TP n° 6 C++
Sous-programmes et paramètres (D) - 3 séances

NB : Les codes sources utiles pour ce TP sont disponibles sur Moodle

Objectifs

1. Organiser son code en sous-programmes.
2. Écrire des sous-programmes en C++.
3. Manipuler des paramètres passés par valeur.
4. Distinguer variable et paramètre.

Consignes générales pour les exercices sur machine

1. **Lire** la question en entier.
2. Écrire la **spécification algorithmique** (à recopier en commentaire sous le prototype)
3. Préparer un **jeu d'essais**.
4. Écrire le **code** de la fonction
5. Faire les tests de chaque sous-programme avec les jeux d'essai prévus dans le main() de façon automatique. Laissez vos tests dans votre rendu.

Travail à rendre : déposer sur Moodle les fichiers de l'exercice 3
inclure les essais en commentaire

Exercice 1 : (sur feuille) du prototype aux spécifications, des spécifications au prototype

a) Le prototype C++ d'un sous-programme doit indiquer complètement son mode d'utilisation. Soit le prototype suivant :

```
bool estPositif(int);
```

Quel est son nom ? En algorithmique, dirait-on qu'il s'agit d'une fonction ou d'une procédure et pourquoi ? Quel est le statut de ses paramètres ? Finalement, donnez les spécifications algorithmiques correspondant à ce prototype.

b) On veut que ce sous-programme retourne vrai si le nombre passé en paramètre est strictement positif, faux sinon. Ecrivez ensuite la définition complète en C++.

Exercice 2 : Premiers sous-programmes avec Code::Blocks : la décomposition en 3 fichiers

Tous les programmes devront respecter l'architecture en 3 fichiers vue en cours, à savoir :

- un fichier source contenant la fonction main
- un fichier d'entête pour les déclarations (prototypes) de sous-programmes et
- un fichier source contenant les définitions des sous-programmes.

a) **Créer un projet "calculs" et y inclure 3 fichiers.** Pour cela, voir encadré **: "Compléments sur Code::Blocks"** ci-dessous.

- **calcul.h :** contiendra les prototypes des sous-programmes. Penser aux directives de pré-compilation.
- **calcul.cpp** : contiendra les définitions des sous-programmes. Penser à inclure le fichier **calcul.h**
- **main.cpp :** contient la fonction main. Penser à inclure le fichier **calcul.h**

b) Recopier le prototype de la fonction **estPositif** dans le fichier **calcul.h** et sa définition dans le fichier **calcul.cpp**.

c) On veut tester le bon fonctionnement de la fonction **estPositif**. Quels sont les différents cas à tester ? Compléter le fichier **main.cpp** pour réaliser ces tests. Ne pas faire de saisie des valeurs : il suffit d'effectuer des appels de la fonction **estPositif** avec des valeurs constantes bien choisies et d'afficher le résultat pour tester les différents cas possibles.
Inspirer vous des exemples vus en amphi.

Compléments sur Code::Blocks

Pour ajouter un fichier d'entête (.h) à un projet existant (voir également le document Codeblocks.pdf disponible sur Moodle)

- sélectionner `File → New → File`
- sélectionner `C/C++ header`
- entrer le chemin complet du fichier à créer, en le mettant dans le même répertoire que les autres fichiers sources du projet
- vérifier que le champ `Header guard word` n'est plus vide,
  et que la case `Add file to active project` est bien cochée.
- sélectionner toutes les cibles de compilation en cliquant sur `All`
- cliquer sur `Finish`
- Procédez de la même manière pour ajouter un fichier de code source à votre projet (cette fois, vous sélectionnerez "source C/C++"). Remarquez que le fichier ajouté est vide.

Exercice 3 : Un menu pour choisir parmi plusieurs traitements

Vous allez écrire un programme qui demandera la saisie d'un nombre **n** positif, puis qui proposera un menu offrant les possibilités suivantes à l'utilisateur :

- Calculer et afficher la somme des entiers de 1 à **n**
- Calculer et afficher la factorielle de **n**
- Afficher la liste des diviseurs de **n**
- Quitter

Rappel : Factorielle de n (notée n!) est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Par exemple, 5! = 1 * 2 * 3 * 4 * 5 = 120

Chacun des calculs sera effectué par un sous-programme dédié. La saisie de **n** et l'affichage du résultat des calculs seront réalisés dans le programme principal (sauf si le but du sous-programme est de déclencher un affichage). Le menu sera implémenté sous forme de fonction indépendante qui retourne un entier correspondant au type de calcul désiré.

