université PARIS-SACLAY

Travaux Pratiques de programmation n°1

Cours de programmation modulaire

—Licence MI/IM - Info 3—

Structures

La présence aux séances de TPs est obligatoire, et l'assiduité sera prise en compte dans l'évaluation pour le contrôle continu.

Les exercices un peu plus difficiles sont signalés par le symbole  . Pour bien suivre la progression des travaux pratiques, il faut au moins avoir fait les exercices non marqués comme difficiles. Si vous n'avez pas eu le temps de les finir en TP, vous devez les finir par vous-même avant le prochain TP.

1 Exercices sur feuille

▸ **Exercice 1.**

Le créateur d'un jeu souhaite pouvoir modéliser des personnages participants à des combats. Un personnage aura un nom, des points de vie, et un bouclier lui permettant d'arrêter les coups. La structure de données choisie pour modéliser un personnage est donc la suivante :

```
1 struct Guerrier {
2   string nom;
3   int ptVie;
4   int bouclier;
5 };
```

La force des coups qu'il portera à son adversaire sera égale à son nombre de points de vie - 1 et il sera blessé s'il reçoit un coup d'une force supérieure à la force du bouclier qu'il porte. Un personnage blessé se verra retirer un nombre de point de vie égal à la différence entre la force du coup reçu et la force de son bouclier.

1. Spécifier et réaliser la procédure **afficheGuerrier** permettant d'afficher l'état d'un guerrier, i.e. son nom, et ses différentes propriétés.
2. Écrire la fonction **créeGuerrier** qui permet d'initialiser un personnage, sachant qu'il doit disposer au départ d'exactement 10 points à répartir entre les points de vie du personnage (au moins deux points) et la force de son bouclier. On devra pouvoir saisir le nom du personnage, par exemple «Elisa», et lui donner des points de vie, par exemple 7 points. Dans ce cas, la fonction **créeGuerrier** devra lui affecter automatiquement un bouclier d'une force égale à $3 = (10 - 7)$.

   Vous pouvez utiliser, sans la redéfinir ni la réaliser, la fonction ci-dessous :

   `int litValeurBornee(string texteAEcrire, int min, int max);`

   Cette fonction permet, après avoir affiché le `texteAEcrire` explicitant la valeur attendue, de saisir une valeur auprès de l'utilisateur, de vérifier qu'elle est bien comprise entre un `Min` et un `Max`, puis de la renvoyer quand elle respecte les bornes données.
3. Spécifier et réaliser la fonction **forceCoup** qui permet de renvoyer la force du coup porté par un guerrier, calculée comme égale à son nombre de points de vie - 1.

4. Spécifier et réaliser une fonction **reçoitCoup** qui prend en paramètre un guerrier et la force d'un coup et qui renvoie le guerrier après avoir reçu le coup.
5. On suppose déjà réalisée la fonction

   ```
   Guerrier initialiseGuerrierAutomatique(Guerrier G1);
   ```

   qui permet au concepteur de créer le guerrier G_2 qui combattra le guerrier G_1 créé par l'utilisateur du jeu. Réaliser le programme principal qui crée les deux guerriers, les affiche, enchaîne les coups qu'ils se portent jusqu'à ce que l'un des deux ne soit plus en état de se battre puis affiche le nom du gagnant et ses points de vie.

2 Exercices sur machine

Pour les redoublants : pour ne pas avoir de collision entre votre travail de l'année dernière et celui de cette année, il faut supprimer ou renommer votre ancien répertoire `ProgMod`, par exemple en tapant, dans un terminal, la commande `mv ProgMod ProgMod-2023`.

▸ **Exercice 2. (Environnement de travail)**

De la même manière qu'au premier semestre, on utilisera un script pour charger les sujets et soumettre vos travaux. Il faudra exécuter les commandes dans un terminal ayant le répertoire `ProgMod` comme répertoire courant (voir détails plus loin).

— Chargement : `./course.py fetch Semaine1`
— Soumission : `./course.py submit Semaine1 MonGroupe`

Toute séance de travail (chez vous ou à l'université) doit commencer par le chargement et se terminer par une soumission.

1. Pour mettre en place les dépots en début de semestre taper la commande suivante

   ```
   git clone https://gitlab.dsi.universite-paris-saclay.fr/L1InfoProgMod/ComputerLab.git ~/ProgMod
   ```

   Cette commande n'est à faire qu'**une seule fois lors du premier TP**.
2. Ensuite tout le travail se fera dans le répertoire `ProgMod`. Pour aller dans ce répertoire taper la commande

   ```
   cd ProgMod
   ```

   dans un terminal.
3. Pour charger le sujet, faire la commande

   ```
   ./course.py fetch Semaine1
   ```

   Ne pas créer de dossier au préalable, un dossier ProgMod sera automatiquement créé la première fois que vous faites cette commande.

