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 2014
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)	Mr B Van Aardt 074 813 6247
External examiner(s)	Mr L.A Machowski
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 about constructors and destructors? (5 marks)

(a) Default constructors can be invoked without supplying arguments.
(b) A class may only have one destructor.
(c) Constructors can only be public.
(d) A class may only have one constructor.

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

(a) A reference cannot refer to null.
(b) A pointer and a reference are different names for the same thing.
(c) Primitive data types cannot be passed by reference.
(d) A pointer may point to another pointer.

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

(a) The terms *free store* and *heap* are equivalent.

(b) There is no need to call `delete` on a smart pointer.

(c) Calling `delete` on a pointer only deletes the pointer, not the data it points to.

(d) It is safe to call `delete` multiple times on a pointer without any side effects.

(e) Smart pointers can guard against memory leaks.

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

(a) The primary purpose of inheritance is to re-use implementation code from a base class.

(b) Classes can derive from multiple parents in C++.

(c) You cannot directly create an object from a class which has a pure virtual member function.

(d) Abstract Base Classes can contain implementation details.

(e) *Overridden* and *overloaded* refer to the same concept.

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

(a) A `constructor` can be declared as `virtual`.

(b) Classes open for extension should have a virtual destructor defined on them.

(c) Destructors must take no arguments and have no return type.

(d) Pure interface classes can contain data members.

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

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

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

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

(d) `begin(container)` and `end(container)` return iterators pointing to the first and one past the last element of a container respectively, creating an asymmetric range.

(e) `begin(container)` and `end(container)` return iterators pointing to the first and last element of a container respectively , creating a symmetric range.

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

(a) Exceptions should not be thrown or caught in a normal run of a program.

(b) Exceptions report errors found at compile time.

(c) Exceptions thrown in a `constructor` will cause the objects `destructor` to be run.

(d) Custom exception classes should be defined to express the logical reason for the error.

(e) Exceptions should be thrown whenever a function or some part of the program can't do what it was asked to.

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

(a) Only commercial software requires a software licence to be specified.

(b) Code released without a software licence attracts an implicit copyright.

(c) A GPL licence means the code may not be used in any proprietary program.

(d) Copyleft licences allow modifications and additions to the software to be released under less permissive conditions than the original software.

[Total Marks Part A: 40]

Part B

Question 2

A magic square is an arrangement of numbers from 1 to n^2 in an [n x n] matrix, with each number occurring exactly once, and such that the sum of the entries of any row, any column, or any main diagonal is the same. One method of generating a magic square in cases where n is **odd** is as follows: Place a 1 in any cell (in the centre cell of the top row, for example), then place subsequent numbers by moving one cell above and to the right. The counting is wrapped around, so that falling off the top returns on the bottom and falling off the right returns on the left. When a cell is encountered which is already filled, the next number is instead placed in the cell below the previous one and the method continues as before.

An example of a magic square is shown in Listing 1.

Provide all the source code necessary for modelling the generation of a magic square.

A partial class diagram is given in Figure 1. 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. Emphasis should be placed in demonstrating your understanding of the design patterns that you learned about during the course.

```
17   24   1    8    15
23   5    7    14   16
4    6    13   20   22
10   12   19   21   3
11   18   25   2    9
```

