# INFORMATION TECHNOLOGY P1

# EXEMPLAR 2018

**NATIONAL**

# SENIOR CERTIFICATE



# GRADE 12

**MARKS: 150**

# TIME: 3 hours

**This question paper consists of 19 pages.**

|  |  |  |
| --- | --- | --- |
| **INSTRUCTIONS AND INFORMATION** |  |  |

|  |  |
| --- | --- |
| 1. | This paper is divided into FOUR sections. Candidates must answer ALL the questions from all FOUR sections. |
|  |  |
| 2. | The duration of this examination is three hours. Because of the nature of this examination it is important to note that you will not be permitted to leave the examination room before the end of the examination session. |
|  |  |
| 3. | This question paper is set with programming terms that are specific to Delphi programming language. The Delphi programming language must be used to answer the questions. |
|  |  |
| 4. | Make sure that you answer the questions according to the specifications that are given in each question. Marks will be awarded according to the set requirements. |
|  |  |
| 5. | Answer only what is asked in each question. For example, if the question does not ask for data validation, then no marks will be awarded for data validation. |
|  |  |
| 6. | Your programs must be coded in such a way that they will work with any data and not just the sample data supplied or any data extracts that appear in the question paper. |
|  |  |
| 7. | Routines, such as search, sort and selection, must be developed from first principles. You may NOT use the built-in features of the Delphi programming language for any of these routines. |
|  |  |
| 8. | All data structures must be defined by you, the programmer, unless the data structures are supplied. |
|  |  |
| 9. | You must save your work regularly on the disk/CD/DVD/flash disk you have been given, or on the disk space allocated to you for this examination session. |
|  |  |
| 10. | Make sure that your examination number appears as a comment in every program that you code, as well as on every event indicated. |
|  |  |
| 11. | If required, print the programming code of all the programs/classes that you completed. Your examination number must appear on all the printouts. You will be given half an hour printing time after the examination session. |
|  |  |
| 12. | At the end of this examination session you must hand in a disk/CD/DVD/ flash disk with all your work saved on it OR you must make sure that all your work has been saved on the disk space allocated to you for this examination session. Ensure that all files can be read. |

|  |  |
| --- | --- |
| 13. | The files that you need to complete this question paper have been provided to you on the disk/CD/DVD/flash disk or on the disk space allocated to you. The files are provided in the form of password-protected executable files.  Do the following:   * Double click on the following password-protected executable file:   **DataENGExem2018.exe**.   * Click on the extract button. * Enter the following password: **Exemplar2018**   Once extracted, the following list of files will be available in the folder  **DataENGExem2018**: |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Question1:** | **Question3:** |  |  |
|  | Question1\_P.dpr  Question1\_P.dproj  Question1\_P.res  Question1\_U.dfm  Question1\_U.pas | Player\_U.pas  Players.txt  Question3\_P.dpr  Question3\_P.dproj  Question3\_P.res  Question3\_U.dfm  Question3\_U.pas |  |  |
|  |  |  |  |  |
|  | **Question2:** | **Question4:** |  |  |
|  | dbConnection\_U.dcu  LanFans.mdb  LanFansBackup.mdb  Question2\_P.dpr  Question2\_P.dproj  Question2\_P.res  Question2\_U.dfm  Question2\_U.pas | Question4\_P.dpr  Question4\_P.dproj  Question4\_P.res  Question4\_U.dfm  Question4\_U.pas  Scores.txt |  |  |

|  |
| --- |
| **SCENARIO**  LAN FANatics is a group of young IT entrepreneurs who created a business that hosts LAN gaming parties at different venues in their home town. |

|  |  |  |
| --- | --- | --- |
| **SECTION A** |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **QUESTION 1: GENERAL PROGRAMMING SKILLS** |  |  |
|  |  |  |
| The LAN FANatics group is in the process of creating software that can be used in the registration and planning of the LAN party events that they host. |  |  |

