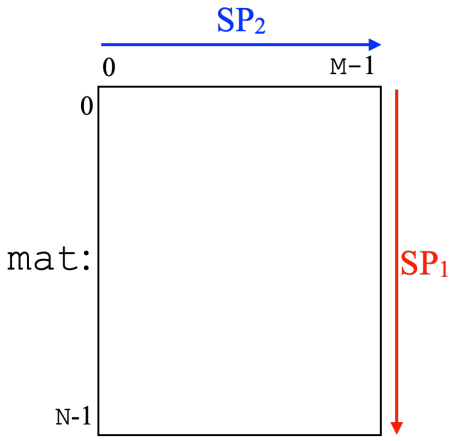

FIGURE 5.1 – Illustration des SPs pour l'affichage d'une matrice.

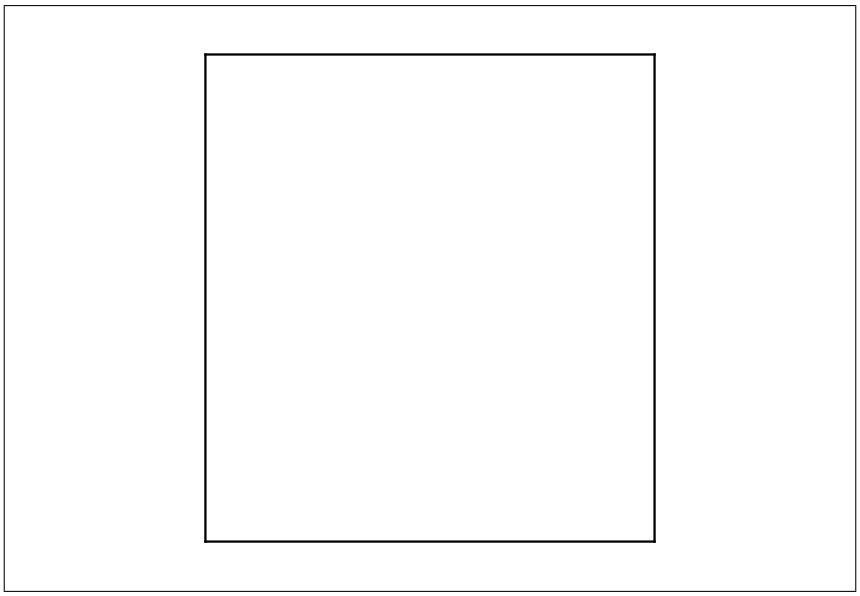

FIGURE 5.2 – Invariant Graphique partiel pour le SP_1.

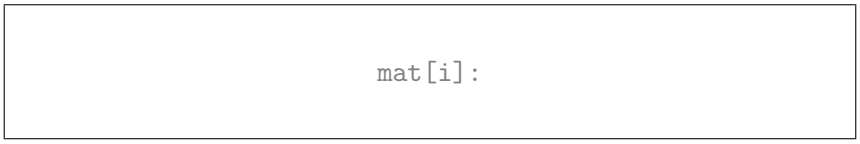

FIGURE 5.3 – Invariant Graphique partiel pour le SP_2.

5.2.2 Construction par Invariant

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [3]
4. Fonction de Terminaison
5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** Écrivez un programme qui remplit un tableau de 12 lignes et 12 colonnes à l'aide des caractères '1', '2' et '3' tel que :

```
1
1 2
1 2 3
1 2 3 1
1 2 3 1 2
1 2 3 1 2 3
1 2 3 1 2 3 1
1 2 3 1 2 3 1 2
1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1
1 2 3 1 2 3 1 2 3 1 2
1 2 3 1 2 3 1 2 3 1 2 3
```

3. cfr. Cours théorique, Chapitre 5, Slide 22.

Rappel préliminaire sur les matrices [a]

On appelle matrice un ensemble de scalaires, i.e. des nombres réels ou complexes, ordonnés sous la forme d'un tableau rectangulaire de m lignes et n colonnes. On note la matrice :

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Les nombres a_{ij} sont les *éléments* de la matrice. On note les matrices en majuscule. Leurs éléments sont notés au moyen de la minuscule correspondante. Les nombres m et n sont les dimensions de la matrice. Si $m = n$, on parle de matrice *carrée*. La dimension $m = n$ est appelée *l'ordre* de la matrice.

Les éléments $a_{ii} (i = 1, \ldots, n)$ d'une matrice carrée forment la *diagonale principale* de la matrice.

Une matrice *identité* est une matrice dont les éléments sont tous nuls sauf ceux de la diagonale principale, qui valent 1, c'est-à-dire, telle que :

$$i_{jk} = \begin{cases} 1, si\ j = k \\ 0, si\ j \neq k \end{cases}$$

Par exemple, si $n = 3$:

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La *transposée* d'une matrice A de dimensions $(m \times n)$, est la matrice A^T de dimensions $(n \times m)$ dont les éléments sont donnés par :

$$(A^T)_{ij} = a_{ji}$$

Exemple, si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, alors $A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$.

L'addition matricielle est définie comme suit : Si A et B sont deux matrices de dimension $N \times M$, alors la matrice C est obtenue via $\forall i \in 1, \ldots, N, \forall j \in 1, \ldots, M, c_{i,j} = a_{i,j} + b_{i,j}$.

Par exemple,

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$$

a. d'après E.J.M.Delhez, *Algèbre*, Tome 1, Centrale des Cours.

▷ **Exercice 2** Écrivez un programme qui met à zéro les éléments de la diagonale d'une matrice carrée A donnée (i.e., remplie par l'utilisateur au clavier).

▷ **Exercice 3** Écrivez un programme qui construit et affiche une matrice *carrée identité* I de dimension N.

▷ **Exercice 4** Écrivez un programme qui effectue la transposition A^T d'une matrice A de dimensions $N \times M$ en une matrice B de dimension $M \times N$. La matrice A sera remplie au clavier par l'utilisateur.

▷ **Exercice 5** Écrivez un programme qui effectue l'addition de deux matrices A et B de même dimension $N \times M$. Les deux matrices A et B seront construites au clavier par l'utilisateur et le résultat de l'addition sera placé dans une matrice C avant d'être affiché à l'écran.

▷ **Exercice 6** Écrivez un programme qui recherche dans une matrice A donnée (i.e., in-

troduite au clavier par l'utilisateur) les éléments qui sont à la fois maximum sur leur ligne et minimum sur leur colonne. Ces éléments sont appelés des *points-cols*. Votre programme doit afficher les positions et les valeurs de tous les points-cols de votre matrice A.

Par exemple, pour la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ le résultat sera le suivant :

```
point-col: (0,2): 3
```

5.3 Chaines de Caractères

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [4]
4. Fonction de Terminaison.
5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** Écrivez un programme déterminant le nombre de lettres "e" (minuscules) présentes dans un texte d'une seule ligne (supposée ne pas dépasser 132 caractères) fourni au clavier.

▷ **Exercice 2** Écrivez un programme qui lit au clavier un mot (d'au plus 30 caractères) et qui l'affiche à l'envers. *Attention : rien ne certifie que le mot contient exactement 30 caractères !*

▷ **Exercice 3** Écrivez un programme qui lit au clavier un mot (d'au plus 50 caractères) et qui détermine si le mot entré est un palindrome ou non.

Pour rappel, un palindrome est un texte dont l'ordre des mots reste le même qu'on le lise de gauche à droite ou de droite à gauche. Par exemple : « kayak », « eh ça va la vache » ou encore « Narine alla en Iran ».

▷ **Exercice 4** Écrivez un programme qui lit des mots au clavier et les affiche après les avoir converti en *louchebem* (i.e., « langage des bouchers »).

Cette conversion consiste à :

- reporter la première lettre du mot en fin de mot, suivie des lettres 'e' et 'm'.
- remplacer la première lettre du mot par la lettre 'l'.

4. cfr. Cours théorique, Chapitre 5, Slide 22.

Par exemple, « vision » devient « lisionvem », « vache » devient « lachevem » ou encore « bonne » devient « lonnebem ».

Pour simplifier, les mots entrés par l'utilisateur font cinq caractères.

▷ **Exercice 5** Écrivez un programme permettant de saisir une phrase au clavier (maximum 300 caractères) et qui affiche, ensuite, le nombre de mots contenu dans la phrase.

▷ **Exercice 6** Écrivez un programme permettant de saisir, au clavier, une chaine de caractères (maximum 100 caractères) et une sous-chaine (maximum 5 caractères) et qui indique, ensuite, combien de fois la sous-chaine est présente dans la chaine de caractères principale.

5.4 Enregistrement

Pour chacun des exercices de cette section (du moins, quand cela s'avère nécessaire), il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [5]
4. Fonction de Terminaison.
5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** À l'aide d'une structure, écrivez la déclaration d'un type permettant de représenter une *durée* avec trois champs entiers indiquant (respectivement) les heures, les minutes et les secondes.

▷ **Exercice 2** À l'aide d'une structure, écrivez la déclaration d'un type pour représenter une *personne*. Une personne sera représentée par son nom et son prénom (maximum 20 caractères dans les deux cas), son âge et son sexe ('M' ou 'F').

▷ **Exercice 3** Un grossiste en composants électroniques vend quatre types de produits :

- des cartes mères (code 1)
- des processeurs (code 2)
- des barrettes mémoire (code 3)
- des cartes graphiques (code 4)

Chaque produit possède une référence (i.e. son code, qui est un nombre entier), un prix en euros et des quantités disponibles.

Il est demandé de

1. définir type pour représenter un produit

5. cfr. Cours théorique, Chapitre 5, Slide 22.

2. d'écrire un programme qui initialise le stock de produits d'un magasin et qui permette ensuite à un utilisateur de saisir une commande d'un ou de plusieurs produit(s). L'utilisateur saisit les codes des produits et les quantités commandées. L'ordinateur affiche toutes les données de la commande, y compris le prix, ainsi que le prix total de la commande.

