QUESTION 2: OBJECT-ORIENTED PROGRAMMING

```
unit uHouseholdXXXX;

interface
uses SysUtils;
   type
       arrType = array[1..7] of integer;
       THousehold = class (TObject)
         private
            fAccount         :string;
            fMembers         :integer;
            fArrWaterUse     :arrType;
         public
          constructor create(aAccount : string; aMembers :integer;arrWaterUse :
                                                       arrType );
          function calculateTotal:integer;
          function calculateAve:double;
          function determineHighDay:integer;
          function determineHighRisk(dayLimit:real):boolean;
          function toString:string;
       end;

implementation
//==============================================================================
```

// Q 2.1.1 (3)

```
constructor THousehold.create(aAccount : string; aMembers:integer;
                                          arrWaterUse:arrType);
begin
  fAccount  := aAccount; ✓
  fMembers  := aMembers; ✓
  fArrWaterUse := arrWaterUse; ✓
end;
```

Q 2.1.1
(3) Assign parameters to private fields

```
Accept a loop to assign the array
Subtract only 1 mark if the assignment statements are reversed, e.g.
aAccount := fAccount
```

//==

// Q 2.1.2 (4)

```
Ignore any errors in definition (declaration) of method - no marks
Total (or return value) can be double or int
```

```
function THousehold.calculateTotal:integer;
var
  iTotal, k :integer;

begin
  iTotal := 0; ✓
  for k := 1 to length(fArrWaterUse) do ✓
     iTotal := iTotal + fArrWaterUse[k];
      // or inc(iTotal, fArrWaterUse[k]);
  result := iTotal; ✓
end;
```

Q 2.1.2
(1) Initialise total
(1) for loop
(1) Add array element to total
(1) return total (use result or function name)

Accept: iTotal as an instance/global variable.
Accept: loop to <=7 or < 8
Accept: adding individual elements – no loop
Accept: not using a variable iTotal – add up and assign to result- all in one statement

```
Award 4 marks if method/code done correctly but in the main unit
```

```
//========================================================================
```

// Q 2.1.3 (2)

```
function THousehold.calculateAve:double; ✓
begin
        result := calculateTotal / 7; ✓
end;
```

Q 2.1.3
(1) Data type of return value real (or double)
(1) Correct calculation

Accept the use of iTotal only if calculateTotal has been called (can be called in main unit.
Accept if values are added here to get a total.
Accept integer as a return type.

```
Award 2 marks if method/code done correctly but in the main unit
```

```
//============================================================================
```

// Q 2.1.4 (8/2 = 4) (rounded up)

```
function THousehold.determineHighDay:integer; ✓
var
  iHighDay,  iHighAmount, k  :integer;
begin
  iHighDay := 1; ✓
  iHighAmount := fArrWaterUse[1]; ✓
  for k := 2 to 7 do✓
  begin
    if (fArrWaterUse [k] > iHighAmount) ✓ then
     begin
       iHighDay := k; ✓
       iHighAmount := fArrWaterUse[k]; ✓
     end;
     result := iHighDay; ✓
  end;
 end;
```

Q 2.1.4
(1) Return type integer
(1) Initialise iHighDay
(1) Initialise iHighAmount
(1) For loop
(1) if statement
(1) change iHighDay
(1) change iHighAmount
(1) return iHighDay

Accept sorting the amounts, also returned the correct day (full marks)
Accept correct variations of finding highest e.g. start with 0 as highest instead of first element.
Sorting done correctly but correct day not found and returned – 3 out of 4 marks

```
Award 4 marks if method done correctly but in the main unit
```

```
//==============================================================================
```

// Q 2.1.5 (9)

```
function  THousehold.determineHighRisk(dayLimit:real):boolean;
var
  rAve          :real;
  iCount, k     :integer;
begin
       rAve := calculateAve;
       iCount := 0; ✓
       for k := 1 to length(fArrWaterUse) do✓
       begin
         if(fArrWaterUse[k] > dayLimit) then✓
                  inc(iCount); ✓
       end;
       if ((rAve > dayLimit) ✓  OR✓ (iCount > 2)) ✓ then
        result := true✓
```

Q 2.1.5
(1) Initialise iCount
(1) Loop
(1) if array element > dayLimit
(1) increment count
(3) if rAve > dayLimit or iCount > 2
(1) return true
(1) else return false

```
       else
        result := false; ✓
end;
```

```
Accept variables as global
Do not deduct a mark for input of dayLimit
Accept: if ((calculateAve > dayLimit) OR (iCount > 2))
Accept: a single statement that returns a Boolean value
Result = ✓ (rAve > dayLimit)✓ OR ✓(iCount > 2) ✓✓
Accept: Initialising a Boolean variable, return the Boolean variable
```

```
//============================================================================
```

// Q 2.1.6 (6)

```
1 mark for each piece of information = 5 marks
1 mark for adding all the information in one string
```

```
function THousehold.toString:string;
var
   sObjStr:  string;
   k:integer;
begin
   sObjStr := 'Account number : ' + fAccount + #13 + 'Number of members : ' +
            IntToStr(fMembers) + #13;
   sObjStr := sObjStr + 'Daily water usage' + #13 ✓+ 'Days:         ' + #9;
     for k := 1 to 7 do
             sObjStr := sObjStr + intToStr(k) ✓ + #9;

     sObjStr := sObjStr + #13 + 'Water used:'✓ + #9;
     for k := 1 to length(fArrWaterUse) do✓
       sObjStr := sObjStr + IntToStr(fArrWaterUse[k])✓+
                                                         #9;
                                 // Join strings✓
    result :=  sObjStr;
end;
```