   ☞ Si lorsque vous rentrez vos identifiants, vous n'êtes pas reconnu, c'est peut-être que vous ne vous êtes encore jamais connecté au serveur git de l'université Paris-Saclay. Connectez-vous sur `https://gitlab.dsi.universite-paris-saclay.fr` puis une fois la connexion réussie, essayez à nouveau la commande `./course.py fetch Semaine1` dans le terminal, en faisant bien attention de taper les mêmes login et mot de passe que lors de votre connection sur git.

   ☞ Il est possible que sur certaines machines où l'installation de travo se soit mal passée, si c'est le cas, il faut la commande suivante dans le terminal avant de pouvoir utiliser `course.py`.

   ```
   info-111 run bash
   ```

Cette commande peux prendre un peu de temps. Signaler le problème à votre enseignant en lui donnant le numéro de la machine.

4. Vérifier que vous obtenez bien un dossier ProgMod contenant un dossier Semaine1 contenant des fichiers.
5. N'oubliez pas d'exécuter la commande de soumission à la fin de la séance :

```
./course.py submit Semaine1 MonGroupe
```

Pour travailler chez vous : Vous pouvez utiliser le serveur JupyterHub (accessible à l'adresse `https://jupyterhub.ijclab.in2p3.fr/`) à distance. La même procédure est nécessaire pour travailler chez vous, y compris la première fois, la commande `git clone ...` du point 1.

ATTENTION ! La communication entre les salles de TP et JupyterHub, n'est pas automatique. Elle est faites grâce au `fetch` et `submit` que vous faites en début et fin de séance. Il faut donc bien se rappeler de les faire à chaque fois.

Editeur de texte Vous pouvez utiliser un éditeur de texte de votre choix. Selon votre système d'exploitation, vous pouvez choisir : sous Linux, `jedit, gedit, emacs, VScode` et sous Windows, `Notepad++, VScode`. Nous conseillons sur les postes à l'université de se connecter sous Linux et d'utiliser l'éditeur `VScode` qui se lance par la simple commande `code`.

Compilateur Un compilateur est nécessaire pour créer un fichier exécutable à partir d'un code source. Puisque nous développons en `C++`, ce semestre nous conseillons le compilateur `g++`. Il est accessible en ligne de commande (terminal). Normalement, ce compilateur est déjà installé sur votre machine.

La syntaxe à utiliser pour compiler un fichier est la suivante

```
clang++ -std=c++11 -Wall fichier.cpp -o nom_programme
```

Voici quelques explications :

- `-std=c++11` : Pour obliger le compilateur à suivre la norme `ISO C++11`
- `-Wall` : Pour afficher les messages d'avertissement (Warnings) durant la compilation
- `fichier.cpp` : Le code source c++ à compiler
- `-o nom_programme` : Pour spécifier un nom à l'exécutable

Note : en cas de problème, on peut remplacer le compilateur `clang++` par `g++` mais il a tendance à avoir de moins bon messages d'erreur.

Si vous avez des messages d'erreur à la compilation, il faut lire ces messages d'erreur en commençant par le premier pour voir comment corriger votre code ou votre commande de compilation. Vous pouvez consulter et compléter le document partagé qui contient des explications des messages.

À faire pour vérifier que tout est ok :

6. Vérifier que tout va bien en compilant le programme `bonjour.cpp` fourni dans l'archive grâce à la commande :

   ```
   g++ -std=c++11 -Wall bonjour.cpp -o bonjour
   ```

7. Exécuter le programme précédemment compilé (commande `./bonjour`).

▸ **Exercice 3. (Conversion Heures/Minutes/Secondes d'une durée)**

On représente une durée heure, minute, seconde sous la forme d'un type C++ structuré `Duree` où les heures et les minutes seront des entiers et les secondes un réel pour avoir des fractions de seconde. Par exemple : 5h 20min et $10,25$s sera représenté par la structure

```
{ .heures = 5; .minutes = 20; .secondes = 10.25 }
```

1. Ouvrir avec un éditeur de texte le fichier `convertHMS.cpp` fourni.
2. Compléter la définition du type `Duree`. Attention ! les champs `heures`, `minutes` et `secondes` sont au pluriels. Ne pas oublier le «s» à la fin.
3. Écrire une fonction `egalHMS` qui prend deux durées et qui retourne si elles sont égales.
4. Tester cette fonction en complétant les tests de la fonction `testEgalHMS`.

