NATIONAL SENIOR CERTIFICATE EXAMINATION
NOVEMBER 2012

INFORMATION TECHNOLOGY: PAPER II

MARKING GUIDELINES

Time: 3 hours 120 marks

These marking guidelines are prepared for use by examiners and sub-examiners, all of whom are required to attend a standardisation meeting to ensure that the guidelines are consistently interpreted and applied in the marking of candidates' scripts.

The IEB will not enter into any discussions or correspondence about any marking guidelines. It is acknowledged that there may be different views about some matters of emphasis or detail in the guidelines. It is also recognised that, without the benefit of attendance at a standardisation meeting, there may be different interpretations of the application of the marking guidelines.

---

1. Information Technology teachers must mark a sample of five papers of their candidates' work, using the supplied *interim marking guidelines and marksheet* as a guideline. They must then meet with their cluster group to discuss potential problems with the marking guidelines, as outlined in the *Subject Assessment Guidelines* document.

2. The mark scheme and solution given is just one way of solving the problem. If a candidate has solved the problem in a different manner that still uses good programming techniques and incorporates all aspects of the question, then full marks may be awarded.

3. Place **ALL** your candidates' printed work and updated marksheets in **ONE** envelope with the school name and examination number clearly marked on the outside of the envelope.

4. With the exception of the sample of 5 papers, the interim mark scheme may **not** be used to mark the candidates' examinations. The updated mark scheme must be obtained from <http://groups.yahoo.com/group/iebcompza> or <www.ieb.co.za> one week after the examination. A candidate's work must be marked using the mark scheme spreadsheet so that the spreadsheet will calculate the candidate's total mark. Place a printed copy of the spreadsheet in front of each candidate's solution. Make sure the examination number is clearly visible on each marked script. Teachers must show evidence that the print-outs have been marked by indicating ticks and so on. Marking should be done in red pen.

5. You will receive a summary mark schedule for the centre which must be returned with the candidates' marksheets and printouts. These marks should also be entered via the IEB online facility.

PRACTICAL EXAMINATION SUBMISSION CHECK LIST

A copy of this form must be printed, completed and submitted with the examination scripts.

I, _____ (name), of _____ (school), confirm that I have marked these scripts to the best of my ability according to the **revised** marksheet that was published on the IEB website after the standardisation meeting, taking into account all revisions and modifications from the original.

My submission conforms the following requirements:

☐ Each candidate's marked print-outs, stapled into question order. These print-outs show evidence of mark justifications and allocations.

☐ A printed copy of the revised marksheet issued after the standardisation meeting with marks captured electronically for each candidate.

☐ A single compact disk that contains:

1. An electronic copy of the *entire* contents of each candidate's examination folder as it existed at the point when the examination concluded. These should be stored in folders named with the candidates' examination numbers.
2. An electronic copy of the revised marksheet with marks captured electronically for each candidate (either 1 file per candidate, or 1 file with multiple worksheets).

☐ A copy of the IEB marksheet with **printed** marks to confirm accurate capture on the IEB system. This marksheet is also be signed by your school's Head.

☐ The above contents put into a single envelope per 15 candidates. If more than 15 candidates are present, please include the data CD, this form and printed IEB marksheet in the first envelope only.

Teacher's Signature: _____

Head's Signature: _____

Date: _____

Place candidate's barcoded sticker here. This sheet must be stapled to the font of each candidates' submission.	**IT PRACTICAL EXAMINATION 2012**			
	Version:	**Draft Marksheet (April 2012)**	**Prog. Lang.**	
	Candidate:			
	Marker:		**120**	**0**
	Checksum:	0000-0000-0120	**Total**	**Pupil**

