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 2022
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 A Levien
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, 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.
- Reference sheets are provided in a separate appendix.

Part A

Question 1

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

(a) Tests can be written to indirectly test the code within `private` member functions.

(b) Tests should not contain branching logic because then they cannot be run in any order.

(c) A test that takes a second to run would be considered a fast test.

(d) For every `public` class member function there should be a corresponding test.

(e) AAA stands for Anticipate, Act, Assert.

1.2 When writing code, it is considered good practice to: (5 marks)

(a) Rely on comments to explain how a function is implementing its task.

(b) Name types using nouns, and functions using verbs.

(c) Describe the return value type within the function name so that programmers have a better understanding of the function.

(d) Use abbreviations in variable and function names.

(e) Phrase functions that return a Boolean value as questions.

1.3 Given the following program, which of the statements concerning the program are true? (5 marks)

```
1 int calculate_length(const string& word) { return word.length(); }
2
3 int main()
4 {
5    auto words = vector<string>{"This", "is", "a", "short", "sentence"};
6    auto word_lengths = vector<int>(words.size()); // for results
7    transform(begin(words), end(words), begin(word_lengths),
         calculate_length);
8    return 0;
9 }
```

(a) An in-place transformation can be used instead of storing the results in `word_lengths`.

(b) The `transform` function does not call the `calculate_length` function when the iterator moving over the range is equal to: `end(words)`.

(c) The following line of code is equivalent to that on line 6:
`auto word_lengths = vector<int>{words.size()};`

(d) `calculate_length` is known as a *function object*.

(e) The implementation of the `transform` function does not use `vector<T>::push_back`.

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

(a) Inserting elements at the back of a vector is more efficient than inserting at other positions.

(b) Lists are preferred when random access to elements is required.

(c) `begin(container)` and `end(container)` return iterators pointing to the first and last elements of a container, respectively.

(d) The size of a vector is always less than or equal to its capacity.

(e) If the elements of a vector are very large in size then there is good reason for the capacity of the vector not to increase in an exponential fashion.

1.5 SFML is a gaming library which is available for use in C++. Which of the following statements concerning SFML are true? (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) SFML provides inheritance hierarchies which can be extended (derived from) for your own application.

(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 larger than if it were statically linked to the libraries.

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

(a) A pointer and a reference are different names for the same thing.

(b) Primitive data types cannot be passed by reference.

(c) A reference cannot refer to null.

(d) A pointer may point to another pointer.

(e) A reference may refer to another reference.

1.7 Which of the following statements concerning a layered architecture are true ? (5 marks)

(a) The Presentation Layer is typically constructed from existing open-source or commercial libraries and/or frameworks.

(b) Taking a layered approach is applying the Separation of Concerns design principle.

(c) If the Domain/Logic Layer is dependent on the Data Layer, this implies that the Data Layer has no knowledge of the Domain/Logic Layer.

(d) The Data Access Layer refers to the data store (file system, database, etc) of the application.

(e) The standard Windows calculator application has all three layers present.

1.8 Examine the following C++ code:

```
1  void aFunction(const vector<int>& v)
2  {
3      auto a = begin(v);
4  };
5
6  class Person {};
7  class Student: public Person {};
8
9  int main()
10 {
11     auto b = 17.0;
12
13     auto c = static_cast<int> (b);
14
15     string default_value{""};
16     auto d = default_value;
17
18     auto e = new Student{};
19
20     return 0;
21 }
```

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*`

[Total Marks Part A: 40]

Part B

Question 2

Examine the code for the `Employee`, `Payroll` and `TelephoneGuide` classes in Listing 1 and Listing 2 and answer the following questions:

a) Name two *code smells* that apply to these classes. Note, code smells refer to indicators of poor design, not the use of particular C++ language features. Specifically indicate where the code smells lie, and explain why they are poor design. (6 marks)

b) *Refactor* the code, that is, change the structure but not the external behaviour, to remove the code smells. Refactoring implies that any client code which uses the above classes must continue to work after the changes.

