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 2013
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 R Mangisi
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:
  - 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.
  - Classes may be implemented entirely in their header files, unless you are specifically asked to separate the header from the implementation.
- Reference sheets are provided separately.

Part A

Question 1

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

(a) NSA stands for National Surveillance Agency.

(b) The intention of the NSA's PRISM program is to intercept communications from non-US citizens.

(c) Regional internet traffic to the United States from Asia, Africa and Latin America has increased since Edward Snowden's revelations.

(d) The scope of the US government's monitoring includes not only internet communications but also phone calls.

1.2 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 derived from this class.

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

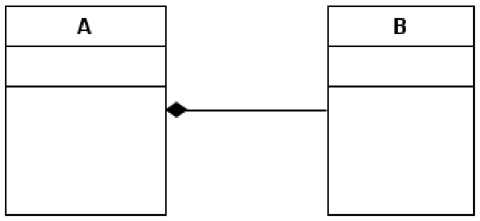
(c) Include the commonality in both classes and track changes to ensure consistency.

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

1.3 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.

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

(a) A hash function is a one way operation — it is impossible to uniquely recover a password from its hash value.

(b) Generating rainbow tables is relatively quick, compared to using them.

(c) Different character sets usually result in rainbow tables of different sizes.

(d) In order to use rainbow tables it is essential to know the hashing algorithm that is in use on the system that you are trying to crack.

(e) Salting a hash implies users adding some unique phrase to their password.

1.5 Examine the following C++ code:

```
void aFunction(const vector<int>& v)
{
    auto a = begin(v);
};

class Person {};
class Student: public Person {};

int main()
{
    auto b = 17.0;

    auto c = static_cast<int> (b);

    string default_value{""};
    auto d = default_value;

    auto e = new Student{};

    return 0;
}
```

Which of the following statements concerning the variables a to e are true? (5 marks)

(a) a is of type `vector<int>::iterator`

(b) b is of type `double`

(c) c is of type `int`

(d) d is of type `string`

(e) e is of type `Person*`

1.6 Which of the following statements are true about a *default* constructor? (5 marks)

(a) It is a constructor with no parameters.

(b) It is constructor with parameters which all have default arguments.

(c) The default constructor of the base class in a class hierarchy is automatically called when constructing one of the derived classes in the hierarchy.

(d) One cannot provide a default constructor for a class which has at least one pure virtual function.

(e) The compiler does not provide a default constructor for a class if the programmer writes their own constructor.

1.7 Examine the following C++ program:

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

int main()
{
    int a = 0;
    for (int i = 2 ; i != 20; ++i)
    {
        cout << ++a << endl;
    }

    cout << endl;

    int b = 0;
    for (int i = 0 ; i <= 17; i++ )
    {
        cout << (b += 1) << endl;
    }

    cout << endl;

    int c = 0;
    for (int i = 0 ; i != 18; i++ )
    {
        cout << c++ << endl;
    }

    cout << endl;

    int d = 0;
    for (int i = 2 ; i != 20; i++)
    {
        cout << (d = d + 1) << endl;
    }

    return 0;
}
```

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

(a) The program will run through each of the `for` loops 18 times.

(b) Only the `for` loops with variables `a`, `d` will produce the same output.

(c) Only the `for` loops with variables `b`, `c` will produce the same output.

(d) Only the `for` loops with variables `a`, `b` and `d` will produce the same output.

(e) Only the `for` loops with variables `b`, `c` and `d` will produce the same output.

1.8 Which of the following statements are <u>true</u>? (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.

[Total Marks Part A: 40]

Part B

Question 2

The following questions are based on a simple graph plotting framework which is very similar to one which was presented in one of the labs. The code for the framework is given in Listings 4 through 8 in the appendix. Listing 9 demonstrates the use of the framework. The framework utilises the SFML library via the `Display` class for doing the actual drawing of the graphs.

Note, only the public interfaces for the `DataPoints` and `Display` classes are given. *You are not required to write out the implementation of either these classes, in any of the questions below — you can assume that these classes have been implemented.*

(a) Draw a sequence diagram showing how a sinusoid graph is plotted when the following line of code is executed in `main` (Listing 9). You need only show the object interactions evident from the line of code below, and from the `Graph::plot` function (Listing 8).