1.1	SELECT * FROM tblWaiters ORDER BY waiterName;	**3**	
1.2	SELECT tableID, tableGuests FROM tblTables WHERE tblGuests = 1 OR tblGuests >= 10;	**3**	
1.3	SELECT menuDescription FROM tblMenuItems WHERE menuDescription LIKE '*chips*';	**3**	
1.4	SELECT menuDescription, menuSalesPrice – menuCostPrice AS profit FROM tblMenuItems WHERE menuCategory = "Drinks";	**3**	
1.5	INSERT INTO tblWaiters (waiterName, waiterPhone) VALUES ("Busi", "083 469 9000");	**3**	
1.6	SELECT menuDescription, (menuSalesPrice / menuCostPrice – 1) * 100 AS MarkUp FROM tblMenuItems ORDER BY (menuSalesPrice / menuCostPrice - 1) * 100 DESC;	**5**	
1.7	SELECT menuDescription, SUM(orderQuantity) AS Quantity FROM tblMenuItems INNER JOIN tblOrders ON tblOrders.orderMenuItemID = tblMenuItems.menuID GROUP BY menuDescription;	**5**	
1.8	SELECT waiterName, COUNT(tableID) AS tablesServed, AVG(tableAmountPaid) AS avgAmountPaid FROM tblWaiters INNER JOIN tblTables ON tblWaiters.waiterID = tblTables.tableWaiterID GROUP BY waiterName;	**7**	
1.9	SELECT waiterName, SUM(orderQuantity)*10 AS Prize FROM tblWaiters INNER JOIN (tblTables INNER JOIN (tblMenuItems INNER JOIN tblOrders ON tblMenuItems.menuID = tblOrders.orderMenuItemID) ON tblTables.tableID = tblOrders.orderTableID) ON tblWaiters.waiterID = tblTables.tableWaiterID WHERE menuDescription LIKE "*Giant Burger*" GROUP BY waiterName;	**8**	
2	Class header is correct; PROPERTIES: all private, all appropriate data types, all named correctly; CONSTRUCTOR: method header is correct, assignments are correct; GETTERS: all getters correct (-1 per error to a max of 2); METHODS: setter is correct; changeQuantity has correct header, increase; toString has correct header, formatting & fields;	**13**	
3.1	Class header is correct with extend	**1**	
3.2	**Properties:** both private, both double, both named appropriately	**3**	
3.3	**Constructor:** header is correct (-1 per error to a max of 2), calls parent constructor, assignments are correct.	**4**	
3.4	**mustOrder method:** method header is correct, if statement with correct condition (getQuantity () < minimumLevel) (-1 per error to a max of 2), return true else, return false.	**4**	

3.5	**getOrderAmount method:** method header is correct, correct calculation for return maximumLevel – getQuantity () (-1 per error to a max of 2)	**3**	
4.1	Class header is correct	**1**	
4.2	**Properties:** both are declared private, correct data type for each (StockItem array, int), both initialised (array of 100, count = 0)	**4**	
4.3	**Constructor:** method header correct, open file for reading, indefinite loop, correct looping condition, parse line on "#", "if" determines object correctly, correctly create a StockItem object with parameters, correctly create a StockItemOrder with parameters, increment counter, read in a new line in the loop	**10**	
4.4	**getStockList:** method header correct, initialise a temporary variable, appropriate for loop, concatenate the toString with a newline, return concatenated variable	**7**	
4.5	**getOrderingList:** method header correct, for loop to loop through each element, if-statement to check object type, type-casting, if-statement to check for order, concatenate correct fields (getDescription () + ": " + getOrderAmount() + " " + getUnit ()) to return var with a newline (-1 for errors to a max of 2), return	**8**	
4.6	**findStockItem:** method header correct, for-loop to iterate through all records, compare to search string, return found object, return null if none.	**5**	
5.1	Class header correct	**1**	
5.2	Instantiate a StockManager object	**1**	
5.3	Print both headings, print stock list, print ordering list	**3**	
6.1	**updateStockLevels:** method header correct, open file for reading, indefinite loop with correct condition, parse text, check for "used" and reduce, check for "bought" and increase, otherwise set level; read next line.	**10**	
6.2	Perform stock take, display info as before	**2**	

SOLUTIONS

SECTION A – Structured Query Langauge

NB: All queries here are guidelines as other possible solutions may exist. Provided that the query given meets the requirements of the question, full marks may be awarded. For example, a WHERE clause may be used instead of an INNER JOIN clause.

```
1.1   SELECT * FROM tblWaiters ORDER BY waiterName;                    (3)

1.2   SELECT tableID, tableGuests FROM tblTables WHERE tblGuests = 1 OR
      tblGuests >= 10;                                                 (3)

1.3   SELECT menuDescription FROM tblMenuItems WHERE menuDescription LIKE
      '*chips*';                                                       (3)

1.4   SELECT menuDescription, menuSalesPrice – menuCostPrice AS profit
      FROM tblMenuItems WHERE menuCategory = "Drinks";                 (3)

1.5   INSERT INTO tblWaiters (waiterName, waiterPhone) VALUES ("Busi",
      "083 469 9000");                                                 (3)
```


