© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

Fiche de TDM : Programmation Extrême et Défensive

Il est question dans cette fiche de développer deux outils

1. Un outil de tests unitaires
2. Un outil de gestion d'erreur

Ces outils ont une place fondamentale dans la programmation défensive, ils constituent un des principes souteneurs des méthodes agiles dont la vocation est de fournir des logiciels prêts à usage à chaque étape du développement.

Thème - 1 *Génération de Projet*

Dans cette première partie, dite préliminaire, est apparaît tout naturel de générer un projet selon les règles standards de la distribution GNU `http://www.gnu.org` où `http://autotoolset.sourceforge.net/tutorial.html#SEC68`

Exercice-1 **:** Génération du squelette

1. Placez-vous dans un répertoire de travail. Et exécuter la commande suivante :

```
autoproject
```

Vous devez répondre aux questions posées, comme illustré dans le tableau suivant

```
 What is the program name? defensive_prog

Please describe in one line what defensive_prog will do:
realise un test unitaire et une gestion d'erreur
The main program will be written in what language?
    select from:  c sh c++ fortran lex yacc awk  [c]: c++
What other languages will be used in the package?
    select from:  c sh c++ fortran lex yacc awk  none  [none]: c
What other languages will be used in the package?
    select from:  c sh c++ fortran lex yacc awk  none  [none]:
What command line parser generator will be used?
    select from:  argp autogen clig none  [none]:
Please indicate which of the following standard options ↩
   defensive_prog will use:
    dry-run? [yN]
    no-warn? [yN]
    output? [yN]
    brief? [yN]
    quiet? [yN]
    verbose? [yN]
    directory? [yN]
    cd? [yN]
    interactive? [yN]
```

```
What is the name of the author? [Jean-Baptiste APOUNG KAMGA]:
What is the email address of the author? [apoung@apoungPCMomas.math.u↩
    -psud.fr]: jean-baptiste.apoung@math.u-psud.fr
```

2. Qu'observez-vous ?
3. Rentrez dans le répertoire créé : ici **defensive_prog** .
   (a) Qu'observez-vous ? Analysez le contenu des fichiers AUTHORS, COPYING, INSTALL, NEWS, README. Il faudra grandement faire attention au contenu du fichier COPYING, qui contient la licence, qui contrôle l'usage du logiciel en cours de développement.
   (b) Exécutez et commentez le résultat de la commande

```
./configure --help
```

   (c) Exécutez **`./configure en donnant une valeur à l'option --prefix`** : cette valeur devra être un chemin absolu, vers un répertoire. Vous pourrez exécuter

```
./configure --prefix=`pwd`/..
```

   Assurez-vous alors de la présence d'un fichier **`Makefile`**
   (d) Entrer ensuite les commandes

```
make
```

   Qu'observez-vous ?
   (e) Entrer maintenant la commande

```
make install
```

   Qu'observez-vous ?
4. Après avoir nettoyé le projet

```
make distclean
```

   Reprendre les commandes précédentes (de compilation et d'installation, mais cette-foi-ci à partir d'un répertoire différent, dit répertoire de construction (ou *build directory* en anglais)

Exercice-2 : Modification du squelette et Ajouts de fichiers

1. Ouvrir le fichier **`defensive_prog.cc`** et supprimer y les lignes de codes inutiles.
2. Renommer le fichier **`defensive_prog.cpp`** et modifier le fichier **`Makefile.am`** pour prendre en compte cette modification. Puis recompiler le projet.
3. Essayer d'ajouter des fichiers au projet. Vous pourrez par exemple créer une classe (soit dit le cours fait entièrement usage du language `C++`) (fichier entête .hpp et source .cpp) et l'évaluer dans le le fichier **`defensive_prog.cpp`**.

Exercice-1 : Interfaces

Un outils de tests unitaire est constitué de essentiellement de

1. D'une classe de base **`Test`**, dont le but est de charger et d'exécuter un test (méthode virtuelle **`run`**) et de fournir une rapport **`report`**) consistant en le nombre d'échecs et le nombre de succès. Ainsi que les fonctions responsables de l'échec. Ces fonctions sont répertoriées par leur nom, le fichier les contenants et la ligne où l'erreur c'est observée. Ces dernières informations nécessites l'utilisation des **`macros`** emprunté au langage **`C`**, comme illustré ci-dessous.