Le programme devra être implémenté dans les fichiers suivants :

- **main.cpp** : qui contient l'implémentation du programme principal.
- **menu.cpp** et **menu.h**: qui contiennent respectivement l'implémentation de la fonction menu, et la déclaration de la fonction et **d'une constante par option proposée par le menu**.
- **calculs.cpp** et **calculs.h** : qui contiennent respectivement l'implémentation des fonctions de calcul, et leurs déclarations.

Travail à réaliser :

a) Reprendre et compléter le projet **calculs** de l'exercice précédent incluant les 5 fichiers :
   - **main.cpp** (commenter les instructions utilisées pour l'exercice précédent),
   - **menu.h et menu.cpp** (à télécharger à partir de Moodle), prenez le temps de regarder ce qu'ils contiennent. Notez que des constantes sont définies.
   - **calculs.h, calculs.cpp** (que vous avez créés pour l'Exercice 2 :)

b) En utilisant le sous-programme **estPositif,** écrire le début du programme **main.cpp** pour qu'il fasse saisir à l'utilisateur un nombre et vérifie qu'il est bien positif. Le programme doit recommencer si ce n'est pas le cas. Testez.

c) En utilisant les fonctions définies dans **menu.cpp,** compléter le programme principal du fichier **main.cpp**. Après la saisie du nombre n, faire appel à la fonction **menu** (affichage du menu et lecture de la réponse de l'utilisateur) et effectuer une boucle répéter pour savoir quel traitement l'utilisateur veut effectuer jusqu'à ce que l'utilisateur demande de quitter (en choisissant la réponse QUITTER).
Quels sont les cas à tester ? Tester les différents cas possibles, en confirmant par un simple affichage le choix de l'utilisateur.

d) Écrire les spécifications algorithmiques de la fonction **somme**. Ensuite compléter les fichiers **calculs.cpp** et **calculs.h** en y plaçant respectivement la définition et la déclaration de la fonction **somme**.

Modifier ensuite le programme principal pour que le choix **1** provoque l'appel de la fonction **somme** puis l'affichage du résultat. Testez les différents cas possibles.

e) Procéder de même pour ajouter les 2 autres types de calculs (factoriel et diviseurs).

Travail à rendre : déposer sur Moodle les fichiers de l'exercice 3
inclure les essais en commentaire

Exercice 4 : Des spécifications aux prototypes, avec des paramètres (R) et (D/R)

En utilisant le poly volume 2 du cours de C++ pages 23, reprendre les spécifications algorithmiques suivantes (provenant du TD4) et les traduire en prototypes C++ :

Procédure afficheSpecial (x, y)
*{Affiche **x** fois la valeur **y**}*
paramètres (D) x, y : entiers

Fonction estDiviseur(x,y) retourne booléen
*{Retourne VRAI si **x** est un diviseur de **y**, et FAUX sinon}*
paramètres (D) x, y: entiers

Fonction encore() **retourne booléen**
{Retourne VRAI si l'utilisateur veut recommencer le traitement en cours, FAUX sinon}
paramétres (aucun)

Procédure sommeDesDiviseurs (nb, somme)
{renvoie dans **somme** la somme des diviseurs de l'entier **nb**}
paramètres (D) nb : **entier**
(R) somme : **entier**

Fonction nbOccurrences(tab, val) retourne entier
{Retourne le nombre d'occurrences d'un entier donné dans un tableau d'entiers}
paramètres (**D**) tab : **tableau d'entiers**
(**D**) val : **entier**

Procédure doubler(tab2D)
{prend un tableau 2D d'entiers et remplace toutes les valeurs de ce tableau par son double.}
parametres (D/R) tab2D : **tableau d'entier à 2 dimensions**

Fonction tousEgaux (tab2D) retourne **booléen**
{ prend un tableau 2D d'entiers et retourne vrai si ce tableau contient des valeurs qui sont toutes identiques. On suppose que le tableau n'est pas vide.}
paramètres (**D**) tab2D: **tableau d'entiers à 2 dimension** *{non vide}*