



ATELIER DE PROGRAMMATION

GUIDE DE L'ÉTUDIANTE ET DE L'ÉTUDIANT S1 – APP4snte

Automne 2022

Auteur: D. Palao Muñoz, J. Vincent, J.-S. Lauzon, A. Huppé, E. Morin, A. Marchese, C.-A. Brunet
Version: GitLab (28 octobre 2022)

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

©2000 Tous droits réservés. Département de génie électrique et de génie informatique,
Université de Sherbrooke.

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	7
7	LOGICIELS ET MATÉRIEL	8
8	SANTÉ ET SÉCURITÉ	9
9	SOMMAIRE DES ACTIVITÉS	10
10	PRODUCTIONS À REMETTRE	11
11	ÉVALUATIONS	13
12	POLITIQUES ET RÈGLEMENTS	15
13	INTÉGRITÉ, PLAGIAT ET AUTRES DÉLITS	16
14	PRATIQUE PROCÉDURALE	17
15	PRATIQUE EN LABORATOIRE	22
16	VALIDATION AU LABORATOIRE	27
A	FORMATS ET TRAITEMENT DES IMAGES	28
B	DESCRIPTION DES OPÉRATIONS	31
C	BONNES PRATIQUES DE PROGRAMMATION	36

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	13
11.2	Grille d'indicateurs utilisée pour les évaluations	14
B.1	Constantes et paramètres utilisés dans les prototypes des opérations	31
B.2	Valeurs de retour de l'opération lire	32

1 ACTIVITÉS PÉDAGOGIQUES ET COMPÉTENCES

GEN145 – Atelier de programmation

1. Résoudre un problème informatique en développant un algorithme et en exécutant sa programmation, sa validation et sa documentation en exploitant un système de développement de programme avec interface graphique et outils de débogage.

Description officielle : usherbrooke.ca - GEN145

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 décrites à la section 1. Le sommaire de l'évaluation de l'unité d'APP est donné au tableau 2.1.

TABLEAU 2.1 : Sommaire de l'évaluation de l'unité.

Évaluation	GEN145-1
Rapport et livrables associés	40
Évaluation sommative théorique	40
Évaluation sommative pratique	80
Évaluation finale théorique	50
Évaluation finale pratique	90
Total	300

À moins de circonstances exceptionnelles, une cote ou un niveau d'atteinte d'une *qualité* est calculé à partir du tableau 2.2. La grille des *indicateurs* utilisée pour les évaluations est donnée 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

Manipulation d'images

La compagnie MesImages inc. vous a engagé comme stagiaire dans le domaine du **traitement des images**. Le traitement d'images est une branche de l'informatique et des mathématiques qui étudie les **images numériques** et leurs transformations. On peut analyser l'information fournie par une image, mais aussi la transformer avec l'aide de différents calculs mathématiques.

Une image numérique est composée **d'échantillons de l'image réelle**. La représentation de l'image dans la mémoire de l'ordinateur peut être faite avec une **matrice à deux dimensions composée de valeurs numériques**. Chaque valeur dans la matrice représente un *pixel*, un **point à l'écran**. Un pixel est l'unité minimale adressable et gérée par le contrôleur vidéo (la carte vidéo).

Dans une première étape, seules les **images en noir et blanc**, enrichies avec des tons de gris, sont traitées. Chaque image est sauvegardée dans un fichier de format standard, le **format PGM (*Portable Gray Map*)**. Les fichiers PGM sont en format texte avec **l'extension pgm**. Les fichiers PGM sont relativement faciles à manipuler, car ils ont une structure standard connue et simple. Cette structure est explicitée à l'annexe [A](#).

Pour cette étape, il faut **programmer un ensemble d'opérations** applicables sur une image en noir et blanc donnée. Pour représenter l'image dans la mémoire de l'ordinateur, il faut **créer une matrice d'entiers**. Pour simplifier le traitement, **la taille maximale de la matrice est fixée (256 par 256)**.

Dans une deuxième étape, le propriétaire de la compagnie aimerait faire quelques tests avec des **images en couleurs**. Pour traiter ce cas, l'image est sauvegardée dans un fichier de format **PPM (*Portable Pixel Map*)**. Les fichiers PPM ont une structure standard semblable au format PGM et elle est aussi explicitée à l'annexe [A](#). Les fichiers PPM sont en format texte avec **l'extension ppm**.

Pour cette étape, il faut programmer un ensemble d'opérations applicables sur une image en couleur donnée. Pour représenter l'image dans la mémoire de l'ordinateur, vous allez **créer une matrice dont chaque élément est une structure (**struct**) du langage C**. La structure sera composée de trois valeurs entières nommées valeurR, valeurG et valeurB qui représentent **les valeurs RGB (*Red, Green, Blue*) de la couleur** (voir l'annexe [A](#)).

Les opérations qui peuvent être effectuées sur une image sont les suivantes. Le détail de ces opérations est donné à l'annexe [B](#).

Format PGM : lire, écrire, copier, créer l'histogramme, trouver la couleur prépondérante, éclaircir et noircir, créer le négatif, extraire, détecter deux images identiques et pivoter de 90 degrés.

Format PPM : lire, écrire, copier, détecter deux images identiques et pivoter de 90 degrés.

L'ensemble de ces opérations constitue une bibliothèque de fonctionnalités pour la gestion (traitement) d'images. Pour la création de la bibliothèque, une démarche uniformisée est proposée par la compagnie. Un ensemble de fichiers pour démarrer le projet est fourni : **un fichier d'entête (.h) pour les prototypes de fonctions et des fichiers de code (.c) pour l'implémentation.** Cette structure de fichier doit être conservée et elle ne peut être modifiée. Ces **fichiers sont incomplets** et ils doivent donc être complétés. Ces fichiers permettront éventuellement de faciliter l'automatisation des tests avec d'autres fichiers que la compagnie fournira à un moment ultérieur.

