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 2012
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 P Mendes
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.
- Reference sheets are provided separately.

Part A

Question 1

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

(a) "Magic numbers" are also known as "literals".

(b) The following line of code will produce an output of "6": `cout « 12.5/2;`

(c) `string::size_type` represents a kind of magic number.

(d) The output of the following program is the maximum value that the `int` type can hold.

```
#include <iostream>
#include <climits>

int main()
{
   int x = INT_MIN;
    std::cout << --x;
    return 0;
}
```

1.2 Consider the following extract from Wikipedia:

"In computer programming, the adapter pattern (often referred to as the wrapper pattern or simply a wrapper) is a design pattern that translates one interface for a class into a compatible interface. An adapter allows classes to work together that normally could not because of incompatible interfaces, by providing its interface to clients while using the original interface. The adapter translates calls to its interface into calls to the original interface (the adaptee), and the amount of code necessary to do this is typically small."

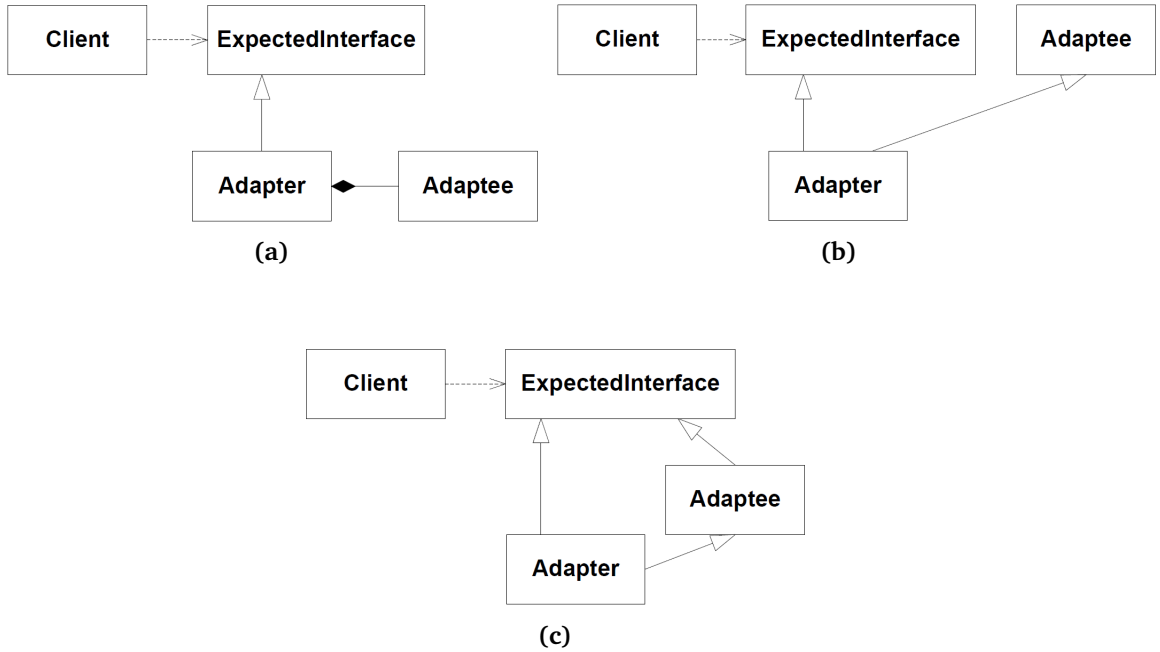
There are two types of adapter patterns:

Object Adapter Pattern

In this type of adapter pattern, the adapter contains an instance of the class it wraps. In this situation, the adapter makes calls to the instance of the wrapped object.

Class Adapter pattern

This type of adapter uses multiple polymorphic interfaces to achieve its goal. The adapter is created by implementing or inheriting both the interface that is expected and the interface that is pre-existing. It is typical for the expected interface to be created as a pure interface class, especially in languages such as Java that do not support multiple inheritance."



