Introduction à la programmation et aux algorithmes

GUIDE DE L'ÉTUDIANTE ET DE L'ÉTUDIANT
S1 – APP2sntetm (GEGIGRO)

Automne 2022

Département de génie électrique et de génie informatique
Faculté de génie
Université de Sherbrooke

Auteur : D. Palao Munõz, A. Huppé, E. Morin, A. Marchese et C.-A. Brunet
Version : A22-01 (2022-09-21)

Ce document est réalisé avec l'aide de LaTeX et de la classe `gegi-app-guide`.

TABLE DES MATIÈRES

1 ACTIVITÉS PÉDAGOGIQUES ET COMPÉTENCES 1

2 SYNTHÈSE DE L'ÉVALUATION 2

3 QUALITÉS DE L'INGÉNIEUR 3

4 ÉNONCÉ DE LA PROBLÉMATIQUE 4

5 CONNAISSANCES NOUVELLES 6

6 GUIDE DE LECTURE 8

7 LOGICIELS ET MATÉRIEL 9

8 SANTÉ ET SÉCURITÉ 10

9 SOMMAIRE DES ACTIVITÉS 11

10 PRODUCTIONS À REMETTRE 12

11 ÉVALUATIONS 15

12 POLITIQUES ET RÈGLEMENTS 17

13 INTÉGRITÉ, PLAGIAT ET AUTRES DÉLITS 18

14 PRATIQUE PROCÉDURALE 1 19

15 PRATIQUE PROCÉDURALE 2 22

16 PRATIQUE EN LABORATOIRE 1 25

17 PRATIQUE PROCÉDURALE 3 27

18 PRATIQUE EN LABORATOIRE 2 28

19 VALIDATION AU LABORATOIRE 30

A DESCRIPTION DES OPÉRATIONS 31

B PLAN DE TESTS 33

C NOTIONS MATHÉMATIQUES 34

D BONNES PRATIQUES DE PROGRAMMATION 35

LISTE DES RÉFÉRENCES 38

LISTE DES FIGURES

14.1 Diagramme d'activités mystère de l'exercice P1.E5 20

LISTE DES TABLEAUX

2.1 Sommaire de l'évaluation de l'unité 2
2.2 Calcul d'une cote et d'un niveau d'atteinte d'une qualité 2

11.1 Sommaire de l'évaluation du rapport et des livrables associés 15
11.2 GEN146 : Grille d'indicateurs utilisée pour les évaluations 16

B.1 Exemple de plan de tests sous forme de tableau 33

1 ACTIVITÉS PÉDAGOGIQUES ET COMPÉTENCES

GEN146 – Introduction à la programmation et aux algorithmes

1. Faire la synthèse et l'implémentation d'un algorithme en vue de résoudre un problème selon une approche procédurale.
2. Développer un logiciel composé d'un programme principal et de fonctions sur la base d'un algorithme spécifié de complexité élémentaire en exploitant un système de développement de programme avec interface graphique.

Description officielle : https://www.usherbrooke.ca/admission/fiches-cours/GEN146

2 SYNTHÈSE DE L'ÉVALUATION

La note attribuée aux activités pédagogiques de l'unité est une note individuelle. L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques à la section 1. La pondération de chacune d'entre elles dans l'évaluation de cette unité est donnée au tableau 2.1.

Tableau 2.1 : Sommaire de l'évaluation de l'unité

Évaluation	GEN146-1	GEN146-2
Rapport et livrables associés	40	30
Validation		10
Évaluation sommative théorique	120	10
Évaluation sommative pratique		110
Évaluation finale théorique	140	50
Évaluation finale pratique		90
Total	300	300

À moins de circonstances exceptionnelles, une cote ou un niveau d'atteinte d'une *qualité* est calculé à partir du tableau 2.2. Les grilles des *indicateurs* utilisées pour les évaluations sont données au tableau 11.2.

Tableau 2.2 : Calcul d'une cote et d'un niveau d'atteinte d'une qualité

Note(%)	<50	50	53	57	60	64	68	71	75	78	81	85
Cote	E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
Niveau	N0	N1	N1	N1	N2	N2	N2	N3	N3	N3	N4	N4
Libellé	Insuffisant	Passable (seuil)			Bien			Très bien (cible)			Excellent	

3 QUALITÉS DE L'INGÉNIEUR

Les qualités de l'ingénieur visées et évaluées par cette unité d'APP sont données dans le tableau un peu plus bas. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant : qualités et BCAPG

Qualité	Libellé	Touchée	Évaluée
Q01	Connaissances en génie	✓	✓
Q02	Analyse de problèmes	✓	✓
Q03	Investigation		
Q04	Conception		
Q05	Utilisation d'outils d'ingénierie	✓	✓
Q06	Travail individuel et en équipe		
Q07	Communication		
Q08	Professionnalisme		
Q09	Impact du génie sur la société et l'environnement		
Q10	Déontologie et équité		
Q11	Économie et gestion de projets		
Q12	Apprentissage continu		

4 ÉNONCÉ DE LA PROBLÉMATIQUE

Librairie de fonctionnalités

Vous êtes engagé pour faire un stage chez XY-calculus, une firme spécialisée dans les calculs mathématiques de toute sorte. En effet, l'informatique a toujours été fortement liée aux mathématiques. Au fil du temps, la compagnie a développé une librairie des formules mathématiques impliquées dans les calculs. La compagnie compte sur un groupe de personnes spécialisées dans l'analyse et la création des algorithmes demandés par les clients. Voici une liste non exhaustive des opérations mathématiques développées par la compagnie :

Arithmétique (complexité basse)
: Addition, soustraction, multiplication, division et modulo.

Géométrie (complexité basse)
: Distance euclidienne, circonférence et aire d'un cercle, aire et volume d'une sphère.

Chaines de caractères (complexité basse)
: Longueur, inversion, recherche, détection de palindrome et compteur de mots.

Résolution d'équation (complexité moyenne)
: Résolution d'équations du deuxième degré.

Trigonométrie à partir de séries (complexité moyenne)
: Calcul du sinus, du cosinus et de la tangente.

Traitement de matrices : (complexité haute)
: Détection d'une matrice diagonale, unitaire et triangulaire supérieure, addition et soustraction, multiplication par un nombre, transposition, multiplication et calcul du déterminant.

Dernièrement, la compagnie a décidé de changer sa philosophie afin de réduire sa dépendance à des librairies externes et afin de réduire significativement la taille de ses livrables, car les clients les utilisent de plus en plus sur des systèmes embarqués. Pour atteindre ce but, les opérations doivent donc maintenant être réalisées sans l'usage de librairies standards ou externes ou de variables globales.

Chaque opération est répertoriée et soigneusement documentée. Comme la compagnie subit une croissance importante de clients, elle voudrait utiliser une démarche ordonnée et contrôlée afin d'assurer le succès du changement. Après quelques rencontres, le directeur a décidé que pour faire l'analyse de chaque opération une notation standardisée sera utilisée et il faudra créer le pseudocode et le diagramme d'activités UML correspondant. Cette

notation permettra à différents programmeurs de faire le codage dans une étape ultérieure. De plus, le directeur aimerait valider s'il existe une correspondance entre les différentes représentations : le pseudocode, les diagrammes d'activités et le code programmé.

La compagnie aimerait produire du code modulaire et facile d'entretien, le langage de programmation sélectionné est le langage C et l'environnement de travail (IDE) est Geany en utilisant le compilateur gcc. Un des consultants experts en programmation a parlé du besoin de préparer de bons plans de tests pour assurer un bon niveau de qualité dans les programmes à produire. Ainsi, chaque opération avec ses tests sera réalisée dans un fichier. Les tests devront être automatisés et donc se dérouler sans aucune intervention de l'usager. Les opérations elles-mêmes ne font donc pas d'affichage ou de sollicitation de l'usager. Le respect des spécifications de chacune des opérations est donc très important.