|  |  |  |
| --- | --- | --- |
| Do the following: |  |  |
| * Compile and execute the program found in the **Question1** folder. The user interface displays FOUR different sections named Question 1.1 to Question 1.4.  The program has no functionality currently.   Example of graphical user interface (GUI): |  |  |
| * Complete the code for each section of QUESTION 1, as described in QUESTION 1.1 to QUESTION 1.4. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.1 | **Button [Register Player]**  Write code to use the information entered in the **Player Name** and **E-mail** edit boxes to compile a registration message. A message that indicates that the registration was successful must only be displayed if information has been entered in both the edit boxes, otherwise an error message must be displayed. The message must be compiled in the following format:  <Player Name> with e-mail address <e-mail address> has been registered. |  |  |
|  |  |  |  |
|  | Example of output for Peter Joubert with the e-mail address pete@gmail.com: |  |  |
|  |  |  |  |
|  | Example of output if NO information was entered in any one of the edit boxes: |  |  |
|  |  |  |  |
|  |  |  | (6) |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.2 | **Button [Calculate Cost of Meals]**  The LAN party organisers provide meals to the players. The meals have to be ordered from the following menu:   * Toasted sandwich – R25.00 * Burger and chips – R32.80 * Medium-size pizza – R45.75   The user must select one of the meals from the combo box and the number to be ordered from the spin edit box. The user must tick the checkbox if the vegetarian version of the selected meal is preferred.  Write code to calculate the amount to be paid based on the selected information. Decrease the amount by 10% if the vegetarian option has been selected.  Display the calculated amount in the edit box provided, formatted to currency with TWO decimal places and in blue text. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Example of output if four vegetarian toasted sandwiches were ordered: |  |  |
|  |  |  | (14) |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.3 | **Button [Calculate Tables and Power]**  The number of tables required for the number of players who will take part must be calculated. Each table at the venue can accommodate six players. |  |  |
|  |  |  |  |
|  | A constant variable **Tablesize** is provided. Write code to calculate the minimum number of tables that will be required to accommodate the number of players that was entered in the relevant edit box.  Also write code to calculate the total amount of power that will be required by the computers used by the players, rounded to the nearest kilowatt (kW). Assume that the average amount of power required by a computer is  0,66 kW. Also assume that each player will use one computer during the event.  Display the calculated values in the edit boxes provided. |  |  |
|  |  |  |  |
|  | Example of output if the number of players that has been entered is 86: |  |  |
|  |  |  | (8) |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.4 | **Button [Calculate Days]**  The organisers must buy units of electricity before hosting a LAN party. The user must enter the total number of units of electricity required to host the event in the provided edit box.  Write code to randomly generate a value in the range from 50 to 149, which represents the estimated daily usage of electricity during the event, and display the value in the relevant edit box.  Write code to calculate the number of units left after each day's game play using the estimated usage figure. Display the information as shown in the example below. The memo box must be cleared before the output is displayed. |  |  |
|  |  |  |  |
|  | Example of output if a value of 500 was entered for the number of units the organisers plan to buy and the random value of 89 has been generated as the estimated daily usage figure: |  |  |
|  |  |  | (12) |
|  |  |  |  |
|  | **NOTE:** The output will depend on the units entered and the randomly generated daily usage figure. |  |  |

|  |  |  |
| --- | --- | --- |
| * Enter your examination number as a comment in the first line of the program file. * Save your program. * Print the code if required. |  |  |

|  |  |  |
| --- | --- | --- |
| **TOTAL SECTION A:** |  | **40** |

|  |  |  |
| --- | --- | --- |
| **SECTION B** |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **QUESTION 2: SQL AND DATABASE** |  |  |

|  |
| --- |
| The LAN FANatics group has developed a database called **LanFans.mdb**, which contains information about players and the games they have played. |

|  |  |  |
| --- | --- | --- |
| The **LanFans.mdb** database contains two tables called **tblPlayers** and **tblGames**. |  |  |
|  |  |  |
| The records contained in the **tblPlayers** table consist of the following fields:   * PlayerID – a unique five-character identification code (text data type) which is the primary key (PK) * Name – name of player (text data type) * Email – e-mail address of player (text data type) * DateOfBirth – date of birth of player (date/time data type) |  |  |
|  |  |  |
| Example data from the **tblPlayers** table: |  |  |
|  |  |  |
|  |  |  |
| The records contained in the **tblGames** table consist of the following fields:   * GameIndx – unique index code for each game (number data type) starting from the value of 1(PK) * GameDate – date on which game was played (date/time data type) * PlayerID – code of player who played the game (text data type) * Duration – number of minutes the game lasted (integer data type) * Score – score obtained by the player who played the game (integer data type) |  |  |
|  |  |  |
| Example data from the **tblGames** table: |  |  |
|  |  |  |
|  |  |  |
| The **PlayerID** field has been used to link the two tables. |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **NOTE:**   * The date format might be displayed differently on your computer due to the regional settings on your computer. * Connection code has been provided. * The database is password-protected. Therefore you will not be able to access the database directly. |  |  |

