| | | | |
|---|---|---|---|
| hrs | / /20 | | Exams Office Use Only |

University of the Witwatersrand, Johannesburg

| | |
|---|---|
| Course or topic No(s) | ELEN3009 |
| Course or topic name(s) Paper Number & title | Software Development II |
| Examination/Test* to be held during month(s) of (*delete as applicable) | November 2015 |
| Year of Study (Art & Sciences leave blank) | Third |
| Degrees/Diplomas for which this course is prescribed (BSc (Eng) should indicate which branch) | BSc(Eng)(Elec) |
| Faculty/ies presenting candidates | Engineering and the Built Environment |
| Internal examiners and telephone number(s) | Dr SP Levitt x77209 |
| External examiner(s) | Mr J Lewis |
| Special materials required (graph/music/drawing paper) maps, diagrams, tables, computer cards, etc) | Computer card for multiple-choice questions |

Time allowance

| | | | |
|---|---|---|---|
| Course Nos | ELEN3009 | Hours | 3 |

Instructions to candidates (Examiners may wish to use this space to indicate, inter alia, the contribution made by this examination or test towards the year mark, if appropriate)

(a) Read instructions on page 1 of exam
(b) Available marks: 110 - Full marks: 100
(c) Closed-book exam
(d) Basic scientific calculator allowed

Internal Examiners or Heads of School are requested to sign the declaration overleaf

1. As the Internal Examiner/Head of School, I certify that this question paper is in final form, as approved by the External Examiner, and is ready for reproduction.

2. As the Internal Examiner/Head of School, I certify that this question paper is in final form and is ready for reproduction.

(1. is applicable to formal examinations as approved by an external examiner, while 2. is applicable to formal tests not requiring approval by an external examiner—Delete whichever is not applicable)

Name:________________________________________ Signature:

(THIS PAGE NOT FOR REPRODUCTION)

**Instructions**

- Answer *all* questions. The questions do not carry equal weight.
- The paper is divided into two parts:
  - **Part A** consists of several multiple choice questions which are to be answered on the computer card provided. You must fill in your student number on the card. There may be more than one correct answer for each of the multiple choice questions, for example (a) and (e). Indicate ALL the correct answers for each question by clearly marking the appropriate blocks of the chosen letters on the card with a dark HB pencil. A negative marking system will be adopted so **DO NOT GUESS**. Marks will be subtracted for incorrect answers. You cannot, however, get less than zero for any single multiple choice question.
  - **Part B** consists of three (3) questions to be answered legibly in the answer book provided.
- For questions which require you to write **source code**, note that:
  - Pencil may be used.
  - 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, following good coding practices, and the use of idiomatic C++.
  - Your code must be easily understandable or well commented.
- Reference sheets are provided in a separate appendix.

