READ THE FOLLOWING SCENARIO CAREFULLY BEFORE ANSWERING THE RELEVANT QUESTION.

The Golden Pangolin Competition

Game Lodges in Southern Africa have collaborated to organise a competition to determine which lodges offer the best facilities. They will rate the lodges according to the game that they offer. To do this a competition was started in 2006 called the ***The Golden Pangolin*** (because the winning lodge gets a trophy in the shape of a Pangolin made of solid gold).

In 2006 the competition only counted animals and awarded points based on whether the animal was one of the big five (elephant, leopard, lion, buffalo, rhino). This year the competition has been extended to include birds. The points for birds are based on the distance sighted, whether the bird is one of the big six (Lappetfaced Vulture, Martial Eagle, Saddlebilled Stork, Kori Bustard, Ground Hornbill, Pel's Fishing Owl), and if the bird was sighted in flight. The competition organisers decided to define a different set of birds to comprise the big six based on how often they were sighted in the area.

You have been approached to write the software that will be used to analyse the results and determine the winner.

Each guide logs important and rare sightings and these are sent to ***SuperFast Computing Corp*** where they are consolidated into a database table. You have been provided with a database file called **GoldenPangolin.mdb** containing two tables called **Lodges** and **Sightings**.

The **Lodges** table has the following fields:

- **LodgeNo** *the number of the lodge* *Type: Number*
- **LodgeName** *the name of the lodge* *Type: Text*

A sample of the table called **Lodges**[1] in the **GoldenPangolin** database is given in Figure 1.

LodgeNo	LodgeName
1	Treetop Lodge
2	Dinonyane
3	Babanango Lodge
4	St Lucia Lodge
5	Madibane Lodge
6	Madikwe Lodge
7	Hillside Haven
8	Sangbonani Lodge

Figure 1

[1] The names of the Lodges have been shortened for the purpose of this examination.

The **Sightings** table has the following fields:

- **SightingNo** *the number of the sighting*
- **LodgeNo** *the number of the lodge*
- **Species** *a character indicating if the sighting was a bird ('b') or a mammal ('m')*
- **SpeciesDesc** *a description of the animal*
- **ValCode** *a code to ensure the sighting was valid*
- **Big6** *a Boolean true or false indicating if the bird was one of the big 6 – note that this field only applies to birds*
- **Flight** *a Boolean true or false indicating if the bird was in flight when it was sighted – note that this field only applies to birds*
- **Distance** *a number indicating the distance from the observer to the species – note that this field only applies to birds*

A sample of the table called **Sightings** is given in Figure 2.

SightingNo	LodgeNo	Species	SpeciesDesc	ValCode	Big6*	Flight*	Distance
1	5	m	Sable	WYPJZTTP78			
2	13	b	Ground Hornbill	JWGXGTWU23	False	True	10
3	12	m	Pangolin	HBVLSDFJ91			
4	19	m	Banded Mongoose	NRTLNELK78			
5	9	b	Redcrested Korhaan	HQHHUQRF71	True	True	29
6	18	b	European Roller	KJUFSDET80	True	False	7
7	13	m	Aardwolf	ISQJDKYL67			
8	13	m	Steenbok	ZGEXXRKV59			
9	16	m	Roan	IIZSSLTH90			
10	13	m	Impala	RADDMYEG67			
11	9	b	Blue Waxbill	XUBAFOUD76	True	True	22
12	4	b	Bateleur	MHWQXRDN71	False	True	8
13	1	m	Spotted Hyena	DRUZOGKE59			
14	10	b	Namaqua Dove	RGKLXKKJ88	False	True	28

Figure 2

Note that not all the lodges will have sightings recorded and some lodges may have more than one recorded sighting.

* Your database will display true as a ticked checkbox, and false as an unchecked checkbox. Your database may interpret no value for the **Big6** and **Flight** fields as false.

QUESTION 2 DELPHI

This section is to be answered by candidates using DELPHI.

1. **Database Connectivity**

1.1 Load the **DB** class that has been provided for you. In this class, code a method named **update()** which will allow for the insertion, editing and deletion of records in the tables. This method will be sent a SQL statement as a String parameter. (5)

1.2 Code a method named **query()** in the **DB** class which will execute any SQL query sent as a String parameter. (5)

2. **Application Program**

2.1 Code an application named **GPApp** and add the menu options shown below using a main menu component named **mnuFile** to display the menu.

A List all Lodges
B List all Lodges and Sightings
C Add a Lodge
D Delete a Sighting
E View Sightings
F Show Lodge Scores (4)

2.2 Provide the code needed to instantiate a **DB** object at the beginning of the **GPApp** application. (2)

All of the solutions in questions A – D are dependent on providing the appropriate SQL statements in your code.