Par exemple, l'utilisateur pourrait utiliser le programme comme suit :

```
Que souhaitez-vous commander ?
(Carte mere = 1, Processeur = 2, Memoire = 3, Carte graphique = 4, Fini = 0)
Choix  : 2
Entrez une quantité : 200

Que souhaitez-vous commander ?
(Carte mere = 1, Processeur = 2, Memoire = 3, Carte graphique = 4, Fini = 0)
Choix  : 0

Voici le detail de votre commande :
Processeurs 200 x 100.00 = 20000.00 euros
TOTAL = 20000.00 euros
```

▷ **Exercice 4** Écrivez un programme qui lit au clavier des informations dans un tableau de structures de type `point` défini comme suit :

```c
typedef struct{
  int num;
  float x;
  float y;
}point;
```

Le nombre d'éléments du tableau sera fixé au début du programme.

Ensuite, le programme affiche à l'écran l'ensemble des informations précédentes.

▷ **Exercice 5** Dans un premier temps, définissez un type permettant de représenter une fraction mathématique. Écrivez ensuite un programme qui place n (e.g., $n = 50$) fractions dans un tableau et les affiche ensuite à l'écran.

Dans un second temps, complétez votre programme pour qu'il affiche à l'écran toutes les fractions simplifiées.

▷ **Exercice 6** On souhaite construire une structure de données, appelée `EtatCivil`, permettant d'enregistrer le nom d'une personne, sa date de naissance, la ville où elle réside et le code postal de la ville. Définissez un type pour cette structure ainsi qu'un tableau qui permet de saisir ces dix ensembles d'informations et qui les affiche ensuite à l'écran.

▷ **Exercice 7** On souhaite enregistrer les notes de mathématique et de physique pour une classe de 35 élèves et calculer, pour chacun d'eux, la moyenne de ses notes. Proposez une structure de données pertinente et écrivez un programme permettant d'effectuer la saisie des notes puis l'affichage des résultats.

5.5 Énumérations

Pour chacun des exercices de cette section (du moins, quand cela s'avère nécessaire), il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))

2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)

3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [6]

4. Fonction de Terminaison.

5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !

- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** Écrivez une déclaration de type permettant de représenter un booléen. Pour rappel, un booléen est un type de variable à deux états : vrai ou faux. Attention, le type créé doit correspondre à la définition de « vrai » (et « faux ») en C.

▷ **Exercice 2** Écrivez une déclaration de type permettant de représenter, sous la forme d'une énumération, les différents jours de la semaine

▷ **Exercice 3** Écrivez une déclaration de type permettant de représenter, sous la forme d'une énumération, les différents mois de l'année. Attention, janvier doit commencer à 1.

▷ **Exercice 4** On considère un jeu de cartes dans lequel les cartes sont de quatre couleurs possibles : « trefle », « pique », « carreau » et « coeur ». Pour chacune des couleurs, une carte peut prendre les valeurs suivantes : « as », « roi », « dame », « valet », « dix », « neuf », « huit » et « sept ». Proposez une (ou plusieurs) structure(s) de données pour représenter un tel jeu de cartes.

Ensuite, écrivez un programme qui crée un jeu de cartes et qui, ensuite,

- mélange les cartes (à vous de voir comment)

- affiche le jeu de cartes mélangé

6. cfr. Cours théorique, Chapitre 5, Slide 22.

Comment contrôler l'aléatoire en C ?

En C, on peut produire une suite pseudo-aléatoire d'entiers en utilisant la fonction `rand()` de la bibliothèque standard (`stdlib.h`). A chaque fois qu'un programme appelle cette fonction, elle retourne l'entier suivant dans la suite. Les entiers de la suite sont compris entre `0` et `RAND_MAX` (une constante prédéfinie).

Par exemple, le programme suivant :

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

int main(){
  int i;

  for(i=0; i<10; i++){
    int a = rand();
    printf("%d\t", a);
  }//fin for - i

  printf("\n");

  return 0;
}//fin programme
```

produit la sortie suivante :

```
16807    282475249        1622650073       984943658        1144108930
470211272        101027544        1457850878       1458777923       2007237709
```

Si on cherche à tirer aléatoirement un nombre entre 0 et 2 (compris), il suffit d'utiliser le reste de la division par 3 :

```
int coffre_tresor = rand()%3;
```

Si on veut tirer un entier entre A et B compris, on peut faire :

```
int x = rand() % (B-A+1) + A;
```

Pour produire un flottant entre A et B, on peut faire :

```
double x = (double) rand() / RAND_MAX * (B-A) + A ;
```

La suite produite par `rand()` n'est pas vraiment aléatoire. En particulier, si on lance le programme précédent plusieurs fois, on obtiendra exactement la même suite.
Pour rendre le résultat apparemment plus aléatoire, on peut initialiser la suite en appelant la fonction `srand()` (définie aussi dans `stdlib.h`), à laquelle on passe en paramètre une valeur d'initialisation. La suite produite par `rand()` après cet appel dépendra de la valeur que l'on passe à `srand()` lors de l'initialisation. Une même valeur d'initialisation produira la même suite.
Par exemple, le programme suivant :

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

int main(){
  int i;
  srand(2);

  for(i=0; i<5; i++){
    int a = rand();
    printf("%d\t", a);
  }//fin for - i

  printf("\n");
  srand(2);

  for(i=0; i<5; i++){
    int a = rand();
    printf("%d\t", a);
  }//end for - i

  printf("\n");

  return EXIT_SUCCESS;
}//fin programme
```

produit la sortie suivante :

```
33614    564950498        1097816499       1969887316       140734213
33614    564950498        1097816499       1969887316       140734213
```

Pour que la suite soit vraiment imprévisible, on peut passer à `srand()` une valeur dépendant de l'heure. Par exemple en utilisant la fonction `time()` de la librairie standard :

```
srand(time(NULL));
```

5.6 Fichiers

Pour chacun des exercices de cette section (du moins, quand cela s'avère nécessaire), il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [7]
4. Fonction de Terminaison.
5. code C.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., ZONE 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

▷ **Exercice 1** Écrivez un programme permettant d'afficher, à l'écran, le contenu d'un fichier en numérotant les lignes. Ces lignes ne devront jamais dépasser plus de 80 caractères.

▷ **Exercice 2** Écrivez un programme qui permet de créer séquentiellement un fichier "répertoire" comportant pour chaque personne

- nom (20 caractères maximum)
- prénom (15 caractères maximum)
- âge (entier)
- numéro de téléphone (11 caractères maximum)

Les informations relatives aux différentes personnes seront lues au clavier.

▷ **Exercice 3** Écrivez un programme permettant, à partir du fichier créé par l'exercice précédent, de retrouver les informations correspondant à une personne de nom donné (l'information sur le nom est donnée au clavier).

▷ **Exercice 4** Écrivez un programme permettant de saisir 20 mots de 20 caractères maximum et de les enregistrer ligne par ligne dans un fichier appelé `mots.txt`. Écrivez ensuite un autre programme qui permet de relire le fichier et qui en affiche le contenu à l'écran.

▷ **Exercice 5** On propose d'ajouter au programme écrit à l'Exercice 8 (Sec. 5.4) une séquence d'instructions permettant d'écrire les 35 enregistrements contenant les noms, les notes et moyennes de chaque élève dans un fichier appelé `scol.txt`.

Une fois ces modifications effectuées, écrivez un nouveau programme permettant, à partir de ce fichier `scol.txt` d'afficher les résultats scolaires de la classe.

7. cfr. Cours théorique, Chapitre 5, Slide 22.

5.7 Modélisation

▷ **Exercice 1** Soit une société qui doit gérer des informations sur son personnel. Elle emploie au maximum 100 personnes. Pour chaque employé, elle enregistre le nom, le prénom, l'adresse, le sexe et, s'il s'agit d'un homme, la situation militaire (libéré, exempté, réformé ou incorporable), s'il s'agit d'une femme, son nom de jeune fille. Une adresse contient un numéro, un nom de rue, un code postal et une localité. Un nom ne dépasse jamais 30 caractères.

Proposez les structures de données nécessaires pour représenter ces informations. Soyez très précis sur les types utilisés et les noms des différentes structures.

Chapitre 6

Modularité du Code

Jusqu'à présent, les programmes réalisés étaient constitués d'un seul et unique bloc de code principal. Cependant, il est possible de découper le code en divers *modules*, modules pouvant être appelés n'importe quand dans le programme. Une programme C devient alors une succession de modules, le module `int main()` étant le principal. Les modules se présentent sous deux formes : *fonction* (i.e., un module permettant d'effectuer un travail et renvoyant un résultat au code appelant) et *procédure* (i.e., un module permettant d'effectuer un travail mais ne retournant pas de résultat au code appelant). Un exemple de fonction est la fonction `fopen()` que nous avons manipulée avec les fichiers. Un exemple de procédure est `printf()` que nous connaissons depuis le Chapitre 1.

L'avantage de cette découpe en modules, c'est qu'elle s'applique parfaitement à la découpe en sous-problèmes. Chaque sous-problème est représenté, maintenant, par un module. Et pour peu que le sous-problème soit un minimum générique, il pourra être réutilisé maintes fois durant la vie du programme.

Dans ce chapitre, nous allons nous exercer à définir, documenter, et utiliser des modules. Tout d'abord, nous allons commencer par lire et comprendre du code qui implique plusieurs modules (Sec. 6.1). Ensuite, nous allons voir comment documenter précisément un module via des spécifications (Sec. 6.2). Nous allons ensuite écrire des modules et les utiliser dans du code (Sec. 6.3). Enfin, nous allons nous attarder sur la modélisation de problèmes d'une certaine taille. Ces problèmes nécessitent des structures de données et devront être découpés en sous-problèmes et pour chaque sous-problème, il faudra indiquer son *interface* (Sec. 6.4). Pour rappel, l'interface d'un module est sa spécification et son *prototype* (type de retour, identifiant et liste des paramètres formels).

6.1 Lecture de Code

L'objectif de cette section est de vous apprendre à lire et comprendre du code écrit par un autre programmeur. Soyez clair et précis dans vos réponses. Inutile d'exprimer les choses en paraphrasant le code C (i.e., *Si la variable `i` est plus petite que ...*), ce qui correspondrait à une traduction *littérale* du code. Pensez plutôt en terme de *transcodage* (ou traduction *littéraire*) en indiquant l'objectif final du bout de code.

La meilleure façon de résoudre les exercices qui suivent est de travailler avec des valeurs pour chacune des variables, d'exécuter le code avec ces valeurs et, ensuite, essayer d'inférer une relation entre les différentes variables.