## Part A

### Question 1

1.1 Which of the following statements are true? (5 marks)

(a) The following statement will compile without an error:

```
int& r = 5;
```

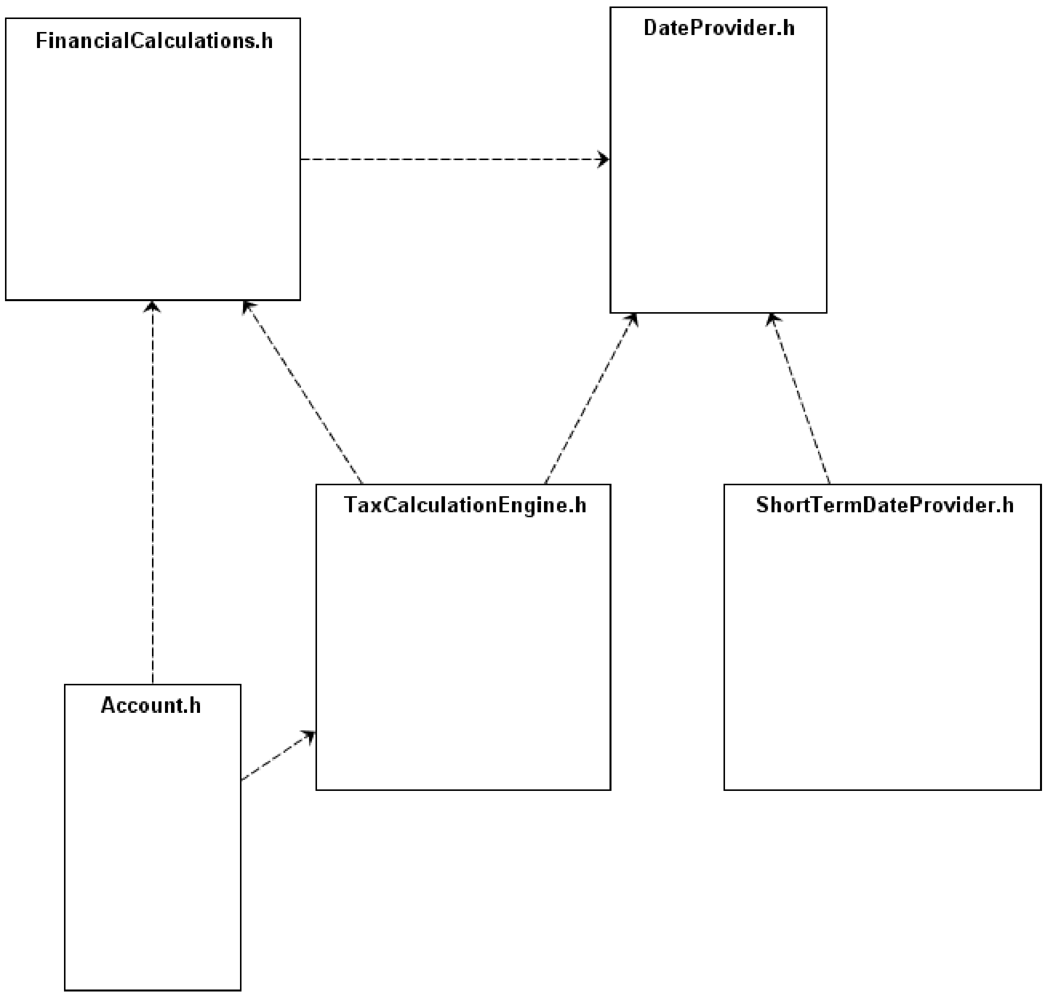
(b) It is possible to have a pointer, to a pointer, to an integer.

(c) It is possible to have a reference, to a reference, to an integer.

(d) It is possible to have a pointer, to a reference, to an integer.

(e) It is possible to have a reference, to a pointer, to an integer.

1.2 Examine the following diagram depicting dependencies among C++ *header* files.



Which of the following statements concerning the diagram are true? (5 marks)

(a) Header include guards will prevent the linker from multiply including `FinancialCalculations.h` in `Account.h`

(b) The diagram correctly depicts the dependency between `DateProvider.h` and `ShortTermDateProvider.h`, given that class `ShortTermDateProvider` inherits from class `DateProvider`. Assume that `DateProvider` is declared in `DateProvider.h` and `ShortTermDateProvider` is declared in `ShortTermDateProvider.h`.

(c) Code in `TaxCalculationEngine.h` and `TaxCalculationEngine.cpp` is able to make use of `ShortTermDateProvider` objects.

(d) Code in the `Account` module makes use of financial calculation functions but it is preferable not to include `FinancialCalculations.h` in `Account.h` as it will be included anyway via `TaxCalculationEngine.h`.

1.3 SFML is a gaming library which is available for use in C++. Which of the following statements concerning SFML are <u>true</u>? (5 marks)

(a) When setting up a C++ project which makes use of SFML there is no need to add the SFML header files to the project in the IDE, if the include path to the header files has been specified in the project settings.

(b) You need to specify the location of the SFML library files for the pre-processor.

(c) The functionality provided by SFML is part of the `std` namespace.

(d) Linking with the SFML library files is done after the code is compiled.

(e) Dynamically linking to the SFML libraries means that the size of the executable using SFML is smaller than if it were statically linked to the libraries.

