education

Department:
Education
PROVINCE OF KWAZULU-NATAL

JUNE EXAMINATION 2017
INFORMATION TECHNOLOGY
GRADE 12
PAPER 1

EXAMINER: V. B. RAMKILAWAN (*Stanger Secondary School*)
MODERATOR: R. PILLAY (*Mountview Secondary School*)

MARKS: 150
DURATION: 3 HOURS

INSTRUCTIONS AND INFORMATION:

1. This question paper is divided into THREE questions. Answer ALL THREE questions.
2. This paper is set in programming terms that are specific to the Delphi Programming Language (utilizing the IDE *Embarcadero Delphi 2010*).
3. Make sure that you answer the questions according to the specifications that are given in each question. Marks will only be awarded according to the set requirements.
4. 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.
5. 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.
6. Routines such as search, average and selection must be developed from first principles. You may not use the built-in features of a programming language for any of these routines.
7. You must save your work regularly (at least once every 5 minutes) on the disk you have been given, or the disk space allocated to you for this examination.
8. Make sure that your name appears as a comment in every program that you code as well as on every event indicated.
9. 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.

DATA FILES

The files that you need to complete this question paper have been given to you on a disk / CD / DVD / flash disk or on the disk space allocated to you.
The files are provided in a folder named: "**G12JuneDataFiles**". Inside this folder are THREE subfolders: **Question1, Question2** and **Question3**. DO NOT MODIFY THE STRUCTURE OF THESE FOLDERS. Ensure that ALL data files are present before beginning the examination.

IMPORTANT FILES PROVIDED:

Folder	Files	Purpose
Question1	`Question1P.dproj`	Main Project File (Open this file to answer Question 1)
	`players.txt`	Text File used in Questions 1.3 and 1.4
Question2	`Question2P.dproj`	Main Project File (Open this file to answer Question 2)
	`clsTeam.pas`	Object Class for Question 2
Question3	`Question3P.dproj`	Main Project File (Open this file to answer Question 3)

THIS QUESTION PAPER CONSISTS OF 13 PRINTED PAGES.

SCENARIO

LAN Gaming has become one of the fastest growing sectors of the Computer Industry in the past decade. The premise behind LAN Gaming is simple: Gamers connect compatible computers or gaming consoles to a common Switch or Router and then play co-operatively (together) or competitively (against each other).

Shujin Academy, a local high school, is hosting a LAN Party with the aims of fundraising and increasing interest in Computer Science. For a week, players both internal and external (aged 13 years or older) will be invited to participate in the LAN Party with prizes for players and teams emerging at the top of the leaderboards.

QUESTION 1

1.1 Each player's performance is tracked using a generated code which will be unique to that player. The code that is generated is based on specific personal data provided by the participant.

There are two basic types of entrants:

INTERNAL – A player attending Shujin Academy.
EXTERNAL – A player not in school, or attending another school.

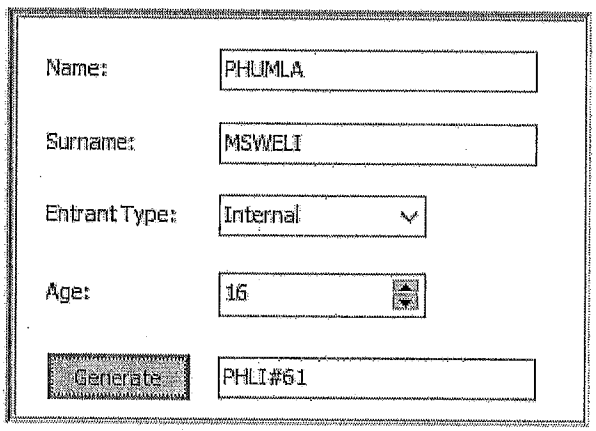
Write code for the "GENERATE" button (btnQ1_1) which will:

Extract the player's name, surname, entrant type and age from the associated components.

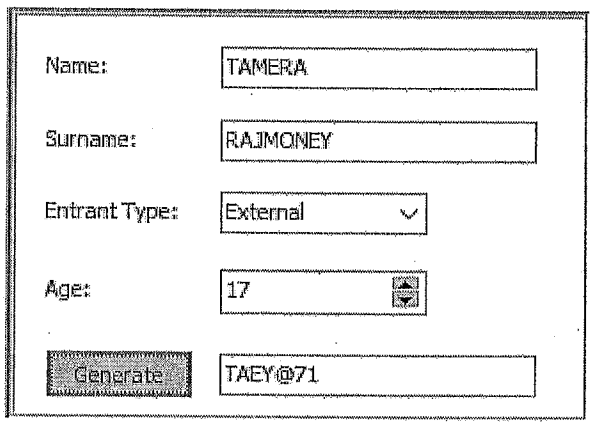
Generate a code for the player by combining:

- First 2 letters of the player's first name.
- Last 2 letters of the player's surname.
- The character # if the entrant is INTERNAL
- The character @ if the entrant is EXTERNAL.
- The digits of the player's age reversed (swopped).

Display the generated code in the Edit Box **edtQ1_1**. **[15]**



SAMPLE DATA #1



SAMPLE DATA #2

1.2 A LAN-Party caters for two broad categories:

COMPETITIVE: Players compete against each other.
CO-OPERATIVE: Players work together trying to achieve a common goal.

Competitive Matches tend to put a greater strain on servers since they require each player to be individually tracked to a greater detail.

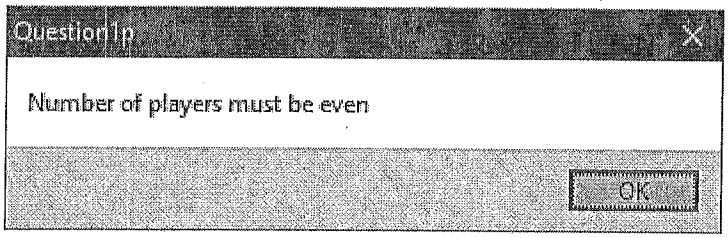
Hence, the entry fees are as follows:

COMPETITIVE: R1000 per player
CO-OPERATIVE: R850 per player

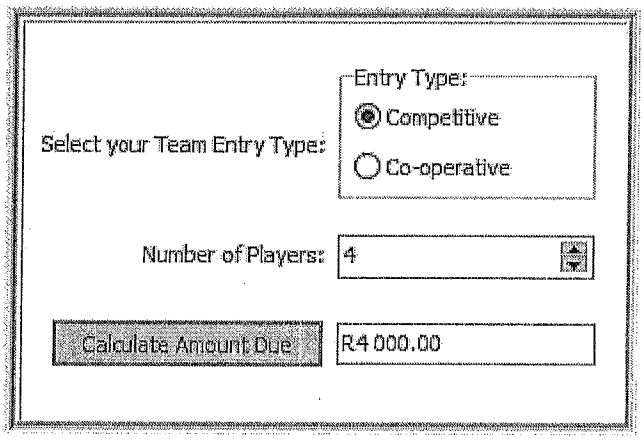
When the "CALCULATE AMOUNT DUE" button (btnQ2_2) is clicked:

- Obtain data from the Radio Group and determine the type of match being registered for.
- Obtain data from the spinner to determine the number of players.