Quelques particularités sont imposées au sujet des fichiers d'images (PGM et PPM). Ces fichiers ne doivent **pas contenir de commentaires**, sauf pour la première ligne, afin de simplifier le traitement. En effet, la première ligne d'un fichier d'image peut contenir les métadonnées.

Les **métadonnées**, si elles sont présentes, sont des informations sur l'image qui sont insérées et sauvegardées dans le fichier d'image. Les métadonnées prennent la forme d'une ligne de commentaire située à la première ligne du fichier d'image. Les métadonnées contiennent les trois valeurs suivantes : **l'auteur de l'image, la date de création et le lieu de création.** Ces trois valeurs sont du texte en format libre et elles sont séparées par un point virgule. Il est possible qu'une valeur soit vide et donc ne contienne pas de caractères. Ces trois valeurs sont propres à la problématique et elles ne font pas partie des standards PPM et PGM, bien que les commentaires en fassent partie.

Pour ce projet, la compagnie demande aussi de tenir compte de quelques **bonnes pratiques de programmation.** Les bonnes pratiques sont des règles informelles adoptées par la grande majorité de la communauté de développeurs de logiciels et qui permettent d'augmenter la qualité des logiciels. La liste des bonnes pratiques peut être très longue, mais la compagnie demande de **se conformer au moins à celles présentées à l'annexe C.**

5 CONNAISSANCES NOUVELLES

Connaissances déclaratives (quoi)

- Le langage C
 - Pointeurs.
 - Structures (struct).
 - Lecture et écriture de fichiers texte.
 - Implémentation d’algorithmes structurés manipulant des tableaux d’une et deux dimensions.
 - Compilation séparée et programmation modulaire (fichiers d’entête .h et de code source .c).
- Développement de logiciels
 - Création de fonctions selon des spécifications fournies.
 - Développer un logiciel composé d’un programme principal et de fonctions.

Connaissances procédurales (comment)

- Utiliser les structures pour traiter des groupes de données.
- Utiliser les fichiers pour sauvegarder de données (lecture et écriture).
- Construire des programmes en utilisant une approche modulaire à l’aide de fonctions.
- Utiliser la compilation séparée avec un environnement de développement.

Connaissances conditionnelles (quand)

- Écrire un programme afin de résoudre un problème de complexité limitée en utilisant les instructions appropriées.
- Faire des tests pour valider que le programme résolve le problème.

6 GUIDE DE LECTURE

6.1 Références essentielles

Livre du langage C : Le livre de référence est celui de Jones et al. [?]. Les parties suivantes du livre seront abordées dans les activités de l'APP.

- Chapitre 9
- Chapitre 11 : sauf la partie sur les *unions*.
- Chapitre 12 : jusqu'à la page 285.
- Chapitre 17 : jusqu'à la page 431 et les pages 443 et 444 sur la détection de fin de fichier.

Lectures supplémentaires sur langage C : liens internet consultés le 28 octobre 2022.

- Les structures : zestedesavoir.com - Tutoriel structures
- Les pointeurs : zestedesavoir.com - Tutoriel pointeurs
- Les flots de fichiers : openclassroom.com - Tutoriel flots de fichiers
- La programmation modulaire : openclassroom.com - Tutoriel programmation modulaire

Guide de pseudocode : Ce document montre la manière standard de produire le pseudo-code pour la session. Il est offert sur la page web de l'unité.

Guide de diagrammes d'activités : Ce document montre la manière de construire des diagrammes d'activités en UML. Il est offert sur la page web de l'unité.

Fichiers de démarrage : Un ensemble de fichiers de départ est distribué sur la page web de l'unité pour la résolution de la problématique.

Autovalidation : Un ensemble de fichiers afin de faire une autovalidation de votre résolution de la problématique est distribué sur la page web de l'unité pour la résolution de la problématique. Consultez aussi la section 16 pour plus de détails.

6.2 Documents complémentaires

Les documents suivants sont disponibles sur la page web de l'unité ou aux endroits indiqués 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.

Définition des formats PGM et PPM : Consulter l'annexe A pour débiter et ensuite, si nécessaire, le site https://fr.wikipedia.org/wiki/Portable_pixmap.

Introduction au traitement d'images : Le site suivant en fait une petite introduction : https://fr.wikipedia.org/wiki/Traitement_d'images.

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. Pour installer ces outils sur vos ordinateurs personnels, consultez la section des logiciels offerts du site web de la session.

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 !

Pseudocode : Vous pouvez utiliser le logiciel que 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.

Visualisation d'images : Une grande quantité de logiciels existent pour visualiser des images de format PPM et PGM. À défaut d'en avoir un préféré, GIMP est le logiciel suggéré. On le trouve sur site suivant : <https://www.gimp.org/>.

Conversion de format d'images : L'annexe A discute brièvement de ce sujet. À défaut d'avoir un logiciel préféré de conversion, ImageMagick est suggéré. On le trouve sur le site suivant : <http://www.imagemagick.org/>.

Notepad++ : Cet éditeur de texte peut être utilisé pour consulter en format texte un fichier d'image de format PGM ou PPM.

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

Activités de la semaine

- Présentation de la problématique
- Étude personnelle
- Formation à la pratique procédurale
- Laboratoires avec support
- Évaluation formative
- Consultation générale
- Évaluation sommative théorique et pratique

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 de la présentation du début de l'unité d'APP. 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 12h00 (midi), la journée de l'examen sommatif. Tout retard entraîne 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 entraîner 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.

Les livrables remis sont un rapport, le code en langage C produit en réponse à la problématique et le fichier résultant de l'autovalidation. Ces livrables, leurs spécifications et la méthode de dépôt sont détaillés dans ce qui suit.