1.6
```
SELECT menuDescription, (menuSalesPrice / menuCostPrice – 1) * 100
AS MarkUp FROM tblMenuItems ORDER BY (menuSalesPrice /
menuCostPrice - 1) * 100 DESC;
```
(5)

1.7
```
SELECT menuDescription, SUM(orderQuantity) AS Quantity FROM
tblMenuItems INNER JOIN tblOrders ON tblOrders.orderMenuItemID =
tblMenuItems.menuItemID GROUP BY menuDescription;
```
(5)

1.8
```
SELECT waiterName, COUNT(tableID) AS tablesServed,
AVG(tableAmountPaid) AS amountPaid FROM tblWaiters INNER JOIN
tblTables ON tblWaiters.waiterID = tblTables.tableWaiterID GROUP BY
waiterName;
```
(7)

1.9
```
SELECT waiterName, SUM(orderQuantity)*10 AS Prize FROM tblWaiters
INNER JOIN (tblTables INNER JOIN (tblMenuItems INNER JOIN tblOrders
ON tblMenuItems.menuItemID = tblOrders.orderMenuItemID) ON
tblTables.tableID = tblOrders.orderTableID) ON tblWaiters.waiterID
= tblTables.tableWaiterID WHERE menuDescription LIKE "*Giant
Burger*" GROUP BY waiterName;
```
(8)

40 marks

SECTION B OBJECT ORIENTED PROGRAMMING

JAVA SOLUTION:

QUESTION 2

```
public class StockItem
{
    private String description;
    private double quantity;
    private String units;   (all correctly named)

    public StockItem (String description, double quantity, String units)   header
                        correct
    {
        this.description = description;
        this.quantity = quantity;
        this.units = units;   (all assignments correct) ("this." Is not necessary!)
    }

    public String getDescription ()
    {
        return description;
    }

    public double getQuantity ()
    {
        return quantity;
    }

    public String getUnits ()  (all "getters" correct, -1 per error to a max of 2)
    {
        return units;
    }

    public void setQuantity (double quantity)
    {
        this.quantity = quantity;  (setter is correct)
    }

    public void changeQuantity (double quantity)   (method header correct)
    {
        this.quantity += quantity;   (increase correct)
    }

    public String toString ()  (method header correct)
    {
        return description + ": " + quantity + " " + units;   (formatting, fields, return all
                        perfect)
    }

}
```

[13]

QUESTION 3

```
public class StockFoodItem extends StockItem (class header correct, including extend)
{
    // Question 3.2
    private double minimum_level;
    private double maximum_level;

    // Question 3.3
    public StockFoodItem (String description, double quantity, String units,
                          double minimum_level, double maximum_level)  (-1 if no "super"
                          fields)
    {
        super (description, quantity, units);  (calls parent constructor)
        this.minimum_level = minimum_level;
        this.maximum_level = maximum_level;  ("this." Is not necessary!)
    }

    // Question 3.4
    public boolean mustOrder () (method header correct)
    {
        if (getQuantity () < minimum_level)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // Question 3.5
    public double getOrderAmount () (method header correct)
    {
        return this.maximum_level – this.getQuantity ();
    }
}
```

[15]