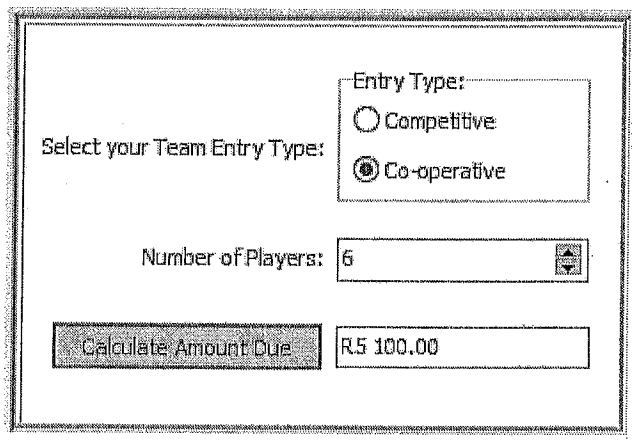
NOTE: The number of players must be even. If the number of players is an odd number, display a Dialog Box with the message "Number of players must be even" and no further processing should take place.



If the number the of players is even, calculate the amount due and display the value in the Edit Box **edtQ2_2**, formatted as currency to 2 decimal places. **[10]**



SAMPLE DATA #1



SAMPLE DATA #2

1.3 A text file named **PLAYERS.TXT** has been compiled with contains a list of player data for all currently registered players. The data is captured in the following format:

REG NUMBER # GAMER TAG # GENDER # PAYMENT STATUS

FIRST TWO ENTRIES FROM THE TEXT FILE:

```
1#SKULL#M#NOTPAID
2#PANTHER#F#PAID
```

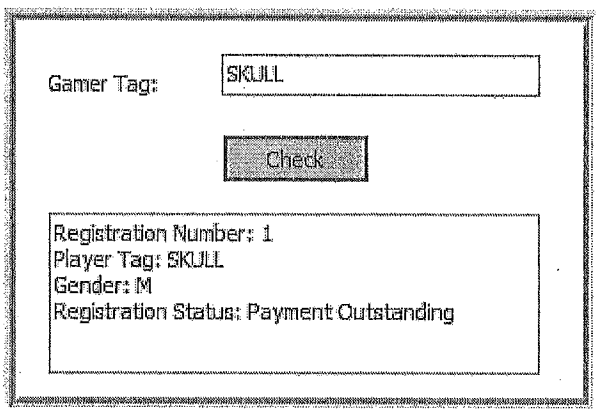
A facility is required for players to check if they are already registered.

When the "CHECK" button (btnQ1_3) is clicked:

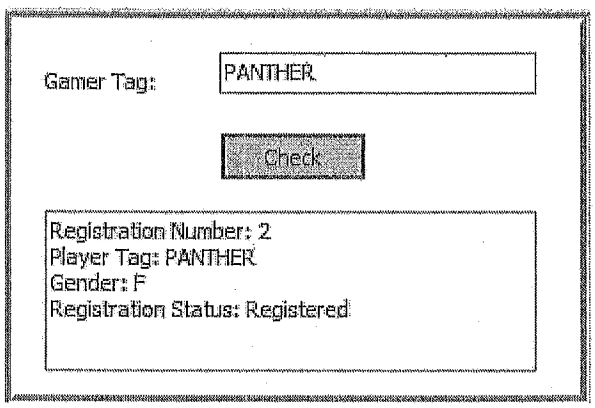
- Check whether the text file (PLAYERS.TXT) exists. If it does not exist, display the message "File Not Found" and processing should stop.
- If the file exists, connect to the text file and open it.
- Extract the tag the user provided in the Edit Box (**edtGamerTag**).
- Loop through the text file, reading and processing each line.
- Search for the tag in the text file.
- If the tag is found, display the following data in the Rich Edit Box (**redQ1_3**):

Registration Number
Player Tag
Gender ("M" or "F")
Registration Status ("Registered" or "Payment Outstanding")

NOTE: A player is considered registered if they have paid the required fee in full.

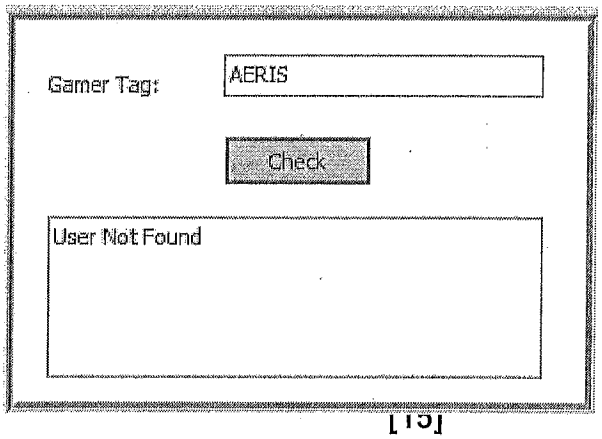


TA #1



SAMPLE DATA #2

If the tag is not found in the text file, display the message "User Not Found" in the Rich Edit Box (**redQ1_3**).



[15]

1.4 A system is required to add new players to text file PLAYERS.TXT. Newly registered players have to choose a GamerTag by which they are identified on the network.

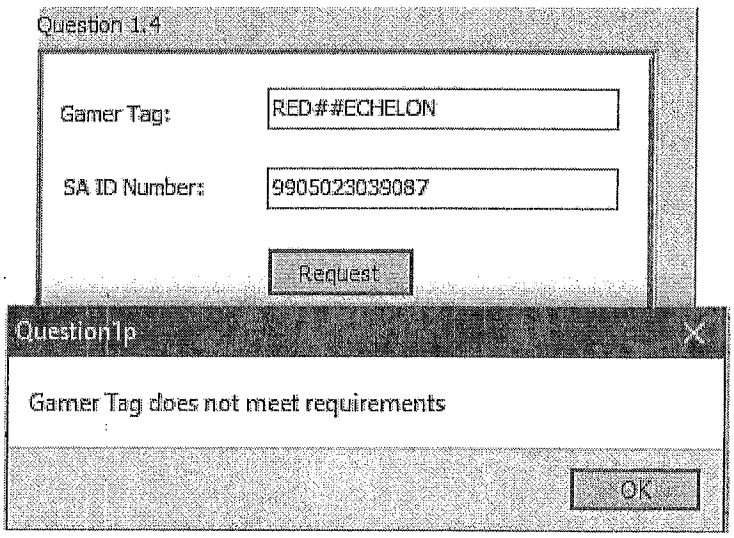
A GamerTag is valid if it meets the following requirements:

- Be a minimum of 6 characters
- Be a maximum of 14 characters.
- Should consist of letters of the alphabet ONLY; both uppercase and lowercase letters are acceptable.
- No digits or special characters are allowed.

When the "REQUEST" button (btnQ1_4) is clicked:

Extract the requested Gamer Tag from the Edit Box (edtReqTag) and validate it, checking whether it meets the requirements stated above.

If it fails to meet the requirements, display a Dialog Box with the message "Gamer Tag does not meet requirements" and no further processing should take place.



Extract the player's South African ID from the Edit Box edtSAID.

If the Gamer Tag meets the requirements, append the new tag to the Text File PLAYERS.TXT using the format:

REG NUMBER # GAMER TAG # GENDER # PAYMENT STATUS

IMPORTANT NOTES:

a. The REG NUMBER should be determined by counting the number of lines in the text file to determine the next valid number.