Listing 1: Magic Square

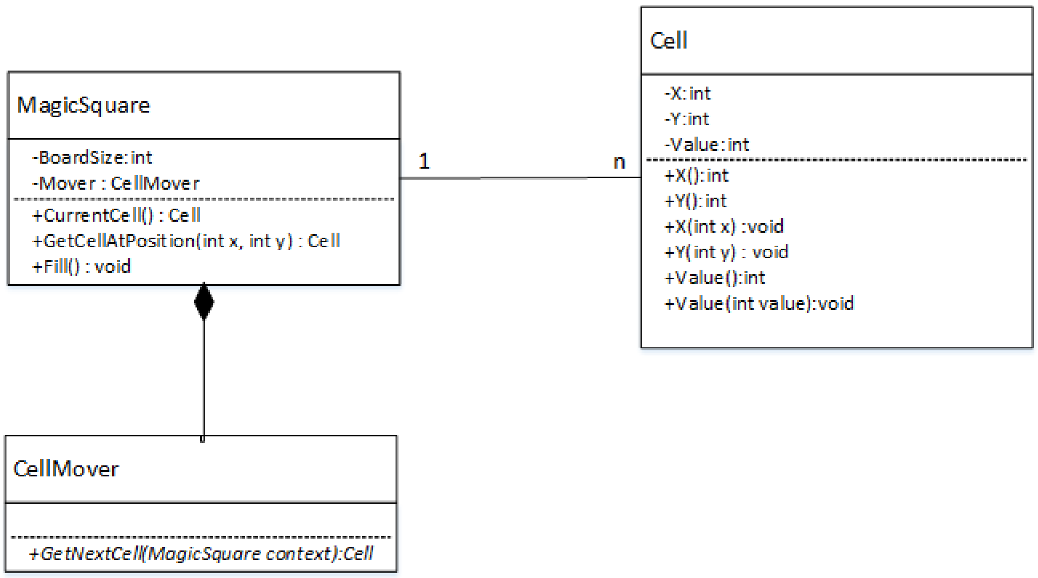

Figure 1: Magic Square Solver Classes

[Total Marks 25]

Question 3

The interface of a class which stores and validates a South African identity number is given in Listing 2.

```
class IdentityNumber
{
public:

   // validates the id number. throws InvalidIdentityNumber exception
   // if the id number is not valid.
   IdentityNumber(const string& idNumber);

   // returns a string representation of the date of birth in YYMMDD format
   Date DateOfBirth() const;

   // returns "male" for male  and "female" for female
   string Gender() const;

   // true if SA citizen, false otherwise.
   bool IsRSACitizen() const;
}
```

Listing 2: 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. **Calculation** 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 won't apply to your actual ID number)

You may assume that all birthdates represented by this ID format are in the range 1900 - 1999. An example of a valid ID for a South African male born on 2 March 1982 using the rules above would be `8203025071091`. The public interface for the `Date` class is given in the appendix as Listing 6.

You are required to:

(a) Provide a reasonable number of tests, using the `googletest` framework, to verify that the identity number works as expected. The framework's syntax and assertions are given in Listing 3. *You do not need to write unit tests for the `Date` class.* (11 marks)

(b) Write the source code for the `IdentityNumber` class. You may use the C++11 `stoi` function to convert from string to integer in your solution. An example usage of `stoi` is shown in Listing 4. Note that `stoi` throws an `invalid_argument` exception if the conversion can not be performed. *You are not required to provide source code for the `Date` class* (11 marks)

(c) Comment on alternatives for the data type of `Gender` on the `IdentityNumber` class. (3 marks)

```
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 3: Googletest framework: syntax and assertions

```
#include <iostream>
#include <string>

using namespace std

int main()
{
    string test = "45";
    int converted_integer = stoi(test);
    cout << converted_integer << '\n';
}
```

Listing 4: C++11 stoi usage

[Total Marks 25]

Question 4

The following questions relate to the modelling and display of a rudimentary grocery list. Three different designs are given in Appendix 2, each being clearly indicated. In all cases, the Canvas class is identical and this class's interface is given below. All three designs produce the same output, that is, there is no difference in the list's appearance.

```
enum Font {Arial, TimesNewRoman, Helvetica};

// Canvas makes use of a particular graphics library/framework
// and system fonts in order to render the list on the screen
// in a window. All the necessary includes to use the library/framework
// will be present in this file.
class Canvas
{
public:
   // pass in the pen thickness to write with as well as the font
   // used to display each list item
   Canvas(float graphics_pen_thickness, Font font);
   void drawListItemWithAdjacentEmptyCheckBox(const string& item);
   void drawListItemWithAdjacentTickedCheckBox(const string& item);
   //...
};
```

Listing 5: Canvas public interface

(a) Draw three sequence diagrams, one for each design alternative, illustrating the object interactions that occur when a populated list is displayed on the screen. (13 marks)

(b) Evaluate each of the design alternatives in terms of their strengths and weaknesses and justify which option you would choose. In performing your evaluation make reference to relevant design principles that have been covered in the course. (7 marks)

[Total Marks 20]

[Total Marks Part B: 70]

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

Appendix 1: Date Class

```
class Date
{
public:

   // validates the date. throws InvalidDate exception if the date is not valid.
   // dateString must be in the form 'YYMMDD' or 'YYYYMMDD'
   Date(const string& dateString);