QUESTIONS 4 AND 6.1

```
// Question 4.1
public class StockManager (class header correct)
{
    // Question 4.2
    private StockItem []inventory = new StockItem [100]; (both must be private)
    private int stockcount = 0;

    // Question 4.3
    public StockManager () (method header correct)
    {
        try
        {
            BufferedReader br = new BufferedReader (new FileReader
                        ("stocklist.txt")); (open file for reading)

            String line = br.readLine ();
            while (line != null) (loop, terminating condition)
            {
                String[] bits = line.split ("#"); (parse based on '#')
```

```
                if (bits.length == 3)   (check for which object)
                {
                    inventory [stockcount] = new StockItem (bits[0],
                        Double.parseDouble (bits[1]), bits[2]);   (instantiate an object)
                    stockcount ++; (increment)
                }
                else if (bits.length == 5)
                {
                    inventory [stockcount] = new StockItemOrder (bits[0],
                        Double.parseDouble (bits[1]), bits[2],
                        Double.parseDouble (bits[3]), Double.parseDouble
                        (bits[4]));
                    stockcount ++;
                }

                line = br.readLine (); (read next line)
            }
        }
        catch (IOException ioe)
        {
            System.out.println (ioe);
        }
    }

    // Question 4.4
    public String getStockList ()  method header correct
    {
        String temp = ""; initialise temp variable

        for (int i = 0; i < stockcount; i ++)  appropriate loop
        {
            temp += inventory [i].toString () + "\n";  append, toString, new line
        }

        return temp;   return
    }

    // Question 4.5
    public String getOrderingList ()  method header correct
    {
        String temp = ""; initialise a temporary variable
        for (int i = 0; i < stockcount; i ++)
        {
            if (inventory[i] instanceof StockItemOrder)   check type
            {
                StockItemOrder tempItem = (StockItemOrder) inventory[i];
                        (typecast)
                if (tempItem.mustOrder ())  check for reorder
                {
                    temp += tempItem.getDescription () + ": " +
                        tempItem.getOrderAmount () + " " + tempItem.getUnit () +
                        "\n";
                }
            }
        }

        return temp;
    }

    // Question 4.6
    public StockItem findStockItem (String description)   method header correct
```

```
    {
        for (int i = 0; i < stockcount; i ++) for-loop
        {
            if (inventory[i].getDescription ().equalsIgnoreCase (description))
                        comparison
            {
                return inventory[i];   return found object
            }
        }
        return null;   return null if nothing found
    }

    // Question 6.1
    public void updateStockLevels () method header
    {
        try
        {
            BufferedReader br = new BufferedReader (new FileReader
                        ("stockusage.txt")); open file

            String line = br.readLine ();

            while (line != null)  (indefinite loop)
            {
                String[] bits = line.split (":");

                String item = bits[0].trim ();
                String change = bits[1].trim (); (parse correctly)

                StockItem temp = findStockItem (item);

                if (temp != null)
                {
                    if (change.length () >= 5 && change.substring (0,
                        5).equalsIgnoreCase ("used ")) (check for used)
                    {
                        String amount = change.substring (5);
                        double amt = Double.parseDouble (amount);
                        temp.changeQuantity (-amt);   (reduce)
                    }
                    else if (change.length () >= 7 && change.substring (0,
                        7).equalsIgnoreCase ("bought ")) (check for bought)
                    {
                        String amount = change.substring (7);
                        double amt = Double.parseDouble (amount);
                        temp.changeQuantity (amt);   increase
                    }
                    else
                    {
                        double amt = Double.parseDouble (change);
                        temp.setQuantity (amt);   set
                    }
                }

                line = br.readLine (); next line
            }
        }
        catch (IOException ioe)
        {
            System.out.println (ioe);
        }
    }
}
```

[35]

QUESTIONS 5 AND 6.2

```
// Question 5.1
public class RestaurantDriver  class header correct
{
    public static void main (String[] args)
    {
        // Question 5.2
        StockManager sm = new StockManager ();  instantiate

        // Question 5.3
        System.out.println ("STOCK LIST");  both headings
        System.out.println (sm.getStockList ());
        System.out.println ("ORDER LIST");
        System.out.println (sm.getOrderingList ());

        // Question 6.2
        sm.updateStockLevels ();
        System.out.println ("STOCK LIST");  same as before
        System.out.println (sm.getStockList ());
        System.out.println ("ORDER LIST");
        System.out.println (sm.getOrderingList ());
    }
}
```

[7]

DELPHI SOLUTION:

QUESTION 2

```
unit uStockItem;

interface

uses SysUtils;

type TStockItem = class (class header)
  private (properties private)
    description (all names correct) : String (all types correct);
    quantity : real;
    units : String;
  public
    constructor Create(d : String; q : real; u : String);
    function getDescription : String;
    function getQuantity : real;
    function getUnit : String;
    procedure setQuantity(q : real);
    procedure changeQuantity(diff : real);
    function toString : String;
end;

implementation

{ TStockItem }

constructor TStockItem.Create(d: String; q: real; u: String);  (constructor)
begin
  description := d;
  quantity := q;
  units := u;  (all assignments correct)
end;

function TStockItem.getDescription: String;
begin
  Result := description;
end;

function TStockItem.getQuantity: real;
begin
  Result := quantity;
end;

function TStockItem.getUnit: String;
begin
  Result := units;
end;  (all getters correct)

procedure TStockItem.setQuantity(q: real);
begin
  quantity := q;
end;  (setQuantity is correct)

procedure TStockItem.changeQuantity(diff: real);  (correct header)
begin
  quantity := quantity + diff;  (increase)
end;

function TStockItem.toString: String;  (correct header)
```

```
begin
  Result := description + ': ' + FloatToStr(quantity) + ' ' + units;   (correct
                          formatting and fields)
end;

end.
```