```
graph.plot(sine_points, solid_red);
```

(8 marks)

(b) Why is it inadvisable to use protected data members in a class hierarchy?

(2 marks)

(c) Assume that the `LineStyle` class (Listing 7) is modified so that its protected data members are now private. Give all the code changes that are required in the rest of framework for it to still compile and run. (4 marks)

(d) The Display class (Listing 8) attempts to shield the rest of the framework code from the fact that the underlying graphics library that is being used is SFML. Does it succeed in this regard? Explain your answer. If you feel that it does not succeed than suggest how you would overcome any shortcomings. (5 marks)

(e) Explain why a virtual destructor is provided for both the `Function` (Listing 5) and the `LineStyle` (Listing 7) classes. (2 marks)

(f) *Extend* the given framework by adding code (and *not* modifying any existing code) to allow graphs to be plotted that are the sum of an arbitrary number of functions. Add code to the `main` function (Listing 9) which plots

$$f(x) = x + sin(x)$$

over the range $[0, 6\pi]$.

(9 marks)

[Total Marks 30]

Question 3

The interface of a class which models a *recently-used list* of *strings* is given in Listing 1. A recently-used list allows items to be retrieved in the reverse order of insertion. For example, the last item added (the most recent item) is at the front of the list and is the first item to be retrieved.

A recently-used list may not contain duplicate items. If an attempt is made to add an item to the list which is already contained in the list, that item is simply moved to the front of the list.

```
class RecentlyUsedList
{
public:

    // returns the total number of items in the list
    int count() const;

    // clears the list
    void clear();

    // adds a new item to the list
    void add(const string& item);

    // returns the item indexed, the index starts from 0
    // an out_of_range exception is thrown for an invalid index
    string itemAt(int index) const;
};
```

Listing 1: RecentlyUsedList's public interface

Provide a reasonable number of tests, using the `googletest` framework, to verify that the recently-used list works as expected. The framework's syntax and assertions are given in Listing 2.

```
TEST(TestCaseName, TestName)
{
    EXPECT_TRUE(condition);             // condition is true
    EXPECT_FALSE(condition);            // condition is false
    EXPECT_EQ(expected, actual);        // expected == actual
    EXPECT_FLOAT_EQ(expected, actual);  // expected ≈ actual
    EXPECT_NE(val1, val2);              // val1 != val2
    EXPECT_LT(val1, val2);              // val1 < val2
    EXPECT_LE(val1, val2);              // val1 <= val2
    EXPECT_GT(val1, val2);              // val1 > val2
    EXPECT_GE(val1, val2);              // val1 >= val2
    EXPECT_THROW({code which throws an exception}, TypeOfExpectedException);
    EXPECT_NO_THROW({code which does not throw an exception});
}
```

Listing 2: Googletest framework: syntax and assertions

[Total Marks 15]

Question 4

The Game of Life is a famous "game" developed by the mathematician John Conway. The rules are simple but result in a variety of continuously evolving patterns. Wikipedia explains the game as follows:

> The universe of the Game of Life is a two-dimensional grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:
>
> - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
> - Any live cell with two or three live neighbours lives on to the next generation.
> - Any live cell with more than three live neighbours dies, as if by over-crowding.
> - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
>
> The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed — births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Provide all the source code necessary for modelling a tick in the Game of Life that will produce a new generation. You may assume that the size of the board is 10×10 and is already seeded. Cells which lie along the boundary of the board will have less than eight neighbours.

The skeleton of a class modelling a cell is given in Listing 3. You must use this in your solution but you may add additional functions and data members as required. You may also create additional classes as needed.

```
class Cell
{
public:
    Cell(int x, int y): _x(x), _y(y)
    {}

    int getX() const { return _x; }
    int getY() const { return _y; }

private:
    // (x,y) represents the cell location on the grid
    int _x;
    int _y;
};
```

Listing 3: Modelling a Cell

[Total Marks 25]

[Total Marks Part B: 70]

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

Appendix: Source Code for the Graph Plotter

```
struct Point
{
   float x;
   float y;
};

class PointPair
{
public:
   PointPair(Point p1, Point p2):
     _p1(p1),
     _p2(p2)
     {}

     Point first() const { return _p1; }  // return first point in pair
     Point second() const { return _p2; } // return second point in pair

private:
   Point _p1;
   Point _p2;
};

class Range
{
public:
   Range(float start, float end):
     _start(start),
     _end(end)
     { if (end <= start) throw "Not a range."; }

     float getStart() const { return _start; }     // return start of range
     float getEnd() const { return _end; }         // return end of range
     float size() const { return _end - _start; }  // return size of range

private:
   float _start;
   float _end;
};
```

Listing 4: Point structure, PointPair and Range classes

```
class Function
{
public:
    virtual float evaluate (float x) const = 0;
    virtual ~Function () {};
};
```

