Programmation pour le calcul scientifique
Année : 2019-2020
Formation : L3 Ingénierie Mathématique

TP5 : Classes

1 Une classe de nombres complexes

1. Dans un fichier header, déclarer une classe `complex` avec comme données privées deux réels représentant la partie réelle et la partie imaginaire d'un nombre complexe.
2. Dans un programme, créez une instance de la classe complex. Que se passe-t-il lorsque vous tentez de modifier la partie réelle ou imaginaire du nombre de la même manière que vous le faisiez pour les types structurés ?
3. Définir les fonctions suivantes comme membres publics de la classe `complex` :

   ```
   double& real() {return _re;};
   double  imag() {return _im;};
   ```

   Testez-les en utilisant

   ```
   complex p;
   p.real() = 1.;
   p.imag() = 1.;
   ```

   Que constatez-vous ? Comment l'expliquez-vous ?
4. Modifiez une des deux fonctions pour que l'on soit en mesure de changer la valeur des deux données.
5. Déclarez les fonctions comme privées. Que se passe-t-il ?
6. Écrire une fonction membre `display()` qui ne renvoie rien mais affiche à l'écran le nombre complexe au format $a + bi$. Quelle doit être la visibilité de cette fonction ?
7. Écrire un constructeur par défaut qui initialise le nombre à 0.
8. Écrire un constructeur avec un réel en argument qui n'initialise que la partie réelle avec l'argument, la partie complexe étant à 0.
9. Écrire un constructeur qui prend deux réels en argument pour les parties réelles et complexes respectivement.
10. Écrire un constructeur par copie.

2 Gestion de fichier de paramètres

Les codes de calcul scientifique ont la plupart du temps un nombre d'options et de paramètres très important pour contrôler le comportement des calculs. C'est pourquoi on privilégie souvent le regroupement de ces paramètres dans un fichier texte lu au début de l'éxécution du programme [1].

Voici un exemple de fichier de paramètres :

1. De plus, cela améliore la reproductibilité des résultats, qui est une des problématiques majeures de la science actuelle.

params.pm

```
# Output controls
output directory = ./myResults/
output period = 10

# Simulation times
start time = 0.
end time = 100.
time step = 0.1
```

Nous allons gérer un tel ensemble de paramètres à l'aide d'une classe `Parameters` que l'on souhaite utiliser de la manière suivante :

```
Parameters p("params.pm"); // Configuration is read from input file at instanciation
p.display(); // Prints on screen all parameters
std::string outDir = p.getString("output directory"); // The key "output directory"
                                                      // must be in the file,
                                                      // otherwise the program stops
int outNIters = p.getInt("output period",1); // We give a default value in case
                                             // the key "start time" is not in file.
                                             // Note the conversion to int.
double t0 = p.getReal("start time",0.); // Also works for doubles
double tf = p.getReal("end time",1.);
double dt = p.getReal("time step",.01);

// Modify configuration
p.set("start time","1.");
p.set("a new key","whatever");

// Save modified configuration in a new file
p.save("params2.pm");
```

1. Téléchargez l'ébauche de classe ainsi que les fichiers contenant des utilitaires pour traiter les chaînes de caractères.
2. Éditez un `Makefile` pour compiler tout ceci.
3. Regardez la composition de la classe `Parameters`. La structure de données sur laquelle elle se base est une `std::map`. On peut voir les `map` (tables de hachage) comme un tableau dont les indices ne sont pas les entiers 0, 1, 2, etc. mais des variables d'un type dont on demande seulement de posséder une relation d'ordre (`<`). C'est le cas du type `std::string` (ordre ∼alphabétique). Ainsi, de la même manière que l'on accède à un élément d'un `std::vector<double>` en écrivant `double a = v[3];`, on accède à un élément d'une `std::map<std::string,double>` en faisant `double a = m["hello"]`. Ici, nous avons une `std::map<std::string,std::string>`, la `map` associe donc une valeur de chaîne de caractère à une autre chaîne de caractère, que l'on appelle souvent la clé (*key*).
   Pour faciliter la vie des utilisateurs, qui peuvent ajouter accidentellement des espaces ou des majuscules aux clés dans le fichier de paramètres, toutes les clés sont passées en majuscules à l'aide de la fonction `toUpper` et les espaces sont retirés avec la fonction `stringTrim`.
   Voici quelques opérations qui vous serons utiles : si vous travaillez avec une `map` qui a des clés de type `T1` et des valeurs de type `T2` :

   ```
   std::map<T1,T2> m;

   // Add an element to the map:
   T1 k = ...;
   T2 v = ...;
   ```

```
m.insert(std::make_pair(k,v));

// Access an element of the map:
T2 w = m[k]; // BEWARE! If the key with the value of k does not exist
             // a new element is added to the map, and the corresponding
             // value may not be properly initialized.
             // This is a potential source of bugs that are hard to track
             // To avoid this, you can use
T2 x = m.at(k); // Safe access

// Test if the map has a key with the same value as k2
T1 k2 = ... ;
bool k2Found = (m.find(k2) != m.end());

// Loop through map, using C++11 syntax
for (auto& p: m) {
  // p's type is actually std::pair<T1,T2>&
  // The reference & can be dropped if the content will not be modified
  // we get key and value like this:
  T1 key = p.first;
  T2 val = p.second;

  // do something with key and val

}
```

4. Écrire une fonction membre `hasKey` qui retourne un booléen qui indique si une clé donnée en argument est présente dans les paramètres.
5. Écrire la fonction membre `display()` qui affiche à l'écran le contenu du fichier tel que stocké dans la classe. Écrire la fonction `save` qui fait de même dans un fichier.
6. Écrire la fonction `getString(std::string key, std::string default)` qui retourne la chaîne de caractères liée à l'entrée `key` si elle existe, `default` sinon.
7. Écrire une version de cette fonction sans la valeur par défaut. Cette version interrompt l'exécution du programme si la clé n'est pas donnée dans le fichier d'entrées.
8. Écrire les fonctions `getInt` et `getReal`, avec et sans valeur par défaut. Vous aurez besoin des fonctions standard de conversion `stoi` et `stod`.
9. Écrire la fonction membre `set` qui écrase la valeur liée à une clé donnée en argument si elle existe, ou crée cette entrée sinon.