▷ **Exercice 1** Soit le code suivant

```
1 #include <stdio.h>
2
```

```c
int fct(int r){
  return 2 * r;
}//fin fct()

int main(){
  int n, p = 5;

  n = fct(p);

  printf("p = %d, n = %d\n", p, n);

  return 0;
}//fin programme
```

Quel est le résultat, à l'écran, de ce programme ?

▷ **Exercice 2** Soit le code suivant :

```c
#include <stdio.h>
int n=10, q=2;

int fct(int p){
  int q;
  q = 2 * p + n;
  printf("B: dans fct, n = %d, p = %d, q = %d\n", n, p, q);

  return q;
}//fin fct()

void f(){
  int p = q * n;
  printf("C: dans f, n = %d, p = %d, q = %d\n", n, p, q);
}//fin f()

int main(){
  int n=0, p=5;

  n = fct(p);
  printf("A: dans main, n = %d, p = %d, q = %d", n, p, q);
  f();

  return 0;
}//fin programme
```

Quel est le résultat, à l'écran, de ce programme ?

6.2 Spécifications

Pour les exercices qui suivent, il n'est pas demandé d'écrire le code C des fonctions.

▷ **Exercice 1** Spécifiez complètement une fonction C qui calcule l'arc tangente d'un angle orienté.

▷ **Exercice 2** Spécifiez complètement une fonction C qui calcule la moyenne des valeurs d'un tableau.

▷ **Exercice 3** Spécifiez complètement une fonction C qui calcule la variance des valeurs d'un tableau.

▷ **Exercice 4** Spécifiez complètement des fonctions C correspondant aux problèmes suivants. Il est clair que, dans ces énoncés, subsistent des ambigüités, des imprécisions. Il vous appartient justement de les éliminer.

1. Rechercher l'emplacement d'une valeur donnée dans un tableau d'entiers.
2. Rechercher l'emplacement d'une valeur donnée dans une portion d'un tableau d'entiers.
3. Calculer le nombre d'éléments communs à deux tableaux d'entiers.
4. Tester si deux tableaux d'entiers comprennent les mêmes valeurs
5. Réaliser une copie inversée d'un préfixe d'un tableau d'entiers
6. Trouver l'entier apparaissant le plus souvent dans un tableau d'entiers

▷ **Exercice 5** En programmation défensive, comment feriez-vous pour écrire les assertions permettant de vérifier les préconditions de la fonction suivante :

```
/*
 * @pre: a>0, b>2*a, b est pair
 * @post: ...
 */
void ma_fonction(int a, int b);
```

6.3 Implémentation et Utilisation de Fonctions/Procédures

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème). Identifiez les sous-problèmes que vous allez implémenter sous la forme de modules et, pour chaque module (fonction ou procédure), donnez la spécification (pensez à faire le lien entre la définition du module et sa spécification)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [1]
4. les Fonctions de Terminaison.
5. code C des modules (fonctions/procédures) et du programme principal.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

En outre, vous penserez à appliquer les principes de la programmation défensive (vérification des préconditions, retour de fonctions, ...) chaque fois que cela s'avère pertinent.

▷ **Exercice 1** Ecrivez un programme qui saisit trois nombres au clavier et affiche leur moyenne.

▷ **Exercice 2** Ecrivez un programme qui saisit, au clavier, la base et la hauteur d'un triangle et affiche l'aire du triangle.

1. cfr. Cours théorique, Chapitre 3, Slide 100 et Chapitre 5, Slide 22.

Pour rappel, l'aire d'un triangle se calcule comme suit :

$$aire \ = \ \frac{base \times hauteur}{2}.$$

▷ **Exercice 3** Dans cet exercice, il vous est demandé d'écrire un programme qui prend, en entrée, deux nombres flottants et un caractère et qui fournit un résultat correspondant à une des quatre opérations appliquées aux deux nombres. A savoir : addition pour le caractère `'+'`, soustraction pour le caractère `'-'`, multiplication pour le caractère `'*'` et division pour le caractère `'/'`. Attention, tout autre caractère que l'un des quatre cités sera interprété comme une addition. De plus, on ne tiendra pas compte des risques de division par zéro.

▷ **Exercice 4** Écrivez un programme permettant de calculer la valeur du nombre e, base des logarithmes népériens, en vous appuyant sur le fait que le nombre e peut être approximé par un développement en série.

Pour rappel, le développement en série du nombre e est le suivant :

$$e = \sum_{k=0}^{+\infty} \frac{1}{k!}.$$

▷ **Exercice 5** Écrivez un programme permettant de calculer la valeur de e^x. Pour ce faire, appuyez vous sur le fait que e^x peut être approximé à l'aide d'un développement en série.

Pour rappel, le développement en série du nombre e^x est le suivant :

$$e^x = \sum_{k=0}^{+\infty} \frac{x^k}{k!}.$$

▷ **Exercice 6** Soit la fonction mathématique f définie par

$$f(x) = \frac{(2x^2+3)(x^2-1)}{\sqrt{3x^2+1}}.$$

1. Écrivez une fonction C qui retourne la valeur de $f(x)$ pour un point x passé en paramètre.
2. Une approximation de la dérivée f' de la fonction f est donnée en chaque point x pour h assez petit (i.e., proche de 0) par :

   $$f'(x) \approx \frac{f(x+h)-f(x)}{h}.$$

   Écrivez la fonction C qui calcule une approximation de la dérivée f' de f en un point x entré au clavier. On passera la valeur de h et de x en paramètre de la fonction.
3. La dérivée seconde de f est la dérivée de la dérivée. Écrivez une fonction C qui calcule une approximation de la dérivée seconde f'' de f en un point x entré au clavier. L'approximation est calculée comme suit :

   $$f''(x) \approx \frac{f(x+h)+f(x-h)-2f(x)}{h^2}.$$

   On passera la valeur de h et de x en paramètre de la fonction.
4. Écrivez une fonction C qui détermine le signe de la dérivée seconde de f en fonction de x. On pourra faire un programme principal qui lit x au clavier et affiche le résultat.
5. Écrivez une fonction C qui donne le choix à l'utilisateur d'afficher la valeur de la fonction f, de sa dérivée première ou de sa dérivée seconde en point x lu au clavier.

▷ **Exercice 7** Comment écririez-vous la fonction dont l'interface[2] est la suivante :

```c
/*
 * @pre: t est un tableau valide, n>0
 * @post: retourne 1 si les donnees du tableau t sont en ordre
 *       strictement decroissant.  -1 sinon.
 */
int ordonne(double t[], int n);
```

▷ **Exercice 8** Écrivez le corps d'une fonction répondant à l'interface de la fonction suivante :

```c
/*
 * @pre: t est un tableau valide, n>0
 * @post: r est un tableau de meme taille que le tableau passe en
 *        arguments mais dont les elements sont inverses (le dernier
 *        element du tableau passe en arguments est le premier du
 *        tableau resultat, l'avant-dernier devient le deuxieme, ...)
 */
void reverse(int t[], int r[], int n);
```

Ensuite, écrivez un programme qui utilise cette fonction. Pensez à appliquer les principes de la programmation défensive.

▷ **Exercice 9** Comment feriez-vous pour écrire une fonction C répondant à l'interface suivante :

```c
/*
 * @pre: matrice est de dimension n*n, n>0
 * @post: retourne le nombre d'elements de la matrice qui sont >=0
 */
int nPositifs(int n, int matrice [][n]);
```

▷ **Exercice 10** Les tableaux peuvent servir à stocker des durées au format | j | h | m | s |, où j, h, m et s sont des entiers décrivant respectivement le nombre de jours, d'heures, de minutes et de secondes. Écrivez une fonction qui renvoie la somme de deux durées encodées dans ce format, en prenant garde de respecter les contraintes habituelles sur la représentation des durées (au plus 59 secondes, au plus 59 minutes, au plus 23 heures).

▷ **Exercice 11** Soit T un tableau et M une matrice de même type que T (e.g., entier).

1. Écrivez une fonction vérifiant que M et T possèdent exactement le même contenu (l'ordre des éléments n'est pas important).
2. Donnez la complexité théorique de votre solution.
3. Comment amélioreriez-vous votre fonction dans le cas où T est trié ? Quel impact cela aurait-il sur le temps d'exécution ?

▷ **Exercice 12** Une matrice binaire M possède la *propriété des uns consécutifs* si pour chaque ligne de M, deux 1 apparaissant dans cette ligne ne sont jamais séparés par un 0. Écrivez une fonction vérifiant si une matrice binaire donnée possède la propriété des uns consécutifs.

▷ **Exercice 13** Soit M une matrice réelle représentant une carte géographique (une valeur positive indique la terre, une valeur négative la mer – voir Fig. 6.1). Écrivez une fonction qui modifie M en traçant le contour des continents dessinés sur la carte. Par exemple :

Les contours peuvent être représentés par des zéros.

6.4 Modélisation

▷ **Exercice 1** Vous venez de créer une compagnie du nom de "Les Limonades Inc." et vous pensez pouvoir révolutionner le monde de la limonade par votre gestion efficace de vos produits.

2. Pour rappel, une interface d'une fonction est le *prototype* et les *spécifications* de la fonction

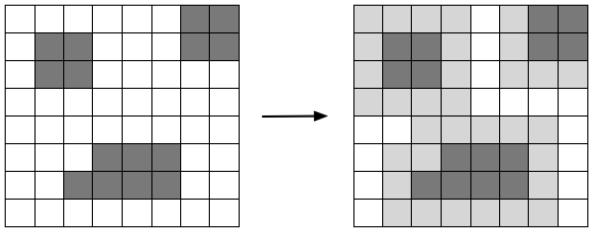

FIGURE 6.1 – Matrice représentant une carte géographique.

Clients	Semaine_1	Semaine_2	Semaine_3	Semaine_4
Gérard	$2\times P_2$	$1\times P_1$	$3\times P_3$	$2\times p_2$
Jules	$3\times P_1$	$2\times P_2$	$2\times P_1$	$3\times p_3$

TABLE 6.1 – Prévision des demandes, par client et produit, pour le mois prochain.

Suite à une étude de marché, votre département de marketing a opté pour la commercialisation de trois nouveaux produits, P_1, P_2 et P_3, qui se vendent, respectivement, aux prix de 3.50$US, 2.50€ et 3.64$CAD.