```
class Sinusoid: public Function
{
public:
   Sinusoid(float amplitude=1.0, float frequency = 1.0, float phase=0.0):
   _amplitude(amplitude),
   _frequency(frequency),
   _phase(phase)
   {}

   virtual float evaluate(float x) const
   { return _amplitude*sin(_frequency * x + _phase); }

private:
   float _amplitude;
   float _frequency;
   float _phase;
};
```

```
class Polynomial: public Function
{
public:
   Polynomial(const vector<float>& coefficients):
     _coefficients(coefficients)
     {}

    virtual float evaluate(float x) const;

private:
   vector<float> _coefficients;
};
```

```
float Polynomial::evaluate(float x) const
{
   auto result = 0.0;
   for (auto i = 0; i != _coefficients.size(); i++)
   {
      result += _coefficients[i]*pow(x,i);
   }

   return result;
}
```

Listing 5: Function, Sinusoid, and Polynomial classes

```
class DataPoints
{
public:
   DataPoints(vector<Point> datapoints):
     _datapoints(datapoints), _transformed(false)
     {}
     vector<Point> getAsPoints() const { return _datapoints; }
     vector<PointPair> getAsPointPairs() const; // groups adjacent points into pairs
     // transforms data points - required before points can be plotted
     void transformToDisplayCoordinateSystem(int display_width, int display_height);
};
```

```
class Sampler
{
public:
   // performs uniform sampling
   DataPoints generateSamples(const Function& func, const Range& range) const;

private:
   static const int TOTAL_POINTS = 100;
};

// standalone function for generating data points
DataPoints generateDataPoints(const Function& func, const Range& range, const
    Sampler& sampler);
```

```
DataPoints Sampler::generateSamples(const Function& func, const Range& range) const
{
   // using TOTAL_POINTS-1 ensures that x_max is the last point in the range
   auto increment = range.size()/(TOTAL_POINTS-1);
   auto current_x = range.getStart();
   vector<Point> points;

   for (int i = 0 ; i != TOTAL_POINTS ; i++) {
      Point newpoint{current_x, func.evaluate(current_x)};
      points.push_back(newpoint);
      current_x = current_x + increment;
   }

   DataPoints data_points(points);
   return data_points;
}

// standalone function generating data points
DataPoints generateDataPoints(const Function& func, const Range& range, const
    Sampler& sampler)
{
   DataPoints data_points(sampler.generateSamples(func, range));
   return data_points;
}
```

Listing 6: DataPoints' public interface, the Sampler class, and generateDataPoints function

```
class LineStyle
{
public:
   LineStyle(sf::Color colour, shared_ptr<Display> display_ptr);
   virtual void plotLine(PointPair end_points) = 0;

   virtual ~LineStyle() {};

protected:
   sf::Color _colour;
   shared_ptr<Display> _display_ptr;
};
```

```
LineStyle::LineStyle(sf::Color colour, shared_ptr<Display> display_ptr):
   _colour(colour),
   _display_ptr(display_ptr)
{
   if (display_ptr == nullptr) throw "A valid display is required.";
}
```

```
class SolidLineStyle: public LineStyle
{
public:
   SolidLineStyle(sf::Color colour, shared_ptr<Display> display_ptr):
     LineStyle(colour,display_ptr)
     {}
   virtual void plotLine(PointPair end_points)
   { _display_ptr->drawLine(end_points, _colour); }
};
```

Listing 7: LineStyle and SolidLineStyle classes

```
class Display
{
public:
   Display(int display_width, int display_height);
   int getWidth() const { return _display_width; }
   int getHeight() const { return _display_height; }

   void drawLine(PointPair end_points, sf::Color colour);
   void drawDot(Point point, sf::Color colour);

   void clear();   // clears the current display
   void update(); // updates the display by rendering all drawn shapes
   void pause();  // pauses execution of the program so that the display can be
       viewed
};
```

```
class Graph
{
public:
   Graph(shared_ptr<Display> display):
     _display(display)
     {}
   void plot(DataPoints data_points, LineStyle& line_plotter);

private:
   shared_ptr<Display> _display;
};
```

```
void Graph::plot(DataPoints data_points, LineStyle& line_plotter)
{
    _display->clear();

   data_points.transformToDisplayCoordinateSystem(_display->getWidth(),
       _display->getHeight());
   auto point_pairs = data_points.getAsPointPairs();
   for (const auto& point_pair : point_pairs)
   {
      line_plotter.plotLine(point_pair);
   }

   _display->update();
   _display->pause();
}
```

Listing 8: Display’s public interface and Graph class

```
int main()
{
   // setup Graph with Display
   const int WIDTH = 800;
   const int HEIGHT = 600;
   shared_ptr<Display> display(new Display(WIDTH, HEIGHT));
   Graph graph(display);

   // plot sine wave
   Sinusoid sine_function;
   Range range(0, 6*PI);
   Sampler sampler;
   DataPoints sine_points(generateDataPoints(sine_function, range, sampler));
   SolidLineStyle solid_red(sf::Color::Red, display);
   graph.plot(sine_points, solid_red);

   // plot polynomial
   vector<float> coefficients{1,2,1};
   Polynomial poly(coefficients);
   Range range2(-20, 20);
   DataPoints poly_points(generateDataPoints(poly, range2, sampler));
   SolidLineStyle solid_blue(sf::Color::Blue, display);
   graph.plot(poly_points, solid_blue);

   return 0;
}
```

Listing 9: The main function