SCHOOL OF **ELECTRICAL** AND
**INFORMATION ENGINEERING**
University of the Witwatersrand
Software Development II

# September Test 2018: 1 Hour – 35 marks

**Instructions**

- Answer *all* questions. The questions do not carry equal weight.
- For questions which require you to write source code, note that:
  - You only need to specify `#include`'s if specifically asked.
  - For classes, you can give the implementation entirely in the header file, unless directed otherwise.
  - Marks are not awarded solely for functionality but also for good design, making appropriate use of library functions, following good coding practices, and using a modern, idiomatic C++ style.
  - Your code must be easily understandable or well commented.
  - You may use pencil but then you forfeit the right to query the marks.
- Reference sheets are provided separately.

## Question 1

a) Given the sequence of `git` commands shown in Listing 1, provide a graphical representation of the commit history. Clearly indicate all commits, branches, and HEAD. Assume that the initial commit takes place on the *master* branch and that the files under version control have changed before each commit. (4 marks)

```
$ git commit -am "Initial commit"
$ git commit -am "first commit"
$ git commit -am "second commit"
$ git branch feature
$ git checkout feature
$ git commit -am "third commit"
$ git commit -am "fourth commit"
$ git commit -am "fifth commit"
$ git checkout master
```

**Listing 1:** A sequence of `git` commands

b) Explain what is meant by the term *fast-forward merge*. (2 marks)

c) Will a fast-foward merge take place, if "`git merge feature`" is input after the sequence in Listing 1?

(1 marks)

[Total Marks 7]

## Question 2

Examine Listings 2 and 3 and give the output of the `main` program, assuming that no compiler optimisations take place. Explain why each line of the output is present.

```cpp
class Person
{
public:
  Person(const string& name): name_{name}
  { cout << "Constructing a person" << endl; }

  Person(const Person& rhs): name_{rhs.name_}
  { cout << "Copying a person" << endl; }

  ~Person()
  { cout << "Destructing a person" << endl; }

private:
  string name_;
};
```

**Listing 2:** `Person` class

```cpp
Person newPerson()
{
   return Person{"Bobby"};
}

int main()
{
   newPerson();

   return 0;
}
```

**Listing 3:** Using `Person`

[Total Marks 8]

## Question 3

a) Write a function called `remove_vowels`, and any other code which may be required, to delete all of the vowels from a given string. The behaviour of `remove_vowels` can be discerned from the tests given in Listing 4.

```
TEST_CASE("Remove all lowercase vowels from string")
{
   auto sentence = string{"This sentence contains a number of vowels."};

   auto result = remove_vowels(sentence) ;

   CHECK(sentence == "This sentence contains a number of vowels.");
   CHECK(result == "Ths sntnc cntns  nmbr f vwls.");
}

TEST_CASE("Remove all upper and lowercase vowels from string")
{
   auto sentence = string{"A sentence starting with the letter 'A'."};

   auto result = remove_vowels(sentence);

   CHECK(sentence == "A sentence starting with the letter 'A'.");
   CHECK(result == " sntnc strtng wth th lttr ''.");
}
```

**Listing 4:** Tests for the `remove_vowels` function

In order to obtain full marks for this question your solution *may not contain* any `if` statements, `switch` statements, or loops of any kind. Partial marks will be awarded for solutions which violate these constraints.

*Hint:* Make use of your reference sheets, and remember that a `string` is essentially a `vector` of characters so all of `vector`'s functions apply to it.

(14 marks)

b) What are the benefits of *not* writing your own branching statements and loops?

(1 marks)

[Total Marks 15]

## Question 4

a) The definition of a Box class is given in Listing 5. Boxes are created and used as shown in Listing 6.

Provide an additional member function for Box that will return the total number of boxes that have been created as well as *all* other code changes that are required for this member function to work correctly. Note, your changes must not break existing client code.

```
class Box
{
public:
   Box(double length, double breadth, double height): length_{length},
      breadth_{breadth},
      height_{height}
      {}

   Box() = default;

   double volume() const { return length_ * breadth_ * height_; }

private:
   double length_ = 1.0;
   double breadth_ = 1.0;
   double height_ = 1.0;
};
```

**Listing 5:** Box class

```
   auto box_1 = Box{3.3, 1.2, 1.5};
   cout << box_1.volume() << endl;

   auto box_2 = Box{};
   cout << box_2.volume() << endl;
```

**Listing 6:** Using Box

[Total Marks 5]