Pour créer ces produits, vous devez mélanger de la matière première. Le produit P_1 contient $2 \times R_1 + 3 \times R_2$. Le produit P_2 contient $1 \times R_1 + 1 \times R_2 + 1 \times R_3$. Le produit P_3, quant à lui, contient $1 \times R_2 + 2 \times R_3$. Vous pouvez acheter les produits R_1 et R_2 chez votre fournisseur "En Poudrie Cie" pour, respectivement, 0.25€ et 0.35€. Et les produits R_2 et R_3 chez un autre fournisseur, "Les Produits Pétri" pour 0.17€ et 0.33€, respectivement.

Pour le moment, vous avez deux fidèles clients : Gérard mansoif et Jules Piler.

Commencez par définir les structures de données nécessaires pour représenter ce problème. Attention, vos structures doivent représenter les produits, les fournisseurs et les clients.

Sachant que vous ne pouvez vous réapprovisionner qu'une fois par mois et que la demande pour le prochain mois est donnée par le Tableau 6.1.

Définissiez et spécifier les fonctions dont vous aurez besoin pour gérer vos produits. Il vous est demandé de séparer votre problème en sous-problèmes et, pour chaque sous-problème, de donner l'interface. Attention, pour gérer une demande, vous avez besoin au moins d'une fonction qui traite les entrants (i.e., inputs) et une qui traite les extrants (i.e., outputs).

Le gouvernement Belge est plutôt strict. Il stipule clairement que les finances des entreprises doivent être maintenues en €. Le taux de conversion $US-€ est de 1.396 et le $CAD-€ est de 1.4123. Cependant, la banque "MegaPognon" qui exécute les transferts bancaires facture 0.15€ par transaction. Ecrivez et spécifiez les fonctions dont vous aurez besoin lors de votre rapport d'impôts. Attention, il ne vous est demandé que de séparer le problème en sous-problèmes et de donner l'interface de chaque sous-problème (pas d'implémentation, donc). N'oubliez pas, aussi, que vous avez besoin de convertir votre argent et de calculer votre profit.

▷ **Exercice 2** Pour le magasin "LouScoot", commerce de location de scooters, il faut réaliser un programme pour gérer son parc de scooters. Un scooter est caractérisé par les éléments suivants :

- le modèle du scooter
- son numéro d'identification
- son kilométrage
- son état (i.e., disponible ou en cours de location)

Le programme que vous demande “LouScoot” doit répondre aux impératifs suivants :

- Il doit être possible de louer un scooter. Lors de la demande de location, le numéro d'identification du scooter est entré, ce qui permet de vérifier si le scooter existe bien et si il est déjà en location (ou pas).
- Il doit être possible de ramener un scooter, à la fin de la location. A nouveau, le numéro d'identification du scooter est utilisé pour vérifier si le scooter existe bien et si il était bien en cours de location. Si le scooter est bien en cours de location, le fait de ramener le scooter change son kilométrage (en fonction de ce que le client aura parcouru) et son état (il passe de l'état “en location” à l'état “libre”).
- Il doit être possible de connaître l'état du scooter sur base de son numéro d'identification. Si le scooter existe bien, alors il faut indiquer le modèle du scooter, son kilométrage et son état.
- On doit pouvoir effectuer l'action précédente pour tous les scooters gérés par “LouScoot”. En outre, quand l'état de tous les scooters aura été indiqué, cette fonctionnalité indique le kilométrage moyen des scooters, ainsi que le nombre de scooters loué et le nombre de scooter actuellement au dépôt.

L'utilisateur interagit avec le programme de la société “LouScoot” via un menu, comme par exemple :

```
*MENU*     LouScoot
  1: Louer un scooter
  2: Ramener un scooter
  3: Etat d'un scooter
  4: Etat du parc de scooters
  0: Quitter
```

Dans cet exercice, il vous est demandé de proposer une structure de données permettant de représenter un scooter ainsi que le parcs de scooters de la société “LouScoot”. Il vous est aussi demandé d'identifier les différents sous-problèmes et, pour chaque sous-problème, de fournir une interface claire et précise. Attention, il ne vous est pas demandé d'implémenter les différents sous-problèmes.

Chapitre 7

Pointeurs

Le cœur du chapitre est la notion de *pointeur*, i.e., une variable qui contient non pas une valeur mais, plutôt, l'adresse d'une autre zone mémoire qui, elle, contient la valeur. La Sec. 7.1 va nous permettre de manipuler les pointeurs. La Sec. 7.2 va s'intéresser au passage de paramètres des modules. En particulier, nous verrons comment effectuer un passage de paramètres par *adresse* et, donc, comment un module (fonction ou procédure) peut renvoyer plus d'un résultat.

7.1 Arithmétique des Pointeurs

▷ **Exercice 1** Pour chaque programme ci-dessous :

1. donnez la représentation graphique de la mémoire à la fin du programme
2. indiquez ce qu'affiche le programme

```c
#include <stdio.h>

int main(){
  int i = 3;
  int *p;
  p = &i;
  printf("*p = %d\n", *p);

  return 0;
}//fin programme
```

```c
#include <stdio.h>

int main(){
  int i = 3;
  int *p;
  p = &i;
  *p = 5;

  printf("i=%d, *p=%d\n", i, *p);
  i = 7;
  printf("i=%d, *p=%d\n", i, *p);
  return 0;
}//fin programme
```

▷ **Exercice 2** Commentez l'affichage du programme suivant, et comparez les valeurs des pointeurs.

```c
#include<stdio.h>
int main(){
  double var = 3.14;
  double *point_var, *pointeur2;
  int var2 = 5, *pointeur3;

  point_var = &var;
  pointeur2 = point_var+1;
  pointeur3 = &var2+1;

  printf("%d %d \n",sizeof(double),sizeof(int));
  printf("%p  %p %p \n", point_var, &var, pointeur2);
  printf("%p %p \n",&var2,pointeur3);
```

```
14
15   return 0;
16 }//fin programme
```

▷ **Exercice 3** Comparez ces deux programmes en donnant la représentation graphique de leur mémoire en fin d'exécution.

```
1  int main(){
2    int i = 3, j = 6;
3    int *p1, *p2;
4    p1 = &i;
5    p2 = &j;
6    *p1 = *p2;
7
8    return 0;
9  }//fin programme
```

```
1  int main(){
2    int i = 3, j = 6;
3    int *p1, *p2;
4
5    p1 = &i;
6    p2 = &j;
7    p1 = p2;
8
9    return 0;
10 }//fin programme
```

▷ **Exercice 4** Ce programme n'est pas correct. Corrigez-le.

```
1  #include <stdio.h>
2
3  typedef struct complexe{
4    float reel;
5    float img;
6  }Complexe;
7
8  int main(){
9    Complexe C, *p;
10
11   C.reel = 3;
12   C.img = 2;
13   p = &C;
14
15   printf("reel=%f, img=%f\n", p.reel, p.img);
16
17   return 0;
18 }//fin programme
```

▷ **Exercice 5**

Soit le programme suivant :

```
1  int main(){
2    float a = 1.5;
3    float b = 3.5;
4    float c, *p1, *p2;
5
6    p1 = &a;
7    *p1 *= 2;
8    p2 = &b;
9    c = 3 * (*p2 - *p1);
10   p1 = p2;
11   *p1 = 1.5;
12   (*p2)++;
13
14   return 0;
15 }//fin programme
```

Complétez le Tableau 7.1. Que se passe-t-il si on enlève les parenthèses à la douzième ligne du programme ?

▷ **Exercice 6** En sachant qu'un `int` est codé sur 4 bytes et qu'un `double` est codé sur 8

ligne(s)	a	b	c	p1	p2
2,3,4	1,5	3,5	/	/	/
6	1,5	3,5	/	&a	/
7					
8					
9					
10					
11					
12					

TABLE 7.1 – Exercice 5

bytes et en supposant que `i` est stocké en mémoire à l'adresse 0xCDEF3210 [1], qu'affichent ces deux programmes ?

```c
#include <stdio.h>

int main(){
  int i = 3;
  int *p1, *p2;

  p1 = &i;
  p2 = p1 + 1;

  printf("p1 = %p , p2 = %p\n", p1, p2);
  return 0;
}//fin main()
```

```c
#include <stdio.h>

int main(){
  double i = 3;
  double *p1, *p2;

  p1 = &i;
  p2 = p1 + 1;

  printf("p1 = %p , p2 = %p\n", p1, p2);
  return 0;
}//fin programme
```

7.2 Passage de Paramètres

▷ **Exercice 1** On a demandé à un étudiant d'écrire une fonction permettant d'échanger les valeurs de 2 variables. Voici le code, erroné, produit par l'étudiant :

```c
void swap(int a, int b){
  a ^= b;
  b ^= a;
  a ^= b;
}//fin swap()
```

Aidez l'étudiant à corriger sa solution.

▷ **Exercice 2** Soit le code suivant :

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

1. Les nombres qui débutent par `0x...` sont des nombres hexadécimaux, i.e. écrits en base 16, plus commode pour représenter des adresses (voir INFO0061).

```
2
3  char fonc1(char a, char b){
4    a = 'P';
5    b = 'Q';
6
7    if(a<b)
8      return a;
9    else
10     return b;
11 }//fin fonc1()
12
13 char fonc2(char *c1, char *c2){
14   *c1 = 'P';
15   *c2 = 'Q';
16
17   if(*c1==*c2)
18     return *c1;
19   else
20     return *c2;
21 }//fin fonc2()
22
23 int main(){
24   char a = 'X';
25   char b = 'Y';
26   char i, j;
27
28   i = fonc1(a, b);
29   printf("a␣=␣%c,␣b␣=␣%c\n", a, b);
30   j = fonc2(&a, &b);
31   printf("a␣=␣%c,␣b␣=␣%c\n", a, b);
32
33   return 0;
34 }//fin programme
```

Répondez à ces questions :

1. Quelle est la valeur affectée à `i` ?
2. Quelle est la valeur affectée à `j` ?
3. Qu'affiche ce programme ?

▷ **Exercice 3** Soit la fonction C suivante :

```
1 void modifie(int a, int *res){
2   if(a>0)
3     *res = a+1;
4   else
5     if(a<0)
6       *res = a-1;
7     else
8       *res = a;
9 }//fin modifie()
```

Pour chacun des programmes ci-dessous :

1. indiquez les valeurs des variables à la fin du programme (si celui-ci est correct)
2. sinon expliquez pourquoi ce n'est pas correct (soyez précis avec le vocabulaire que vous utilisez).

```c
int main(){
    int a = 3;
    int res;

    modifie(a, res);

    return 0;
}//fin programme
```

```c
int main(){
    int a = 3;
    int res, *p;

    p = &res;
    modifie(a, p);

    return 0;
}//fin programme
```

```c
int main(){
    int a = 3;

    modifie(a, a);

    return 0;
}//fin programme
```

```c
int main(){
    int a = 3;
    int res;

    modifie(a, &res);

    return 0;
}//fin programme
```

```c
int main(){
  int a = 3;
    int res, *p;

    p = &res;

    modifie(a, &p);

    return 0;
}//fin programme
```

```c
int main(){
    int a = 3;

    modifie(a, &a);

    return 0;
}//fin programme
```

▷ **Exercice 4** Soit la fonction `main()` suivante appelant une fonction `calcul()` qui réalise deux opérations : la somme et la différence de deux entiers.

```c
#include <stdio.h>

int main(){
  int a = 12, b = 45;
  int somme, difference;

  calcul(a, b, &somme, &difference);

  printf("a + b = %d, a - b = %d\n", somme, difference);

  return 0;
}//fin programme
```

Donnez le prototype de la fonction `calcul()` et, ensuite, écrivez le code de cette fonction.