|  |  |  |
| --- | --- | --- |
| Do the following: |  |  |
|  |  |  |
| * Compile and execute the program in the **Question2** folder. The program limited functionality. |  |  |
| * Complete the code for each question as described in QUESTION 2.1 and QUESTION 2.2. |  |  |

|  |  |
| --- | --- |
| The program provides a GUI with two tab sheets: **Qst2.1 SQL** and **Qst2.2 DB**. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1 | **SQL** |  |  |

|  |  |
| --- | --- |
| Select tab sheet **Qst2.1 SQL**, which displays the following GUI: |  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **NOTE:** When the **Restore Database** button is clicked, the data in the database tables will be restored to the original data. |  |

|  |  |
| --- | --- |
| Complete the SQL code to meet the requirements specified in QUESTION 2.1.1 to QUESTION 2.1.5. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1.1 | **Button [Show All Players]** |  |  |
|  |  |  |  |
|  | All the fields must be displayed for all the players in the **tblPlayers** table. The records must be displayed in ascending order according to the **Name** field. |  |  |
|  |  |  |  |
|  | Example of output of the first four records: |  |  |
|  |  |  |  |
|  |  |  | (3) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1.2 | **Button [Players Born in September]** |  |  |
|  |  |  |  |
|  | Display the **Name** and **DateOfBirth** fields for all players born in September. |  |  |
|  |  |  |  |
|  | Example of output: |  |  |
|  |  |  |  |
|  |  |  | (3) |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1.3 | **Button [Average Game Times]** |  |  |
|  |  |  |  |
|  | Display the average duration of the games played each day in a new field called **AverageDuration**. The values must be rounded off to ONE decimal place. |  |  |
|  |  |  |  |
|  | Example of output: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (5) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1.4 | **Button [Highest Scores]** |  |  |
|  |  |  |  |
|  | Display the **Name** of the player and the highest score that the player obtained in any game so far in a field called **HighestScore**. |  |  |
|  |  |  |  |
|  | Example of output: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (5) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1.5 | **Button [Add Game]** |  |  |
|  |  |  |  |
|  | Add a new game with the number 76 as **GameIndx** to the **tblGames** table. The game was played by a player with **PlayerID** 'HM008' on 24 December 2017. The player scored 6 655 points and the game lasted  250 minutes. |  |  |
|  |  |  |  |
|  | Example of output: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (4) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.2 | **DATABASE MANIPULATION** |  |  |

|  |  |
| --- | --- |
| Select tab sheet **Qst2.2 DB**, which displays the content of both tables as well as the three buttons for the questions that need to be completed: |  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **NOTE:** NO marks will be allocated for the use of SQL statements for QUESTION 2.2. The names of the tables to be used in your code must be **tblPlayers** and **tblGames**.  Complete the code to meet the requirements specified in QUESTION 2.2.1 to QUESTION 2.2.3. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.2.1 | **Button [Domain Users Count]** |  |  |
|  |  |  |  |
|  | * An input box must be used to enter an e-mail domain name, e.g. 'gmail.com'. * Display the names and e-mail addresses of players who use an e-mail address which contains the domain that was entered. * Display headings, as shown in the example below. * Show the total number of addresses at the bottom of the list. |  |  |
|  |  |  |  |
|  | Example of output if the domain name 'gmail.com' has been entered: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (9) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.2.2 | **Button [Change Score]** |  |  |
|  |  |  |  |
|  | An additional 250 points can be added to a player's score in some cases, after a dispute has been resolved in favour of the player. Write code to increase the score of the player of the current active record in the  **tblGames** table display grid, with 250 points. |  | (4) |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.2.3 | **Button [Add Game]** |  |  |
|  |  |  |  |
|  | Write code to add a new game to the **tblGames** table according to the following requirements:   * The **GameIndx** assigned to the new game must be compiled automatically and follow on the number that was assigned to the previous game.   **NOTE:** Records in a database table are indexed starting at 1 up to  the number of records in the table.   * The **GameDate** must be the date the game was added to the table using the 'today's date' function. * The **PlayerID** must be obtained from the selected record in the **tblPlayers** table. * Input boxes in the supplied code must be used to allow the user to enter values for the **Duration** field and **Score** field. |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Example of a new record that has been added for Sue Prins who played  a game for 111 minutes and obtained a score of 2 222: |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (7) |