Which of the following are true? (5 marks)

(a) The adapter design pattern is a special case of the proxy design pattern.

(b) Two out of the three UML diagrams correctly illustrate the Adapter Pattern.

(c) The UML diagram shown in Figure (b) illustrates the *Class* Adapter Pattern.

(d) The UML diagram shown in Figure (c) illustrates the *Class* Adapter Pattern.

(e) A "pure interface class" is also known as an "abstract base class".

1.3 Examine the following C++ program:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a = 10;
7      int& b = a;
8
9      cout << &b << endl;
10     a++;
11     cout << &a << endl;
12     b = b - 1;
13     cout << b << endl;
14     cout << a << endl;
15     cout << a++ << endl;
16
17     return 0;
18 }
```

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

(a) Line 9 will output the memory address of variable a

(b) Lines 9 and 11 do not produce the same output

(c) The output of line 13 is 9

(d) The output of line 14 is 10

(e) The output of line 15 is 11

1.4 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.

1.5 While working on a software project, you go online and find source code which you are able to use without modification. However, no mention is made of the license under which the code has been published.

What would be correct to assume in this situation? (5 marks)

(a) The code is on the web, and thus in the public domain. You can use it as you want.

(b) It is copyrighted and should be treated as such.

(c) It is open source and you can use it as long as you reference where you got it from.

(d) It is proprietary and you have no legal right to use it without permission.

(e) It has been released under the WTFPL and you do not need to worry about the license.

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

(a) A pointer to a reference, and reference to a pointer, are both valid.

(b) A pointer, to a pointer, to a pointer to an integer is valid.

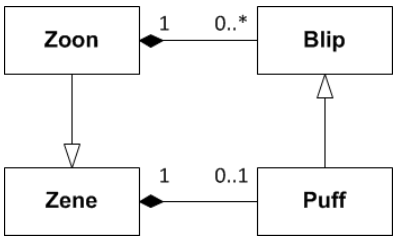
(c) A reference is a constant pointer.

(d) Code containing a vector of references to some type will compile.

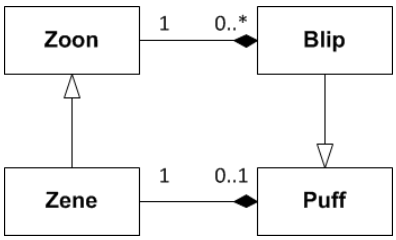
(e) It is possible to create a reference to a reference.

1.7 All Zoons are Zenes. Zenes may or may not have a Puff. A Puff is a Blip, and a Zoon may have any number of Blips. Review the UML diagrams below and identify which of the following statements are correct.

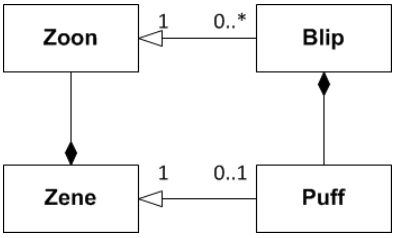
(5 marks)



(d)



(e)



(f)

(a) The UML diagram labelled (d) correctly depicts the given scenario.

(b) The UML diagram labelled (e) correctly depicts the given scenario.

(c) The UML diagram labelled (f) correctly depicts the given scenario.

(d) Aggregation is a form of composition that implies ownership.

(e) The relationship between Zoon and Zene in diagram (f) implies navigability from Zoon to Zene.

1.8 Examine the following C++ program:

```
class Person
{
public:
   Person(const string& name, unsigned int age):
     _name(name),
     _age(age)
     {}

   string name() const { return _name; }
   void name(const string& newname) { _name = newname; }
   unsigned int age() const { return _age; }
   void age(unsigned int new_age) { _age = new_age; }

private:
   string _name;
   unsigned int _age;
};
```

What would be correct to assume given the above definition of the `Person` class?

(5 marks)

(a) `Person`'s compiler-generated copy constructor performs what is termed "default memberwise initialisation".

(b) There is no possibility that an exception will be thrown when a `Person` object is copied.

(c) No state will be shared between two `Person` objects if one is assigned to the other.

(d) A `Person` can be held directly as a data member of another class which requires atomic assignment.

(e) A destructor is automatically provided by the compiler for the `Person` class.

[Total Marks Part A: 40]

Part B

Question 2

(a) The classes and main program given in Listings 1 and 2 compile and run. Give the output of the main program.