Chapitre 8

Allocation Dynamique

Ce chapitre se concentre sur les structures de données *dynamiques*. Jusqu'à présent, les structures de données manipulées avaient un côté statique, e.g., la taille d'un tableau devait être définie directement dans le code et connue à la compilation. On va voir, dans ce chapitre, comment on peut créer (par exemple) des tableaux à la volée, en cours d'exécution du programme.

La Sec. 8.1 va être une première entrée dans la création, à la volée, de tableaux tandis que la Sec. 8.2 nous permettra de mettre les mains dans le cambouis.

8.1 Allocation Dynamique de Mémoire

▷ **Exercice 1** Ces programmes sont-ils correct ? Expliquez.

```c
#include <stdlib.h>
int main(){
   int i, *p;

   i = 3;
   *p = 5;

   return 0;
}//fin programme
```

```c
#include <stdlib.h>
int main(){
   int i, *p;

   i = 3;
   p = &i;
   *p = 5;

   free(p);
}//fin programme
```

▷ **Exercice 2** Soit le programme suivant :

```c
#include <stdlib.h>

int main(){
  int **p;

  p = (int *)malloc(sizeof(int));
  **p = 5;

  free(p);

  return 0;
}//fin programme
```

Dans ce programme, quel est le type de `p`, `*p`, `**p` ? En outre, ce programme est-il correct ? Justifiez.

8.2 Écriture de Code

Pour chacun des exercices de cette section, il vous est demandé de suivre avec rigueur la méthologie vue au cours. En particulier, la réponse à chacun des exercices devra comprendre (explicitement) les étapes suivantes :

1. définition (i.e., Input, Output, Objet(s) Utilisé(s))
2. analyse (i.e., découpe en sous-problèmes – pensez à indiquer des noms pertinents pour chaque sous-problème). Identifiez les sous-problèmes que vous allez implémenter sous la forme de modules et, pour chaque module (fonction ou procédure), donnez la spécification (pensez à faire le lien entre la définition du module et sa spécification)
3. Invariants Graphiques (veillez bien à nommer correctement les objets que vous manipulez et à placer correctement et précisement les différentes bornes) [1]
4. les Fonctions de Terminaison.
5. code C des modules (fonctions/procédures) et du programme principal.

Attention, pour chaque segment de code impliquant une boucle (et donc un Invariant Graphique), il vous est demandé de :

- déterminer les variables nécessaires et leurs valeurs d'initialisation (i.e., Zone 1). Justifiez sur base de l'Invariant Graphique !
- déterminer le Critère d'Arrêt et la Fonction de Terminaison ; Jusitifiez sur base de l'Invariant Graphique !

Faites valider chacune de vos étapes par un membre de l'équipe pédagogique avant de passer à l'étape suivante.

En outre, vous penserez à appliquer les principes de la programmation défensive (vérification des préconditions, retour des fonctions – e.g., retour du `malloc()` –, ...) chaque fois que cela s'avère pertinent.

▷ **Exercice 1** Soit une fonction `int rand(int min, int max)` qui retourne un nombre entier aléatoire compris dans l'intervalle [`min`, `max`]. Écrivez une fonction qui alloue un tableau d'entiers de taille donnée et qui remplit celui-ci avec des entiers aléatoires compris dans un intervalle donné.

La fonction demandée est déclarée comme suit :

```
1 int rand_tab(unsigned int n, int min, int max, int **dest);
```

où

- `n` est la taille du tableau,
- `min` et `max` sont les bornes inférieure et supérieure des nombres aléatoires à insérer dans le tableau,
- `dest` la destination du tableau généré.

La fonction retourne zéro si le tableau a été généré et -1 si ce n'est pas le cas (si `min` $>$ `max` ou `dest == NULL`).

▷ **Exercice 2** Le triangle de Pascal est une représentation des coefficients binomiaux dans un triangle. Le nombre à la ligne i et la colonne j tel que $0 \leq j \leq i$ est égal à C_i^j. Le triangle de Pascal peut être construit en utilisant les règles suivantes :

$$\begin{cases} C_i^0 &= C_i^i = 1 \text{ avec } i \geq 0 \\ C_n^p &= C_{n-1}^{p-1} + C_{n-1}^p \text{ avec } p \geq 1,\ n \geq 1,\ n-1 \geq p \end{cases}.$$

1. cfr. Cours théorique, Chapitre 3, Slide 100 et Chapitre 5, Slide 22.

Écrivez une fonction qui génère un tableau triangulaire contenant les n premières lignes du triangle de Pascal. Celle-ci est déclarée comme suit :

```
1 int **tri_pas(unsigned int n);
```

où `n` est le nombre de lignes du triangle de Pascal à générer. La fonction retourne un tableau triangulaire de `n` lignes généré suivant la définition donnée ci-dessus.

▷ **Exercice 3** Soit une fonction `int f(int *x)` donnée. Cette fonction est capable de produire un nombre limité et a priori inconnu d'entiers. Lorsque `f` est appelée, si un entier est produit, il est stocké dans l'espace pointé par `x` et `f` retourne une valeur différente de zéro. Sinon, `f` retourne zéro et aucun appel suivant ne produira d'entier.

Écrivez un programme qui remplit, au moyen de `f`, un tableau d'entiers t de taille `n` donnée (`n` ≥ 0) à partir de l'indice zéro et tant que `f` produit un nombre. À la fin de l'exécution du programme, la variable `lim` contient le nombre d'entiers que `f` a produit (`lim` $\leq$ `n`).

▷ **Exercice 4** Soit la suite u_n définie par $u_0 = 1$ et $u_{n+1} = 3u_n^2 + 2u_n + 1$. Dans un premier temps, spécifiez et écrivez une fonction C qui prend en paramètre un entier n et qui retourne un tableau contenant les n premiers termes de la suite u_n.

Dans un second temps, écrivez le programme principal qui va utiliser la fonction que vous venez de définir. L'entier n sera passé en paramètre de votre programme. Votre fonction principale devra donc récupérer l'argument passé. Ensuite, appeler la fonction de construction de la suite u_n et afficher les résultats à l'écran.

▷ **Exercice 5** Commencez par définir une structure de données permettant de gérer un étudiant. Un étudiant est représenté par son nom, son prénom, son matricule, 4 notes d'examen, sa moyenne, et son classement.

Spécifiez et écrivez ensuite une fonction qui remplit un tableau d'étudiants tant que l'utilisateur souhaite en ajouter et que le tableau n'est pas rempli. Attention, la moyenne et le classement ne sont pas des données à lire au clavier.

Spécifiez et écrivez ensuite une fonction qui prend en argument un tableau d'étudiants et qui calcule la moyenne de chaque étudiant ainsi que la moyenne de la promo.

Sur base de cette fonction, spécifiez et écrivez une fonction qui calcule le classement de chaque étudiant.

Enfin, spécifiez et écrivez une fonction qui réordonne les étudiants en fonction de leur classement (du premier au dernier).

▷ **Exercice 6** On veut organiser les fiches de renseignements des élèves dans un fichier.

1. Ecrivez un programme qui permet
   (a) la saisie et la sauvegarde des fiches dans un fichier nommé `Eleves.txt`. La fin de la saisie est possible si nous répondons par 'N' à la question "Voulez-vous saisir une nouvelle fiche ?".
   (b) l'affichage des fiches.
2. Chaque fiche comporte les éléments suivants : nom, prénom, numéro de téléphone, âge et adresse. Proposez une structure de données permettant de manipuler une fiche.

▷ **Exercice 7** Dans cet exercice, il est demandé de :

1. Définir une structure de données appelée `Rectangle` qui contient trois champs réels : longueur, largeur et périmètre.
2. Demander à l'utilisateur combien de rectangle il veut définir. Soit `n` ce nombre.
3. Construire un tableau dynamique de rectangle, appelé `T` et de taille `n`.
4. Spécifier et écrire une fonction `calcul_perimetre` qui calcule le périmètre d'un rectangle donné.

5. Spécifier et écrire une fonction `creer_rectangle` qui demande à l'utilisateur la longueur et la largeur d'un rectangle et écrit dans `T` les valeurs saisies dans la première case vide de `T`.
6. Spécifier et écrire une fonction `modifier` qui transforme un rectangle en un rectangle plus petit en divisant sa longueur et sa largeur par deux.
7. Spécifier et écrire une fonction `affiche` qui affiche un rectangle.
8. Afficher tous les rectangles, avant et après modification.
9. Appliquer la fonction `modifier` à tous les rectangles du tableau.
10. Finalement, écrire le programme principal qui, en se servant des fonctions précédemment définies, demande à l'utilisateur de saisir les longueurs et largeurs de `n` rectangles, puis calcule leur périmètre, puis affiche les rectangles, les modifie, (il faut aussi mettre à jour leur périmètre) et effectue enfin un dernier affichage de tous ces rectangles.

