CE320-6-AU

UNIVERSITY OF ESSEX

Undergraduate Examinations 2014

LARGE SCALE SOFTWARE SYSTEMS AND EXTREME PROGRAMMING

Time allowed: **TWO** hours

Candidates must answer **ALL** questions**.**

The paper consists of **FOUR** questions.

The questions are **NOT** of equal weight.

The percentages shown in brackets provide an indication of the proportion of the total marks for the **PAPER** which will be allocated.

Please do not leave your seat unless you are given permission by an invigilator.
Do not communicate in any way with any other candidate in the examination room.
Do not open the question paper until told to do so.
All answers must be written in the answer book(s) provided.
All rough work must be written in the answer book(s) provided. A line should be drawn through any rough work to indicate to the examiner that it is not part of the work to be marked.
At the end of the examination, remain seated until your answer book(s) have been collected and you have been told you may leave.

Candidates must answer ALL questions

Question 1

Extreme programming (XP) designs software in a very different way from more traditional forms of software development. Explain what are they key design strategies and mechanisms in XP. [20%]

Question 2

Clean code is an essential element of XP.

a) Explain why clean code is so important. [10%]

b) List what are the requirements of clean code in relation to:

1) Naming of variables, functions/methods and classes [5%]

2) Comments [5%]

3) Size and organisation of methods/functions [5%]

Question 3

You are part of an extreme programming team that has been asked to build the control software for a new type of ironing machine for laundrettes. When fully developed this will be able to automatically and rapidly iron t-shirts, shorts, trousers, shirts, skirts and bed sheets. The machine has a touch screen which will allow interactions with its customers.

You are asked to sketch a release plan for the software that will control the machine. Given the specific nature of the project you can act both as a programmer and as a client (the company making the ironing machine). The hardware for the fundamental functions of the machine (e.g., ironing different types of items) will be provided to you.

In your release plan please specify:

(a) A list of the features identified for the release. If you require further special hardware to implement a feature, indicate so in the plan. [15%]

(b) A priority for each item in terms of value for the client (the company making the ironing machine) using a point system where 3 = very valuable, 2 = valuable, 1 = optional. [5%]

(c) A list of the features that should be implemented in the first iteration. [5%]

Question 4

Test driven development (TDD) is a key practice in extreme programming. Your XP team has started developing a simple CPU simulator using TDD.

At the moment the CPU has two registers `R0` and `R1` which are meant to be used for calculations. The CPU can execute programs (linear sequences of low-level instructions) and has a program counter which points to the next instruction to be executed in a program and is automatically incremented by the CPU after executing the instruction.

At the moment the team has defined a small set of tests and produced corresponding production code that passes them. In particular only the

```
SET Register Value
```

instruction (which sets register `Register` to value `Value`) has been implemented.

For easier testability, the simulator has been provided with the methods `getReg(registerNumber)` which returns the value of register `registerNumber` and `getPC()` which returns the program counter.

Below you will find **three implementations** of the simulator class: one in Java, one in C++ and one in C.

Choose one implementation, study it, then do the following:

(a) **write a test** that would force the addition of a third register in the CPU and **indicate how the existing implementation of the CPU would need to be modified** in order to pass this new test without failing any of the previous ones. [10%]

(b) **write a test** that will force the implementation of an addition instruction of the following form [20%]

```
ADD ResultRegister Operand1Register Operand2Register
```

which allows adding the content of any register to any other register and saving the result in any register, and **indicate how the existing implementation of the CPU would need to be modified** in order to pass this new test without failing any of the previous ones.