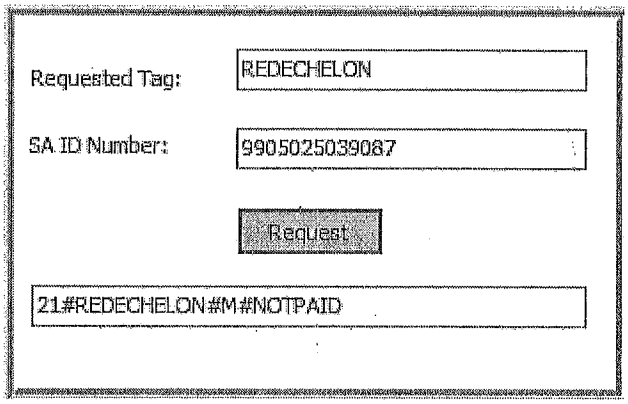
b. The Gender of the Player is determined by extracting the user's ID Number from the Edit Box edtSAID and checking digits 7-10.
If the extracted digits are **> 4999**, the player is **male**. If the extracted digits are **< 5000**, the player is **female**.

FOR EXAMPLE:

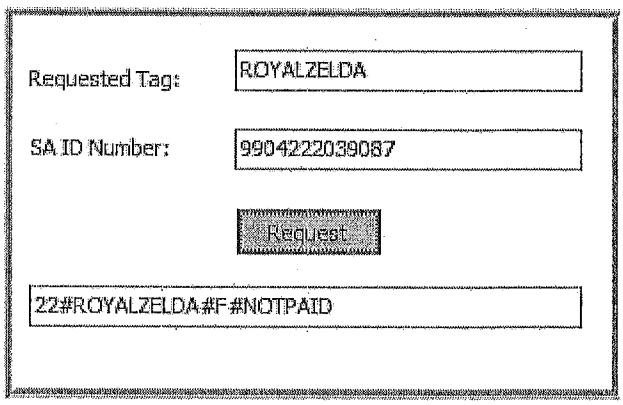
990402**5059**087 – is male.
990301**2020**081 – is female.

c. Payment Status should be set to "**NOTPAID**" for all new registrations.

Append the generated code to the text file and display it in the Edit Box **edtQ1_4**.



SAMPLE DATA #1



SAMPLE DATA #2

[20]

SUBTOTAL: [60]

QUESTION 2

A team consists of 2 or more players. A system is required to process data on teams, so that important details like score calculation and leaderboards can be catered for.

The following details (attributes) are required for a team:

Attribute	Example	Description
TeamName	PhantomThieves	Name of the Team
NoPlayers	6	Number of players in the Team
Score	30	Team's current score
FoulStatus	FALSE	Determines whether or not, the team has committed any actions which break the rules of the game.

NOTE: 2.1.1 to 2.1.7 should be coded in the object class: CLSTEAM.PAS.

2.1.1 Declare the attributes listed above using suitable data types. (5)
NOTE: Marks are allocated for the use of the best-suited data type for each attribute.

2.1.2 Write code for a parameterized constructor which will initialize all attributes. (6)

2.1.3 Write a mutator method called **setFoulStatus** to receive a Boolean value as a parameter and update the *foulStatus* attribute. (3)

2.1.4 Write an accessor method named **getScore** which will return the team's current score. (3)

2.1.5 Write method named **calcAve** which will determine and return the average score per player. (3)

2.1.6 When a team commits a "foul" (breaks the rules of the tournament), the team's score is decreased by 10%.
Write a method named **processFoul** which will evaluate the *FoulStatus* attribute. If the attribute is TRUE, 10% should be subtracted from the *Score* attribute. (5)

2.1.7 Write a **toString** method which will return the values of all attributes formatted as follows:

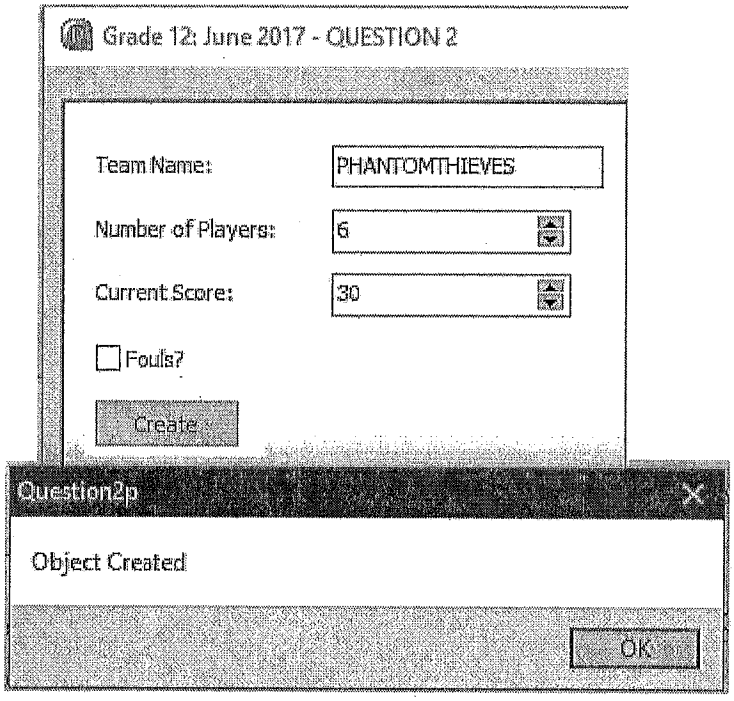
```
TEAM:      PhantomThieves
PLAYERS:   6
SCORE:     30
Fouls?     No
```
(9)

[34]

NOTE: 2.2.1 to 2.2.6 should be answered in Form Class: QUESTION2U.PAS.

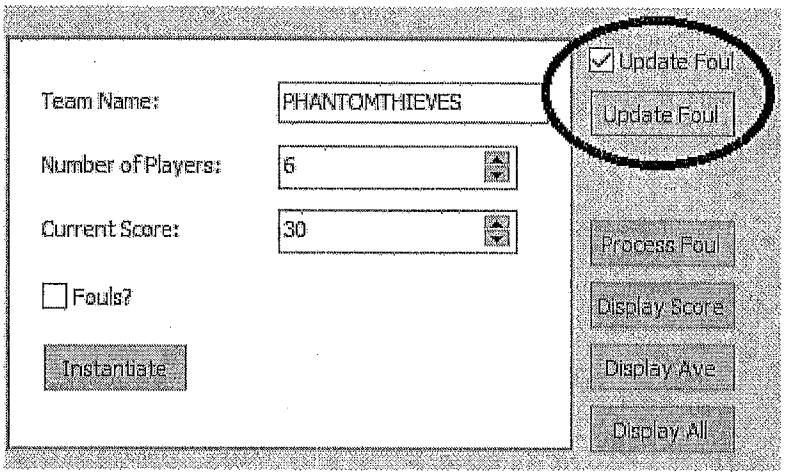
2.2.1 **Write code for the "CREATE" button (btnCreate) which will:**

- Extract data from the components in Panel *pnlInput* and instantiate the global object *objTeam*. (*objTeam* has been declared globally.)
- Display a message using a dialog box to indicate that the object has been successfully created. (7)



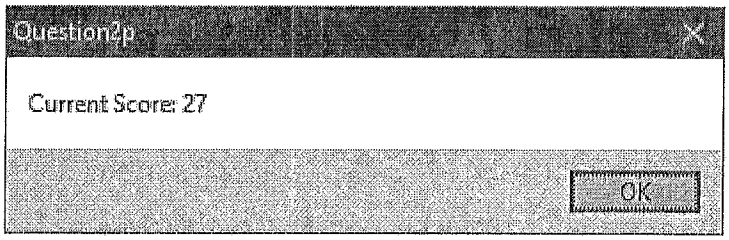
2.2.2 **When the "UPDATE FOUL" button (btnUpdate) is clicked:**