10.1 Rapport

Le rapport remis est un fichier **PDF** nommé **rapport.pdf**. Les noms et CIP de tous les membres de l'équipe doivent être **présents sur la page couverture**. Le rapport contient, à part la page couverture, **uniquement le pseudocode de l'opération `pgm_lire` et le diagramme d'activités de l'opération `pgm_extraire`**. C'est tout, pas même d'introduction, de conclusion ou de table des matières. Le pseudocode et le diagramme d'activités doivent suivre les standards **des guides disponibles sur la page web de l'unité**.

10.2 Code source

Le code source (fichiers de langage C) remis sont dans un répertoire nommé **Code**. Il y a uniquement trois fichiers dans ce répertoire et ils sont nommés comme suit :

- **bibliotheque_images.h**
- **bibliotheque_images.c**
- **gestion_images.c**

Le code remis doit évidemment respecter les bonnes pratiques de programmations mentionnées à l'annexe **C**. La fonction `main` sert à appeler à plusieurs reprises les fonctions réalisant les différentes opérations afin de les tester selon vos plans de tests. Entre autres, aucune interaction avec l'utilisateur ne doit être faite : **les tests sont automatisés**. Des descriptions plus détaillées des opérations sont données à l'annexe **B**.

10.3 Autovalidation

Un fichier de journal est produit lors de l'autovalidation, un fichier avec l'extension **log**. Ce fichier est aussi remis, mais séparément du rapport et du code source.

10.4 Remise des productions

Il y a deux dépôts à faire :

1. Une **archive zip** dont le contenu est le **rapport PDF et le code source**, comme mentionné aux sections **10.1** et **10.2**. Vous devez nommer l'archive zip en accord avec les **CIP de tous les membres de l'équipe**, par exemple **CIP1-CIP2.zip**. L'archive zip est déposée dans le répertoire associé au rapport de cette unité d'APP.
2. Le fichier de journal qui est produit lors de l'autovalidation est un fichier avec l'extension **log**. Le fichier est déposé séparément du rapport dans le répertoire associé à l'autovalidation de cette unité d'APP. Le fichier doit être renommé en accord avec les **CIP des membres de l'équipe**, par exemple **CIP1-CIP2.log**. La section 16 discute de l'autovalidation.

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. Les dépôts sont téléversés en utilisant l'outil de dépôt du département et qui est accessible avec le lien *Dépôt des travaux* sur le site web de la session. 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 avant la date limite.

11 ÉVALUATIONS

11.1 Rapport et livrables associés

L'évaluation du rapport et des livrables associés (voir la section 10) portera sur les compétences figurant dans la description des activités pédagogiques décrites à la section 1. La pondération de chacune d'entre elles dans l'évaluation est indiquée au tableau 11.1.

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

Élément	GEN145-1
Diagrammes d'activités et pseudocode	5
Utilisation des structures	4
Utilisation des pointeurs	4
Utilisation de flots de fichiers sur disque	4
Compilation séparée	2
Bonnes pratiques de programmation	6
Autovalidation	15
Total	40

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. Les évaluations se font sans documentation.

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 : 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	Q04.4	... 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 adéquatement, mais avec quelques écarts au standards, et permettant l'implémentation à l'aide d'un langage de programmation.	... définit une conception détaillée selon les standards et les règles de l'art et permettant l'implémentation à l'aide d'un langage de programmation.
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	Q04.5	... ne démontre pas la capacité à 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	Q04.5	... 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, postconditions 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.	... connaît et est en mesure d'utiliser avec assistance mineure les techniques et outils spécifiés.	... connaît et est en mesure d'utiliser les techniques et outils spécifiés.	... connaît et est en mesure d'utiliser efficacement les techniques et outils spécifiés.
Bonnes pratiques de programmation	Q01.2	... ne démontre pas la capacité à appliquer les bonnes pratiques de programmation.	... démontre de la difficulté à appliquer les bonnes pratiques de programmation.	... sait appliquer la majorité des bonnes pratiques de programmation.	... sait appliquer efficacement la majorité des bonnes pratiques de programmation.	... sait appliquer efficacement les bonnes pratiques de programmation.

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

Normes d'agrément

- Processus d'agrément et qualités du BCAPG
- Ingénieurs Canada – À propos 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

Buts de l'activité

- Comprendre les structures et leur utilisation.
- Comprendre les pointeurs et leur utilisation.
- Comprendre les flots de fichiers et leur utilisation.
- Comprendre la portée des variables et son impact.
- Comprendre la programmation modulaire en langage C.

Suggestions

- Pour cette activité, il est avantageux d'avoir accès à votre livre du langage C.
- Les structures : zestedesavoir.com - Tutoriel structures
- Les pointeurs : zestedesavoir.com - Tutoriel pointeurs
- Les flots de fichiers : openclassroom.com - Tutoriel flots de fichiers
- La programmation modulaire : openclassroom.com - Tutoriel programmation modulaire

14.1 EXERCICES

P.E1 Les structures

- Qu'est-ce qu'une structure?
- Quelle est l'utilité des structures?
- Quels seraient les éléments à inclure dans une structure représentant une **adresse civique**?

P.E2 Les structures en langage c

- Comment peut-on **définir** une structure en langage c (utilisez l'exemple de l'adresse civique de la question précédente)?
- Comment peut-on **écrire** dans une structure?
- Comment peut-on **lire** le contenu d'une structure?
- Comment peut-on passer une structure en paramètre d'une fonction?
- Qu'est-ce qu'un « typedef » et comment l'utiliser avec une structure?

P.E3 Les structures (analyse de code)

- Quelle est la sortie à l'écran du code suivant utilisant une structure?

```
1 #include <stdio.h>
2
3 struct Personne {
4     char nom[128];
5     char prenom[128];
6     int age;
7 };
8
9 void afficher(struct Personne pers) {
10     printf("Nom: %s\n", pers.nom);
11     printf("Prenom: %s\n", pers.prenom);
12     printf("Age: %d\n", pers.age);
13 }
14
15 int main() {
16     struct Personne p;
17     printf("Nom: ");
18     scanf("%s", p.nom);
19     printf("Prenom: ");
20     scanf("%s", p.prenom);
21     printf("Age: ");
22     scanf("%d", &p.age);
23     afficher(p);
24     return 0;
25 }
```

P.E4 Les pointeurs

- Qu'est-ce qu'un pointeur?
- Quelle est la représentation d'un pointeur en mémoire?
- Quels sont les avantages d'utiliser des pointeurs?