A **List All Lodges**
Provide the code in the **GPApp** application needed to display all of the names of lodges (**LodgeName**) in the **Lodges** table, sorted alphabetically according to the name of the lodge (**LodgeName**). (9)

B **List All Lodges and Sightings**
Provide the code in the **GPApp** application needed to display all the names of lodges (LodgeName) in the **Lodges** table together with the total number of the sightings in the **Sightings** table for each lodge. (8)

C **Add a Lodge**
Provide the code in the **GPApp** application needed to add a new record to the **Lodges** table. To do this you need to request the user to input the name of the lodge (**LodgeName**). The **LodgeNo** field is set to **Autonumber**, so your SQL statement must be written so that **LodgeNo** is given the next available number by the database. DO NOT try to insert a value for it. (5)

D Delete a Sighting

Provide the code in the **GPApp** necessary to delete a record from the **Sightings** table given the **SightingNo**. To do this you need to request the user to input the number of the sighting (**SightingNo**). Search for the Sighting number using the method you created in Question 1.2 and either display a messages if the record is not found, or display the fields of the record. The user must confirm that the record is to be deleted before the record is actually deleted. You may assume that there are no two sightings with the same number. (6)

E View Sightings

An array of Sightings will be instantiated to determine the following results. To do this you will first need to code the **MammalSighting** and **BirdSighting** classes to provide the fields and methods needed to calculate scores.

E.1 MammalSighting Class

E.1.1 Create a new class called **MammalSighting** with instance variables (fields) to store the sighting number, the lodge number, the species, the species description and the validation code. Declare these variables so that they can be accessed by classes that inherit from this class. Use suitable types for each of these variables. (4)

E.1.2 Code a parameterised constructor method to assign values to these variables. (3)

E.1.3 Code a **toString()** method to return the fields of the class separated by tabs. (4)

E.1.4 Code a typed method called **calcScore** that returns an integer. This method must be able to be overridden in descendant classes. If the animal is one of the big 5 then the method should return 20 points otherwise it should return 10 points. Marks will be awarded for efficient and elegant code. (5)

E.1.5 Code an accessor method for the **LodgeNo** field. (2)

E.2 BirdSighting Class

E.2.1 Code a subclass (descendant) of the **MammalSighting** class called **BirdSighting** with instance variables (fields) to store the **Distance**, **Big6** and **Flight** values. Declare these variables so that they can be accessed only in this class. (4)

E.2.2 Create a parameterised constructor method which accepts parameters for ***ALL*** the fields (including those used for the **MammalSighting** class). Use the inherited constructor to assign values to the fields in the **MammalSighting** class and assign the remaining parameters to the appropriate variables. (5)

E.2.3 Create a **toString()** method to combine the fields of this class, separated by tabs, with those of the superclass by overriding the toString() method in the superclass. (5)

E.2.4 Code a method called **calcScore** that overrides the method in the **MammalSighting** class. This calculates the score as follows:

Each sighting gets a basic score of 15 points and:

- if **Distance** < 10 then add 20 to the score
- if **Flight** is true then add 5 to the score
- if **Big6** is true add 15 to the score

Return the calculated score as an integer. (4)

E.3 SightingArray Class

E.3.1 Create a class called **SightingArray** with private instance variables (fields) to store an array of up to 100 sightings and an integer field to track how many items are in the array. The base type for this array is the **MammalSighting** class. The array will hold both **MammalSighting** and **BirdSighting** objects. (4)

E.3.2 Code an appropriate method named **ValidSighting** that accepts a **ValCode** parameter and returns the Boolean value, true, if the code is valid and returns the Boolean value, false, if it is not.

The sighting validation code is validated by first adding up the ASCII (UNICODE) values of the even letters and the ASCII (UNICODE) values of the odd letters (excluding the digits at the end of the code). The difference of the two totals minus 64 (**EvenTotal - OddTotal - 64**) is then calculated.

The absolute value* of this result must be equal to the number represented by the digits at the end of the validation code.

For example the validation code: **WYPJZTTP78**

W	Y	P	J	Z	T	T	P
1	2	3	4	5	6	7	8

The ASCII values are given in brackets after each letter.

Odd letters W (87), P (80), Z (90), T (84) giving a total of 341
Even letters Y (89), J (74), T (84), P (80) giving a total of 327

327 - 341 - 64 = -78

The absolute value (i.e. converting -78 to a positive number) gives 78 which is the number at the end of the validation code.

Invoke (call) this method from the Constructor method and only instantiate an array element if the code is successfully validated. (14)

*Absolute value ignores the sign. The absolute value of +78 is 78. The absolute value of -78 is also 78.

E.3.3 Code a Constructor method that populates the array with sightings information from the **Sightings** table in the **GoldenPangolin** database.

- Check if the sighting is valid using the **ValidSighting** method described in E.3.2 above. If the sighting is valid then:
- Check if the sighting is a Bird or a Mammal and then instantiate the appropriate type of object into the array.
- Remember to increase the counter as you go. (10)

E.3.4 Code a **toString()** method in the **SightingArray** class to return all the elements in the array each on a new line. Invoke (call) this method in the **GPApp** class to display the array on the screen. (4)

F **Show Lodge Scores**

Code a method in the **SightingArray** class to determine the total number of points for each lodge. Display all lodge names and total points for each lodge. (8)

120 marks

Total: 120 marks