Use the value from checkbox (*chbUpdate*) to update the *foulStatus* attribute. (2)

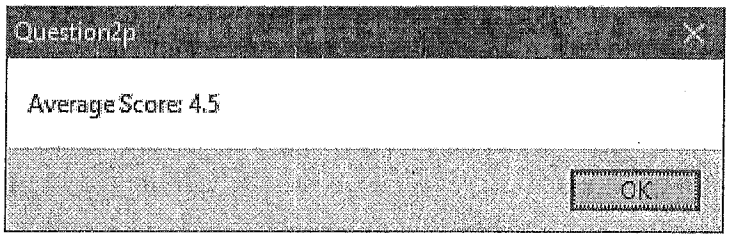


2.2.3 **When the "PROCESS FOUL" button (btnProcess) is clicked:**
When a team breaks the rules of fair-play during a match (commits a "foul"), use the *processFoul* method to update the team's score. (1)

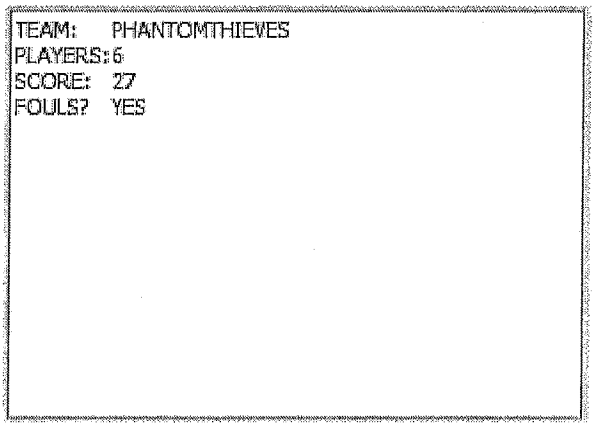
2.2.4 **When the "DISPLAY SCORE" button (btnScore) is clicked:**
Display the current score of the team with a suitable message using a dialog box (2)



2.2.5 **When the "DISPLAY AVE" button (btnAverage) is clicked:**
Use a dialog box to display the average score using the *calcAve* method. (2)



2.2.6 **When the "DISPLAY ALL" button (btnDisplay) is clicked:**
Write code to use the *toString* method to display the details of the team in the Rich Edit Box (**redOutput**). (2)

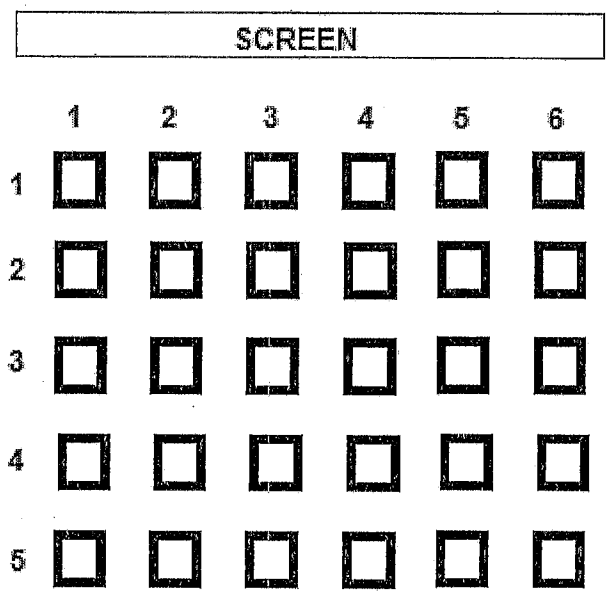


[16]

SUBTOTAL: [50]

QUESTION 3

Learners will be able to watch streamed matches of the local tournaments on the Smart Boards in select classrooms. Seating in these classrooms will be arranged in the following manner:



GIVEN DECLARATIONS AND STRUCTURES:

- A 2-Dimensional array **arrSeating** has been declared with the intention of holding the seating plan of the class. Note that the given code has initialized all cells to store the "#" symbol. The # symbol means that the seat is vacant.
- A 1-Dimensional array **arrNames** has been declared with 10 names belonging to learners from 12A. These learners will be allocated seats first. In order for the system to be fair, seats will be allocated randomly.
- A procedure called **dispData** has been provided which displays all necessary data. Call this method when required to refresh the displayed data.
- A 1-Dimensional array named **arrEarnings** which will be used to determine earnings from the showcased matches.

IMPORTANT NOTE:

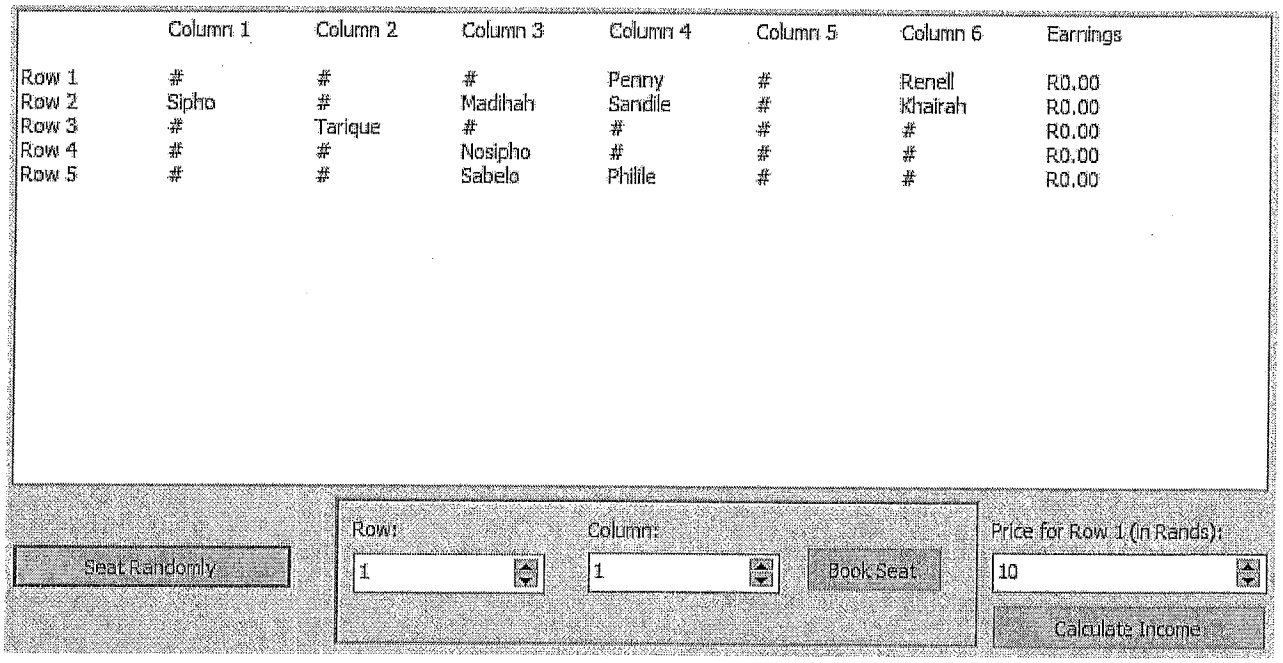
As this question involves the use of random numbers, the screenshots will not necessarily match your output.

3.1 **When the "SEAT RANDOMLY" button (btnSeatRandom) is clicked:**
Write code which will take each name from the 1-Dimensional array *arrNames* and randomly allocate a seat to that learner. This seat is reserved for that learner by placing the learner's name in a randomly chosen position in *arrSeating*.