Vous arrivez donc chez XY-calculus et vous êtes responsable de valider la faisabilité du changement de philosophie. On ne vous demande pas de réaliser toutes les opérations, mais quelques-unes de différents niveaux de complexité. Ce sont deux opérations de complexité basse, de deux opérations de complexité moyenne et deux opérations de complexité haute. Ces opérations sont les suivantes :

- complexité basse : recherche d'un caractère et détection de palindrome
- complexité moyenne : calcul du sinus et calcul du cosinus à l'aide de séries
- complexité haute : addition de matrices et multiplication de matrices carrées

Pour ces opérations, vous devez faire le pseudocode, le diagramme d'activités, le code en langage C et les tests automatisés. Vous devez aussi fournir un plan de tests pour chacune de ces opérations dont les spécifications détaillées sont donnés à l'annexe A. Le plan de tests n'est pas nécessairement exhaustif, mais il faut trouver les cas limites et valider que le programme est capable de bien les gérer. Vous devez utiliser les concepts de précondition (préalable) et de postcondition.

5 CONNAISSANCES NOUVELLES

Connaissances déclaratives (quoi)

- Pseudocode
  - Préconditions et postconditions
  - Commentaires
  - Énoncés, opérateurs et blocs
  - Variables
  - Opérations d'entrées et de sorties
  - Énoncés de sélection
  - Énoncés de répétition
  - Membres de structures et de classes
  - Fonctions et procédures
- Diagrammes d'activités UML
  - Point de départ et de finalisation
  - Activités
  - Transitions
  - Gardes
  - Point de jonction
  - Note
- Notion de programme
  - composition, compilation et exécution d'un programme.
- Le langage C
  - Afficher des informations à l'écran.
  - Obtenir des données à partir du clavier.
  - Directive `#include`.
  - Variables : types de données, assignation et initialisation.
  - Opérateurs : arithmétiques, incrémentation, décrémentation, affectation, relationnels, égalité, logiques et résolution de portée.
  - Priorité et associativité des opérateurs.
  - Commentaires.
  - Expressions mathématiques et leur évaluation.
  - Structures de contrôle : séquence, sélection et répétition.
  - Chaines de caractères.
  - Bibliothèques : entrées et sorties, fonctions mathématiques et autres.

    - Portée des variables.
    - Fonctions, passage de paramètres par référence et par valeur.
    - Tableaux.
- Développer des logiciels.
    - Utiliser un environnement de développement.
    - Écrire des séquences d'instructions.
    - Utiliser les énoncés conditionnels.
    - Utiliser les structures de répétition.
    - Déclarer, définir et appeler une fonction.
    - Développer un logiciel en langage C composé d'un programme principal et de fonctions sur la base d'un algorithme.

Connaissances procédurales (comment)

- Utiliser des diagrammes d'activités UML comme aide à concevoir des programmes.
- Utiliser du pseudocode comme aide à concevoir des programmes.
- Tester et valider un algorithme.
- Utiliser un environnement de développement pour programmer un logiciel.
- Écrire un programme en langage C pour résoudre des problèmes de complexité élémentaire.
- Tester et valider un programme.

Connaissances conditionnelles (quand)

- Choisir les bons éléments des diagrammes UML pour concevoir un programme.
- Choisir les bons éléments de pseudocode pour concevoir un programme.
- Choisir les bonnes instructions afin d'implémenter un programme.
- Utiliser une approche modulaire pour la construction de programmes à l'aide de fonctions.

6 GUIDE DE LECTURE

6.1 Références essentielles

Livre du langage C : Les chapitres suivants du livre de référence [1] seront abordés dans les activités de l'APP : 2, 3, 4, 5, 6, 7, 8 et 10 (sauf les pages 221 à 226 sur la fonction `malloc`) et 12.

Guide de l'étudiant pour le pseudocode : ce document montre la manière standard de produire le pseudocode pour la session. Il est disponible sur la page web de l'unité (GEGI/GRO).

Guide de l'étudiant pour les diagrammes d'activités : Ce document montre la manière de construire des diagrammes d'activités en UML. Il est disponible sur la page web de l'unité (GEGI/GRO).

Aide-mémoire du langage C : Ce petit document résume l'essentiel de la syntaxe du langage C. Il est disponible sur la page web de l'unité (GEGI/GRO) et il vous sera fourni lors des examens pratiques.

6.2 Documents complémentaires

Les documents suivants sont offerts sur la page web de l'unité (GEGI/GRO) et peuvent vous être utiles.

Débogage de base : ce document explique quelques techniques de base pour la recherche et la correction de bogues que vous pourriez trouver dans vos programmes.

6.3 Guide de lecture

Procédural 1 : la lecture préalable du guide de pseudocode et du guide de diagrammes d'activités, mentionnés à la section 6.1, est obligatoire avant cette activité. Pour un procédural, lorsqu'il y a des exercices préparatoires, ce sont des exercices facultatifs que vous pouvez réaliser avant le procédural.

Procédural 2 et laboratoire 1 : la lecture préalable des chapitres 2, 3, 4, 5, 6, 7 et 12 du livre de référence [1] est obligatoire avant cette activité. Pour un procédural, lorsqu'il y a des exercices préparatoires, ce sont des exercices facultatifs que vous pouvez réaliser avant le procédural.

Procédural 3 et laboratoire 2 : la lecture préalable des chapitres 8 et 10 (sauf les pages 221 à 226 sur la fonction `malloc`) du livre de référence [1] est obligatoire avant cette activité.

7 LOGICIELS ET MATÉRIEL

Programmation : Geany est l'environnement de développement utilisé pour programmer en langage C dans cette unité d'APP. Geany ainsi que MinGW, pour utiliser le compilateur gcc, sont installés dans les laboratoires du département. Pour installer ces outils sur vos ordinateurs personnels, consultez la section des logiciels offerts du site web de S1 (GEGI/GRO).

Diagrammes d'activités : Tout bon logiciel de dessins peut être utilisé pour faire un diagramme d'activités. Vous pouvez utiliser un logiciel gratuit et adapté pour UML, comme UMLet et Dia qui sont offerts sur internet, ou encore un logiciel de dessin à tout faire, comme Visio. Assurez-vous que les diagrammes créés sont conformes au standard UML ! Parmi ces logiciels, seul Visio est installé dans les laboratoires du département.

Pseudocode : Vous pouvez utiliser le logiciel qui vous voulez pour écrire le pseudocode. Vous pouvez utiliser un logiciel de traitement de texte, comme Word, ou encore utiliser un éditeur de texte, comme Notepad++, qui s'adapte bien pour la création de pseudocode. Word et Notepad++ sont installés dans les laboratoires du département.

8 SANTÉ ET SÉCURITÉ

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques et directives concernant la santé et la sécurité. Ces documents sont disponibles sur les sites web de l'Université de Sherbrooke, de la Faculté de génie et du département. Les principaux sont mentionnés ici et sont disponibles dans la section *Santé et sécurité* du site web du département.