▷ **Exercice 8** On considère un tableau à m lignes et n colonnes entrées dans un fichier. La première ligne du fichier contient les nombres m et n. Les lignes suivantes du fichier contiennent les coefficients du tableau. Les colonnes sont séparées par des espaces.

```
m n
a_0,0    a_0,1    a_0,2     ...   a_0,n-1
a_1,0    a_1,1    a_1,2     ...   a_1,n-1
...      ...      ....      ...   ...
a_m-1,0  a_m-1,1  a_m-1,2   ...   a_m-1,n-1
```

1. Spécifiez et écrivez une fonction d'allocation du tableau à deux dimensions de m lignes et n colonnes, les nombres m et n étant passés en paramètres.
2. Spécifiez et écrivez une fonction de libération de mémoire pour un tableau à deux dimensions.
3. Spécifiez et écrivez une fonction qui réalise le chargement du fichier dans un tableau de tableaux (ou matrice) A.
4. Spécifiez et écrivez une fonction qui calcule la somme des coefficients de chaque ligne de la matrice A et qui met les résultats dans un tableau de m nombres. La fonction retournera ce tableau.
5. Écrivez le programme principal qui affiche la somme des coefficients de chaque ligne d'une matrice stockée dans un fichier, et libère la mémoire.

▷ **Exercice 9** Une *permutation* est un tableau T contenant chaque nombre de $0, 1, \ldots, n-1$ exactement une fois, et peut être interprétée de la façon suivante : T est une fonction envoyant l'élément i vers l'élément $T[i]$. L'*inverse* de cette permutation s'obtient en échangeant chaque élément de T avec sa position. Par exemple :

$$T = \begin{array}{|c|c|c|c|c|}\hline 0&1&2&3&4\\\hline 2&1&4&3&0\\\hline\end{array} \rightarrow \begin{array}{|c|c|c|c|c|}\hline 2&1&4&3&0\\\hline 0&1&2&3&4\\\hline\end{array} = \begin{array}{|c|c|c|c|c|}\hline 0&1&2&3&4\\\hline 4&1&0&3&2\\\hline\end{array} = T^{-1}$$

Ecrivez une fonction renvoyant l'inverse d'une permutation donnée.

▷ **Exercice 10** Le produit de deux permutations P et Q est une permuation exprimant la composée des fonctions représentées par P et Q appliquées dans cet ordre. Par exemple :

$$P = \begin{array}{|c|c|c|c|}\hline 0&1&2&3\\\hline 1&3&0&2\\\hline\end{array}, Q = \begin{array}{|c|c|c|c|}\hline 0&1&2&3\\\hline 2&1&3&0\\\hline\end{array} \Rightarrow P \times Q = \begin{array}{|c|c|c|c|}\hline 0&1&2&3\\\hline 1&0&2&3\\\hline\end{array}$$

Écrivez une fonction renvoyant le produit de deux permutations de même taille données.

▷ **Exercice 11** Écrivez une fonction renvoyant la matrice M d'une permutation P donnée, définie comme suit :

$$M[i][j] = \begin{cases} 1 & \text{si } P[i] = j \\ 0 & \text{sinon} \end{cases} .$$

Deuxième partie

Séances de Laboratoire

Laboratoire 1

Prise en main

L'objectif de cette première séance de laboratoire est de prendre en main le système. Le système d'exploitation avec lequel nous allons travailler durant les séances de laboratoire est Linux, une implémentation libre du système UNIX. Vous avez, en outre, suivi une formation Linux donnée. Les informations fournies lors de cette formation vous seront utiles durant les différentes séances de laboratoire mais, aussi, durant toutes vos études et votre vie professionnelle.

La réalisation des laboratoires nécessite l'obtention d'un PMLI (login + mot de passe). Ce PMLI (personnel) vous est donné lors de la première séance du Premier Projet d'Informatique (INFO2056). Ne le perdez pas !! Il vous sera utile durant tout le cours mais aussi durant toute l'année académique.

Si vous le désirez, vous pouvez utiliser votre propre laptop plutôt que les machines du laboratoire. Dans ce cas, assurez-vous d'avoir installé au préalable tous les outils nécessaires (cfr. eCampus, Sec. Install Party). Nous vous conseillons d'utiliser le logiciel Atom pour taper votre code source (à nouveau, reportez-vous à eCampus, Sec. Install Party).

Durant les séances de laboratoire, nous allons taper, compiler et exécuter des programmes dans l'environnement Linux. La Sec. 1.1 présente un premier programme trivial qu'il va falloir recopier, compiler et exécuter. La Sec. 1.1 vous guide pas à pas dans ce travail.

Enfin, la Sec. 1.2 vous propose différents exercices à résoudre sur machine. Attention, la bonne pratique de l'informaticien recommande de, d'abord, travailler sur papier avant de se jeter, tête baissée, sur son clavier. Commencez donc par réfléchir sur papier, par noter quelques idées sur papier pour, enfin, attaquer le problème avec la machine.

Il est possible de se connecter à distance (i.e., depuis chez soi) sur les machines du laboratoire d'informatique. La procécure de connexion à distance est décrite sur eCampus, Sec. Répétitions/Exercices

1.1 Un Premier Programme

Téléchargez, depuis eCampus, le fichier `prog1.c`. [1]

Le compilateur utilisé est `gcc` et, pour compiler un fichier source, nous devons appeler `gcc` en lui passant comme paramètre le nom du fichier à compiler. D'autres paramètres peuvent être passés en argument à la commande `gcc` :

- `-Wall` (*Warnings : all*) demande au compilateur d'effectuer des vérifications et de signaler, sous forme d'avertissements, des constructions du code qui sont peut-être incorrectes. Il est fortement conseillé d'utiliser toujours cette option et de corriger le programme si un avertissement est donné par le compilateur.

1. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

- `-o nom_exécutable` permet de choisir le nom de l'exécutable créé. Le nom de l'exécutable doit suivre le `-o`. Attention, si un fichier de ce nom existe déjà, il est écrasé lors de la création de l'exécutable. Si `-o` n'est pas utilisé, l'exécutable est nommé `a.out`.

Lancez la compilation du programme précédent avec la commande :

```
$> gcc -Wall -o prog1 prog1.c
```

Si des erreurs sont détectées, corrigez le fichier et relancez la compilation. Si aucune erreur n'est détectée, `gcc` n'affiche aucun message. Vous pouvez alors lancer l'exécution du fichier exécutable produit par :

```
$> ./nom_de_l'executable
```

Pour lancer l'exécution, il faut toujours saisir `./` suivi du nom de l'exécutable. Le caractère `.` repère en fait le répertoire courant (de la même manière que `..` repère le répertoire « parent »). La syntaxe `./` permet donc de signaler qu'on désire exécuter le fichier dont le nom suit et qui se trouve dans le répertoire courant.

1.2 Exercices de Programmation C

▷ **Exercice 1** Téléchargez, depuis eCampus, le fichier `Labo1 Exercice 1`.[2] Si la compilation échoue, essayez de comprendre pourquoi en lisant le message d'erreur du compilateur. Corrigez ensuite le code.

▷ **Exercice 2** Écrivez un programme qui permet de saisir, au clavier, deux entiers et qui affiche la somme et le produit de ces deux entiers.

▷ **Exercice 3** Écrivez un programme qui demande de saisir au clavier une valeur entière et qui affiche, ensuite, à l'écran « nombre pair » ou « nombre impair » selon la valeur saisie.

▷ **Exercice 4** Écrivez un programme qui permet de saisir deux valeurs réelles et qui affiche le maximum de ces deux valeurs.

▷ **Exercice 5** Écrivez un programme qui affiche les trente plus petits multiples de sept parmi les naturels non nuls.

▷ **Exercice 6** Écrivez un programme qui demande une valeur et calcule sa factorielle.

▷ **Exercice 7** Écrivez un programme qui permet à l'utilisateur de saisir des notes (entiers). La saisie se termine quand l'utilisateur entre -1. Le programme affiche alors à l'écran la moyenne des notes saisies. Conseil : ce programme se prête bien à l'utilisation d'une boucle `do ... while`.

▷ **Exercice 8** On demande d'écrire un programme qui offre le choix à l'utilisateur de :

1. déterminer si une année est bissextile ou non (Exercice 7, Chapitre 2, Sec. 2.1.2)
2. retrouver la décomposition en puissances de 2 d'un entier non signé (Exercice 6, Chapitre 3, Sec. 3.2)
3. calculer la racine carrée d'un nombre réel au moyen d'une recherche dichotomique (Exercice 8, Chapitre 3, Sec. 3.2.5).

Un exemple d'utilisation à titre illustratif est fourni ci-dessous.

```
Que souhaitez-vous faire?

1. Déterminer si une année est bissextile ou non.
2. Retrouver la décomposition en puissances de 2 d'un entier non signé.
3. Calculer la racine carrée d'un nombre réel au moyen d'une recherche dichotomique.
0. Quitter.

Choix : 1
```

2. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

```
Entrez une année : 2100

Cette année n'est pas bissextile.

Que souhaitez-vous faire?

1. Déterminer si une année est bissextile ou non.
2. Retrouver la décomposition en puissances de 2 d'un entier non signé.
3. Calculer la racine carrée d'un nombre réel au moyen d'une recherche dichotomique.
0. Quitter.

Choix : 3

Entrez un nombre réel : 2

La racine carrée à un centième près est 1.414063.

Que souhaitez-vous faire?

1. Déterminer si une année est bissextile ou non.
2. Retrouver la décomposition en puissances de 2 d'un entier non signé.
3. Calculer la racine carrée d'un nombre réel au moyen d'une recherche dichotomique.
0. Quitter.

Choix : 0
```

Laboratoire 2

Manipulation de Tableaux

L'objectif de cette deuxième séance est de travailler des programmes manipulant des tableaux unidimensionnels. L'objectif est aussi de voir quand les tableaux sont utiles (ou pas).