Your code MUST fulfil the following requirements:

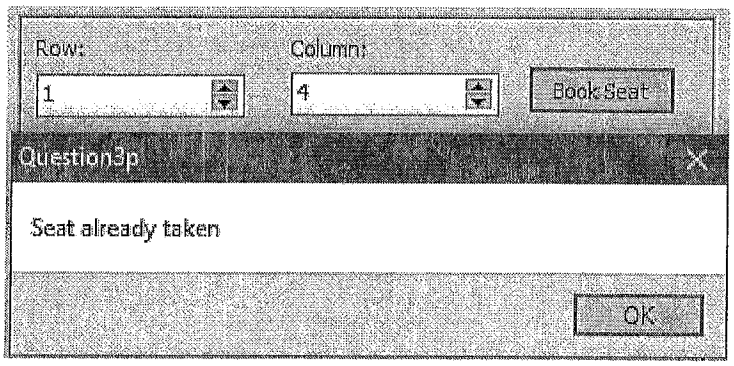
- The row AND the column must be chosen randomly.
- You must ensure that all 10 learners in *arrNames* are allocated a seat. i.e.: because of the randomization, there is a chance that a learner may be allocated a seat that is already taken.
- Your code must ensure that a seat is vacant before allocating the seat to a learner. Each seat should be allocated to 1 learner only.

Call the given procedure *dispData*. (14)

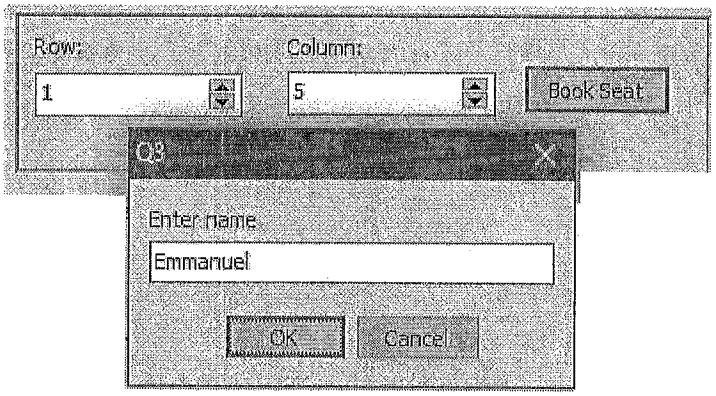


3.2 When the "BOOK SEAT" button (btnBook) is clicked:

Extract the Row and Column values from the Spinners *sedRow* and *sedColumn*. Check whether the required seat is available. If it is not available, display the message "Seat already taken".



If the seat is available, display an Input Dialog Box prompting the user to enter their name.



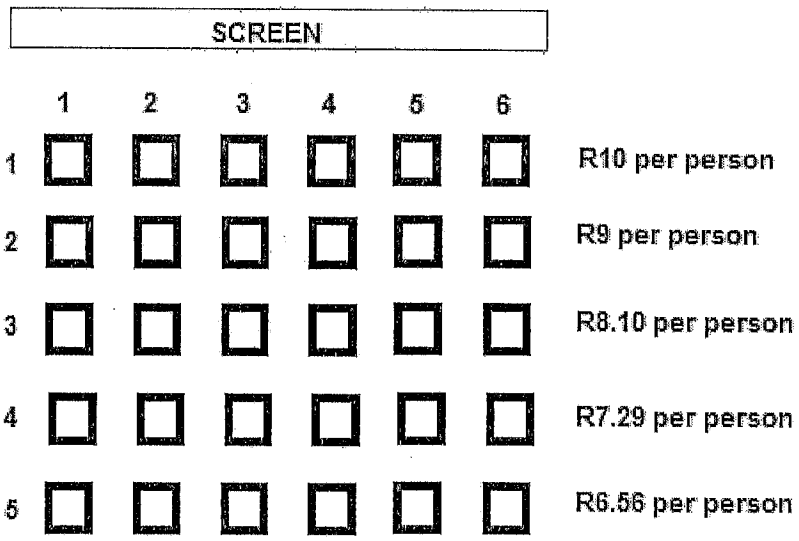
- Update *arrSeating* by placing the user's name in the specified cell.
- Call the given procedure *dispData.* (10)

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Earnings
Row 1	#	#	#	Penny	Emmanuel	Renell	R0.00
Row 2	Sipho	#	Madihah	Sandile	#	Khairah	R0.00
Row 3	#	Tarique	#	#	#	#	R0.00
Row 4	#	#	Nosipho	#	#	#	R0.00
Row 5	#	#	Sabelo	Phillile	#	#	R0.00

3.3 Learners will be charged a small fee for viewing the streamed games in order to raise funds and offset the data costs.
The fees vary based on a number of factors. Since viewers in the front row are closer to the action, a higher fee is charged in comparison to those learners further back.

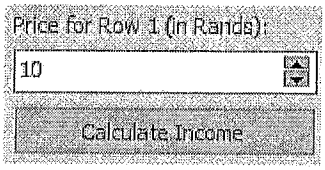
As a general rule, each row further away from the screen is charged 10% less.

EXAMPLE (Assuming Row #1 costs R10 per person):



When the "CALCULATE INCOME" button (btnCalcIncome) is clicked:

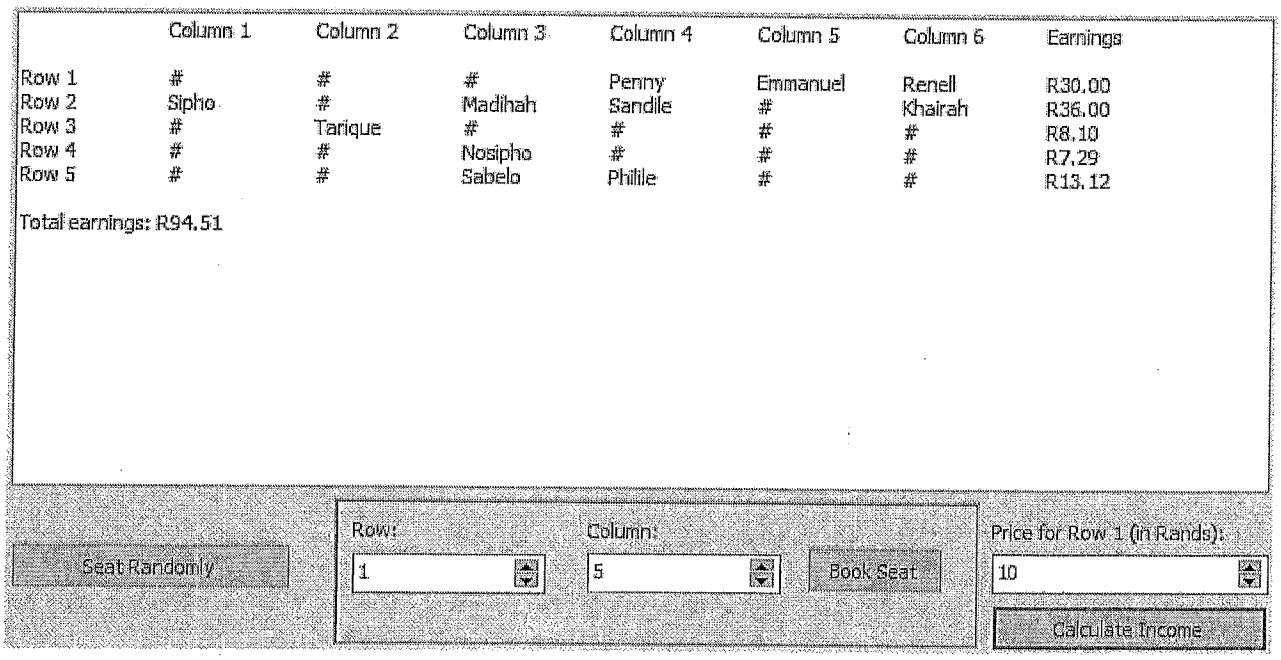
Extract the Price for Row 1 from the Spinner (sedPrice).