- Politique 2500-004 : Politique de santé et sécurité en milieu de travail et d'études
- Directive 2600-042 : Directive relative à la santé et à la sécurité en milieu de travail et d'études
- Sécurité en laboratoire et atelier au département de génie électrique et de génie informatique

9 SOMMAIRE DES ACTIVITÉS

Semaine 1

- Première rencontre de tutorat
- Étude personnelle et exercices

Semaine 2

- Étude personnelle et exercices
- Formation à la pratique procédurale 1
- Formation à la pratique procédurale 2
- Formation à la pratique au laboratoire 1
- Formation à la pratique procédurale 3
- Formation à la pratique au laboratoire 2

Semaine 3

- Laboratoire avec support
- Étude personnelle et exercices
- Validation pratique de la solution
- Rédaction finale du rapport
- Production du schéma de concepts
- Deuxième rencontre de tutorat
- Évaluation formative
- Consultation générale
- Évaluation sommative

10 PRODUCTIONS À REMETTRE

- Les livrables sont réalisés en équipe de 2. La même note sera attribuée à chaque membre de l'équipe.
- L'identification des membres de l'équipe doit être faite avant 16h00, le lendemain du tutorat d'ouverture (jour ouvrable). Passé cette date limite, les personnes seules ou dans des équipes ne respectant pas les contraintes de formation d'équipe peuvent être affectées à une équipe par la tuteure ou le tuteur. Le cas échéant, cette assignation est sans appel.
- La date limite du dépôt des livrables est 08h00 (pas 20h00), le jour du tutorat de fermeture. Tout retard entraine une pénalité immédiate de 20% et de 20% par jour supplémentaire.
- La remise des livrables se fait **uniquement avec l'outil de dépôt** du département. Conséquemment, toute remise de livrables, complète ou partielle, faite autrement que par ce moyen sera ignorée.
- L'outil de dépôt ne permet qu'un seul dépôt. Si vous voulez faire un autre dépôt, il faut contacter un tuteur ou une tuteure avant la date limite.
- Le non-respect des directives et des contraintes, comme le nom d'un répertoire ou d'un fichier, peut entrainer des pénalités.
- Seules les collaborations intraéquipe sont permises. Cependant, vous devez résoudre la problématique de façon individuelle pour être en mesure de réussir les évaluations.
- Les productions soumises à l'évaluation doivent être originales pour chaque équipe, sinon l'évaluation sera pénalisée.

10.1 Schéma de concept

Le schéma de concept est une production individuelle optionnelle en vue de la deuxième rencontre de tutorat. Le schéma de concepts à faire lors de l'étude personnelle cible la question suivante :

> *Comment les différents éléments du pseudocode, d'un diagramme d'activités et du langage C permettent-ils de représenter un algorithme, ses comportements et sa structure pour développer un logiciel ?*

10.2 Rapport et livrables associés

Les livrables sont un rapport en format PDF et le code en langage C produit en réponse à la problématique. Ces deux parties sont détaillées dans les paragraphes qui suivent.

Rapport

Le rapport est réalisé dans un seul fichier et le nom et le CIP de tous les membres de l'équipe doivent être présents sur la page couverture. Le rapport contient la représentation des six opérations demandées dans l'énoncé de la problématique dans un des deux formalismes, pseudocode ou diagramme d'activités, et aussi son code en langage C et son plan de tests. Pour trois des six opérations, il faut fournir le pseudocode et pour les trois autres, le diagramme d'activités. Quelles opérations doivent être données en pseudocode ou en diagramme d'activités ? C'est votre choix. Des descriptions plus détaillées des opérations et des explications sur ce qu'est un plan de tests sont données respectivement aux annexes A et B.

Si une opération est réalisée avec l'aide d'autres opérations ou fonctions, alors il faut aussi en fournir les détails (pseudocode ou diagramme d'activités et code en langage C). Par exemple, si l'opération `sinus` utilise une autre opération, comme `factorielle`, alors il faut aussi en fournir les détails, et ainsi de suite. Il n'est pas nécessaire de fournir les détails de la fonction `main` ou de toutes autres fonctions appelées du `main` et qui ne sont pas reliées à la réalisation d'une des opérations. Par exemple, il ne faut pas fournir les représentations des fonctions qui réalisent et automatisent des tests.

La présentation dans le rapport se fait sur deux colonnes afin de pouvoir aisément comparer les représentations : pseudocode versus langage C ou diagramme d'activités versus langage C. Il faut s'assurer que le tout est lisible et, autant que possible, sur une même page pour une opération donnée. Si nécessaire, pour cette partie du rapport, l'orientation de la page (portrait ou paysage) et la dimension de la page peuvent être ajustées.

Code source

Tous les fichiers de code source (fichiers de langage C) remis, un total de six, doivent être dans un répertoire nommé `Code`. Comme mentionné dans la problématique, le code qui réalise les six opérations est réalisé **sans l'usage de librairies, comme `string.h` ou `math.h`, ou de variables globales**. Les autres parties du code qui ne sont pas reliées à la réalisation des opérations peuvent utiliser des librairies. Par exemple, le `main` pourrait utiliser des fonctions de librairies afin de faciliter les tests.

Les fonctions qui réalisent les opérations doivent respecter les spécifications qui sont données à l'annexe A. Cela inclut de respecter le nombre et le type des paramètres et, le cas échéant, de la valeur de retour. Comme chaque opération est réalisée dans un fichier, il y a une fonction `main` pour chacune qui sert à appeler à plusieurs reprises l'opération réalisée afin de la tester en accord avec votre plan de tests.

Les tests doivent se dérouler sans que l'usager ait à fournir de valeur. Comme annoncé dans la problématique, les tests sont automatisés. Dans le code remis, les tests effectués dans le `main` sont ceux de votre plan de tests

Le code source de chaque fichier doit respecter les bonnes pratiques de programmation, telles que décrites dans l'annexe D. En particulier, chaque fichier doit être identifié avec un entête de fichier, en prenant soin d'y mettre à jour les informations, et une fonction ne doit faire qu'une seule tâche.

Remise des productions

Vous devez produire une archive zip dont le contenu est mentionné plus haut (un fichier PDF pour le rapport et un répertoire nommé `Code` contenant les six fichiers de code). Vous devez nommer l'archive zip en accord avec les CIP des membres de l'équipe, par exemple `CIP1-CIP2.zip`. Notez le tiret entre les différents CIP. L'outil de dépôt n'accepte pas le dépôt si le CIP de la personne faisant le dépôt n'est pas dans le nom du fichier. L'archive zip est téléversée en utilisant l'outil de dépôt du département et qui est accessible avec le lien *Dépôt des travaux* sur le site web de S1 (GEGI/GRO). Ce lien vous mène à la page principale pour les dépôts électroniques des travaux des différentes sessions. Choisissez le bon trimestre, la bonne session et la bonne unité d'APP. L'outil de dépôt ne permet qu'un seul dépôt. Si vous voulez faire un deuxième dépôt, vous devez contacter un des tuteurs afin qu'il retire votre premier dépôt.

11 ÉVALUATIONS

La synthèse des évaluations est donnée à la section 2 et certains détails sont donnés ici.

11.1 Rapport et livrables associés

Cette évaluation porte sur les compétences figurant dans la description des activités pédagogiques (section 1) et la pondération est indiquée au tableau 11.1. L'évaluation est directement liée aux livrables demandés à la section 10.2 et à la séance de validation.

Tableau 11.1 : Sommaire de l'évaluation du rapport et des livrables associés

Élément	GEN146-1	GEN146-2
Conception détaillée (diagrammes et pseudocode)	40	
Implémentation (langage C)		20
Respect des spécifications		5
Plan de tests et tests		5
Les éléments suivants sont évalués pendant la validation :		
Technique et outils		5
Exécution		5
Diagrammes d'activités et pseudocode	(évaluation formative)	
Total	40	40

La qualité de la communication technique n'est pas évaluée de façon sommative, mais si votre rapport est fautif sur le plan de la qualité de la communication et de la présentation, des pénalités pourraient être données ou encore, il pourrait vous être retourné afin que vous le repreniez pour être noté.

11.2 Évaluation sommative et évaluation finale

L'évaluation sommative et l'évaluation finale ont une partie théorique écrite sur papier et une partie pratique sur ordinateur. Ce sont des évaluations qui portent sur tous les éléments de compétences de l'unité mentionnées dans la section 1. La seule documentation permise est l'aide-mémoire du langage C pendant les évaluations pratiques.

11.3 Qualités de l'ingénieur

La grille d'indicateurs utilisée aux fins de l'évaluation est donnée au tableau 11.2. Il est à noter qu'un niveau d'atteinte d'un *indicateur* dans cette grille n'a pas la même signification qu'un niveau d'atteinte d'une *qualité* dans le tableau 2.2. Cela est normal, un indicateur et une qualité, ce sont deux choses différentes.