P.E5 Les pointeurs en langage c

- Comment peut-on déclarer un pointeur?
- Comment initialiser la valeur d'une variable identifiée par un pointeur?
- Comment attribuer la valeur null à un pointeur (quel est l'utilité)?
- Comment obtenir ou modifier la valeur associée à un pointeur?
- Comment peut-on obtenir l'adresse d'une variable?
- Comment peut-on passer un pointeur en paramètre d'une fonction?
- Comment une fonction peut retourner un pointeur?

P.E6 Les pointeurs (analyse de code)

– Quelle est la sortie à l'écran du code suivant ?

```
1  #include <stdio.h>
2
3  struct Personne {
4      char nom[128];
5      char prenom[128];
6      int age;
7  };
8
9  void f(int x, int *ptr) {
10     x = 4;
11     printf("%d\n", x);
12     *ptr = 10;
13     printf("%d\n", *ptr);
14 }
15
16 void lire(struct Personne * p) {
17     printf("Nom: ");
18     scanf("%s", p->nom);
19     printf("Prenom: ");
20     scanf("%s", p->prenom);
21     printf("Age: ");
22     scanf("%d", &p->age);
23 }
24
25 int main() {
26     int i = 1, j = 1;
27     int *p1, *p2;
28     struct Personne pers;
29
30     p1 = &i;
31     p2 = p1;
32     f(j, p1);
33     printf("%d %d %d %d\n", i, j, *p1, *p2);
34
35     lire(&pers);
36     printf("Nom: %s\n", pers.nom);
37     printf("Prenom: %s\n", pers.prenom);
38     printf("Age: %d\n", pers.age);
39     return 0;
40 }
```

P.E7 Les fichiers

- Quelle est l'utilité de lire et d'écrire dans des fichiers sur disque?
- Quel est le résultat dans le fichier de sortie à la suite de l'exécution du code donné plus bas (en admettant que le contenu du fichier donnees.txt est le suivant)?

Contenu du fichier donnees.txt :

4

5

Code :

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     int a, b, res;
6     char nom_data[128];
7     FILE *flot_entree, *flot_sortie;
8
9     strcpy(nom_data, "donnees.txt");
10    flot_entree = fopen(nom_data, "r");
11    flot_sortie = fopen("resultats.txt", "w");
12
13    if(flot_entree == NULL || flot_sortie == NULL)
14    {
15        printf("Erreur d'ouverture de fichier.\n");
16    }
17    else
18    {
19        fscanf(flot_entree, "%d %d", &a, &b);
20        res = a + b;
21        fprintf(flot_sortie, "%d\n", res);
22    }
23
24    if (flot_entree != NULL) fclose(flot_entree);
25    if (flot_sortie != NULL) fclose(flot_sortie);
26    return 0;
27 }
```


P.E8 Les fichiers .h et .c du langage c

- Qu'est-ce que la programmation modulaire en langage C?
- Qu'est-ce qu'un fichier .h en langage C?
- Quel est le contenu d'un fichier .h
- Qu'est-ce qu'un fichier .c en langage C?
- Quel est le contenu d'un fichier .c

15 PRATIQUE EN LABORATOIRE

But de l'activité

- Développer des programmes en langage C qui utilisent des structures de données.
- Développer des programmes en langage C qui utilisent des fichiers en entrée et en sortie.

Suggestions

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

15.1 EXERCICES

Les exercices suivants sont facultatifs. *Cependant, ils sont fortement suggérés, car ils permettent de vérifier votre compréhension de l'usage de fichiers et la manipulation de structure avec pointeur en langage C. **Au lieu de faire ces exercices, certains préfèrent commencer la résolution de la problématique.**

PARTIE 1 : MANIPULATION DE FICHIERS

L.E1 FICHIER - Copie de fichier de matrice

- Développer un programme qui copie d'un fichier source une matrice d'entiers de taille 3x3 dans un fichier destination.
- Les noms des deux fichiers sont donnés par l'utilisateur.
- Évidemment, après la copie, le contenu du fichier en sortie doit être identique à celui en entrée.
- Un exemple de fichier en entrée est le suivant :

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

L.E2 FICHIER - Calculer la moyenne (version 1)

- Développer un programme qui calcule la moyenne d'un nombre indéterminé de nombres réels stockés dans un fichier en entrée.
- Le nom du fichier en entrée est donné par l'utilisateur.
- La moyenne calculée est affichée à l'écran.
- Un exemple de fichier en entrée est le suivant :

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

L.E3 FICHIER - Calculer la moyenne (version 2)

- En utilisant le programme développé à la question précédente.
- Calculer la moyenne d'un nombre indéterminé de nombres réels, stockés dans un fichier en entrée, **et étant séparé par un point-virgule dans le fichier.**
- POUR CET EXERCICE, faire une lecture par caractère dans le fichier texte (utiliser la fonction **fgetc**)
- La moyenne calculée est affichée à l'écran.
- Un exemple de fichier en entrée est le suivant :

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

PARTIE 2 : MANIPULATION DE STRUCTURES DE DONNÉES

L.E4 STRUCTURES - Utiliser des structure de données (mise en contexte)

Une structure de données vraiment utile en informatique est la liste chaînée. Il s'agit d'une liste ordonnée d'éléments qui, contrairement aux tableaux, ne sont pas nécessairement contigus en mémoire. Au plus simple, chaque élément possède une donnée (un nom dans notre cas) et une référence à l'élément suivant dans la chaîne. Lorsque l'on travaille avec une chaîne de données, seule la référence du premier élément est nécessaire.

- Une structure appelée Maillon est déjà fournie dans le code (voir la distribution sur le site de l'App).
- Dans le "main", on instancie plusieurs maillons (A, B, C, D).
- Pour les questions suivantes, on assume que le maillon A est la premier élément (la tête) de la chaîne. Voir la figure 15.1.
- Chaque structure maillon est initialisée avec une référence "prochain" pointant vers une valeur NULL.