Determine the earnings for each row and store the calculated value in *arrEarnings*.
NOTE: A seat should only be counted if it is booked. Vacant seats cannot be counted.

Call the *dispData* procedure.

Determine the total earnings (earnings from all the rows) and display the value in the Rich Edit Box. (16)



SUBTOTAL: [40]

GRAND TOTAL: 150

education

Department:
Education
PROVINCE OF KWAZULU-NATAL

JUNE EXAMINATION 2017
INFORMATION TECHNOLOGY
GRADE 12
PAPER 1

MARKING GUIDE AND POSSIBLE SOLUTIONS

QUESTION 1

NO	TASKS	Max	Mark
1.1	Extracts name from edtName for processing ✓ Extracts Surname from edtSurname for processing ✓ Conditional Statement to evaluate cmbType ✓ Sets "#" symbol for Index 0 ✓ Sets "@" symbol for Index 1 ✓ Extracts Age from sedAge ✓ Inverts age digits ✓✓ Generates code by combining: First two letters of name✓ Last two letters of surname✓✓✓ Type symbol✓ Age digits inverted✓ Displays generated code in edtQ1_1. ✓	15	
1.2	Extracts value from sedNoOfPlayers for processing✓ Conditional structure to test rgbEntryType for Index 0✓ Assigns Price to 1000 ✓ Conditional structure to test rgbEntryType for Index 1✓ Assigns Price to 850✓ Tests Number of Players for divisibility by 2✓ Calculates total cost correctly✓ Displays total cost to edtQ1_2✓ formatted✓. Error message if Num of Players not divisible by 2. ✓	10	
1.3	Checks if players.txt exists ✓ Displays error message and stops execution if not✓ Assigns file to players.txt✓ Opens file for reading✓ Loops to end of text file✓ Extracts line from text file✓ Splits line from text file based on delimiter✓✓ Tests correct extract from line with Input GamerTag (from edtGamerTag) ✓ If found: Displays Registration Num with caption✓ Player Tag with caption ✓	15	

	Gender with caption✓ Evaluates Registration Status and displays "Registered" OR "Payment Outstanding". ✓ Closes file✓ Displays "User Not Found" if entry is not found. ✓		
1.4	Extracts input data from edtReqTag and edtSAID for processing ✓ Sets a Boolean Flag for testing validity of input / OR uses an alternative structure correctly✓ Tests Length with correct range✓✓ Loops from 1 to Length of input tag for evaluation ✓ Tests each letter for validity changing the flag variable, if necessary✓ Error message✓ if validation failed ✓ Counter variable to count number of lines in text file assigned to 0. ✓ Text file assigned and opened✓ Loop through text file✓ Counter increased✓ Counter increased by 1✓ Extracts characters 7-10 from ID✓ Evaluates extracted data ✓ Sets F correctly✓ Sets M correctly✓ Combines data to form output string✓ Writes output string to text file✓ Displays output string in edtQ1_4✓	20	
		[60]	

POSSIBLE SOLUTION : QUESTION 1

```
procedure TForm1.btnQ1_1Click(Sender: TObject);
var
   sAge, sCode, sName, sSurname, sType, sInv : String;

begin
  sName := edtName.Text; ✓
  sSurname := edtSurname.Text; ✓
  if cmbType.ItemIndex = 0 then✓
  begin
     sType := '#'; ✓
  end
  else
  begin
     sType := '@'; ✓
  end;

  sAge := IntToStr(sedAge.Value); ✓

  sAge := sAge[2] ✓ + sAge[1]; ✓

  sCode := sName[1] + sName[2] ✓ + copy(sSurname✓,
(Length(sSurname)-1✓), 2✓) + sType✓ + sAge✓;

  edtQ1_1.Text := sCode; ✓
end;

procedure TForm1.btnQ1_2Click(Sender: TObject);
var
   iNum, iPrice : Integer;
   rTotal : Real;
begin
   iNum := sedNoOfPlayers.Value; ✓

   if rgbEntryType.ItemIndex = 0 then✓
   begin
     iPrice := 1000; ✓
   end
   else✓
   begin
     iPrice := 850; ✓
   end;

   if iNum mod 2 = 0 then✓
   begin
     rTotal := iPrice * iNum; ✓
     edtQ1_2.Text := FloatToStrF(rTotal✓, ffCurrency, 8, 2) ✓;
   end
```



```
   else
   begin
     showMessage('Number of players must be even'); ✓
   end;

end;

procedure TForm1.btnQ1_3Click(Sender: TObject);
var
  t : TextFile;
  sSplit : TStrings;
  s : String;
begin

  if not fileexists('players.txt') then ✓
  begin
    showMessage('File Not Found'); ✓
  end
  else
  begin
    AssignFile(t, 'players.txt'); ✓
    Reset(t); ✓

    redQ1_3.Clear;

    while not eof(t) do✓
    begin
      ReadLn(t, s); ✓
      sSplit := TStringList.Create; ✓

      ExtractStrings(['#'],[],PChar(s),sSplit); ✓

      if edtGamerTag.Text = sSplit[1] then✓
      begin
        redQ1_3.SelText := ('Registration Number: '+sSplit[0] +
                            #13+✓
                             'Player Tag: '+ sSplit[1] + #13+✓
                             'Gender: '+sSplit[2] +  #13+✓
                             'Registration Status: ');
        if sSplit[3] = 'PAID' then
           redQ1_3.SelText := 'Registered'
        else
           redQ1_3.SelText := 'Payment Outstanding';  ✓

      end;

    end;
    CloseFile(t); ✓
```

```
    if redQ1_3.Text = '' then
       redQ1_3.Text := 'User Not Found'; ✓
  end;
end;

procedure TForm1.btnQ1_4Click(Sender: TObject);
const
   alphabet = ['a'..'z','A'..'Z'];
var
   sReqTag, sGenTag, sID, sGender, s : String;
   iGenCode, i, c : Integer;
   bCheck : Boolean;
   t : TextFile;
begin
   sReqTag := edtReqTag.Text;      ✓
   sID := edtSAID.Text;

   bCheck := TRUE; ✓

   if (Length(sReqTag) >= 6) ✓ AND (Length(sReqTag) <= 14) ✓  then
   begin
      for i := 1 to Length(sReqTag) do✓
      begin
        if not(sReqTag[i] in alphabet) then bCheck := FALSE; ✓
      end;
   end
   else
   begin
     bCheck := FALSE;
   end;

   if not bCheck then ✓
   begin
     showMessage('Gamer Tag does not meet requirements'); ✓
   end
   else
   begin
      c := 0; ✓
      AssignFile(t, 'players.txt');
      Reset(t); ✓
      while not eof(t) do✓
      begin
        ReadLn(t, s);
        inc(c); ✓
      end;
      CloseFile(t);

      inc(c); ✓

      iGenCode := StrToInt(copy(sID, 7, 4)); ✓
```



```
      if iGenCode < 5000 then✓
         sGender := 'F'✓
      else
         sGender := 'M'; ✓

      sGenTag := IntToStr(c)+'#'+sReqTag+'#'+sGender+'#NOTPAID'; ✓
      Append(t);
      WriteLn(t, sGenTag); ✓
      CloseFile(t);
      edtQ1_4.Text := sGenTag; ✓
   end;
end;
```

