École de technologie supérieure

Service des enseignements généraux Tél : 396-8938 http://www.seg.etsmtl.ca/

EXIGENCES CONCERNANT LA REMISE DES TRAVAUX COURS

INF-130, INF-135, INF-145 et INF-155

INTRODUCTION

Le présent document offre des précisions sur les attentes des enseignants face aux travaux pratiques en informatique. Il décrit les exigences concernant la qualité des travaux devant être remis. Les indications mentionnées plus-bas ne constituent que le minimum attendu et des demandes supplémentaires peuvent s'ajouter.

Le genre masculin est employé tout au long du texte dans le seul et unique but d'alléger ce dernier.

DESCRIPTION DES ATTENTES

Biens livrables

Lors de la remise d'un travail, à moins d'un avis contraire, l'enseignant s'attend à recevoir :

- L'impression de tout le code composé pour ce travail;
- L'ensemble des fichiers sources nécessaires à l'exécution du programme envoyé par courriel à l'enseignant de votre groupe dans un document compressé (.zip ou .rar);
- Dans le cas d'un travail en équipe, sur un document à part : une description sommaire du travail réalisé par chacun des membres du groupe.

Dans une enveloppe identifiée avec les informations suivantes :

- Le nom, le code permanent de toutes les personnes ayant collaboré au travail;
- Le sigle du cours et le numéro de groupe
- Le nom de l'enseignant

Qualité du code attendue

Dans le but d'enseigner aux étudiants les méthodes de programmation modernes, voici la description de la plupart des caractéristiques qui permettront de rendre leur code clair, propre et efficace. La plupart des points qui suivent servent directement à l'évaluation des étudiants.

NOTIONS DE BASE

1) Commentaires

Dans le but de simplifier la lecture du code, ce dernier devrait contenir un bon nombre de commentaires. Plus précisément, chaque programme devrait se caractériser par :

a) Un commentaire au tout début du programme identifiant le ou les créateurs, l'objectif visé, le cours pour lequel il fut réalisé et la date de création,

b) Un commentaire pour chacun des prototypes de fonctions et procédures décrivant ce qu'elle permet d'accomplir, chacun des paramètres qu'elle attend et, dans le cas d'une fonction, la valeur retournée,

c) Un commentaire pour chacun des corps de fonctions décrivant la stratégie employée,

d) Un commentaire accompagnant chaque déclaration de variable justifiant sa création (la tâche qu'elle permet de remplir),

e) Un commentaire pour chaque instruction (ou groupe d'instructions) pouvant sembler complexe.

f) Il est à noter que les commentaires DOIVENT être insérés dans le code au fur et à mesure (et non pas lorsque l'écriture du code est terminée).

2) Indentation

L'indentation rend la lecture du code plus aisée en permettant très rapidement de déterminer si une instruction se situe **à l'intérieur d'une boucle ou d'une condition**. Un espacement de 4 caractères par tabulation est recommandé. Par exemple :

```
POUR i allant de 0 à 99 BOUCLE
    SI tableau(i) > 0
        Afficher(tableau(i));
    FIN SI
FIN BOUCLE
```

3) L'aération

Espacer les différents blocs d'instructions en insérant des lignes blanches. Même chose pour les variables et les constantes.

À éviter :	À favoriser :

//un point (x,y dans l'espace)
int x;
int y;
//Accumulateur
int somme;
//itérateurs
int i;
int j;
	//un point (x,y dans l'espace)
	int x;
	int y;
	
	//Accumulateur
	int somme;
	
	//itérateurs
	int i;
	int j;

4) Disposition des instructions

Même si le langage le permet, on évitera de mettre plus d'une instruction sur la même ligne de code. Cela permettra de conserver un code clair et facile à lire.

5) Noms significatifs

Tout sous-programme, paramètre de fonction/procédure et variable doit posséder un nom significatif se rattachant à l'utilisation que l'on souhaite en faire. Il faut donc éviter d'utiliser des noms tels que *val1*, *val2* et *val3*. Les noms *i*, *j* et *k* ne seront tolérés QUE pour servir de compteurs de boucle.

6) Utilisation de constantes

Un programme ne devrait pas contenir de valeur littérale (nommée «magic number » en anglais). De telles valeurs devraient être placées à l'intérieur de constantes de façon à pouvoir les modifier aisément sans avoir à fouiller dans le code. De plus, cela augmente la lisibilité du code (la valeur 0.07 à moins de sens que *TPS*). Par convention, les constantes seront toujours mises en majuscules et seront les seules à l'être.

NOTIONS AVANCÉES

7) Variables globales

L'utilisation de variables globales est FORMELLEMENT INTERDITE. Les données encapsulées dans une librairie (ou module) constituent l'UNIQUE EXCEPTION à cette règle.

8) Le *goto*

L'utilisation de l'instruction *goto* est FORMELLEMENT INTERDITE. Bien qu'il soit possible de s'en servir de façon correcte, AUCUN des programmes que vous aurez à écrire n'aura besoin d'en contenir.

9) Sous-programmes

Un sous-programme (procédure ou fonction) ne devrait réaliser qu'une seule et unique tâche (évitez les fonctions comme *lire_valider_et_afficher*). De façon générale, un sous-programme (fonction/procédure) ne devrait pas dépasser 30 à 50 lignes (incluant les commentaires), soit environ un écran d'ordinateur. S'il arrivait que cette valeur soit excédée, alors il faudrait préférablement découper le sous-programme en sous-sous-programmes et ainsi de suite.

10) Paramétrisation

Les fonctions et procédures ont pour objectif de clarifier le code et de permettre la réutilisation de code déjà écrit. Pour ce faire, l'utilisation de paramètres est généralement essentielle. Le nombre de paramètres d'une fonction/procédure ne devrait pas excéder 7. Si un tel cas se présentait, il serait judicieux de créer des structures de données permettant de réduire la quantité de paramètres transmis.

11) Encapsulation

Cette notion ne s'applique qu'en présence de plusieurs librairies/modules. Si une librairie (ou module) *X* conserve de l'information qui lui est privée, alors tout sous-programme désirant accéder à ces données doit le faire en passant par les fonctions offertes en interface par *X*.