Tableau 11.2 : GEN146 : Grille d'indicateurs utilisée pour les évaluations

Indicateur	Qualité	Aucun (N0)	Insuffisant (N1)	Seuil (N2)	Cible (N3)	Excellent (N4)
Formalismes de conception	Q01.1	... ne démontre pas la compréhension des formalismes de conception.	... démontre une compréhension insuffisante des formalismes de conception.	... démontre une compréhension minimale des formalismes de conception.	... démontre une bonne compréhension des formalismes de conception.	... démontre une excellente compréhension des formalismes de conception.
Faire la conception détaillée	Q02.2	... ne démontre pas la capacité à faire une conception détaillée.	... définit une conception détaillée ne permettant pas l'implémentation directe à l'aide d'un langage de programmation.	... définit une conception détaillée comportant des écarts marqué au standard, mais permettant l'implémentation à l'aide d'un langage de programmation.	... définit une conception détaillée déquatement, mais avec quelques écarts au standards, et permettant l'implémentation à l'aide d'un langage de programmation.	... definit une conception détaillée selon les standards et les règles de l'art et permettant l'implémentation à l'aide d'un langage de programmation.
Valider la solution	Q02.3	... ne démontre pas la capacité à valider une solution.	... valide inadéquatement ou de manière non pertinente que la solution répond aux exigences.	... valide minimalement que la solution répond aux exigences.	... valide que la solution répond aux exigences, sauf dans certains cas non critiques.	... valide complètement que la solution répond aux exigences à l'aide de tests couvrant les préconditions, postonditions et cas limites.
Notions du langage C	Q01.1	... ne démontre pas une compréhension des notions du langage C.	... démontre une compréhension insuffisante des notions du langage C.	... démontre une compréhension minimale des notions du langage C.	... démontre une bonne compréhension des notions du langage C.	... démontre une excellente compréhension des notions du langage C.
Utilisation du langage C	Q01.2	... ne démontre pas la capacité d'utilisation du langage C.	... n'est pas en mesure d'utiliser le langage C de manière adéquate.	... sait utiliser le langage C, mais démontre de la difficulté dans certains cas.	... sait utiliser le langage C efficacement sauf dans certains cas particuliers.	... sait utiliser efficacement le langage C.
Implémenter la solution	Q01.2	... ne démontre pas la capacité d'élaborer un programme.	... élabore un programme qui ne répond pas aux exigences.	... élabore un programme fonctionnel qui répond partiellement aux exigences.	... élabore un programme fonctionnel qui répond aux exigences.	... élabore un programme fonctionnel qui répond aux exigences et respecte les bonnes pratiques.
Valider la solution	Q01.2	... ne démontre pas la capacité de valider une solution.	... valide inadéquatement ou de manière non pertinente que la solution répond aux exigences.	... valide minimalement que la solution répond aux exigences.	... valide que la solution répond aux exigences, sauf dans certains cas non critiques.	... valide complètement que la solution répond aux exigences à l'aide de tests couvrant les préconditions, postonditions et cas limites.
Utiliser les techniques et outils sélectionnés selon les protocoles établis	Q05.2	... ne démontre pas la capacité d'utiliser les techniques et outils spécifiés.	... ne démontre pas suffisamment la capacité d'utiliser les techniques et outils spécifiés.	... connait et est en mesure d'utiliser avec assistance mineure les techniques et outils spécifiés.	... connait et est en mesure d'utiliser les techniques et outils spécifiés.	... connait et est en mesure d'utiliser efficacement les techniques et outils spécifiés.

12 POLITIQUES ET RÈGLEMENTS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques, règlements et normes d'agrément ci-dessous :

Règlements de l'Université de Sherbrooke

- Règlement des études

Règlements facultaires

- Règlement facultaire d'évaluation des apprentissages / Programmes de baccalauréat
- Règlement facultaire sur la reconnaissance des acquis

Normes d'agrément

- Processus d'agrément et qualités du BCAPG
- Ingénieurs Canada – Àpropos de l'agrément

Enfin, si vous êtes en situation de handicap, assurez-vous d'avoir communiqué avec le *Programme d'intégration des étudiantes et étudiants en situation de handicap* à l'adresse : prog.integration@usherbrooke.ca.

13 INTÉGRITÉ, PLAGIAT ET AUTRES DÉLITS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance de la page Intégrité intellectuelle des Services à la vie étudiante.

14 PRATIQUE PROCÉDURALE 1

Buts de l'activité

- Connaitre une méthode d'analyse pour produire un logiciel de qualité.
- Être capable de concevoir un algorithme.
- Écrire du pseudocode selon un standard.
- Créer des diagrammes d'activités selon le standard UML.

Suggestions

- Pour cette activité, il est avantageux d'avoir accès au guide de pseudocode et au guide de diagrammes d'activités.
- Pour les exercices, il est assumé que l'utilisateur fournit le bon type de donnée. Par exemple, lorsque l'utilisateur est sollicité pour un entier, il donne un entier.

14.1 EXERCICES PRÉPARATOIRES

P1.P1 Activités pour venir à l'université

Décrire de manière détaillée toutes les activités réalisées à partir du moment du réveil jusqu'à l'arrivée à un procédural qui a lieu le matin.

14.2 EXERCICES

P1.E1 La moyenne

Écrire le pseudocode d'un algorithme qui demandera deux valeurs numériques entières à l'utilisateur et qui calcule ensuite la moyenne. Une fois la moyenne calculée, l'algorithme doit afficher la valeur à l'utilisateur. L'algorithme doit être le plus détaillé possible. Faire ensuite le diagramme d'activités en n'oubliant pas, entre autres, le point départ et de terminaison.

P1.E2 La plus grande valeur

Modifier le diagramme d'activités de l'exercice précédent pour que l'algorithme sélectionne et affiche la plus grande des deux valeurs entrées par l'utilisateur. Faire aussi le pseudocode.

P1.E3 La somme

Écrire le pseudocode d'un algorithme qui demande à l'utilisateur une valeur entière positive et qui calcule la somme des valeurs entières positives plus petites ou égales à celle-ci et qui affiche le résultat à l'écran.

Il y a deux instructions possibles pour représenter les répétitions, l'énoncé **TANT QUE** et l'énoncé **POUR**. Faire les deux implémentations. Faire aussi le diagramme d'activités.

Le calcul peut se représenter à l'aide de la formule mathématique suivante en sachant que $n \geq 1$:

$$total = \sum_{i=1}^{n} i$$

P1.E4 La somme de nombres pairs

Écrire le pseudocode d'un algorithme qui demande à l'utilisateur une valeur numérique positive nommée n, qui calcule la somme des nombres pairs plus petits ou égaux à n mais supérieurs à zéro et qui affiche le résultat à l'écran. Par exemple, si l'utilisateur entre la valeur 10, l'algorithme calcule la somme de $2+4+6+8+10$. L'algorithme de l'exercice P1.E3 peut être un point de départ.

Pour cet exercice, il faut utiliser une seule boucle pour la répétition et la validation d'une condition. Aussi, l'opérateur *modulo* est très utile, car il donne comme résultat le reste de la division entière. Par exemple, 5 modulo 2 donne 1 alors que 4 modulo 2 donne 0. Faire aussi le diagramme d'activités.