```
struct Maillon
{
    char nom[50];
    struct Maillon* prochain;
};
```

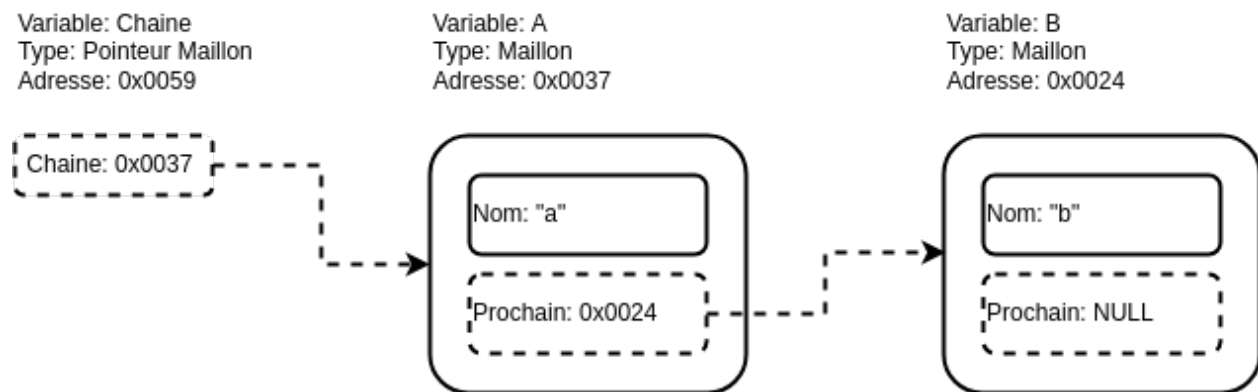


FIGURE 15.1 : Exemple de liste chaînée

L.E5 STRUCTURES - Afficher les données

- On désire afficher les noms des maillons de la chaîne dans l'ordre.
- Compléter la fonction **imprimer_chaine** prenant en référence le premier élément de la chaîne.
- Une chaîne contenant un seul élément devrait sortir le texte suivant :

Chaine: [a]

- En ajoutant l'élément, B après l'élément A, la sortie devrait maintenant afficher un rendu respectant l'exemple suivant :

Chaine: [a, b]

L.E6 STRUCTURES - Ajouter un élément à une chaîne

- Compléter la fonction **ajout_maillon** permettant d'ajouter un maillon a la fin de la chaîne.
- Cette fonction prend en paramètre une référence à la chaîne et la référence de l'élément à ajouter.
- Après avoir ajouté C et D à la fin de la chaîne, la sortie attendue est :

Chaine: [a, b, c, d]

L.E7 STRUCTURES - Retirer un élément d'une chaîne

- Comment peut-on retirer un élément de la chaîne.
- Compléter la fonction **retirer_maillon** prenant la référence de la chaîne (un pointeur de pointeur) et un indice en paramètre.
- Celle-ci devra permettre de retirer uniquement l'élément à l'indice précis.
- Après avoir retirés les éléments B et D, la sortie attendue est :

Chaine: [a, d]

L.E8 STRUCTURES - Retirer le premier élément d'une chaîne

- Comment pourrait-on retirer le premier élément de la chaîne.
- Après avoir retiré le premier élément, la sortie attendue est :

Chaîne: [d]

- Selon vous, pourquoi cette fonction doit-elle absolument prendre un pointeur de pointeur de struct Maillon?
- Quelle serait la sortie si on utilisait seulement le pointeur **une_chaine**?

L.E9 STRUCTURES - Écrire la sortie dans un fichier

- Comment pourrait-on écrire la sortie du terminal (console) dans le fichier "sortie.txt"?

16 VALIDATION AU LABORATOIRE

Pour cette unité d'APP, il y a une **autovalidation à réaliser**, donc à faire par vous-même. Cette autovalidation peut être faite pendant les séances de laboratoires. C'est une autovalidation faite de façon logicielle avec l'aide d'une **librairie qui est distribuée sur la page web de l'unité**. La librairie est accompagnée d'un document pour expliquer la manière de l'utiliser.

Avant de faire l'autovalidation, il est fortement suggéré de faire une validation préliminaire de vos opérations avec vos propres tests. Il est facile de créer des fichiers d'images avec un éditeur de texte et d'en créer avec de petites images, comme de taille 2 par 3. Il est alors facile de valider le bon fonctionnement de vos opérations. Ensuite, une fois convaincu, vous pouvez passer à l'étape d'autovalidation.

La librairie d'autovalidation appelle les fonctions que vous avez implémentées et elle affiche à l'écran des messages explicatifs sur la validité des résultats. Étant donné cette méthode de fonctionnement, il est très important que les spécifications des fonctions demandées à **l'annexe B** soient respectées, car dans le cas contraire, il y aura des erreurs de compilation ou des erreurs dans les résultats et des points pour la validation seront perdus. **Il est donc très important de faire l'autovalidation.**

Après l'exécution de votre code avec la librairie d'autovalidation, un fichier de journal sera créé, **un fichier avec l'extension log**. Ce fichier de journal doit être déposé avec les autres documents exigés (voir la section 10).

A FORMATS ET TRAITEMENT DES IMAGES

Les images en noir et blanc (format PGM) et en couleur (format PPM) peuvent être visualisées avec une bonne quantité de logiciels. Elles peuvent aussi être éditées directement par un éditeur de texte, car le contenu de ces fichiers est du texte.

Les informations dans les sections qui suivent résument une partie de l'information donnée sur WIKIPEDIA aux liens suivants :

- https://fr.wikipedia.org/wiki/Portable_pixmap
- https://fr.wikipedia.org/wiki/Rouge_vert_bleu

Les formats utilisés pour la problématique sont les versions **ASCII (*plain*)** et non pas les versions binaires (*raw*) des formats PGM et PPM. Les versions ASCII permettent de **facilement consulter ou de modifier les données du fichier avec un éditeur de texte**. Avec le format **ASCII, aucune ligne ne doit dépasser 70 caractères**. Cependant, un bon nombre de logiciels ne respectent pas cette contrainte de **70 caractères lorsque les fichiers d'images sont créés**. Il faut donc que les logiciels qui lisent les images soient adaptatifs et qu'ils puissent lire les fichiers d'images qui ne respectent pas cette contrainte.

A.1 Format noir et blanc

Avec ce format, le format PGM, toutes les lignes qui commencent par le caractère # sont ignorées. Elles peuvent donc servir pour insérer des commentaires ou des informations supplémentaires dans le fichier d'images. Aussi, l'usage de caractères d'espacement est requis. Un caractère d'espacement peut être un espace, une tabulation ou une nouvelle ligne.

Outre les commentaires, la structure d'un fichier PGM est la suivante :

- Le *nombre magique* du format pour identifier le type de fichier : P2.
- Un caractère d'espacement
- La largeur de l'image (un entier)
- Un caractère d'espacement
- La hauteur de l'image (un entier)
- Un caractère d'espacement
- La valeur maximale utilisée pour coder un pixel, les niveaux de gris. Cette valeur (un entier) est positive et inférieure à 65536.
- Un caractère d'espacement (souvent un changement de ligne)
- Les données de l'image. L'image est codée ligne par ligne en partant du haut et chaque ligne est codée de gauche à droite. Chaque pixel est codé par une valeur numérique précédée et suivie par un caractère d'espacement. Un pixel noir est codé par la valeur zéro, un pixel blanc est codé par la valeur maximale et chaque niveau de gris est codé par une valeur entre ces deux extrêmes proportionnellement à son intensité.

Ce qui suit est un exemple de fichier d'image qui respecte cette description et dont la première ligne est un commentaire.