fichier:Test.hpp

```
#ifndef _TEST_HPP_
#define _TEST_HPP_
#include <string>
#include <iostream>
#include <cassert>
using std::string;
using std::ostream;
using std::cout;

#define test_(cond) \
  do_test(cond, #cond, __FILE__, __LINE__)
#define fail_(str) \
  do_fail(str, __FILE__, __LINE__)

namespace TestSuite{
class Test
{
public:
Test(ostream* osptr = &cout) {
    this->osptr = osptr;
    nPass = nFail = 0;
  }
  virtual ~Test() {}
  virtual void run() = 0; // fonction virtuelle
  long getNumPassed() const { return nPass; }
  long getNumFailed() const { return nFail; }
  const ostream* getStream() const { return osptr; }
  void setStream(ostream* osptr) { this->osptr = osptr; }
  void succeed_() { ++nPass; }
  long report() const;
  virtual void reset() { nPass = nFail = 0; }

protected:
void do_test(bool cond, const string& lbl,
    const char* fname, long lineno);
  void do_fail(const string& lbl,
    const char* fname, long lineno);
private:
ostream* osptr;
long nPass;
long nFail;
// Disallowed:
Test(const Test&);
Test& operator=(const Test&);
};

}//TestSuites
#endif // _TEST_HPP_
```

2. D'une classe de base **Suite**, dont le but est d'enregistrer et de stocker un ensemble de tests (méthodes **addTest, addSuites**) , puis de les exécuter (méthode **run**) à la demande tout en prenant soin de fournir un rapport détaillé (méthode **report**).

fichier:Suites.hpp

```
#ifndef SUITE_H
#define SUITE_H
#include <vector>
#include <stdexcept>
#include "Test.h"
using std::vector;
using std::logic_error;

namespace TestSuite {

class TestSuiteError : public logic_error {
public:
  TestSuiteError(const string& s = "")
  : logic_error(s) {}
};

class Suite {
  string name;
  ostream* osptr;
  vector<Test*> tests;
  void reset();
  // Disallowed ops:
  Suite(const Suite&);
  Suite& operator=(const Suite&);
public:
  Suite(const string& name, ostream* osptr = &cout)
  : name(name) { this->osptr = osptr; }
  string getName() const { return name; }
  long getNumPassed() const;
  long getNumFailed() const;
  const ostream* getStream() const { return osptr; }
  void setStream(ostream* osptr) { this->osptr = osptr; }
  void addTest(Test* t) throw(TestSuiteError);
  void addSuite(const Suite&);
  void run();  // Calls Test::run() repeatedly
  long report() const;
  void free();  // Deletes tests
};

} // namespace TestSuite
#endif // SUITE_H ///:~
```

Exercice-2 : Implantations

1. En vous référant au cours, fournir des implantations (fichiers .cpp) des classes ci-dessus.
2. Insérer ces classes dans le projet initiale (voir plus haut).

Exercice-3 : Utilisations

Supposons que l'on dispose d'une classe Date comme ci-dessous (vous pourrez utiliser une classe de votre choix) accessible dans mon répertoire associé au présent cours (vous prendrez les fichiers Date.hpp, Date.cpp,) :

fichier:Date.hpp

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- ↩
   */
/*
 * DefenSiveProg
 * Copyright (C) Jean-Baptiste Apoung Kamga 2011 <jean-baptiste.math.u-↩
    psud.fr>
 *
 * DefenSiveProg is free software: you can redistribute it and/or modify ↩
    it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DefenSiveProg is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#ifndef _DATE_HPP_
#define _DATE_HPP_
#include <string>
#include <stdexcept>
#include <iosfwd>

class Date {
  int year, month, day;
  int compare(const Date&) const;
  static int daysInPrevMonth(int year, int mon);
public:
  // A class for date calculations
  struct Duration {
    int years, months, days;
    Duration(int y, int m, int d)
    : years(y), months(m) ,days(d) {}
  };
  // An exception class
  struct DateError : public std::logic_error {
    DateError(const std::string& msg = "")
      : std::logic_error(msg) {}
  };
  Date();
  Date(int, int, int) throw(DateError);
  Date(const std::string&) throw(DateError);
  int getYear() const;
  int getMonth() const;
  int getDay() const;
  std::string toString() const;
  friend Duration duration(const Date&, const Date&);
  friend bool operator<(const Date&, const Date&);
  friend bool operator<=(const Date&, const Date&);
  friend bool operator>(const Date&, const Date&);
  friend bool operator>=(const Date&, const Date&);
  friend bool operator==(const Date&, const Date&);
  friend bool operator!=(const Date&, const Date&);
  friend std::ostream& operator<<(std::ostream&,
                                  const Date&);
  friend std::istream& operator>>(std::istream&, Date&);
};
```

```
#endif // _DATE_HPP_
```

Que l'on souhaite tester. On peut fournir un fichier **DateTest.hpp** comme suit :

fichier:DateTest.hpp