JAVA Implementation

```
// ---------------------------------------------------------------------
// CPUshould.java
// ---------------------------------------------------------------------

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

public class CPUshould {
	private CPU cpu;

	@Before
	public void setup() {
		cpu = new CPU();
	}

	@Test
	public void haveItsArithmeticRegistersAndProgramCounterZeroOnCreation() {
		assertEquals(0, cpu.getReg(0) );
		assertEquals(0, cpu.getReg(1) );
		assertEquals(0, cpu.getPC() );
	}

	@Test
	public void haveASetRegisterInstruction() {
		cpu.runProgram("SET R0 10", "SET R1 99");
		assertEquals( 10, cpu.getReg(0) );
		assertEquals( 99, cpu.getReg(1) );
		assertEquals( 2, cpu.getPC() );
	}
}
```

```
// ---------------------------------------------------------------------
// CPU.java
// ---------------------------------------------------------------------
public class CPU {
	private int[] registers = new int[2];
	private int programCounter;

	public int getPC() {
		return programCounter;
	}

	public int getReg(int regNumber) {
		return registers[regNumber];
	}

	public void runProgram(String... program ) {
		for ( programCounter = 0; programCounter < program.length;
		      programCounter++ )
		          execute(program[programCounter]);
	}

	private void execute(String instruction) {
		String[] elements = instruction.split(" ");
		if ( elements[0].equals("SET") )
		       opCodeSET(elements);
	}

	private void opCodeSET(String[] elements) {
		int value = Integer.parseInt(elements[2]);
		int register = regNumber(elements[1]);
		registers[register] = value;
	}

	private int regNumber(String register) {
		// register = "Rx" where x is the register number
		return Integer.parseInt(register.substring(1,2));
	}
}
```

C++ Implementation

```
// ------------------------------------------------------------------
// CPUshould.cpp
// ------------------------------------------------------------------

#include <CppUTest/CommandLineTestRunner.h>
#include <string>
#include <vector>
#include <iostream>
#include "CPU.h"

int main(int argc, char **argv ) {
	return CommandLineTestRunner::RunAllTests(argc,argv);
}

TEST_GROUP(aCPUshould) {
	CPU cpu;
};

TEST(aCPUshould, haveItsArithmeticRegistersAndProgramCounterZeroOnCreation) {
	CHECK_EQUAL(0, cpu.getReg(0) );
	CHECK_EQUAL(0, cpu.getReg(1) );
	CHECK_EQUAL(0, cpu.getPC() );
}

TEST(aCPUshould, haveASetRegisterInstruction) {
	std::vector<std::string> program = {"SET R0 10", "SET R1 99"};
	cpu.runProgram(program);
	CHECK_EQUAL(10, cpu.getReg(0) );
	CHECK_EQUAL(99, cpu.getReg(1) );
	CHECK_EQUAL(2, cpu.getPC() );
}
```

```
// ------------------------------------------------------------------
// CPU.h
// ------------------------------------------------------------------

#include <string>
#include <vector>
using namespace std;

class CPU {
      private:
      unsigned int programCounter;
      int registers[2];

      int regNumber(register string);
      void opCodeSET(string element1, string element2 );
      void execute(string instruction);
      vector<string> splitAtSpaces(string instruction);

      public:
            int getPC();
            int getReg(int regNumber);
            void runProgram(vector<string> program );
};
```

```
// ------------------------------------------------------------------
// CPU.cpp
// ------------------------------------------------------------------

#include "CPU.h"
#include <vector>
#include <string>
#include <iterator>
#include <sstream>
using namespace std;

int CPU::getPC() {
	return programCounter;
}

int CPU::getReg(int regNumber) {
	return registers[regNumber];
}

void CPU::runProgram(vector<string> program) {
	for ( programCounter = 0; programCounter < program.size(); programCounter++) {
		execute(program[programCounter]);
	}
}

void CPU::execute(string instruction) {
	vector <string> element = splitAtSpaces(instruction);
	if ( element[0].compare("SET") == 0 )
		opCodeSET(element[1],element[2]);
}

vector<string> CPU::splitAtSpaces(string instruction) {
	istringstream iss(instruction);
	vector <string> element { istream_iterator<string> {iss},
				  istream_iterator<string> {} };
	return element;
}

void CPU::opCodeSET(string element1, string element2 ) {
	int value = atoi(element2.c_str()); // atoi = ascii string to integer
	int reg = regNumber(element1);
	registers[reg] = value;
}

int CPU::regNumber(string reg) {
	// reg = "Rx" where x is the register number
	return atoi(reg.substr((size_t)1,(size_t)1).c_str());
}
```