ALTERNATIVE SOLUTION : QUESTION 1

```
procedure TForm1.btnQ1_1Click(Sender: TObject);
var
  sName, sSurname, sType, sFirst2, sLast2: String;
  iAge, iLen, iRev: Integer;
  cUser: char;
begin
  { Question 1.1 }
  sName := edtName.Text✓;
  sSurname := edtSurname.Text✓;
  sType := cmbType.items[cmbType.ItemIndex];
  iAge := sedAge.Value✓;
  sFirst2 := copy(sName, 1, 2) ✓;
  iLen := length(sSurname);
  sLast2 := copy(sSurname, iLen - 1) ✓;
  if sType = 'Internal' then✓
  begin
    cUser := '#'✓
  end
  else
  begin
    cUser := '@'; ✓
  end;
  iRev := (iAge mod 10) * 10✓ + iAge div 10✓;
  edtQ1_1.Text := sFirst2 + sLast2 + cUser✓ + IntToStr(iRev) ✓;

end;

procedure TForm1.btnQ1_2Click(Sender: TObject);
var
  iIndex, iNumPlayers: Integer;
  rAmount, rTotal: Real;
begin
  { Question 1.2 }
  iIndex := rgbEntryType.ItemIndex;
```

```
  rAmount := 0;
  rTotal := 0;
  case iIndex of✓
    0:
      rAmount := 1000; ✓
    1:
      rAmount := 850.00✓;
  end;
  iNumPlayers := sedNoOfPlayers.Value✓;
  if iNumPlayers mod 2 = 0 then✓
  begin
    rTotal := iNumPlayers * rAmount✓;
    edtQ1_2.Text := FloatToStrF(rTotal✓, ffCurrency, 8, 2) ✓;
  end
  else
  begin
    ShowMessage('Number of players must be even') ✓;
  end;
end;

procedure TForm1.btnQ1_3Click(Sender: TObject);
var
  tName: Textfile;
  sLine, sTag, sGamerTag, sPaid, Snum, sGender, sStatus: String;
  iNum, iPos: Integer;
  cGender: char;
  bFlag: Boolean;
begin
  { Question 1.3 }
  try
    begin
      sGamerTag := UpperCase(edtGamerTag.Text);
      AssignFile(tName, 'Players.txt'); ✓
      Reset(tName); ✓
      bFlag := false;
      while (NOT EOF(tName)) ✓ do
      begin
        Readln(tName, sLine) ✓;
        // redQ1_3.Lines.Add(sLine);
        iPos := Pos('#', sLine); ✓
        Snum := copy(sLine, 1, iPos - 1); ✓
        Delete(sLine, 1, iPos); ✓
        iPos := Pos('#', sLine);
        sTag := copy(sLine, 1, iPos - 1); ✓
        // redQ1_3.Lines.Add(sTag);
        Delete(sLine, 1, iPos); ✓
        iPos := Pos('#', sLine); ✓
        sGender := copy(sLine, 1, 1); ✓
        sPaid := copy(sLine, iPos + 1); ✓
```



```
        sStatus := 'Registered';
        if sPaid = 'NOTPAID' then
        begin
          sStatus := 'Payment Outstanding'; ✓
        end;
        if sGamerTag = sTag then
        begin
          bFlag := true;
          redQ1_3.Lines.Clear;
          redQ1_3.Lines.Add('Registration Number ' + Snum);
          redQ1_3.Lines.Add('Player Tag ' + sTag);
          redQ1_3.Lines.Add('Gender  ' + sGender);
          redQ1_3.Lines.Add('Registration Status ' + sStatus);
        end
      end;
      close(tName); ✓
    end;
  except✓
    begin
      ShowMessage('File not found') ✓;
    end;
  end;
  if bFlag = false then
  begin
    redQ1_3.Lines.Clear;
    redQ1_3.Lines.Add('User Not Found');
  end;

end;

procedure TForm1.btnQ1_4Click(Sender: TObject);
var
  sReqTag, sGenTag, sID, sGender, s: String;
  iGenCode, i, c: Integer;
  bCheck: Boolean;
  t: Textfile;

begin
  { Question 1.4 }
  sReqTag := edtReqTag.Text;
  sID := edtSAID.Text; ✓
  bCheck := true;
  if (length(sReqTag) >= 6) ✓ AND (length(sReqTag) <= 14) ✓ then
  begin
    for i := 1 to length(sReqTag) ✓ do
    begin
      if not(sReqTag[i] in ['a' .. 'z', 'A' .. 'Z') then
        bCheck := false; ✓
    end;
  end
```

```
  else
  begin
    bCheck := false;
  end;

  if not bCheck then✓
  begin
    ShowMessage('Gamer Tag does not meet requirements'); ✓
  end
  else
  begin
    c := 0; ✓
    AssignFile(t, 'players.txt');
    Reset(t); ✓
    while not EOF(t) do✓
    begin
      Readln(t, s);
      inc(c); ✓
    end;
    CloseFile(t);
    inc(c); ✓
    iGenCode := StrToInt(copy(sID, 7, 4)) ✓;
    if iGenCode < 5000 then
      sGender := 'F'✓
    else
      sGender := 'M'; ✓

 sGenTag := IntToStr(c) + '#' + sReqTag + '#' + sGender +
'#NOTPAID'; ✓
    Append(t);
    WriteLn(t, sGenTag); ✓
    CloseFile(t);
    edtQ1_4.Text := sGenTag; ✓
  end;

end;

End.
```


QUESTION 2

NO	TASKS	Max	Mark
2.1.1	Declares all 4 attributes correctly✓✓✓✓ With the most suitable data types used for all attributes✓	5	
2.1.2	Declares Constructor in Interface✓ Implements Constructor correctly (header) ✓ by assigning received parameters to attributes. ✓✓✓✓	6	
2.1.3	Declares mutator under Interface✓ Implements mutator correctly (header) ✓ Assigns received parameter to attribute ✓	3	
2.1.4	Declares accessor under Interface✓ Implements accessor correctly (header) ✓ Sends Score attribute correctly✓	3	
2.1.5	Declares CalcAve function correctly under Interface✓ Implements CalcAve correctly (header) ✓ Result calculated using correct formula✓	3	
2.1.6	Declares ProcessFoul under Interface✓ Implements ProcessFoul correctly (header) ✓ Conditional statement to evaluate FoulStatus attribute✓ Score decremented✓ by correctly calculated value✓	5	
2.1.7	Declares toString under Interface✓ Implements toString correctly (header) ✓ Outputs Team Name with correct caption + #13✓ Players with correct caption and new line✓ Score with converted value, caption and new line✓ Evaluates Foul Status ✓ Outputting "Yes" with caption if TRUE✓ "No" with caption if FALSE✓ Returns output correctly. ✓	9	
		(34)	
2.2.1	Extracts data from all input components correctly✓✓✓✓ Calls Constructor correctly✓ with arguments in the correct order. ✓ Confirmation Message✓	7	
2.2.2	Mutator called correctly ✓ Sending value from CheckBox chbUpdate✓	2	
2.2.3	ProcessFoul method called correctly✓	1	
2.2.4	Calls Accessor correctly ✓displaying value✓	2	
2.2.5	Calls Average function✓ displaying value correctly✓	2	
2.2.6	Calls toString function✓, displaying value in redOutput✓	2	
		(16)	
		[50]	