   On utilisera cette fonction dans la suite pour les tests.
5. Écrire une fonction

   ```
   float convertHMS2S(Duree hms)
   ```

   qui pour une durée exprimée sous la forme « heures, minutes, secondes » retourne la durée correspondante exprimée en secondes. Ajouter 3 autres tests pour cette fonction. Compiler et exécuter votre programme pour le vérifier ; ceci est à faire à chaque question dans tous les TPs, on ne le rappellera plus dans les énoncés, à vous de bien penser à le faire systématiquement.
6. Écrire une fonction

   ```
   Duree convertS2HMS(float d)
   ```

   qui pour une durée exprimée en secondes retourne la durée correspondante exprimée sous la forme « heures, minutes, secondes ». Ajouter 3 autres tests pour cette fonction.
7. Écrire une fonction

   ```
   void testHMS(Duree hms)
   ```

   qui vérifie qu'une durée est bien une durée sous la forme « heures, minutes, secondes » correcte. Une structure `hms` contient une heure correcte si $0 \leq$ `hms.heures` < 24, $0 \leq$ `hms.minutes` < 60 et $0 \leq$ `hms.secondes` < 60. On écrira les vérifications en utilisant la commande `CHECK` fournie.
   Note : `CHECK(condition)` permet de tester n'importe quelle condition. C'est juste un raccourci pour `if (not (condition)) cout << "Test failed ...";`
8. On va tester à grande échelle nos deux fonctions de conversion. Pour ceci, écrire un programme qui pour toutes les durées de 0 à 80000 secondes, en avançant de $0,25$s à chaque fois,
   — convertit la durée sous la forme « heures, minutes, secondes » ;
   — vérifie grâce à la fonction `testHMS` que le résultat obtenu est bien correct ;
   — re-convertit la durée en seconde et vérifie (avec `CHECK`) que l'on retrouve bien la durée initiale.

▸ **Exercice 4.(Calcul avec les dates).**

Il faut d'abord télécharger le fichier d'exemple du cours sur les dates qui se trouve dans la section Exemples sur le site du module (page du chargé de cours). Ce fichier sera nommé `date.cpp` et devra être déposé dans votre répertoire `ProgMod/Semaine1`. Pour que ce fichier soit pris en compte par le système d'archivage, il faut ensuite taper dans le répertoire du fichier la commande

```
git add date.cpp
```

Cette commande est à noter et à retenir. Elle vous sera en effet utile dès que vous voulez ajouter un fichier par rapport à ceux qui vous ont été fournis.

On vous demande d'ajouter et de tester les fonctions suivantes :

1. `bool avantDate(Date d1, Date d2)` qui dit si la date `d1` est avant la date `d2` ;
2. `Date ajouteDate(Date d, int n)` qui calcule la date qui suit `d` de `n` jours ;
3. `int differenceDate(Date d1, Date d2)` qui calcule le nombre de jours écoulés entre les dates `d1` et `d2` ;
4. Sachant que le 1er janvier 2000 était un samedi, ecrire une fonction `int jourDate(Date d)` qui retourne le jour de la semaine d'une date (renvoie 0 pour lundi, 1 pour mardi, ...).

3 Exercice complémentaire (pas forcément sur les structures)

▸ Exercice 5.Calendrier

Le but de cet exercice est d'afficher un calendrier en affichant mois par mois les jours de la semaine. Voici par exemple l'affichage du mois de février 2013 qui commence un vendredi :

```
       Fevrier
lu ma me je ve sa di
 .  .  .  .  1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28
```

Pour ceci, on déclare le nom des mois et leur longueur (dans une année usuelle) en C++ de la manière suivante :

```
vector<string> nom_mois = {
    "Janvier", "Fevrier", "Mars", "Avril", "Mai", "Juin", "Juillet",
    "Aout", "Septembre", "Octobre", "Novembre", "Decembre"
  };

vector<int> long_mois = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Attention : le mois de janvier porte le numéro 0.

1. Dans un premier temps, on suppose que le premier janvier est un lundi. On affichera les jours du mois en revenant à la ligne chaque dimanche sans se soucier des alignements. On sautera une ligne à la fin de chaque mois. On pourra utiliser trois compteurs :
   — `mois` pour stocker le numéro du mois en cours d'affichage ;
   — `j_mois` pour stocker le numéro dans le mois du jour en cours d'affichage ;
   — `j_sem` pour stocker le numéro dans la semaine du jour en cours d'affichage.
2. Écrire un programme qui demande à l'utilisateur le jour du premier janvier sous la forme (0 = lundi, 1 = mardi, 2 = mercredi, etc.) et qui affiche le calendrier de l'année.