P1.E5 Un diagramme d'activités mystère

Trouvez la valeur finale de a après l'exécution de l'algorithme de la figure 14.1 en supposant que les valeurs initiales sont $a = 1$, $b = 2$ et $c = 3$:

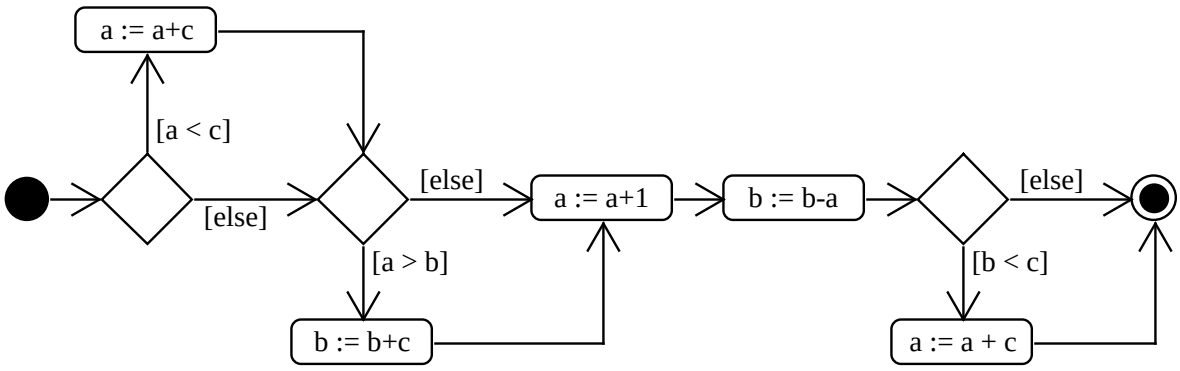

FIGURE 14.1 : Diagramme d'activités mystère de l'exercice P1.E5

14.3 EXERCICES SUPPLÉMENTAIRES

Les exercices suivants sont facultatifs et permettent de continuer à pratiquer les concepts appris lors du procédural.

P1.S1 Les opérations

Faire le diagramme d'activités d'un algorithme qui demande à l'utilisateur deux valeurs entières et qui en fait la somme, la soustraction, la multiplication et la division, et qui ensuite affiche le résultat de chacune de ces opérations. Faites aussi le pseudocode.

P1.S2 La plus petite

Écrire le pseudocode d'un algorithme qui demande trois valeurs entières à l'utilisateur et qui trouve la plus petite des trois. Faites aussi le diagramme d'activités.

P1.S3 Compter les impairs

Écrire le pseudocode d'un algorithme qui demande à l'utilisateur une valeur n et qui trouve le nombre de valeurs impaires plus petites que n.

P1.S4 Somme des pairs et somme des impairs

Écrire le pseudocode d'un algorithme qui demande à l'utilisateur une valeur n et qui calcule séparément la somme des nombres pairs et la somme des nombres impairs inférieurs à n.

P1.S5 Deux valeurs

Écrire le pseudocode d'un programme qui demande à l'utilisateur d'entrer deux valeurs, a et b, qui sont dans l'intervalle $0 \leq a, b < 50$. Une nouvelle valeur est redemandée à l'utilisateur tant que cette contrainte n'est pas respectée. La valeur la plus petite augmente par incrément de 5 et la plus grande par incrément de 2. Les valeurs sont affichées jusqu'à ce que la plus petite rattrape la plus grande.

15 PRATIQUE PROCÉDURALE 2

Buts de l'activité

- Être capable de concevoir un algorithme.
- Être capable de traduire un algorithme dans un langage de programmation.
- Développer l'habilité d'analyser des situations concrètes.
- Créer du pseudocode.
- Créer des diagrammes d'activités.

Suggestions

- Pour cette activité, il est avantageux d'avoir accès aux guides de pseudocode et de diagrammes d'activités et à votre livre du langage C.
- Pour les exercices, il est assumé que l'utilisateur fournit le bon type de donnée. Par exemple, lorsque l'utilisateur est sollicité pour un entier, il donne un entier.

15.1 EXERCICES PRÉPARATOIRES

P2.P1 Activités pour changer les pneus d'hiver

L'hiver approche à grands pas et il faut penser à changer les pneus de votre voiture. Décrivez de manière détaillée toutes les activités à réaliser pour changer les pneus.

15.2 EXERCICES

P2.E1 Exécution main

Quelle est la sortie produite par chacun des pseudocodes suivants ? Quelle est la différence entre les deux pseudocodes ? Quel est le rôle de la fonction `Main` ? Quel est l'avantage d'avoir une fonction `AfficherEcran` ?

Pseudocode 1

```
FONCTION Main
DÉBUT
    Écrire "Bonjour !" à l'écran
FIN
```

Pseudocode 2

```
FONCTION AfficherEcran(message)
    // message (chaine de caractères)
DÉBUT
    Écrire message à l'écran
FIN

FONCTION Main
DÉBUT
    AfficherEcran("Bonjour !")
FIN
```

P2.E2 La factorielle

La factorielle d'un entier naturel (n) est notée $n!$ et elle est définie comme le produit des nombres entiers positifs inférieurs ou égaux à n. Par exemple, $4! = 4 * 3 * 2 * 1$. Écrire le pseudocode d'une fonction nommée *factorielle* qui réalise ce calcul pour une valeur donnée. Quel est le type du paramètre et le type de la valeur de retour ? Est-ce qu'elle sollicite l'usager ou affiche le résultat ? Transformez le pseudocode en une fonction en langage C nommée `factorielle`. Quelles sont les préconditions et postconditions de la fonction ?

P2.E3 La puissance

Écrivez le code en langage C de la fonction `puissance` qui calcule la valeur de x^n en sachant que x est un nombre réel et n un entier. Quels doivent être les arguments de la fonction ? Quel est le type du retour de la fonction ? Quelles sont les préconditions et postconditions ? Est-ce que la fonction devrait solliciter l'usager ou afficher le résultat ?

P2.E4 La suite de Fibonacci

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. La suite commence par les termes 0 et 1. Les premiers termes de la suite sont : 0, 1, 1, 2, 3, 5, 8, 13, 21, etc. Écrire le pseudocode qui vous permettra d'afficher les n premiers termes de la suite. Votre algorithme doit demander à l'utilisateur la valeur de n et afficher les termes.

P2.E5 Portée des variables

Quels sont les avantages et les inconvénients des deux fonctions suivantes qui font une addition d'entiers ? Laquelle est préférable ? Vous voulez un indice ? Écrivez le code qui réalise $d = 1 + 2 + 3 + 4$ avec chacune d'elle.

```
1  int a, b, c;
2
3  void addition1()
4  {
5     a = b + c;
6  }
7
8  int addition2(int x, int y)
9  {
10    return x + y;
11 }
```

15.3 EXERCICES SUPPLÉMENTAIRES

Les exercices suivants sont facultatifs et permettent de continuer à pratiquer les concepts appris lors du procédural.

P2.S1 Est-ce un triangle ?

Écrire le pseudocode d'un programme capable de demander à l'utilisateur 3 nombres, ces 3 nombres représentent la longueur des côtés d'un triangle. Votre programme doit valider que ces nombres sont valides pour un triangle. Pour qu'ils soient valides, il faut que la somme de n'importe lesquels de deux côtés soit plus grande que le troisième côté.

P2.S2 Un triangle de quel type ?

Modifiez le pseudocode de l'exercice précédent pour afficher le type de triangle détecté, si c'est un triangle. Les options d'affichage sont : isocèle, équilatéral et scalène.

16 PRATIQUE EN LABORATOIRE 1

But de l'activité

- Apprendre à utiliser l'environnement de développement Geany.
- Développer des programmes en langage C.

Suggestions

- Pour cette activité, il est avantageux d'avoir accès aux guides de pseudocode, de débogage et de diagrammes d'activités et à votre livre du langage C.
- Sur les ordinateurs au laboratoire, assurez-vous de toujours travailler sur le *thawspace* (le disque K) afin de ne pas perdre vos travaux si l'ordinateur devait redémarrer.

EXERCICES

L1.E1 Hello world !