```
#Prenom Nom;2015-06-17;Sherbrooke
P2
4 4
255
0 64 128 255
0 64 128 255
0 64 128 255
0 64 128 255
```

A.2 Format couleur

La structure d'un fichier d'image couleur (format PPM) est la même que celle d'une image noir et blanc (format PGM), sauf que le *nombre magique* est P3 et les données de l'image sont différentes, car chaque pixel est représenté par trois valeurs numériques, les valeurs RGB (*Red, Green, Blue*), précédées et suivies par un caractère d'espacement. Un pixel noir est codé par les valeurs 0 0 0 et un pixel blanc est codé par trois fois la valeur maximale.

Voici un exemple de fichier d'image qui respecte cette description et dont la première ligne est un commentaire :

```
#Prenom Nom;2015-06-17;Sherbrooke
P3
2 2
255
0 255 0 12 22 55
0 0 127 255 0 0
```

A.3 Conversion de formats d'images

Les images sont souvent offertes dans un autre format que PGM ou PPM. Comme les formats traités dans la problématique sont PGM et PPM, il faut possiblement faire une conversion de format. De plus, la taille de l'image ne respecte peut-être pas les limites imposées par la problématique.

Il existe plusieurs moyens de redimensionner une image et de faire une conversion de format. Un logiciel de dessin ou d'autres logiciels spécialisés peuvent être utilisés. Des procédures suivent, mais celles-ci ne sont que des suggestions.

Changement de taille

S'il est nécessaire de changer la taille de l'image, le programme Windows Paint peut être utilisé. Il faut alors changer la taille de l'image et la sauvegarder afin qu'elle soit plus petite ou égale à celle spécifiée dans la problématique.

Changement de format

Si vous avez une image dans le format JPG, alors la procédure suivante peut être utilisée pour la convertir en format PGM ou PPM.

1. Dans un répertoire de travail, déposer une copie de l'image et une copie du programme `convert.exe` qui est fournie avec les fichiers de démarrage (section 6.1).

2. Ouvrir une fenêtre DOS et aller dans le répertoire de travail. Ceci peut être fait facilement en faisant un *shift-clic-droit* dans sur le répertoire de travail et en choisissant *Ouvrir une fenêtre de commande ici*.
3. Convertir l'image de format JPG en image de format PGM ou PPM en tapant dans la fenêtre DOS une des commandes suivantes, en remplaçant `nom.jpg` par le vrai nom du fichier d'image, suivi de la touche Enter. Par exemple, pour créer une image en format PGM, il faut utiliser la commande suivante :
`convert nom.jpg -compress none nom.pgm`
Pour créer une image en format PPM, il faut plutôt utiliser l'extension PPM et la commande à utiliser devient alors la suivante :
`convert nom.jpg -compress none nom.ppm`

B DESCRIPTION DES OPÉRATIONS

Cette annexe décrit les opérations à réaliser pour le traitement des images. Certaines des descriptions qui sont données dans cette annexe font référence aux types de données RGB et MetaData dans les prototypes de fonctions. Ces types de données sont définis dans les fichiers de démarrage qui sont distribués (voir la section 6.1). Aussi, les prototypes des fonctions utilisent des valeurs symboliques (en majuscules) et des paramètres (en minuscules) qui sont décrits au tableau B.1 afin d'éviter les répétitions.

TABLEAU B.1 : Constantes et paramètres utilisés dans les prototypes des opérations

Nom	Description
MAX_HAUTEUR	nombre maximum de lignes (voir problématique)
MAX_LARGEUR	nombre maximum de colonnes (voir problématique)
MAX_VALEUR	valeur maximale d'un pixel (standard PPM et PGM)
nom_fichier	nom d'un fichier pour lire ou écrire une image
colonnes colonnes1 colonnes2	nombre effectif de colonnes d'une matrice d'image
lignes lignes1 lignes2	nombre effectif de lignes d'une matrice d'image
maxval	valeur maximale d'un pixel, spécifiée dans le fichier
matrice matrice1 matrice2	noms de matrices d'image
metadonnees	métadonnées d'une image

Certains des paramètres du tableau B.1 ont des variantes pointeurs, par exemple colonnes et p_colonnes. Lorsque c'est un pointeur qui est donné en paramètre à une fonction, c'est la valeur pointée qui doit être modifiée. Ainsi, afin d'abréger les descriptions de cette annexe, lorsqu'une expression du style « *les paramètres ... sont modifiés* » ou encore « *les paramètres ... contiennent* » est rencontrée et que des paramètres pointeurs sont visés, il faut interpréter l'expression pour les paramètres pointeurs dans le sens de « *les valeurs pointées par les paramètres ...* ».

B.1 Opérations pour les images en noir et blanc

B.1.1 Lire

Cette opération ouvre en lecture le fichier en format PGM spécifié et charge les données dans la matrice. La signature de la fonction correspondante est la suivante :