```
/*
 * DateTest.hpp
 *
 * Copyright (C) 2011 - Apoung Kamga Jean-Baptiste
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DATETEST_HPP
#define DATETEST_HPP

#include "Date.hpp"     // class a tester
#include "Test.hpp"     // test simple
class DateTest : virtual public TestSuite::Test {
  Date mybday;
  Date today;
  Date myevebday;
public:
  DateTest(): mybday(1951, 10, 1), myevebday("19510930") {}
  void run() {
    testOps();
    testFunctions();
    testDuration();
  }
  void testOps() {
    test_(mybday > today);
    test_(mybday <= today);
    test_(mybday != today);
    test_(mybday == mybday);
    test_(mybday >= mybday);
    test_(mybday <= mybday);
    test_(myevebday < mybday);
    test_(mybday > myevebday);
    test_(mybday >= myevebday);
    test_(mybday != myevebday);
  }
  void testFunctions() {
    test_(mybday.getYear() == 1951);
    test_(mybday.getMonth() == 10);
    test_(mybday.getDay() == 1);
    test_(myevebday.getYear() == 1951);
    test_(myevebday.getMonth() == 9);
    test_(myevebday.getDay() == 30);
    test_(mybday.toString() == "19511001");
    test_(myevebday.toString() == "19510930");
  }
```

```
  void testDuration() {
    Date d2(2003, 7, 4);
    Date::Duration dur = duration(mybday, d2);
    test_(dur.years == 51);
    test_(dur.months == 9);
    test_(dur.days == 3);
  }
};

#endif // DATETEST_HPP ///:~
```

Q-3-1 : Implanter cette classe ou une autre de votre choix et intégrer la dans le projet

Les appels peuvent alors se faire de la manière suivante :

Appel Simple

```
#include < iostream >
#include " Date.hpp"
#include " DateTest.hpp "

using namespace std ;
using namespace TestSuite ;

int main ( int argc , char ** ↩
    argv )
{

DateTest test ;

test.run () ;

return test.report () ;
}
```

Appel Evolué

```
# include < iostream >
# include " Date.hpp"
# include " DateTest.hpp"
# include " Suite.hpp"

using namespace std ;
using namespace TestSuite ;
int main ( int argc , char ** argv↩
    ) {
Suite suite (" Quelques Tests ");
// insersion du test de Date
suite.addTest ( new DateTest );
suite.run () ;
long nFail = suite.report () ;
suite.free () ;
return nFail ;
}
```

Q-3-2 : Intégrer ces appels dans le projet précédent. Valider l'outil obtenu. En observant sa réaction en cas d' échec d'un test (on modifiera le test pourqu'il y ait échec !)

Exercice-4 : Allez plus loin

Pour allez plus loin dans les tests, vous pourrez modifier :

1. Modifier le Makefile.am pour que et le projet de sorte que la commande

   ```
   make check
   ```

   Exécute tous les tests du projet.

2. La plupart des projets industriels dans lesquels vous aurez à intervenir, repose leur outils de test sur des bibliothèques extérieur. Il vous appartiendra de vous y conformer. A ce titre, l'exercice suivant voit sont utilité :

   Rechercher sur la toile télécharger puis intégrer dans le projet la bibliothèque de test unitaires suivantes :

   (a) cuttee `http://www.codesink.org/cutee_unit_testing.html`

   (b) cpptest `http://cpptest.sourceforge.net/`

   Pour conclure vous pourrez lire la documentations sur les bibliothèques plus sophistiquées suivantes :

   (a) Boost Test `http://www.boost.org/doc/libs/1_48_0/libs/test/doc/html/index.html`

   (b) QTtest `http://doc.qt.nokia.com/4.7-snapshot/qttest.html`

Thème - 3 *Développent d'un outil de contrôle d'erreur*

Renseignez-vous sur la gestion des exceptions en C++ voir **stdexcept**.

Le travail consistera à

1. Répertorier pour un projet un ensemble d'erreurs courantes, la plupart lié à l'utilisation des bibliothèques tierces.
2. Fournir pour chacune des sous-classes de la classe **std::exception**
3. A chaque fois qu'une erreur particulière est susceptible de se produire, lancer la bonne exception au cas ou elle se produirait.

On prendra soin de faire usage de **macro** pour faciliter l'utilisation de loutil développer.

Le squelette de l'appel ressemblerait à :

```
PROJET_ERR_CHECK(err)
```

où **err** serait un code de retour et la macro PROJET_ERR_CHECK saura lancer la bonne exception.