```
class Sport {
public:
   Sport(string name): name(name)
     { _number_of_sports++; }

   string official() { return "Referee"; }

   virtual void print() { cout << name << " " << official() << endl; }

   static int _number_of_sports;

protected:
   string name;
};

int Sport::_number_of_sports = 0;

class Rugby : public Sport {
public:
   Rugby(): Sport("Rugby"), _number_of_players(15)
    {}

   virtual void print() { cout <<  name << " " << _number_of_players << endl;
       }

   int  _number_of_players;
};

class Sevens : public Rugby {
public:
   Sevens()
     { _number_of_players = 7; }
};

class Hockey : public Sport {
public:
   Hockey(): Sport("Hockey"), _number_of_players(11)
    {}

   string official() { return "Umpire"; }

   int _number_of_players;
};
```

Listing 1: Sports classes

```
int main()
{
   vector<shared_ptr<Sport> > sports;
   sports.push_back(shared_ptr<Sport>(new Hockey()));
   sports.push_back(shared_ptr<Sport>(new Sport("Croquet")));
   sports.push_back(shared_ptr<Sport>(new Sport("Running")));

   scoped_ptr<Rugby> a(new Rugby());
   scoped_ptr<Sevens> b(new Sevens());
   scoped_ptr<Hockey> c(new Hockey());
   shared_ptr<Sport> d(sports[0]);

   cout << "1. " << sports[0]->official() << endl;
   cout << "2. " << c->official() << endl;
   cout << "3. " << d->official() << endl;
   for(int i=0; i != 3; i++) sports[i]->print();
   cout << "a. ";  a->print();
   cout << "b. ";  b->print();
   cout << Sport::_number_of_sports << endl;
}
```

Listing 2: Main program

(14 marks)

(b) Why can the use of `protected` data members be problematic? (3 marks)

(c) In computer programming the “queue" is a very important data structure. A queue can be thought of as being similar to a line of people waiting to pay in a supermarket. A queue has a front and a back. Data enters the queue at the back and leaves at the front.

Create a class which models a queue of integers and give the complete implementation. Ensure that you provide methods for adding items to the queue and for removing items from the queue. There should also be a method which determines if the queue is empty or not.

You may assume that objects of this class will only store a small number of integers at any one time so performance in terms of speed is not a critical issue.

(8 marks)

[Total Marks 25]

Question 3

The following questions relate to the modelling and display of a rudimentary grocery list. *Three* different designs are given in Appendix 1, 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 3: `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]

Question 4

Figure 1 depicts a screen shot from a game called "Dodge".

Dodge consists of:

- A single player who can only move left and right and is controlled via the keyboard.
- Raindrops which appear in random positions in the top quarter of the screen and fall vertically downwards. The number of raindrops on the screen at any one time remains constant — whenever a raindrop disappears off the bottom of the screen, a new one appears at the top of the screen.

The objective of the game is to move the player so that he or she does not get wet, that is, the player must dodge the raindrops. If a raindrop touches the player then the game is over.

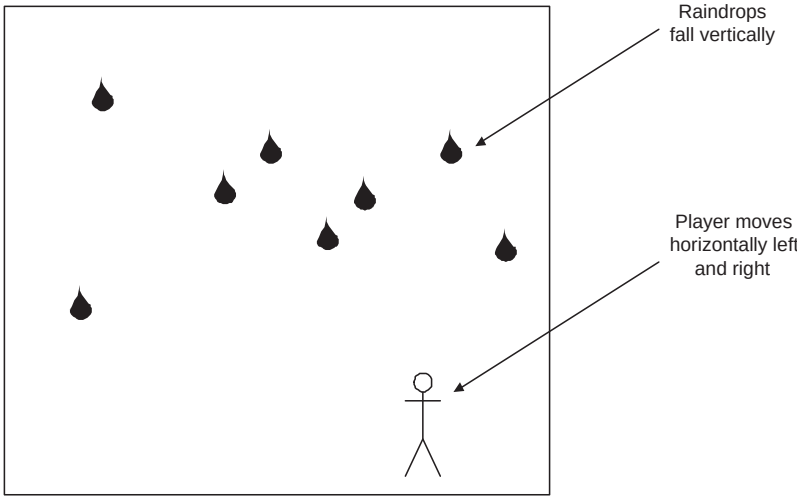


Figure 1: Dodge screenshot

Provide the complete source code for the Dodge game. Your solution must be object-oriented. Certain classes *have already been implemented* and you should use these, where appropriate, in your solution. The interfaces for these classes are given in Listings 17 and 18 in Appendix 2.

You may use *very simple* graphics for the game elements and you can assume that the player will not move beyond the borders of the screen. You are not required to separate the display logic from the game logic.

[Total Marks 25]

[Total Marks Part B: 70]

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

Appendix 1: 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 4: 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 5: 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 6: Main program

Appendix 1: 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 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);
   }
}

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

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

Listing 8: GroceryList implementation file

