Travaux Pratiques
Programmation Multi-Paradigme
Licence 3 Informatique

Julien BERNARD et Arthur HUGEAT

Table des matières

Consignes communes à tous les projets

Au cours de cette UE, vous avez **trois** projets à réaliser à raison d'un projet pour deux séances de trois heures de travaux pratiques encadrées. Les projets sont à faire et à rendre dans l'ordre du présent sujet. Vous avez le droit de vous mettre en binôme, dans ce cas vous devez préciser vos deux noms dans un fichier à la racine de vos rendus. Les projets faits en binôme ou en monôme seront évalués de la même manière.

Pour chaque projet, vous devrez implémenter une interface donnée dans un fichier d'en-tête, ainsi qu'un ensemble de tests unitaires pour cette interface. Les tests serviront à montrer que votre implémentation est correcte et complète.

Les sources doivent être écrites par vous-même, l'utilisation d'IA génératrice (ChatGPT, Copilot, Claude, ...) est donc strictement interdite. Si vous ré-utilisez un bout de code provenant d'un site web (Stack Overflow par exemple), vous devez mettre en commentaire le lien de votre source.

Il est attendu que vos codes sources et les commentaires soient rédigés en anglais et uniquement en anglais.

Projet n°1 : Sérialisation dans un fichier

Le but de ce projet est d'implémenter des classes pour lire et écrire des fichiers binaires, pour réaliser une bibliothèque de sérialisation. La sérialisation consiste à stocker des données en format binaire portable. Elle est utilisée pour la sauvegarde de fichiers, les protocoles réseau, etc.

Étape 1 : Fichier binaire

Dans cette étape, vous devrez implémenter deux classes, `OBinaryFile` pour les fichiers en écriture, et `IBinaryFile` pour les fichiers en lecture. Comme il s'agit de classes représentant des ressources, il sera nécessaire d'appliquer la *Rule of Five*.

L'implémentation de ces deux classes utilisera l'API C qui se trouve dans `<cstdio>`, il est donc formellement interdit d'utiliser toute autre API.

Pour permettre une meilleure compatibilité, le format du fichier binaire doit respecter ces règles :

1. Les données sont stockées dans l'ordre d'insertion : *First In, First Out (FIFO)*
2. Les collections d'objets doivent stocker d'abord le nombre d'éléments puis les objets dans le même ordre que leur itérateur.

Étape 2 : Opérateurs de sérialisation

Dans cette étape, vous utiliserez les deux classes définies précédemment pour sérialiser et désérialiser un ensemble de types de base en surchargeant les opérateurs `<<` et `>>`. Attention, le format de sérialisation devra être en *big endian*. On ne prendra pas en charge les pointeurs et les références.

Puis vous aurez à sérialiser des types conteneurs de la bibliothèque standard de manière générique.

Exemple d'utilisation

```
#include <cassert>
#include "Serial.h"

struct Foo {
  int16_t i;
  double d;
  std::string s;
};

serial::OBinaryFile& operator<<(serial::OBinaryFile& file,
    const Foo& foo) {
  return file << foo.i << foo.d << foo.s;
}

serial::IBinaryFile& operator>>(serial::IBinaryFile& file,
    Foo& foo) {
  return file >> foo.i >> foo.d >> foo.s;
}
```

```
int main() {
  Foo foo;
  foo.i = 42;
  foo.d = 69.0;
  foo.s = "Hello";

  {
    serial::OBinaryFile out("foo.bin");
    out << foo;
  }

  struct Foo copy;

  {
    serial::IBinaryFile in("foo.bin");
    in >> copy;
  }

  assert(foo.i == copy.i);
  assert(foo.d == copy.d);
  assert(foo.s == copy.s);
}
```