Q 2.1.6
(1) Headings + new line (#13 or #10)
(1) Day numbers
(1) Heading
(2) Values from array
(1) Strings concatenated

```
Accept separate array entries instead of the loop.
Accept any correct form of joining all correct information
```

```
//============================================================================
```

unit Question2XXXX_U;

```
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus,  StdCtrls, ComCtrls;

type
  TfrmHousehold = class(TForm)
    MainMenu1: TMainMenu;
    OptionA: TMenuItem;
    OptionB: TMenuItem;
    redOutput: TRichEdit;
    OptionC: TMenuItem;
    Quit: TMenuItem;
    procedure FormActivate(Sender: TObject);
    procedure QuitClick(Sender: TObject);
    procedure OptionAClick(Sender: TObject);
    procedure OptionBClick(Sender: TObject);
    procedure OptionCClick(Sender: TObject);

  private
  public
    { Public declarations }
  end;

var
  frmHousehold: TfrmHousehold;

implementation
uses
  uHouseholdXXXX;
/===========================================================================
```

// Q 2.2.1 **(2)**

```
var
  Household   :THousehold; ✓
  sAccount    :string;
  iMembers    :integer;
  arrWaterUse :arrType = (481, 438, 454, 353, 421, 396, 432);
{$R *.dfm}

procedure TfrmHousehold.FormActivate(Sender: TObject);
begin
     sAccount := 'AC-23245';
     iMembers := 4;
     Household := THousehold.create(sAccount, iMembers, arrWaterUse); ✓
end;
```

Q 2.2.1
(2) Declare object variable

```
Deduct 1 mark for no parameters.
```

```
procedure TfrmHousehold.QuitClick(Sender: TObject);
begin
  Application.Terminate;
end;
//===========================================================================
```

// Q 2.2.2 **(4)**

```
procedure TfrmHousehold.OptionAClick(Sender: TObject);
begin
   redOutput.Clear;
   redOutput.Lines.Add(Household.toString); ✓
   redOutput.Lines.Add('');
  redOutput.Lines.Add('Total water usage: '✓ +
          IntToStr(Household.calculateTotal) ✓+' litres');
   redOutput.Lines.Add('Average water usage per day: ' +
        FloatToStrF(Household.calculateAve, ✓ ffFixed,8,1) + ' litres');
end;
```

Q 2.2.2
(1) Call the toString method of the object
(1) Display label
(1) Call calculateTotal method
(1) Call calculateAverage method

```
Do not be strict in the wording of the labels and formatting of values
```

```
//===========================================================================
```

// Q 2.2.3 **(6)**

```
procedure TfrmQuestion2.mnuOptionBClick(Sender: TObject);
var
  rAve :real;
  k :integer;
begin
    redOutput.Clear;
    rAve := Household.calculateAve; ✓
    redOutput.Lines.Add('Days and amount of water exceeding the average');
    redOutput.Lines.Add('========================================');
```

Q 2.2.3
(1) Call calculateAve method
(1) Display average
(1) Loop
(1) if
(2) Display number & difference

```
    redOutput.Lines.Add('Average water usage per day: ' +
        FloatToStrF(Household.calculateAve, ffFixed, 8, 1) ✓ + ' litres');
    redOutput.Lines.Add('Days    Value exceeding average by (litres)');
    for k := 1 to length(arrWaterUse) do✓
      begin
        if (arrWaterUse[k] > rAve) then✓
          begin
            redOutput.Lines.Add(IntToStr(k) ✓ + #9 +
            FloatToStrF(arrWaterUse[k]- rAve, ✓ ffFixed,8,1));
          end;
      end;
end;
```

```
No marks for headings
Display average – no matter how average is obtained, mark is not for
formatting
Fourth mark goes for calculation, not formatting
```

```
//=============================================================================
```

// Q 2.2.4 **(5)**

Q 2.2.4
(1) Input rDayLimit
(1) Call toString
(1) Call calculateHighDay
(1) If statement
(1) Display correct message

```
procedure TfrmQuestion2.mnuQuitClick(Sender: TObject);
var
  rDayLimit :double;
begin
  redOutput.Clear;
  rDayLimit := StrToFloat(InputBox('Water Limit',
                       'Enter the limit of water per day', '')); ✓
  redOutput.Lines.Add(Household.toString); ✓
  redOutput.Lines.Add('');
  redOutput.Lines.Add('The day on which the most water was used is: ' +
                   intToStr(household.determineHighDay)); ✓
  redOutput.Lines.Add('');
  if (Household.determineHighRisk(rDayLimit)) ✓ then
      redOutput.Lines.Add('High-risk household')     }
  else                                               }   ✓
      redOutput.Lines.Add('Not a high-risk household');}
end;
end.
```

```
rDayLimit – integer or real
Second mark: For call of toString – no other way accepted to display
Third mark goes for calling method, not label. Accept with no label
Fourth mark: for calling the method as part of an if or assign to variable
Fifth mark: displaying message – mark for two messages with else or second
if
```

[45]