```
int pgm_lire(char nom_fichier[], int matrice[MAX_HAUTEUR][MAX_LARGEUR],
             int *p_lignes, int *p_colonnes, int *p_maxval,
             struct MetaData *p_metadonnees);
```

Si l'opération est réussie, elle retourne zéro et les paramètres sont modifiés pour contenir les informations sur l'image qui est stockée dans le fichier. Ce sont les paramètres matrice, p_lignes, p_colonnes, p_maxval et p_metadonnees. Sinon, l'opération retourne un code d'erreur selon la situation rencontrée, comme spécifié dans le tableau B.2. Il est à noter qu'une erreur dans les métadonnées est considérée comme une erreur de format.

TABLEAU B.2 : Valeurs de retour de l'opération lire

Valeur	Nom	Signification
-1	ERREUR_FICHER	Erreur d'ouverture ou de manipulation de fichier
-2	ERREUR_TAILLE	La taille de l'image est trop grande
-3	ERREUR_FORMAT	Le format de l'image n'est pas respecté

B.1.2 Écrire

Cette opération ouvre le fichier spécifié et y sauvegarde en format PGM l'image dont les données sont fournies par les autres paramètres. La signature de la fonction correspondante est la suivante :

```
int pgm_ecrire(char nom_fichier[], int matrice[MAX_HAUTEUR][MAX_LARGEUR],
               int lignes, int colonnes, int maxval, struct MetaData metadonnees);
```

Si l'opération est réussie, elle retourne la valeur zéro, sinon, elle retourne une valeur négative.

B.1.3 Copier

Cette opération prend une image chargée dans une matrice source et en fait une copie dans une matrice destination. La signature de la fonction correspondante est la suivante :

```
int pgm_copier(
    int matrice1[MAX_HAUTEUR][MAX_LARGEUR], int lignes1, int colonnes1,
    int matrice2[MAX_HAUTEUR][MAX_LARGEUR], int *p_lignes2, int *p_colonnes2);
```

Si l'opération est réussie, elle retourne la valeur zéro, sinon, elle retourne une valeur négative. La matrice source est spécifiée par les paramètres d'indice 1. Les paramètres d'indice 2 sont les destinations et ils sont donc modifiés pour contenir la copie des informations.

B.1.4 Créer l'histogramme

Cette opération comptabilise les données pour l'histogramme de l'image. Un histogramme est le compte du nombre de pixels pour chaque intensité lumineuse. La signature de la fonction correspondante est la suivante :

```
int pgm_creeer_histogramme(int matrice[MAX_HAUTEUR][MAX_LARGEUR],
                           int lignes, int colonnes, int histogramme[MAX_VALEUR+1]);
```

Si l'opération est réussie, elle retourne une valeur de zéro et le paramètre histogramme est modifié pour contenir le nombre de parutions dans l'image de chaque valeur d'intensité. Sinon, l'opération retourne une valeur négative.

B.1.5 Trouver la couleur prépondérante

Cette opération trouve la couleur prépondérante de l'image fournie. Cette opération trouve la valeur de la couleur qui se répète le plus dans l'image. Il est à noter que cette opération peut se réaliser avec l'aide de l'opération *créer l'histogramme*. La signature de la fonction correspondante est la suivante :

```
int pgm_couleur_preponderante(int matrice[MAX_HAUTEUR][MAX_LARGEUR],
                              int lignes, int colonnes);
```

Si le traitement est correct, l'opération donne comme valeur de retour la valeur de la couleur prépondérante. S'il y a une erreur de traitement, l'opération donne une valeur de retour négative.

B.1.6 Éclaircir et noircir

Cette opération modifie la valeur de chaque pixel de l'image d'une valeur spécifiée. Si cette valeur est positive, l'image s'éclaircira. À l'inverse, si la valeur est négative, l'image noircira. La signature de la fonction correspondante est la suivante :

```
int pgm_eclaircir_noircir(int matrice[MAX_HAUTEUR][MAX_LARGEUR],
    int lignes, int colonnes, int maxval, int valeur);
```

Si l'opération est réussie, elle retourne une valeur de zéro et l'image est modifiée directement, sinon, elle retourne une valeur négative. Il faut faire attention aux dépassements, car la valeur d'un pixel est dans l'intervalle $[0, maxval]$, $maxval$ étant la valeur maximale spécifiée dans le fichier et reçue en paramètre. Si la valeur d'un pixel est plus grande que $maxval$, alors le résultat est $maxval$ et si la valeur d'un pixel est plus petite que 0, alors le résultat est 0.

B.1.7 Créer le négatif

Cette opération crée le négatif de l'image en utilisant la formule suivante :

$$valeurPixel = maxval - valeurPixel$$

La valeur $maxval$ étant la valeur maximale spécifiée dans le fichier. La signature de la fonction correspondante est la suivante :