1.4 Which of the following statements are <u>true</u>? (5 marks)

(a) A vector uses more memory than an array of an equivalent size.

(b) The following program demonstrates correct use of the vector class.

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

int main()
{
    vector<int> f;
    f.reserve(5);
    f[0] = 12;
    f[1] = 5;

    return 0;
}
```

(c) A vector's capacity is always greater than its size.

(d) Accessing an element using `v[i]` is faster than using `v.at(i)`.

(e) The time taken to access the last element in a vector is proportional to the total number of elements in the vector.

1.5 Assume that there are two classes that model distinct concepts but have a fair amount of implementation details in common. Which of the following designs for handling this scenario, follow good coding principles? (5 marks)

(a) Capture the commonality in a separate class and let both classes be publicly derived from this class.

(b) Capture the commonality in a separate class and provide it to the two classes via composition.

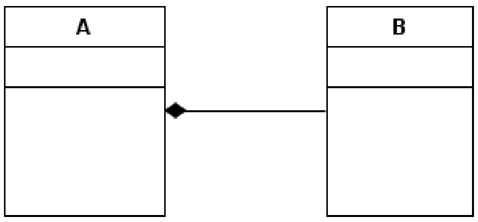
(c) Include the commonality in both classes and track changes to ensure consistency.

(d) Merge the two classes together into a single combined class.

1.6 Which of the following statements, concerning UML class diagrams, are true? (5 marks)

(a) A class diagram presents the dynamic view of a software design.

(b) It is possible to implement the relationship shown in the diagram below by having class `A` have `unique_ptr` of type `B` as a data member.



(c) The diagram in (b) above implies navigability from B to A.

(d) UML is an acronym for "Universal Modelling Language".

(e) When UML is used as a sketch you can expect the notation to vary slightly between authors.

1.7 Which of the following statements are true? (5 marks)

(a) International Atomic Energy Agency personnel were sent to the Natanz nuclear enrichment plant in Iran to investigate the Stuxnet virus.

(b) "Zero-day" exploits rely on the fact that the anti-virus software is not up to date on the machine being exploited.

(c) A virus signature is a tell-tale piece of data or behaviour used by anti-virus software to identify a particular virus.

(d) The Stuxnet virus spread from computer to computer via the internet.

1.8 Which of the following statements are true? (5 marks)

(a) Autonomous vehicles are already available for purchase by the general public in the USA.

(b) Autonomous vehicles have already driven millions of kilometres.

(c) Utilitarianism is a principle that advocates maximising individual happiness.

(d) Deontology argues that some values are categorically always true.

(e) Self-driving cars have been programmed to follow a utilitarian approach.

[Total Marks Part A: 40]

## Part B

### Question 2

The canonical form of the assignment operator is:

```
Class& Class::operator=(const Class& rhs)
{
    Class temp(rhs);
    swap(temp);
    return *this;
}
```

(a) What is meant by an *atomic* operation? (2 marks)

(b) Why does this form of the assignment operator guarantee atomic assignment? (4 marks)

(c) What is the purpose of returning `*this`? (2 marks)

(d) Rewrite the canonical form given above so that it works in an identical fashion but has one line less of code. (3 marks)

[Total Marks 11]

### Question 3

Review the code given in Listings 2 and 3 in the appendix and answer the following questions.

(a) The code compiles and runs as given. Predict the output of the main program. (18 marks)

(b) Explain the difference between the keywords `virtual` and `override`. (4 marks)

(c) Assume that line 3 is uncommented in `main`. Will the main program still compile without errors? Explain your answer. (3 marks)

(d) Will the main program in Listing 3 compile without errors if line 5 is changed to following?