2.1 Exercices de Base

▷ **Exercice 1** Concevez un programme qui détermine la recherche de la valeur minimale contenue dans un vecteur aléatoire de nombres entiers. Pour ce faire, téléchargez le fichier labo2.c sur eCampus et complétez le. A toutes fins utiles, voici le contenu du fichier `labo2.c`. [1]

```c
#include "/home/algo/lib/util.h"

int main(){
  int *vecteur;
  int n;
  int min;

  printf("Entrez une valeur pour n :\n");
  scanf("%d", &n);

  // Remplit le tableau avec des valeurs aléatoires
  remplir_vecteur(&vecteur, n);

  // AJOUTER VOTRE CODE ICI

  // Imprime sur la sortie standard le contenu du tableau
  afficher_vecteur(vecteur, n);
  printf("%d\n", min);

  return 0;
}//fin programme
```

Dans ce code, `vecteur` est votre tableau de `n` nombres entiers. Vous pouvez donc aller lire l'élément stocké en $(i+1)^e$ position grâce à `vecteur[i]`. Les fonctions `remplir_vecteur()` et `afficher_vecteur()` sont déclarées dans `/home/algo/lib/util.h`. `remplir_vecteur()` initialise le tableau en le remplissant de valeurs entières aléatoires dans l'intervalle $[0, 99]$. `afficher_vecter()` affiche le contenu du tableau sur la sortie standard.

▷ **Exercice 2** Concevez un programme qui détermine la différence entre les deux plus grandes composantes d'un vecteur aléatoire de n nombres entiers. Vous pouvez vous appuyer sur le fichier `labo2.c` pour construire votre code.

1. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

▷ **Exercice 3** Concevez un programme qui affiche la liste des carrés des nombres entiers compris entre 0 et 100 et dont la représentation en base 10 est un palindrome (i.e., se lit de droite à gauche comme de gauche à droite). Pour ce programme, on veillera à ne pas utiliser de tableau.

▷ **Exercice 4** Concevez un programme qui affiche la liste des carrés des nombres entiers compris entre 0 et 100 et dont la représentation en base 10 est un palindrome (i.e., se lit de droite à gauche comme de gauche à droite). Contrairement à l'exercice 3, on pourra ici utiliser un tableau pour stocker la représentation du carré courant. On déterminera, à l'aide de ce tableau, si le carré est un palindrome.

Laboratoire 3

Algorithmique et Tableaux

Cette troisième séance fait suite au Laboratoire 2 en approfondissant le travail sur les tableaux.

Les exercices proposés dans cette séance sont plutôt difficiles. Inutile de commencer à essayer de les résoudre en tapant, directement, le code à l'ordinateur. Au contraire, prenez le temps de bien réfléchir (une feuille blanche, un crayon et une gomme vous seront d'une très grande utilité), en faisant divers dessins et en fournissant un (ou plusieurs) invariant(s) graphique(s). Pensez à bien découper votre problème en sous-problèmes et à documenter scrupuleusement chaque sous-problème.

3.1 Minimum et Maximum

Concevez un programme qui détermine les éléments minimum et maximum d'un tableau de n entiers ($n > 0$) en effectuant au maximum $\frac{3 \times n}{2}$ comparaisons d'éléments [1]. Servez-vous du code suivant [2] :

```c
#include <stdio.h>
#include "/home/algo/lib/util.h"

int main(){
    int *vecteur;
    int n;
    int min;
    int max;

    printf("Entrez une valeur pour n :\n");
    scanf("%d", &n);

    // Remplit le tableau avec des valeurs aléatoires
    remplir_vecteur(&vecteur, n);

    // AJOUTER VOTRE CODE ICI

    // Imprime sur la sortie standard le contenu du tableau
    afficher_vecteur(vecteur, n);
    printf("%d %d\n", min, max);

    return 0;
```

1. Indice si vous êtes bloqués :

C'est $n/2$ itérations et 3 comparaisons par itération qui font $3n/2$ comparaisons en tout.

2. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

```
23 }//fin programme
```

Dans ce code, `remplir_vecteur()` et `afficher_vecteur()` sont déclarés dans `/home/algo/lib/util.h`. La procédure `remplir_vecteur()` permet de remplir avec des valeurs aléatoires un vecteur de taille N. La procédure `afficher_vecteur()` permet d'afficher le vecteur de taille NN.

3.2 Spirale

Concevez un programme qui construit une spirale dans un tableau de $N \times N$ éléments ($N > 0$) comme montré dans l'exemple ci-dessous. Servez-vous du code suivant [3] :

```
1  #include "/home/algo/lib/util.h"
2
3  int main(){
4     int **matrice;
5     int n;
6
7     printf("Entrez une valeur pour n :\n");
8     scanf("%d", &n);
9
10    remplir_matrice_zeros(&matrice, n);
11
12    // AJOUTER VOTRE CODE ICI
13
14    // Affiche le tableau
15    afficher_matrice(matrice, n);
16
17    return 0;
18 }//fin programme
```

Dans ce code, `remplir_matrice_zeros()` et `afficher_matrice()` sont déclarées dans `/home/algo/lib/util.h`. La procédure `remplir_matrice_zeros()` permet d'initialiser toutes les cases du tableau à deux dimensions de taille $n \times n$ à zéro. La procédure `afficher_matrice()` permet d'afficher, sur la sortie standard, le tableau à deux dimensions de taille $n \times n$.

Voici un exemple pour `n = 5` :

```
17 16 15 14 13
18  5  4  3 12
19  6  1  2 11
20  7  8  9 10
21 22 23 24 25
```

Voici un exemple pour `n = 4` :

```
16 15 14 13
 5  4  3 12
 6  1  2 11
 7  8  9 10
```

3. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

3.3 Somme des Carrés

Concevez un programme qui calcule les sommes des $\lceil N/2 \rceil$ « carrés » formés par les éléments des contours de plus en plus internes d'un tableau de dimensions $N \times N$ ($N > 0$) comme montré dans les exemples ci-dessous. Servez-vous du code suivant (le tableau est nommé `matrice` dans le code, les sommes doivent être stockées dans le vecteur `sqsum`)[4] :

```
#include "/home/algo/lib/util.h"

int main(){
   int **matrice;
   int n;
   int *sqsum;

   printf("Entrez une valeur pour n :\n");
   scanf("%d", &n);

   int sq = n/2 + n%2; // taille du tableau sqsum

   remplir_matrice(&array, n);

   remplir_vecteur_zeros(&sqsum, sq);

   // AJOUTER VOTRE CODE ICI

   // Affichage du tableau en 2 dimensions
   afficher_matrice(array, n);
   // Affichage des sommes des carrés
   afficher_vecteur(sqsum, sq);

   return 0;
}//fin programme
```

La procédure `remplir_matrice()` initialise chaque case du tableau de taille $N \times N$ avec une valeur comprise dans l'intervalle [0,99]. La procédure `remplir_vecteur_zeros()` initialise toutes les cases du vecteur à zéro. À nouveau, ces deux procédures sont définies dans `/home/algo/lib/util.h`.

Voici un exemple avec `n = 3` :

```
7 5 7
7 1 6
7 6 3

Sums: 48 1
```

Voici un exemple avec $n = 4$.

```
6 3 1 5
6 0 1 9
8 5 1 6
6 8 6 6

Sums: 70 7
```

En effet, $70 = 6 + 3 + 1 + 5 + 6 + 9 + 8 + 6 + 6 + 8 + 6 + 6$
et $7 = 0 + 1 + 5 + 1$

4. cfr. eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

Laboratoire 4

Structures de Données

Cette quatrième séance se focalise sur les structures de données (`struct`, `enum`, ...) et les chaines de caractères (cas particulier des tableaux).

4.1 Répertoire d'Adresses Électroniques

On souhaite mettre en place un répertoire d'adresses électroniques. Pour ce faire, nous allons procéder en plusieurs étapes.

▷ **Exercice 1** Définissez une structure de données permettant de représenter une *personne*. Les informations relatives à une personne sont son nom, son prénom et son adresse email. Toutes ces informations sont des chaines de caractères.

▷ **Exercice 2** Spécifiez et écrivez une fonction C prenant en argument deux chaines de caractères et retournant une valeur entière non-nulle si ces deux chaines sont égales (i.e., elles possèdent le même contenu), une valeur nulle sinon.

Attention, il n'est pas permis de faire appel, pour résoudre ce problème, à des fonctions d'une quelconque bibliothèque C.

Veillez à envisager tous les cas possibles : les deux chaines peuvent avoir des longueurs maximales différentes et le symbole de fin de chaine (« `\0` ») n'est peut-être pas présent dans l'une ou l'autre des chaines. Il est indispensable de **réaliser un schéma** afin de gérer tous ces cas particuliers.

▷ **Exercice 3** Définissez maintenant une structure de données permettant de représenter un *répertoire*. Les informations nécessaires sont les personnes composant le répertoire et le nombre de personnes contenues dans le répertoire.

▷ **Exercice 4** Spécifiez et écrivez une fonction C permettant de recherche l'adresse électronique d'une personne donnée. Plus précisément, la fonction prendra pour argument un répertoire ainsi que la nom de la personne dont on cherche l'adresse. Si la personne se trouve dans le répertoire, la fonction affichera son email et retournera une valeur non nulle. Sinon, la fonction renverra une valeur nulle.

4.2 Chaines de Caractères en Général

▷ **Exercice 1** Spécifiez et écrivez une fonction C prenant en arguments deux chaines de caractères, et retournant -1 si la première est strictement inférieure à la seconde selon l'ordre lexicographique, 1 si la première strictement supérieure, et 0 si les deux chaines sont égales.

Attention, il n'est pas permis de faire appel, pour résoudre ce problème, à des fonctions d'une quelconque bibliothèque C.

▷ **Exercice 2** Spécifiez et écrivez une fonction C permettant de trier un répertoire sur base de l'ordre lexicographique des noms des personnes.

▷ **Exercice 3** Spécifiez et écrivez une fonction C permettant de rechercher, au moyen d'une recherche dichotomique, l'adresse électronique d'une personne donnée dans un répertoire trié selon l'ordre lexicographique.

4.3 Recherche de Programmes Enregistrés

Le but de cet exercice est de réaliser une application permettant de chercher et d'afficher des programmes (de télévision) ayant été enregistrés et correspondant à certains critères. Les programmes sont d'abord saisis par l'utilisateur qui renseignera, pour chacun d'eux, une date et une heure de diffusion, une catégorie et un titre. Dans un deuxième temps, l'utilisateur pourra effectuer une recherche par date, par catégorie ou par titre sur des programmes précédemment saisis.