For your answer, you need only write down the code that you modify. (10 marks)

```
1  using TelephoneNumbers = vector<string>;
2
3  class Employee
4  {
5  public:
6     Employee(const string& name, const string& address):
7       _name(name),
8       _address(address)
9       {}
10
11      void addTelephoneNumber(const string& number)
12      { _telephone_numbers.push_back(number); }
13
14      string name() const
15      { return _name; }
16
17      string address() const
18      { return _address; }
19
20      TelephoneNumbers getTelephoneNumbers() const
21      { return _telephone_numbers; }
22
23 private:
24    string _name;
25    string _address;
26    TelephoneNumbers _telephone_numbers;
27 };
```

Listing 1: `Employee` class

```
1  class Payroll
2  {
3  public:
4     void createEmployeeLabel(const Employee& employee)
5     {
6        _label += employee.name() + "\n";
7        _label += employee.address() + "\n";
8        for (auto e : employee.getTelephoneNumbers())
9        {
10          _label += e;
11          _label += " "; // space used as a separator
12       }
13       _label.erase(_label.length()-1, 1);
14    }
15
16    string getLabel() const { return _label; }
17
18 private:
19    string _label;
20 };
21
22 class TelephoneGuide
23 {
24 public:
25    void setCurrentEmployee(Employee* employee)
26    { _employee = employee; }
27
28    string formatTelephoneNumbers() const
29    {
30       string tel_list;
31       tel_list += _employee->name() +=  "\n";
32       tel_list += _employee->address() +=  "\n";
33       auto tel_numbers = _employee->getTelephoneNumbers();
34       auto it = begin(tel_numbers);
35       for (; it != tel_numbers.end(); it++)
36       {
37          // dashes used as a separator
38          if (it != begin(tel_numbers)) tel_list += "--";
39          tel_list += *it;
40       }
41       return tel_list;
42    }
43
44 private:
45    Employee* _employee;
46 };
```

Listing 2: `Payroll` and `TelephoneGuide` classes

[Total Marks 16]

Question 3

The interface of a class which stores and validates a South African identity number is given in Listing 3. The interface for the `Date` class is given in Listing 4.

```
class InvalidIdNumber {};

class IdentityNumber
{
public:
    // validates the ID number, throws InvalidIdNumber
    // if the ID number is not valid
    IdentityNumber(const string& idNumber);

    // returns the date of birth
    Date dateOfBirth() const;

    // returns "male" for male, "female" for female
    string gender() const;

    // returns true if SA citizen, false otherwise
    bool isRSACitizen() const;
};
```

Listing 3: `IdentityNumber`'s public interface

An identity number is of the form (YYMMDD)(G)(SSS)(C)(A)(Z) where:

- YYMMDD is the date of birth of the ID holder.
- (G) is the gender of the ID holder: 0-4 indicates female, and 5-9 indicates male.
- (SSS) is a sequence number for unique date of birth and gender combinations. It can be any number.
- (C) indicates citizenship type: 0 indicates a South African citizen, 1 indicates other citizenship.
- (A) this is the series of the identity number. It can be 8 or 9.
- (Z) is the control digit for the identity number, used as a checksum.
  The control digit is calculated by summing the first 12 digits of the ID number and calculating the remainder after dividing by 3 (ie. the modulus of 3). The calculated control digit and the actual control digit (Z) must be equal for the ID number to be valid. (Note that this is a simplified algorithm for the purposes of this exam - it will not apply to your actual ID number.)

You may assume that the `IdentityNumber` class will only be used to represent ID numbers for people having birthdates in the years ranging from 1900 to 1999.

An example of a valid ID for a South African male born on 2 March 1982 using the rules above would be 8203025071091.

You are required to:

a) Provide a reasonable number of tests, using the doctest framework, to verify that the `IdentityNumber` class works as expected. *Do not write unit tests for the `Date` class.*
(12 marks)

b) Provide all of the source code necessary for implementing the `IdentityNumber` class. You may use the `stoi` function to convert from a string to an integer in your solution. An example usage of `stoi` is shown below in Listing 5.

Do not provide any implementation code for the `Date` class. (12 marks)

```
class InvalidDate {};

class Date
{
public:
    // validates the date, throws InvalidDate
    // if the day, month and year do not represent a valid date
    Date(int day, int month, int year);

    // constructs the date 1/1/1900
    Date();

    // return the day of the month
    int  day() const;

    // return the month of the year
    int month() const;

    // return the year
    int year() const;
};
```

Listing 4: `Date`'s public interface

```
int main()
{
    // Note: stoi throws std::invalid_argument if the string to be converted contains
        non-numeric characters

    auto test = "045"s;
    auto converted_to_integer = stoi(test);
    cout << converted_to_integer; // prints: 45
}
```

Listing 5: Using the `stoi` function

[Total Marks 24]

Question 4

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 6 through 10 in the appendix. Listing 11 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 11). You need only show the object interactions evident from the line of code below, and from the `Graph::plot` function (Listing 10).

```
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 9) 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 10) 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 7) and the `LineStyle` (Listing 9) 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 11) which plots

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

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

(9 marks)

[Total Marks 30]

[Total Marks Part B: 70]

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

Please fill in the question numbers on the front page of your script.

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 6: 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 7: 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 8: 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 9: 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 10: 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 11: The main function