Cet exercice consiste à exécuter votre premier programme avec Geany. Le programme à développer doit simplement afficher la phrase *Hello world !* à l'écran. Les étapes sont les suivantes :

1. Démarrer Geany.
2. Sélectionner `Nouveau` du menu `Fichier`.
3. Sauvegarder le fichier en sélectionnant `Enregistrer sous` du menu `Fichier` et le nommer `hello.c`.
4. Écrire le code en langage C pour réaliser la tâche qui est demandée. S'inspirer du code dans votre livre de référence, si nécessaire.
5. Cliquer sur le bouton `Construire` pour compiler votre code et pour tenter de générer le fichier exécutable (`hello.exe`).
6. Observer les messages au bas de la fenêtre. S'il n'y a pas d'erreurs, il faut passer à l'étape suivante. S'il y a des messages erreurs, le fichier exécutable n'a pas pu être généré. Il faut tenter de corriger les erreurs dans le code et ensuite retourner à l'étape 5.
7. Cliquer sur le bouton `Executer` et observer le résultat à l'écran. Si le résultat observé ne correspond pas à ce qui est désiré, il faut corriger le code et retourner à l'étape 5.

L1.E2 La moyenne : implémentation

Implémentez en langage C le pseudocode que vous avez conçu à l'exercice P1.E1 du procédural 1. Nommez votre fichier `moyenne-deux-nombres.c`. Vérifiez que votre programme compile et s'exécute correctement. Comment pouvez-vous être certain que votre programme est correct ?

L1.E3 La somme de nombres pairs : implémentation

Implémentez en langage C le pseudocode que vous avez conçu à l'exercice P1.E4 du procédural 1. Nommez votre fichier `somme-pairs.c`. Vérifiez que votre programme compile et s'exécute correctement. Comment pouvez-vous être certain que votre programme est correct ?

L1.E4 L'équation du deuxième degré : implémentation

Implémentez en langage C un programme qui trouve, si elles existent, les racines d'une équation du second degré (voir l'annexe C) une fois que l'utilisateur a fourni les valeurs de a, b et c. Des messages appropriés doivent être affichés à l'écran pour indiquer les réponses trouvées. Considérez tous les cas possibles. Nommez votre fichier `equation.c`

Afin de calculer une racine carrée, la fonction `sqrt` de la librairie `math.h` est très utile. Vérifiez que votre programme compile et s'exécute correctement. Comment pouvez-vous être certain que votre programme est correct ?

17 PRATIQUE PROCÉDURALE 3

But de l'activité

- Connaitre et appliquer les concepts de précondition (préalable) et de postcondition.
- Connaitre et utiliser la programmation modulaire (fonctions).
- Connaitre et utiliser les tableaux

Suggestions

- Pour cette activité, il est avantageux d'avoir accès aux guides de pseudocode et de diagrammes d'activités et à votre livre du langage C.

EXERCICES

P3.E1 La moyenne (version tableau)

Dans l'exercice P1.E1 du procédural 1, la moyenne de deux valeurs numériques était calculée. Qu'est-ce qui se passe s'il faut calculer la moyenne de dix de valeurs ? Il serait trop compliqué de déclarer dix variables pour faire le calcul. Si on avait 100 valeurs ?

Pour résoudre ce problème, l'usage d'un tableau pour stocker les valeurs est tout à fait indiqué. Il suffit alors de parcourir le tableau complet pour calculer la moyenne. Un exemple serait de calculer la note moyenne d'un groupe de dix étudiants.

Écrire le pseudocode d'un algorithme qui calcule la moyenne des valeurs stockées dans un tableau nommé `notes` et qui affiche le résultat à l'écran. Le tableau contient dix valeurs numériques de type réel. Afin de simplifier l'algorithme, on considère que le tableau est préalablement initialisé avec des valeurs, il n'est donc pas nécessaire de solliciter l'usager ou d'initialiser le tableau. Faire aussi le diagramme d'activités.

P3.E2 Longueur d'une chaine de caractères

En langage C, une chaine de caractères est stockée dans un tableau de type **`char`** et le caractère `'\0'` dans le tableau indique la fin de la chaine. Écrire le pseudocode de la fonction nommée `chaineLongueur` qui retourne une valeur numérique indiquant la longueur de la chaine. La chaine vide a une longueur de zéro. Préparer un plan de tests.

P3.E3 Afficher une matrice

Écrire le pseudocode de la fonction nommée `afficherMatrice` qui affiche à l'écran les valeurs d'une matrice de valeurs entières. La matrice fournie en paramètre a deux dimensions et le nombre de lignes et le nombre de colonnes sont fixés (constantes). Que faudrait-il changer au pseudocode de cette fonction afin qu'elle remplisse la matrice de valeurs fournies par l'usager plutôt que de les afficher ?

18 PRATIQUE EN LABORATOIRE 2

But de l'activité

- Connaitre et utiliser la programmation modulaire (fonctions).
- Connaitre et utiliser les tableaux et les matrices.

Suggestions

- Pour cette activité, il est avantageux d'avoir accès aux guides de pseudocode et de diagrammes d'activités et à votre livre du langage C.

EXERCICES

L2.E1 La moyenne (version tableau) : implémentation

Écrire le code en langage C de l'exercice P3.E1 du procédural 3.

L2.E2 L'exponentielle : implémentation

Une manière de calculer la valeur de e^x est avec une série (voir l'annexe C). Écrire le code en langage C de la fonction nommée `e_x` pour trouver la valeur de e^x en utilisant `n` termes. Le prototype de cette fonction est le suivant :

```
double e_x(double x, int n);
```

Vous devez réutiliser les fonctions `factorielle` et `puissance` que vous avez élaborées aux exercices P2.E2 et P2.E3 du procédural 2. Créez un programme principal (`main`) capable de demander à l'utilisateur la valeur de `x` et qui utilisera 12 termes pour le calcul.

L2.E3 Longueur d'une chaine de caractères : implémentation

Écrire le code en langage C de la fonction `chaineLongueur` de l'exercice P3.E2 du procédural 3. Le prototype de la fonction est le suivant :

```
int chaineLongueur(char chaine[]);
```

Il faut aussi programmer une fonction `main` qui demande à l'utilisateur d'entrer une chaine de caractères, l'envoie à la fonction `chaineLongueur` et affiche le résultat reçu.

L2.E4 Afficher une matrice : implémentation

À l'aide du pseudocode de l'exercice P3.E3 du procédural 3, écrire l'implémentation en langage C de la fonction `afficherMatrice` dont le prototype est le suivant :

```
void afficherMatrice(int m[4][3]);
```

Tester votre fonction à l'aide de la matrice A suivante :

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

L2.E5 Multiplication de matrice : implémentation

Écrire le code en langage C permettant d'effectuer une multiplication d'une matrice par un nombre donné, un entier. Vous pouvez supposer que la matrice est initialisée et que sa taille ne change pas, par exemple 4 lignes et 3 colonnes. La matrice est modifiée directement. Utilisez des constantes pour définir la taille de la matrice. Tester votre fonction avec la matrice de l'exercice L2.E4.

19 VALIDATION AU LABORATOIRE

Lors de la validation, chaque équipe doit présenter ses travaux de réalisation de la problématique afin de valider les apprentissages faits. Pour ce faire, une période de temps sera allouée à chaque équipe lors de l'activité de validation. Le moment de cette période seront donnés sur la page web de l'unité. Afin de gagner du temps, l'équipe doit être prête dès l'arrivée de l'évaluateur. Cela veut dire que vos diagrammes et votre pseudocode doivent être prêts à être consultés et aussi que Geany est déjà démarré et prêt à compiler et exécuter vos programmes. Une fois la validation faite, on vous demande que quitter le local dès que possible afin de libérer l'espace pour une autre équipe.

> Les scénarios de tests lors de la validation se dérouleront sans que l'usager ait à fournir de valeurs. Comme annoncé dans la problématique, les tests sont automatisés. Ils sont donc aussi automatisés dans la validation. Les opérations ciblées avec les scénarios de tests pour chacune seront annoncées sur la page web dans les jours qui précèdent la validation.