```
int pgm_creer_negatif(int matrice[MAX_HAUTEUR][MAX_LARGEUR],
    int lignes, int colonnes, int maxval);
```

Si l'opération est réussie, l'image est modifiée directement et elle retourne une valeur de zéro. Sinon, elle retourne une valeur négative.

B.1.8 Extraire

Cette opération prend une partie de l'image pour la replacer au coin supérieur gauche de l'image originale. La partie de l'image extraite doit évidemment être de dimension plus petite ou égale à l'image originale. La signature de la fonction correspondante est la suivante :

```
int pgm_extraire(int matrice[MAX_HAUTEUR][MAX_LARGEUR],
    int lignes1, int colonnes1, int lignes2, int colonnes2,
    int *p_lignes, int *p_colonnes);
```

Le paramètre `matrice` contient l'image originale. Les pixels du coin supérieur gauche et inférieur droit de la partie de l'image à extraire sont (`lignes1`, `colonnes1`) et (`lignes2` et `colonnes2`), valeurs qui sont aussi données en paramètres.

Si l'opération est réussie, elle retourne une valeur de zéro et l'image est modifiée directement. Les paramètres `p_lignes` et `p_colonnes` sont alors modifiés pour contenir les dimensions de l'image extraite. Sinon, l'opération retourne une valeur négative, par exemple, si le nombre de lignes ou de colonnes extraites est trop grand ou s'il y a débordement des images.

B.1.9 Détecter deux images identiques

Cette opération détecte si deux images de même taille sont identiques en les comparant pixel par pixel. La signature de la fonction correspondante est la suivante :


```
int pgm_sont_identiques(  
    int matrice1[MAX_HAUTEUR][MAX_LARGEUR], int lignes1, int colonnes1,  
    int matrice2[MAX_HAUTEUR][MAX_LARGEUR], int lignes2, int colonnes2);
```

L'opération retourne zéro, si les deux images sont identiques et une valeur positive, si elles sont différentes. S'il y a erreur, alors une valeur négative est retournée.

B.1.10 Pivoter de 90 degrés

Cette opération prend une image et la fait tourner de 90 degrés dans le sens horaire ou antihoraire. Voici un exemple de matrice (image) originale avec la matrice (image) résultante pour une rotation dans le sens horaire.

124	112	154	234
112	243	124	111
221	211	21	156
113	156	112	38



113	221	112	124
156	211	243	112
112	21	124	154
38	156	111	234

La signature de la fonction correspondante est la suivante :

```
int pgm_pivoter90(int matrice[MAX_HAUTEUR][MAX_LARGEUR],  
    int *p_lignes, int *p_colonnes, int sens);
```

En entrée, les paramètres contiennent les informations sur l'image originale. Si le paramètre sens a une valeur de 1 alors la rotation est dans le sens horaire et s'il a une valeur de 0, elle est dans le sens antihoraire. Si l'opération est réussie, elle retourne une valeur de zéro et l'image est modifiée directement ainsi que les paramètres p_lignes et p_colonnes afin de refléter les dimensions de la nouvelle image. Sinon, l'opération retourne une valeur négative.

B.2 Opérations pour les images en couleur

Les opérations pour les images en couleur (format PPM) ont les mêmes spécifications que les opérations correspondantes en noir et blanc (PGM). Ainsi, seule la signature des fonctions est donnée et la section précédente donne le détail des spécifications. Évidemment, comme ce sont des images en couleurs, le format des fichiers en entrée ou en sortie est PPM et non PGM et les images sont stockées dans des matrices qui permettent de stocker les pixels en format RGB.

B.2.1 Lire

Les spécifications de cette opération sont données à la section B.1.1. Le format du fichier en entrée est PPM. La signature de la fonction est la suivante :

```
int ppm_lire(  
    char nom_fichier[], struct RGB matrice[MAX_HAUTEUR][MAX_LARGEUR],  
    int *p_lignes, int *p_colonnes, int *p_maxval,  
    struct MetaData *metadonnees);
```

B.2.2 Écrire

Les spécifications de cette opération sont données à la section [B.1.2](#). Le format du fichier en sortie est PPM. La signature de la fonction est la suivante :

```
int ppm_ecrire(  
    char nom_fichier[], struct RGB matrice[MAX_HAUTEUR][MAX_LARGEUR],  
    int lignes, int colonnes, int maxval, struct MetaData metadonnees);
```

B.2.3 Copier

Les spécifications de cette opération sont données à la section [B.1.3](#). La signature de la fonction est la suivante :

```
int ppm_copier(  
    struct RGB matrice1[MAX_HAUTEUR][MAX_LARGEUR],  
    int lignes1, int colonnes1,  
    struct RGB matrice2[MAX_HAUTEUR][MAX_LARGEUR],  
    int *p_lignes2, int *p_colonnes2);
```

B.2.4 Détecter deux images identiques

Les spécifications de cette opération sont données à la section [B.1.9](#). La signature de la fonction est la suivante :

```
int ppm_sont_identiques(  
    struct RGB matrice1[MAX_HAUTEUR][MAX_LARGEUR],  
    int lignes1, int colonnes1,  
    struct RGB matrice2[MAX_HAUTEUR][MAX_LARGEUR],  
    int lignes2, int colonnes2);
```

B.2.5 Pivoter de 90 degrés

Les spécifications de cette opération sont données à la section [B.1.10](#). La signature de la fonction est la suivante :

```
int ppm_pivoter90(struct RGB matrice[MAX_HAUTEUR][MAX_LARGEUR],  
    int *p_lignes, int *p_colonnes, int sens);
```

C 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.

C.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'utilisateur pour les valeurs et afficher le résultat à l'écran.

En effet, si les données ne viennent pas de l'utilisateur 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.

C.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 tâche 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
```

C.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.