|  |  |  |
| --- | --- | --- |
| * Enter your examination number as a comment in the first line of the program file. * Save your program. * Print the code if required. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TOTAL SECTION B:** |  | **40** |

|  |  |  |
| --- | --- | --- |
| **SECTION C** |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **QUESTION 3: OBJECT-ORIENTATED PROGRAMMING** |  |  |

|  |  |  |
| --- | --- | --- |
| The LAN FANatics group is designing software to assist with the registration of players who want to take part in the LAN parties that they host. |  |  |

|  |  |  |
| --- | --- | --- |
| Do the following: |  |  |
| * Open the incomplete program in the **Question3** folder. * Open the incomplete object class **Player\_U.pas**. * Compile and execute the program.   The following user interface is displayed with limited functionality: |  |  |

|  |
| --- |
|  |

|  |
| --- |
| **NOTE:** Code has been supplied to obtain today's date when the program is executed and display the date in the **Date** edit box. This is done to show the date format that is used by your computer. You need to use the displayed format when entering dates to ensure that any date-based calculations will function correctly. |

|  |  |  |
| --- | --- | --- |
| * Complete the code as specified in QUESTION 3.1 and QUESTION 3.2. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 3.1 | The provided incomplete object class (**TPlayer**) contains the declaration of four attributes which describe a **Player** object. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | The attributes for a **Player** object have been declared as follows:   * fEmail – the e-mail address that uniquely identifies the player * fName – the name of the player * fDateOfBirth – the player's date of birth * fApproved – a flag indicating whether the registration of the player has been approved or not |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | The following methods have also been provided:   * **create** – used to instantiate the object * **toString** – used to provide output of all attributes of the object |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Complete the code in the object class as described in QUESTION 3.1.1 to QUESTION 3.1.5. |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3.1.1 | Write an accessor method called **getEmail** for the **fEmail** attribute. | (2) | |
|  |  |  |  | |
|  | 3.1.2 | Write a mutator method called **setApproved** to receive a Boolean value (true/false) as a parameter and assign the received value to the **fApproved** attribute. | (3) | |
|  |  |  |  | |
|  | 3.1.3 | Write a **private** method called **testEmail** that can be used to test the validity of the supplied e-mail address. The **testEmail** method must receive the e-mail address as a parameter and return a Boolean value (true/false) based on the following test:  The address may only start with a letter from the alphabet (upper case or lower case) and with NO other character or digit. | (5) | |
|  |  |  |  |  |
|  | 3.1.4 | Change the provided constructor method called **create** to use the **testEmail** method in order to test whether the e-mail address format is valid before allocating it to the **fEmail** attribute.  If the e-mail address is invalid, the word 'ERROR' must be allocated to the **fEmail** attribute. | (5) | |
|  |  |  |  |  |
|  | 3.1.5 | Write a **public testAge** method that will receive the minimum age requirement for registration as a parameter and determine whether a player is old enough to be registered. The player's age must be calculated using today's date and the player's date of birth.  If the player's age meets the minimum age requirement, the word 'APPROVED' must be returned. If the player is too young, the word 'REJECTED' must be returned. | (7) | |