Pendant la validation, les membres de l'équipe doivent présenter certaines des opérations demandées dans l'énoncé de la problématique. Les opérations ciblées doivent toutes être documentées (pseudocode et diagramme d'activités) et implémentées. Pour chaque opération ciblée, vous devez être en mesure d'exécuter le programme et aussi d'expliquer le pseudocode, le diagramme d'activités et le code. Lors de la validation, l'évaluation du pseudocode et des diagrammes est formative. Finalement, les membres de l'équipe doivent répondre aux questions de l'évaluateur. La validation porte sur les éléments qui sont décrits au tableau 11.1.

A DESCRIPTION DES OPÉRATIONS

Il est important de respecter les spécifications qui suivent afin que les opérations soient réalisées comme ce qui est attendu. Il est donc important de respecter ce qui est spécifié en ce qui concerne le nombre et le type des paramètres et, le cas échéant, des valeurs de retour des fonctions qui réalisent ces opérations. Aussi, comme spécifié dans la problématique, le code des fonctions réalisant les opérations ne doit pas faire usage de variables globales et ne fait pas d'affichage ou de sollicitation de l'usager. Les bonnes pratiques de programmation de l'annexe D discutent d'ailleurs de ce sujet.
Comme annoncé dans la problématique, les tests sont automatisés. Les tests doivent se dérouler sans que l'usager ait à fournir de valeur.

Recherche d'un caractère : Cette opération retourne comme valeur la position (un entier) de la première occurrence d'un caractère dans une chaine de caractères (un tableau de caractères). Le caractère cherché et la chaine de caractères sont fournis en paramètres lors de l'appel. Si le caractère est trouvé, la valeur retournée est supérieure ou égale à zéro, sinon, la valeur retournée est de -1. La première position dans la chaine est la position zéro.
Par exemple, si le caractère `'a'` est cherché dans la chaine `"Bonjour les amis"`, l'opération retourne la valeur 12, car la première occurrence du caractère cherché est à l'index 12.

Détection d'un palindrome : Cette opération détecte si la chaine de caractères (un tableau de caractères) fournie en paramètre lors de l'appel est un palindrome. Cette opération retourne une valeur entière. Elle retourne une valeur de zéro, si la chaine n'est pas un palindrome, sinon, elle retourne une valeur différente de zéro. Le cas simple est considéré : les chaines considérées ne contiennent que des lettres minuscules non accentuées. Elles ne contiennent donc pas de majuscules, d'espaces, de caractères de ponctuation ou autres caractères.

Calcul du sinus avec une série : Cette opération retourne la valeur du calcul du sinus à partir d'une série (voir l'annexe C). L'angle en radians est spécifié en paramètre lors de l'appel et la valeur retournée est le résultat du calcul. Les deux valeurs sont des nombres réels. Le nombre de termes utilisés dans la série pour le calcul est spécifié globalement uniquement par une constante (`const` ou `#define`) et n'est pas passé en paramètre.

Calcul du cosinus avec une série : Cette opération retourne la valeur du calcul du cosinus à partir d'une série (voir l'annexe C). L'angle en radians est spécifié en paramètre lors de l'appel et la valeur retournée est le résultat du calcul. Les deux valeurs sont des nombres réels. Le nombre de termes utilisés dans la série pour le calcul est spécifié globalement uniquement par une constante (`const` ou `#define`) et n'est pas passé en paramètre.

Addition de deux matrices : Cette opération fait l'addition de deux matrices fournies en paramètres lors de l'appel et le résultat est donné dans une troisième matrice aussi fournie en paramètre lors de l'appel. Les matrices ont m lignes et n colonnes et ces

deux valeurs sont des constantes (**`const`** ou `#define`) définies globalement et ne sont pas passées en paramètres. Cette opération n'a pas de valeur de retour.

Multiplication de deux matrices : Cette opération fait la multiplication de deux matrices carrées fournies en paramètres lors de l'appel et le résultat est donné dans une troisième matrice aussi fournie en paramètre lors de l'appel. Les matrices sont de dimensions n qui est une constante (**`const`** ou `#define`) définie globalement et n'est pas passée en paramètres. Cette opération n'a pas de valeur de retour.

B PLAN DE TESTS

En ingénierie, comme dans d'autres domaines, les plans de tests sont une partie importante de tout développement et les logiciels ne font pas exception. Évidemment, la forme du plan de tests et son ampleur dépendent de la nature et de l'ampleur du projet.

Le terme *plan de tests* est utilisé dans plusieurs domaines, dont l'électronique et l'informatique. De domaine en domaine, la signification du terme varie. En développement de logiciel, le plan de tests sert à valider que l'objet soumis aux tests, comme une application ou une fonction, respecte les spécifications.

Le plan de tests peut prendre plusieurs formes, mais peu importe la forme, il véhicule la même information : l'ensemble des entrées à fournir et le résultat attendu dans chaque cas. Un plan de tests peut prendre une forme textuelle ou celle d'un tableau, comme le suivant :

Tableau B.1 : Exemple de plan de tests sous forme de tableau

Fonction	Paramètre	Résultat attendu
arcsinus	0	0
arcsinus	-1	-1.5708
arcsinus	1	1.5708
arcsinus	0.5	0.52359
arcsinus	2	erreur
arccosinus	0	1.5708
arccosinus	-1	-3.14159
arccosinus	1	0
arccosinus	0.5	1.04719
arccosinus	2	erreur

Afin de faire un bon plan de tests, il faut y penser dès le début, dès qu'on se penche sur le projet ou la tâche à réaliser. Il est important de se rappeler que le but d'un test n'est pas de montrer le bon fonctionnement, mais de trouver un problème. Lors de la création d'un plan de tests, l'état d'esprit est très différent entre *créer des tests pour montrer que ça fonctionne* et *créer des tests pour tenter de trouver un problème*. À force de montrer qu'il n'y a pas de problèmes, on peut montrer, avec un certain niveau de confiance, que ça fonctionne.

En se rappelant qu'un plan de tests sert à tenter de trouver un problème, un bon point de départ pour créer de bons tests sont les cas limites, les préconditions (préalables) et les postconditions.

C NOTIONS MATHÉMATIQUES

C.1 Racines d'une équation du second ordre

Une équation du second degré peut être décrite comme suit :

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

Cette équation a deux valeurs de x comme solution (racines). Une manière de trouver les racines est de commencer par trouver la valeur du discriminant (d) avec la formule suivante : $d = b^2 - 4ac$. Si la valeur du discriminant est négative, alors l'équation n'a pas de racines réelles. Si la valeur du discriminant est positive, alors il y a deux racines distinctes, x_1 et x_2, et si elle est nulle, alors l'équation a une racine double $(x_1 = x_2)$. S'il existe des racines $(d \geq 0)$, alors elles se calculent avec les formules suivantes :

$$x_1 = \frac{-b - \sqrt{d}}{2a} \text{ et } x_2 = \frac{-b + \sqrt{d}}{2a}$$

C.2 Géométrie

Circonférence d'un cercle : $C = 2\pi r$, r étant le rayon du cercle.
Aire d'un cercle : $A = \pi r^2$, r étant le rayon du cercle.
Aire d'une sphère : $A = 4\pi r^2$, r étant le rayon de la sphère.
Volume d'une sphère : $V = \frac{4}{3}\pi r^3$, r étant le rayon de la sphère.
Distance entre deux points : Si les coordonnées de deux points (p_1 et p_2) dans le plan cartésien sont respectivement (x_1, y_1) et (x_2, y_2), alors la distance d qui sépare p_1 et p_2 est donnée par la formule suivante :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

C.3 Approximations avec une série

En sachant que x est en **radians** :

$$e^y \approx 1 + y + \frac{y^2}{2!} + \frac{y^3}{3!} + \ldots \text{ pour } -\infty < y < \infty$$

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \text{ avec } 0 \leq x < 2\pi$$

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \text{ avec } 0 \leq x < 2\pi$$

$$\tan(x) \approx x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \ldots, \text{ avec } |x| < \pi/2$$

D BONNES PRATIQUES DE PROGRAMMATION

Les bonnes pratiques de programmation sont un consensus généralement admis par la communauté de programmeurs et elles sont considérées comme indispensables.