C Implementation

```
// ------------------------------------------------------------------
// CPUshould.c
// ------------------------------------------------------------------

#include "cut.h"
#include "CPU.h"
#include <stdio.h>
#include <string.h>

// This macro uses a standard trick to compute
// the number of instructions in a program
#define progSize(program) ( sizeof(program) / sizeof(program[0]))


CPU cpu;

void __CUT_BRINGUP__Pass( void ){
      CPU cpu = createCPU();
}

void __CUT_TAKEDOWN__Pass( void ) {}

void __CUT__haveItsArithmeticRegistersAndProgramCounterZeroOnCreation() {
      ASSERT_EQUALS(0, getReg(&cpu, 0) , "" );
      ASSERT_EQUALS(0, getReg(&cpu, 1) , "" );
      ASSERT_EQUALS(0, getPC(&cpu), "" );
}

void __CUT__haveASetRegisterInstruction() {
      const char *program[] = {
                  "SET R0 10",
                  "SET R1 99" };
      runProgram(&cpu, program, progSize(program) );
      ASSERT_EQUALS( 10, getReg(&cpu, 0) , "" );
      ASSERT_EQUALS( 99, getReg(&cpu, 1) , "" );
      ASSERT_EQUALS(2, getPC(&cpu), "" );
}
```

```
// ------------------------------------------------------------------
// CPU.h
// ------------------------------------------------------------------

typedef struct CPUs {
      int registers[2];
      int programCounter;
} CPU;

CPU createCPU();
int getPC(CPU *cpu );
int getReg(CPU *cpu, int regNumber);
void runProgram(CPU *cpu, const char  *program[], int programLength );
void execute(CPU *cpu, const char *instruction);
char* getInstructionName(const char* instruction);
char* getNextInstructionElement();
void opCodeSET(CPU *cpu, char *element1, char *element2);
int regNumber(char *reg);
```

```c
// ------------------------------------------------------------------
// CPU.c
// ------------------------------------------------------------------

#include "cut.h"
#include "CPU.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

CPU createCPU() {
      CPU cpu = {0,0,0};
      return cpu;
}

int getPC(CPU *cpu ) {
      return cpu->programCounter;
}

int getReg(CPU *cpu, int regNumber) {
      return cpu->registers[regNumber];
}

void runProgram(CPU *cpu, const char  *program[], int programLength ) {
      for ( cpu->programCounter = 0; cpu->programCounter < programLength;
            cpu->programCounter++ )
                  execute(cpu, program[cpu->programCounter]);
}

void execute(CPU *cpu, const char *instruction) {
      char* element = getInstructionName(instruction);
      if ( strcmp(element, "SET") == 0 ) {
            char *e1 = getNextInstructionElement(),
                 *e2 = getNextInstructionElement();
            opCodeSET(cpu,e1,e2);      // Use temp variables as parameter evaluation
                                       // is not guaranteed from left-to-right in C
      }
}

char* getInstructionName(const char* instruction) {
   static char instrCopy[100];   // Making a static copy of string as strtok splits
   strcpy(instrCopy, instruction);// it into elements by modifying its first
                                  // argument
   return strtok(instrCopy, " "); // Return first word in string (instruction type)
}

char *getNextInstructionElement() {
      return strtok(NULL, " ");     // Return next word in string
}

void opCodeSET(CPU *cpu, char *element1, char *element2) {
      int reg = regNumber(element1);
      int value = atoi(element2); // atoi = ascii string to integer
      cpu->registers[reg] = value;
}
```

```
int regNumber(char *reg) {
	if ( reg[1] == '0' ) // reg = "Rx" where x = '0' or '1'
		return 0;
	else
		return 1;
}
```