|  |  |  |  |
| --- | --- | --- | --- |
| 3.2 | An incomplete program has been supplied in the **Question3** folder. The program contains code for the object class to be accessible and declares an object variable called **objPlayer**.  Write code to perform the tasks described in QUESTION 3.2.1 to  QUESTION 3.2.3 on the next page to approve and register players. |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3.2.1 | **Button [Submit Details]** |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | The user needs to enter the player's e-mail address, name and date of birth. The date format that you use for the date of birth must be the same as the format of the current date displayed in the **Date** edit box.  Write code to do the following:   * Use the details that were entered to instantiate a new **objPlayer** object. * Clear the **Display** area and the **Age Test Result** edit box. * Call the **toString** method to display the details of the object that has been instantiated in the **Display** area. |  |  |
|  |  |  |  |  |
|  |  | Example of output in the **Display** area if an object with an invalid  e-mail address has been entered: |  |  |
|  |  |  |  | (7) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3.2.2 | **Button [Test Age]** |  |  |
|  |  |  |  |  |
|  |  | The user must enter the minimum age requirement in the edit box provided. Use the age that was entered as an argument to call the **testAge** method to test whether the player's age adheres to the minimum requirement to be registered for the LAN party. Display the result in the edit box provided. |  |  |
|  |  |  |  |  |
|  |  | Example of output for an age that has been approved: |  |  |
|  |  |  |  | (3) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 3.2.3 | **Button [Approve]** |  |  |
|  |  |  |  |  |
|  |  | Use the result of the age test and the content of the e-mail attribute of the object to determine whether the player's application can be approved or not.  If the age has not been approved or the e-mail address has an error, the following message must be displayed in a message box: **'The player's age or e-mail address has been rejected.'** Also call the correct method to ensure that the **fApproved** status of the object is 'false'.  If the application meets all the requirements use the correct method to change the **fApproved** status of the object to 'true'.  Clear and update the content of the **Display** area to show the latest **fApproved** status of the player object. |  | (8) |
|  |  |  |  |  |
|  |  |  |  |  |
| * Enter your examination number as a comment in the first line of the object class and the form class. * Save your program. * Print the code in the object class and the form class if required. | | |  | |

|  |  |  |
| --- | --- | --- |
| **TOTAL SECTION C:** |  | **40** |

|  |  |  |
| --- | --- | --- |
| **SECTION D** |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **QUESTION 4: PROBLEM-SOLVING PROGRAMMING** |  |  |

|  |
| --- |
| The LAN FANatics group will be hosting an elite Master Players competition. They want to select all the players with scores above that of average players. A text file that contains the names and scores of all the players has been supplied. |

|  |  |  |
| --- | --- | --- |
| Do the following: |  |  |
|  |  |  |
| * Compile and execute the program in the **Question4** folder. The program has no functionality currently. |  |  |
| * Complete the code for each question as described in QUESTION 4.1 and QUESTION 4.2. |  |  |

|  |  |  |
| --- | --- | --- |
| The following GUI is displayed when the program is executed: |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| The following has been provided: |  |  |
|  |  |  |
| * A text file called **Scores.txt**,which contains the names and scores of all the players who participated in the game play events. The information is supplied for each player in the following format in the file:   <name>,<score>   * A **Load Scores** button, which contains code to read the data from the **Scores.txt** file into two parallel arrays called **arrNames** and **arrScores**. * A procedure to display suitable headings and the names and scores contained in the two parallel arrays. |  |  |

|  |  |  |
| --- | --- | --- |
| Write code to perform the tasks described in QUESTION 4.1 and QUESTION 4.2 on the next page. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 4.1 | **Button [Remove Duplicates]** |  |  |
|  |  |  |  |
|  | More than one game score was captured for some of the players in the file. Write code to ensure that only the highest scores and the corresponding names remain **in the provided arrays**. |  |  |
|  |  |  |  |
|  | Example of output that shows only the highest score for each player when the **Display** procedure which displays the content of the arrays are executed: |  |  |
|  |  |  |  |
|  |  |  | (16) |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 4.2 | **Button [Display Master Players]** |  |  |
|  |  |  |  |
|  | * Obtain the names and scores of all players with scores higher than the average score of all players. Use the array, which contains only the highest scores, to calculate the average score. That is the array where lower duplicate scores of the players have been removed. * The average score must be displayed as part of the heading in the output area. * Save the names and scores of these players in another array called **arrHighScores** as a string in the following format: <Name>#<Score>   Example: **John Smith#5623**   * Sort the content of the array **arrHighScores** alphabetically and display the sorted names and scores in the **Display** area. |  |  |
|  |  |  |  |
|  | Example of output: |  |  |
|  |  |  |  |
|  |  |  | (14) |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| * Enter your examination number as a comment in the first line of the program file. * Save your program. * Make a printout of the code if required. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TOTAL SECTION D:**  **GRAND TOTAL:** |  | **30**  **150** |