À quoi servent les bonnes pratiques de programmation ? En sachant que la majorité du développement de logiciels est de la maintenance (retravailler du code existant des mois ou des années plus tard), on veut faciliter cette tâche lorsque vient le temps de revisiter le code.

De la même manière, les bonnes pratiques de programmation sont utiles pour mieux se comprendre entre collègues, pour avoir du code plus uniforme et pour avoir du code plus facile à lire et à comprendre. Les bonnes pratiques sont très nombreuses et en voici quelques-unes.

D.1 Code

Noms

Les noms utilisés doivent être en lien avec l'usage fait dans le code. Ils sont faciles à lire, à comprendre et à interpréter. De bons noms concernent le code, mais ils s'appliquent aussi aux noms de fichiers et de projets.

Le nom d'une variable (ou d'une constante) doit exprimer, le mieux possible, son rôle dans le code. Ainsi, des noms variables tels que `toto`, `entier` et `valeur` ne sont typiquement pas appropriés et doivent être évités. Ce qui est intéressant ce n'est pas que la variable est un entier ou qu'elle contient une valeur, c'est la signification du contenu de cette variable. Un nom comme `vitesse` ou `temperature` est beaucoup plus expressif. Cependant, il est courant de voir des noms comme `i` et `j` pour des variables compteurs dans énoncés de boucles ou encore des noms comme `x`, `y` et `z` dans le cas de coordonnées cartésiennes. Tout est une question de contexte.

En ce qui concerne le nom d'une fonction, il faut en choisir un qui représente bien la tâche réalisée et qui la distingue bien d'une autre fonction. Il faut éviter les abréviations et les noms qui portent à confusion. Lorsqu'il est difficile de trouver un nom simple, c'est souvent un indicateur que la fonction a trop de responsabilités diverses. Les fonctions ont une signature qui est composée du nom de la fonction, du nombre et du type des paramètres et le type de la valeur de retour. Quelle signature (ou prototype) est la plus facile à comprendre ?

```
float fct1(int x, int y);
float moyenne(int valeur1, int valeur2);
```

Fonctions

Une fonction ne devrait faire qu'une seule tâche. Par exemple, une fonction qui calcule la moyenne ne devrait faire que cela, calculer la moyenne. Elle ne devrait pas solliciter l'usager pour les valeurs et afficher le résultat à l'écran.

En effet, si les données ne viennent pas de l'usager ou si le résultat ne doit pas être affiché, quelle serait l'utilité de cette fonction ? Cette fonction est plus difficilement réutilisable

dans un autre contexte ou un autre endroit dans le code. La provenance des données pour effectuer le calcul et ce qui doit être fait avec le résultat n'est pas de la responsabilité de la fonction qui calcule la moyenne. C'est la responsabilité de la fonction appelante. La lecture des données et l'annonce des résultats se font selon la situation ou la préférence de l'utilisateur, comme dans la fonction `main`.

Si les responsabilités sont bien divisées, un nom représentatif de fonction est plus facile à trouver, le code requiert moins de commentaires et lorsque des changements surviennent, la quantité de code à modifier est peut-être diminuée.

Lisibilité

Il est avantageux de garder le code simple, autant que possible, afin de faciliter sa compréhension. Il est important aussi de le garder lisible en le formatant convenablement en y ajoutant des changements de lignes, des espaces et des tabulations aux endroits appropriés. La lisibilité est aussi augmentée lorsqu'il y a cohérence tout au long du code dans la manière de nommer les variables, les fonctions, les constantes, etc.

Il faut programmer dans une seule langue, comme le français, et non pas mélanger deux langues, comme l'anglais et le français. Évidemment, rien ne peut être fait pour les mots-clés du langage C, mais pour le reste, il faut être cohérent. Aussi, les caractères accentués doivent être évités dans le code, car certains compilateurs ne les tolèrent pas bien, ce qui nuit à la portabilité du code.

Afin de garder le code simple, il est important de réutiliser des fonctions existantes, plutôt que de réinventer ce qui existe déjà. Cela aide à la maintenance, la lisibilité et la facilité de compréhension du code.

Autres astuces

Il faut éviter de copier-coller des valeurs identiques à différents endroits dans le code, comme la taille d'un tableau. Il faut plutôt utiliser une variable, une constante ou un `#define`. Plus tard, s'il faut modifier cette valeur, il suffit d'aller à un seul endroit, plutôt que plusieurs et risquer d'en oublier.

Il faut éviter de copier-coller du code identique à différents endroits, comme le calcul de la moyenne. Il faut plutôt faire une fonction avec ce code et ensuite la réutiliser aux endroits nécessaires.

Bien qu'il existe des situations d'exception, en général, il faut éviter d'utiliser des variables globales. Ce type de variables est reconnu pour créer des liens inutilement entre différentes parties du code. Ces liens peuvent être difficiles à défaire plus tard. Les variables globales nuisent typiquement à la modularité et l'indépendance du code. Au lieu d'utiliser des variables globales, il faut favoriser l'usage de paramètres à des fonctions.

D.2 Commentaires

Les commentaires sont nécessaires et ils ont plusieurs utilités, selon l'endroit où ils se situent dans le code. Ils servent à communiquer avec vos collègues et avec ceux qui feront la maintenance du code.

Les commentaires doivent être simples, pertinents, précis, concis et faciles à maintenir. Il ne faut pas exagérer sur les commentaires, car ils doivent ensuite être maintenus. Il ne faut pas oublier que la personne qui lit le code est déjà familière, au moins un peu, avec le langage. Il faut éviter d'ajouter des commentaires inutiles et qui nuisent à la lisibilité du code. C'est là que survient une partie de subjectivité.

En tenant compte de ce qui vient d'être énoncé, il faut donc éviter des commentaires dans le code comme les suivants qui sont inutiles et qui nuisent à la lisibilité.

```
int i = 0; // Déclaration d'une variable entière nommée i initialisée à 0
i = i + 1; // Incrémenter la variable i de 1
```

En début de fichier, les commentaires peuvent prendre la forme d'un *entête de fichier* afin de fournir des informations générales sur le code.

```
/********
Fichier: nomDuFichier
Auteurs: Nom1  CIP1
         Nom2  CIP2
Date:    01 janvier 2000
Description: description sommaire de ce qui est dans le fichier
********/
```

Tout juste avant une fonction, les commentaires peuvent prendre la forme d'un *entête de fonction*. Un entête de fonction est utile pour décrire la tâche réalisée, les préconditions et les postconditions. La description de la tâche est optionnelle lorsque le code de la fonction est court, sa tache bien définie et que son nom est bien choisi.

```
// Description: fonction qui calcule la moyenne de deux valeurs
// Préconditions: les préconditions de la fonction
// Postconditions: les postconditions de la fonction
```

D.3 Écriture du code : petit train va loin...

Idéalement, des tests sont effectués régulièrement sur de petits bouts de code et au fur et à mesure qu'ils sont écrits. En procédant ainsi, les sources d'erreurs sont normalement moins grandes et plus faciles à identifier. Si nécessaire, il est possible de mettre en commentaire des bouts de code afin de limiter les possibilités sur les sources d'erreurs.

C'est toujours une bonne idée de vérifier le bon fonctionnement d'un bout de code avant de continuer plus loin. Il faut bâtir sur du solide. Éventuellement, tout le code est écrit et on peut alors procéder à des tests complets afin de vérifier le comportement global du programme.

LISTE DES RÉFÉRENCES

[1] B. Jones, P. Aitken, et D. Miller, *Sams Teach Yourself C Programming in One Hour a Day*, 7e édition. Pearson, 2014.