   // returns the year component of the date.
   int Year() const;

   // returns the month number of the date.
   int Month() const;

   // returns the day component of the date.
   int Day() const;
}
```

Listing 6: Date Class

Appendix 2: Grocery List Modelling and Display - Design 1

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   // ...
   void displayOn(Canvas& canvas);
private:
   List _list;
   List _removed_item_list;
};
```

Listing 7: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}
void GroceryList::remove(const string& item)
 {
      // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
   if (found != _list.end())
   {
         _removed_item_list.push_back(*found);
      _list.erase(found);
      }
}

void GroceryList::displayOn(Canvas& canvas)
{
   for (int i = 0; i != _list.size(); i++)
   {
      canvas.drawListItemWithAdjacentEmptyCheckBox(_list[i]);
   }
   for (int i = 0; i != _removed_item_list.size(); i++)
   {
      canvas.drawListItemWithAdjacentTickedCheckBox(_removed_item_list[i]);
   }
}
```

Listing 8: GroceryList implementation file

```
int main() {
   GroceryList my_groceries;
   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");
   Canvas canvas(12, Arial);
   my_groceries.displayOn(canvas);
}
```

Listing 9: Main program

Appendix 2: Grocery List Modelling and Display - Design 2

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   List itemsOnList() const;
   List itemsRemovedFromList() const;
   // ...
private:
   List _list;
      List _removed_item_list;
};
```

Listing 10: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}

void GroceryList::remove(const string& item)
{
      // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
   if (found != _list.end())
   {
         _removed_item_list.push_back(*found);
         _list.erase(found);
      }
}

List GroceryList::itemsOnList() const
{
   return _list;
}
List GroceryList::itemsRemovedFromList() const
{
   return _removed_item_list;
}
```

Listing 11: GroceryList implementation file

```
class ListDisplayer
{
public:
   ListDisplayer();
   void display(const GroceryList& list);
private:
   Canvas _canvas;
};
```

Listing 12: ListDisplayer header file

```
ListDisplayer::ListDisplayer(): _canvas(12, Arial)
{}

void ListDisplayer::display(const GroceryList& list)
{
   List items_list = list.itemsOnList();
   for (int i = 0; i != items_list.size(); i++)
    {
      _canvas.drawListItemWithAdjacentEmptyCheckBox(items_list[i]);
    }
   List removed_items_list = list.itemsRemovedFromList();
   for (int i = 0; i != removed_items_list.size(); i++)
   {
      _canvas.drawListItemWithAdjacentTickedCheckBox(removed_items_list[i]);
   }
}
```

Listing 13: ListDisplayer implementation file

```
int main() {
   GroceryList my_groceries;
   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");
   ListDisplayer displayer;
   displayer.display(my_groceries);
}
```

Listing 14: Main program

Appendix 2: Grocery List Modelling and Display - Design 3

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   void display(ListDisplayer& displayer) const;
    // ...
private:
   List _list;
      List _removed_item_list;
};
```

Listing 15: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}

void GroceryList::remove(const string& item)
{
   // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
    if (found != _list.end())
   {
            _removed_item_list.push_back(*found);
            _list.erase(found);
      }
}

void GroceryList::display(ListDisplayer& displayer) const
{
   displayer.displayMe(_list, _removed_item_list);
}
```

Listing 16: GroceryList implementation file

```
class ListDisplayer
{
public:
   ListDisplayer();
   void displayMe(const List& list, const List& removed_item_list);
private:
   Canvas _canvas;
};
```

Listing 17: ListDisplayer header file

```
ListDisplayer::ListDisplayer(): _canvas(12, Arial)
{}
void ListDisplayer::displayMe(const List& list, const List& removed_item_list)
{
   for (int i = 0; i != list.size(); i++) {
   _canvas.drawListItemWithAdjacentEmptyCheckBox(list[i]);
}
   for (int i = 0; i != removed_item_list.size(); i++)
   {
   _canvas.drawListItemWithAdjacentTickedCheckBox(removed_item_list[i]);
   }
}
```

Listing 18: ListDisplayer implementation file

```
int main() {
   GroceryList my_groceries;
   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");
   ListDisplayer displayer;
   my_groceries.display(displayer);
}
```

Listing 19: Main program