POSSIBLE SOLUTION: QUESTION 2

Question 2.1

```
unit clsTeam;

interface

type
  TTeam = class(TObject)

private
{2.1.1}
  FTeamName : String; ✓
  FNoPlayers : Integer; ✓
  FScore : Integer; ✓
  FFoulStatus : Boolean; ✓

public

{2.1.2}
constructor CREATE(sTeamName : String; iNoPlayers, iScore :
Integer; bFoulStatus : Boolean); ✓

{2.1.3}
procedure setFoulStatus(bFoulStatus : Boolean); ✓

{2.1.4}
function getScore : Integer; ✓

{2.1.5}
function calcAve : Real; ✓

{2.1.6}
procedure processFoul; ✓

{2.1.7}
function toString : String; ✓

end;

implementation
uses SysUtils;

{2.1.2}
constructor TTeam.CREATE(sTeamName : String; iNoPlayers, iScore :
Integer; bFoulStatus : Boolean); ✓
begin
  FTeamName := sTeamName; ✓
  FNoPlayers := iNoPlayers; ✓
```

✓


```
  FScore := iScore; ✓
  FFoulStatus := bFoulStatus; ✓
end;

{2.1.3}
procedure TTeam.setFoulStatus(bFoulStatus : Boolean); ✓
begin
  FFoulStatus := bFoulStatus; ✓
end;

{2.1.4}
function TTeam.getScore : Integer; ✓
begin
  Result := FScore; ✓
end;

{2.1.5}
function TTeam.calcAve : Real; ✓
begin
  Result := FScore / FNoPlayers; ✓
end;

{2.1.6}
procedure TTeam.processFoul; ✓
begin
  if FFoulStatus = TRUE then✓
  begin
    FScore := FScore - ✓Round(FScore * 0.1) ✓;
  end;
end;

{2.1.7}
function TTeam.toString : String; ✓
var
   sOutput : String;
begin
   sOutput := 'TEAM:'+#9+FTeamName+#13+✓
              'PLAYERS:'+#9+IntTOStr(FNoPlayers)+#13+✓
              'SCORE:'+#9+IntToStr(FScore)+#13+✓
              'FOULS?'+#9;
   if FFoulStatus = TRUE then✓
      sOutput := sOutput + 'YES'✓
   else
      sOutput := sOutput + 'NO'; ✓

   Result := sOutput; ✓
end;

end.
```

Question 2.2

```
procedure TForm1.btnCreateClick(Sender: TObject);
var
  sTeam : String;
  iScore, iPlayers : Integer;
  bFouls : Boolean;
begin
  sTeam := edtTeamName.Text; ✓
  iScore := sedScore.Value; ✓
  iPlayers := sedPlayers.Value; ✓
  bFouls := chbFouls.Checked; ✓

  objTeam := TTeam.CREATE✓ (sTeam, iPlayers, iScore, bFouls) ✓;

  showMessage('Object Created'); ✓
end;

procedure TForm1.btnUpdateClick(Sender: TObject);
begin
  objTeam.setFoulStatus✓ (chbUpdate.Checked✓);
end;

procedure TForm1.btnProcessClick(Sender: TObject);
begin
  objTeam.processFoul; ✓
end;

procedure TForm1.btnScoreClick(Sender: TObject);
begin
  showMessage('Current Score: '+✓IntToStr(objTeam.getScore) ✓);
end;

procedure TForm1.btnAverageClick(Sender: TObject);
begin
  showMessage('Average Score: '+✓FloatToStr(objTeam.calcAve) ✓);
end;

procedure TForm1.btnDisplayClick(Sender: TObject);
begin
  redOutput.Text := ✓objTeam.toString✓;
end;
```


QUESTION 3

NO	TASKS	Max	Mark
3.1	Loop✓ to Length of arrNames✓ Generates random number for row in correct range✓ Generate random number for col in correct range✓ Structure to ensure row / col combination has not been chosen already (NOTE: This can be achieved in a variety of different ways – Trace learner solutions to check viability of solution and allocate marks at your discretion) ✓✓✓✓✓✓✓ Assign arrSeating ✓to value from arrNames✓ (allocate marks if array and index is used correctly) DispData method called ✓	14	
3.2	Extracts data from sedRow✓ Extracts data from sedColumn✓ Checks ✓if seat is vacant✓ If not vacant, displays "Booked" message✓ If seat is vacant✓, gets learner's name from Dialogue Box✓ Assigns arrSeating [Correct Indices] ✓ to received name ✓ DispData method called✓	10	
3.3	Extracts Price from sedPrice✓ Sets TotalEarnings variable to 0✓ Loop from 1 to 5 (Loop A) ✓ Sets arrEarnings[correct index] to 0✓ Loops from 1 to 6 (Loop B) ✓ Checks✓ if seat at pos [A][B] is taken✓ If taken, adds Price value to ✓ arrEarnings [correct index] ✓ Increases ✓TotalEarnings value correctly✓ *(NOTE: Some learners may do this separately outside of the nested loops; with its own loop)* Decreases price correctly (-10%)✓✓ DispData method called✓ Displays TotalEarnings formatted as currency✓ with suitable caption✓	16	
		[40]	
GRAND TOTAL:		**150**	

POSSIBLE SOLUTION: QUESTION 3

```
procedure TForm1.btnSeatRandomClick(Sender: TObject);
var
  i, j, c : Integer;
begin

  for c :=✓ 1 to 10 do✓
  begin

    i := Random(5) + 1; ✓
    j := Random(6) + 1; ✓

    while not✓ (arrSeating[i][j] ✓ = '#') do✓
    begin
        i := ✓Random(5) + 1; ✓
        j := ✓Random(6) + 1; ✓
    end;

    arrSeating[i][j] ✓:= arrNames[c]; ✓

  end;

  dispData; ✓
end;
```

Repeat loop may be used in many learner solutions instead of the While..Do

```
procedure TForm1.btnBookClick(Sender: TObject);
var
  i, j : Integer;
  sName : String;
begin
  i := sedRow.Value; ✓
  j := sedColumn.Value; ✓

  if arrSeating[i][j] ✓<> '#' then✓
  begin
    showMessage('Seat already taken'); ✓
  end
  else✓
  begin
    sName :=✓ InputBox('Q3','Enter name',''); ✓
    arrSeating[i][j] := sName; ✓
    dispData; ✓
  end;

end;
```



```
procedure TForm1.btnCalcIncomeClick(Sender: TObject);
var
  i, j : Integer;
  rPrice, rTotal : Real;
begin
  rPrice := sedPrice.Value; ✓
  rTotal := 0; ✓

  for i := 1 to 5 do✓
  begin

    arrEarnings[i] := 0; ✓

    for j := 1 to 6 do✓
    begin

      if arrSeating[i][j] ✓ <> '#' then✓
         arrEarnings[i] :=✓ arrEarnings[i] + rPrice; ✓

    end;
    rTotal := ✓rTotal + arrEarnings[i]; ✓
    rPrice := ✓rPrice - (rPrice * 0.1); ✓

  end;

  dispData; ✓
  redOutput.Lines.Add(#13+'Total earnings: ✓ '+FloatToStrF(rTotal,
ffCurrency, 8, 2)); ✓
end;
```