```
unique_ptr<Shape> ptr = make_unique<Rectangle>{10, 5};
```

Note, the only change in this line is the curly braces around `10` and `5`. Justify your answer. (3 marks)

[Total Marks 28]

## Question 4

You are developing a computer simulation in which a robot has to navigate a maze. The maze is represented by a square 5×5 grid of cells. Each cell in the grid is either a passage way or a wall. For the first development iteration, passage ways are to be represented by the character 'P' and walls are to be represented by the character 'W'.

The robot is able to move one cell at a time in four possible directions: North, East, South, and West. The robot can move along passages but is blocked by walls. For example, if the robot tries to move North from its current cell and the cell to the North is a passage then the move is successful. However, if the cell to the North is a wall then the move is blocked. Additionally, the robot may not move beyond the boundaries of the maze. A robot controller will be written to place the robot at a location in the maze and control its movements.

You may assume that the following constructor and data members exist for the `Maze` class.

```
class Maze
{
public:
    Maze(const string& text_file_name)
    { // DO NOT WRITE THIS CONSTRUCTOR - ASSUME THAT IT EXISTS
      // the constructor reads the maze stored in the text file
      // into the 2D character array: _maze_map
    }

private:
    const static int WIDTH = 5;
    const static int HEIGHT = 5;
    char _maze_map[WIDTH][HEIGHT];
}
```

**Listing 1:** The Maze class

(a) Complete the `Maze` class, except for the constructor, and provide a `Robot` class as well as any other classes or types which are deemed necessary or useful for allowing the robot controller to work. You are *not* required to write the robot controller.

Ensure that the implementation details of the `Maze` class are hidden, so that if there are implementation changes in future versions (to a matrix of ones and zeros, for example), the `Robot` class will not have to change.

(20 marks)

(b) Provide tests which verify the robot's movement in a North direction only. Clearly explain any assumptions that you make concerning the text file containing the maze map, and give the file contents for the maps used in your tests.

(11 marks)

[Total Marks 31]

[Total Marks Part B: 70]

(Exam Total: Four Questions – 110 : Full Marks – 100 marks)

## Appendix: Source code for the `Shape` hierarchy

```
class Shape
{
public:
   Shape(){ cout << "Shape constructor" << endl; _count++; }
   virtual ~Shape() { _count--; cout << "Shape destructor" << endl; }

   void anyFunction() { cout << "Shape function" << endl; }
   virtual void draw() = 0;
   virtual void area() = 0;
   virtual void perimeter() = 0;
   static int getCount() { return _count; }

private:
   static int _count;
};

int Shape::_count = 0;

/*******************************************************/

class Rectangle : public Shape
{
public:
   Rectangle(int w, int h): _w{w},_h{h} {cout << "Rectangle constructor" << endl;};
   virtual ~Rectangle() { cout << "Rectangle destructor" << endl; };

   void anyFunction() { cout << "Rectangle function" << endl; }
   void draw() override { cout << "Drawing rectangle" << endl; }
   void area() override { cout << "Rectangle area: " << _w * _h << endl; }
   void perimeter() override  { cout << "Rectangle perimeter: " << 2*(_w+_h) <<
       endl; }
   void setWidth(int w) { _w = w; }
   void setHeight(int h) {_h = h; }

protected:
   int _w, _h;
};

/*******************************************************/

class Square : public Rectangle
{
public:
   Square(int s): Rectangle{s, s} { cout << "Square constructor" << endl; }
   virtual ~Square() { cout << "Square destructor" << endl; }

   void draw() override { cout << "Drawing square" << endl; }
   void perimeter() override {   cout << "Square perimeter: " << 4*_w << endl; }
};
```

**Listing 2:** Shape hierarchy

```
int main ()
{
    /* cout << "Number of shapes: " << Shape::getCount() << endl; */

    unique_ptr<Shape> ptr = make_unique<Rectangle>(10, 5);
    ptr->draw();
    ptr->anyFunction();
    cout << "Number of shapes: " << ptr->getCount() << endl;
    ptr.reset(nullptr);
    cout << "Number of shapes: " << Shape::getCount() << endl;

    cout << endl;

    Square a_square{10};
    cout << "Number of shapes: " << Shape::getCount() << endl;
    a_square.anyFunction();
    a_square.perimeter();
    a_square.area();
    return 0;
};
```

**Listing 3:** `main` program using the `Shape` hierarchy