Voici un exemple de déroulement du programme (les réponses de l'utilisateur sont soulignées) :

```
Voulez-vous saisir un programme (o/n)? \underline{o}
Date du programme (JJ MM AAAA)?  \underline{08 07 2011}
Heure du programme (HH MM)?  \underline{12 00}
Catégorie du programme (sans espaces)? \underline{serie}
titre du programme (sans espaces)?  \underline{GameofThrones}

Voulez-vous saisir un programme (o/n)? \underline{o}
Date du programme (JJ MM AAAA)?  \underline{08 07 2011}
Heure du programme (HH MM)?  \underline{12 45}
Catégorie du programme (sans espaces)? \underline{serie}
titre du programme (sans espaces)?  \underline{BattleStarGalactica}

Voulez-vous saisir un programme (o/n)? \underline{o}
Date du programme (JJ MM AAAA)?  \underline{10 08 2011}
Heure du programme (HH MM)?  \underline{15 30}
Catégorie du programme (sans espaces)? \underline{documentaire}
titre du programme (sans espaces)?  \underline{DebatParlementaire}

[08/07/2011] (serie) Game of Thrones
[08/07/2011] (serie) BattleStarGalactica
[10/08/2011] (documentaire) DebatParlementaire

Voulez-vous effectuer une recherche (o/n)? \underline{o}
   Une recherche par Date (0), Categorie (1), Titre (2)? \underline{0}
      Date à chercher (JJ MM AAAA)?  \underline{08 07 2011}

(serie) Game of Thrones
(serie) BattleStarGalactica

Voulez-vous effectuer une recherche (o/n)? \underline{n}
```

▷ **Exercice 1** Définissez une structure de données adaptée (cfr. exemple supra) permettant de mémoriser une date (trois entiers) et l'heure (deux entiers).

▷ **Exercice 2** Définissez une structure de données adaptée permettant de prendre en compte les différentes catégories d'un programme télévisuel.

Seules les catégories suivantes sont envisageables : série, documentaire, journal télévisé, jeu, film.

▷ **Exercice 3** Définissez une structure de données permettant d'encoder un programme télévisuel. Pour rappel, les informations relatives à un programme sont : une date et une heure de diffusion, une catégorie et un titre.

▷ **Exercice 4** Définissez une structure de données permettant de maintenir en mémoire les différents programmes. On peut considérer que le système ne peut se souvenir que d'un certain

nombre, maximum, de programme télévisuel (disons 20). Votre structure de données doit donc contenir tous les programmes enregistrés et un entier indiquant combien de programmes sont maintenus en mémoire.

▷ **Exercice 5** Spécifiez et écrivez les fonctions C permettant de gérer les divers menus apparaissant à l'écran.

▷ **Exercice 6** Spécifiez et écrivez une fonction C qui permet d'encoder en mémoire un programme entré par l'utilisateur. Cette fonction renvoie une valeur entière non-nulle si l'encodage a réussi, une valeur nulle sinon.

▷ **Exercice 7** Spécifiez et écrivez les fonctions C nécessaires à la recherche d'un programme télévisuel.

Laboratoire 5

Allocation Dynamique

Cette dernière séance s'organise en deux temps. Tout d'abord, on vous demande de créer un programme qui peut prendre des arguments (Sec. 5.1). Ensuite, on va revenir sur les notions de structures de données, et en particulier la création dynamique de structure de données (Sec. 5.2 et 5.3).

5.1 Argc/Argv

Écrivez un programme qui prend comme paramètres du programme une expression arithmétique en notation infixe, formée de deux opérandes entiers et d'un opérateur (+, −, ×, /), et qui affiche sur la sortie standard le résultat de son évaluation. Par exemple, si le programme exécutable s'appelle `calc`, on aura :

```
1 $> ./calc 3 x 5
```

Le résultat, à l'écran, sera

```
3 x 5 = 15
```

Attention ! N'utilisez pas le caractère `*` pour indiquer une multiplication : celui-ci est interprété différemment par le shell. En effet, si votre dossier courant contient les fichiers suivant :

```
calc
exo1.c
exo2.c
```

Invoquer

```
1 $ >./calc 3 * 5
```

sera traduit par le shell :

```
1 $> ./calc 3 calc exo1.c exo2.c 5
```

vu que `*` représente tous les fichiers dont le nom comprend *n'importe quel caractère* – rappelez-vous de l'effet de « `rm *` » ! Dans cet exemple, `argv[2]` contient alors la chaine `calc`, `argv[3]` contient `exo1.c`, etc.

5.2 Jeu de Rôle

Dans un jeu de rôle, un personnage est représenté par un nom, des attributs de base (force, dextérité, magie, vitalité, esprit, chance), une arme et une armure équipées d'une liste de compétences ou « materia ». Les armes peuvent augmenter les attributs force, dextérité et magie. Quant aux armures, elles augmentent les attributs vitalité, esprit et chance. Les armes comme les armures contiennent un certain nombre de slots materia qui représentent le nombre maximal de materia équipables à un objet. Chaque materia équipée sur une arme ou une armure fournit un bonus de force et de magie. Ces structures sont définies ci-dessous :

```
#define MAX_NAME 64
#define MAX_SLOTS 8

typedef struct materia_t{
  char name[MAX_NAME];
  int str, mag;
}Materia;

typedef struct weapon_t{
  char name[MAX_NAME];
  int str, dex, mag;
  int slots;
}Weapon;

typedef struct armor_t{
  char name[MAX_NAME];
  int vit, sp, luck;
  int slots;
}Armor;

typedef struct character_t{
  char name[MAX_NAME];
  Weapon *weapon;
  Armor *armor;
  Materia *mat_w[MAX_SLOTS];
  Materia *mat_a[MAX_SLOTS];
  int str, dex, mag, vit, sp, luck;
}Character;
```

Des listes de materias, d'armes et d'armures sont disponibles dans le répertoire `/home/algo/rpg` [1] dans les fichiers `materia.dat`, `weapon.dat` et `armor.dat` respectivement. Il vous est demandé de concevoir un programme qui fournit les fonctionnalités suivantes :

- écrire un personnage dans un fichier. *Attention ! Il est inutile de stocker les adresses des armes, armures et matérias : mieux vaut stocker leurs positions dans leurs tableaux respectifs* ;
- lire un personnage à partir d'un fichier ;
- afficher à l'écran les données d'un personnage, c'est-à-dire son nom, son arme, son armure, ses attributs finaux ainsi que ses materias. La valeur finale d'un attribut est calculée en sommant la valeur de base du personnage, le bonus donné par son équipement ainsi que celui fournit par les matérias équipant son arme ou son armure.

Chaque materia dans `materia.dat` est encodée sur 2 lignes. Une première ligne contient le nom et une deuxième contient les bonus force et magie séparés par un espace. Les armes

1. En outre, ce dossier contient le fichier `rpg.h` qui contient les définitions de structures ci-dessus. Les fichiers sont aussi disponibles sur eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos

dans `weapon.dat` sont encodées sur 3 lignes. Une première ligne contient le nom, une deuxième contient les ajouts de force, de dextérité et de magie séparés par des espaces et une troisième contient le nombre de slots materia. Un format d'encodage similaire est utilisé pour les armures dans `armor.dat`. Notez bien que la première ligne des fichiers `.dat` contient le nombre d'éléments décrits dans ce fichier.

Rappel sur `scanf` et ses arguments : Pour lire, depuis `stdin` une série de n'importe quel caractère, excepté le retour à la ligne et au maximum X caractères et stocker le tout dans une chaine `str`, utilisez la commande ci-dessous (dans notre exemple, $X = 64$) :

```
scanf("␣%64[^\n]", str);
```

Demandez de l'aide à l'équipe pédagogique si vous ne comprenez pas cette commande ou consultez les pages de manuels.

5.3 Nombres Complexes

Rappel sur les nombres complexes
Tout nombre complexe z admet une représentation *cartésienne*, $x + iy$, et une représentation *polaire*, $\rho e^{i\theta}$ où ρ est le *module* et θ l'*argument* de z. L'argument n'est défini que si $z \neq 0$. Le passage d'un système de coordonnées à l'autre se fait à l'aide des formules de conversion suivantes :

Soit la définition de structure de données suivantes décrivant un nombre complexe (la définition se trouve dans le fichier d'en-tête `complexe.h`, disponible sur eCampus, Sec. Répétitions/Exercices/Fichiers utilisés aux labos) :

```
typedef struct complexe_t{
  double reel;
  double img;
}Complexe;
```

Pour les exercices qui suivent, n'oubliez pas de séparer les interfaces de l'implémentation des différentes fonctions.

▷ **Exercice 1** Spécifiez et écrivez la fonction qui permet d'écrire, à l'écran, sous la forme (r, i) un nombre complexe passé en paramètre.

▷ **Exercice 2** Spécifiez et écrivez une fonction "constructeur" qui retourne un pointeur sur nombre complexe initialisé aux valeurs des parties réelle et imaginaire passées en paramètre. Ce constructeur retourne `NULL` en cas d'erreur.

▷ **Exercice 3** Spécifiez et écrivez les fonctions C qui retournent les valeurs des parties réelle et imaginaire d'un nombre complexe passé en argument.

▷ **Exercice 4** Spécifiez et écrivez la fonction C qui calcule le module d'un nombre complexe passé en paramètre.

▷ **Exercice 5** Spécifiez et écrivez la fonction C qui calcule l'argument d'un complexe passé en paramètre. Attention, la fonction `atan` (présente dans `math.h`) est définie de $\mathbb{R}$ vers $]-\frac{\pi}{2}, \frac{\pi}{2}[$. Pensez à utiliser la fonction `atan2` qui règle ce problème.

▷ **Exercice 6** Spécifiez et écrivez une fonction C qui possède deux paramètres de type réel représentant le module et l'argument d'un complexe polaire et qui renvoie un complexe en coordonnées cartésiennes.

▷ **Exercice 7** Spécifiez et écrivez les fonctions C qui effectuent la somme, la différence et le produit de deux nombres complexes. Ces fonctions C retournent un complexe résultant de l'opération mathématique.

Notez que le produit de deux nombres complexes est plus simple à écrire en utilisant les coordonnées polaires :

$$\begin{aligned} \rho z_1 \times z_2 &= \rho z_1 \times \rho z_2 \\ \theta(z_1 \times z_2) &= \theta z_1 \times \theta z_2 \end{aligned}$$

▷ **Exercice 8** Spécifiez et écrivez les fonctions qui testent l'égalité et la différence entre deux complexes donnés en argument.

▷ **Exercice 9** Écrivez un programme qui teste vos différentes fonctions.