```
class ListDisplayer
{
public:
   ListDisplayer();
   void display(const GroceryList& list);

private:
   Canvas _canvas;
};
```

Listing 9: 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 10: 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 11: Main program

Appendix 1: 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 12: 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 13: GroceryList implementation file

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

private:
   Canvas _canvas;
};
```

Listing 14: 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 15: 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 16: Main program

Appendix 2: Existing Dodge Game Classes

```
// This structure is used in the Screen class interface (see drawRectangle)
struct Rectangle
{
   unsigned int x_pos;    // top-left x-coordinate
   unsigned int y_pos;   // top-left y-coordinate
   unsigned int width;   // rectangle width
   unsigned int height;  // rectangle height
};

class Screen
{
public:
   // Screen constructor
   // A "ScreenInitialisationError" exception is thrown
   // if the Screen cannot be initialised
   // For the default screen, the top-left co-ordinate is (0,0) and
   // the bottom-right co-ordinate is (640,480)
   Screen(int width = 640, int height = 480);

   // Screen destructor, frees up all of Screen's resources
   ~Screen();

   // Returns the Screen width
   int width() const;

   // Returns the Screen height
   int height() const;

   // Updates the Screen with all the changes that have occurred since
   // the previous display call
   void display();

   // Clears the Screen, by drawing over it in black
   void clear();

   // Sets the foreground colour (the background colour is black)
   // R represents the red component (0 = minimum, 255 = maximum)
   // G represent the green component (0 = minimum, 255 = maximum)
   // B represents the blue component (0 = minimum, 255 = maximum)
   void setDrawingColour(int R, int G, int B);

   // Draws a rectangle on the screen in the foreground colour
   void drawRectangle(Rectangle rect);

   // Draws a circle on the screen in the foreground colour
   void drawCircle(unsigned int centre_x, unsigned int centre_y,
               unsigned int radius);
};
```

Listing 17: `Screen` class

```
// The KeyBoardInputHandler class recognises the following keys
enum Key {None, Space, Escape, LeftArrow, RightArrow, UpArrow, DownArrow};

class KeyBoardInputHandler
{
public:
   // Returns the most recent key that was pressed
   // Only the keys specified in the enumeration type above are recognised
   // If no key has been pressed "None" is returned
   Key getKeyPressed() const;
};
```

Listing 18: `KeyboardInputHandler` class