[13]

QUESTION 3

```
unit uStockItemFood;

interface

uses uStockItem;

// Question 3.1
type TStockItemFood = class(TStockItem)   (class with extend)
  private  (prop declared as private)
    // Question 3.2
    minLevel  (name): real;   (type)
    maxLevel : real;
  public
    constructor Create(d: String; q: real; u: String; min : real; max : real);
    function mustOrder : boolean;
    function getOrderAmount : real;

end;

implementation

{ TStockItemFood }

// Question 3.3
constructor TStockItemFood.Create(d: String; q: real; u: String; min,
  max: real);   (constructor header)
begin
  inherited Create(d, q, u);   (parent constructor)
  minLevel := min;
  maxLevel := max;   (assignments)
end;

// Question 3.4
function TStockItemFood.mustOrder: boolean;   (method header)
begin
  Result := getQuantity < minLevel;   (calculates and returns appropriate boolean – could also use an
                          if-statement)
end;

// Question 3.5
function TStockItemFood.getOrderAmount: real;   (method header correct)
begin
  Result := maxLevel - getQuantity;   (correct calculation, return)
end;
end.
```

[15]

QUESTIONS 4 AND 6.1

```
unit uStockManager;

interface

uses uStockItem, uStockItemFood, classes, SysUtils, Dialogs;

// Question 4.1
type TStockManager = class (class deifnition)
  private (private)
    // Question 4.2
    arrStock : array[1..100] of TStockItem;  (TStockItem array)
    counter : integer;  (int)
  public
    constructor Create;
    function getStockList : String;
    function getOrderingList : String;
    function findStockITem(description : String) : TStockItem;
    procedure updateStockLevels;
end;

implementation

{ TStockManager }

// Question 4.3
constructor TStockManager.Create;  (method header correct)
var
  infile : textfile;
  fname : String;
  line : String;
  description : String;
  quantity : real;
  units : String;
  minimum : real;
  maximum : real;
begin
  counter := 0;  (4.2 – initialisation)
  fname := 'stocklist.txt';

  if (FileExists(fname)) then
    begin
      AssignFile(infile, fname);
      Reset(infile);  (open file for reading)

      while NOT(EOF(infile)) do (indefinite loop)  (condition)
        begin
          Readln(infile, line);  (read new line)
          Inc(counter);  (increment counter)

          description := Copy(line, 1, Pos('#', line) - 1);  (parse line on #)
          Delete(line, 1, Pos('#', line));

          quantity := StrToFloat(Copy(line, 1, Pos('#', line) - 1));
          Delete(line, 1, Pos('#', line));

          if (Pos('#', line) > 0) then (if determines object correctly)
            begin
              units := Copy(line, 1, Pos('#', line) - 1);
              Delete(line, 1, Pos('#', line));
```

```
              minimum := StrToFloat(Copy(line, 1, Pos('#', line) - 1));
              Delete(line, 1, Pos('#', line));

              maximum := StrToFloat(line);

              arrStock[counter] := TStockItemFood.Create(description, quantity,
                       units, minimum, maximum);   (create TStockItemFood with parameters)
            end
          else
            begin
              units := line;
              arrStock[counter] := TStockItem.Create(description, quantity,
                       units);   (create TStockItem with parameters)
            end;
        end;

      CloseFile(infile);
    end
  else
    begin
      ShowMessage('File not found');
    end;

end;

// Question 4.4
function TStockManager.getStockList: String;   (method header correct)
var
  rString : String;   (temp variable – if uses "Result" directly give the mark)
  loop : integer;
begin
  for loop := 1 to counter do  (for loop)
    rString := rString +  (concatenate) arrStock[loop].toString  (toString) + #13
                          (newline);

  Result := rString;   (return – if uses Result directly, give the mark)
end;

// Question 4.5
function TStockManager.getOrderingList: String;   (method header correct)
var
  rString : String;
  loop : integer;
  tmpItem : TStockItemFood;
begin
  for loop := 1 to counter do  (appropriate loop)
    begin
      if (arrStock[loop] is TStockItemFood) then  (if to check object type)
        begin
          tmpItem := arrStock[loop] as TStockItemFood;   (type cast)

          if (tmpItem.mustOrder) then  (if to check for order)
            begin
              rString := rString + tmpItem.getDescription + ': ' +
                       FloatToStr(tmpItem.getOrderAmount) + ' ' +
                       tmpItem.getUnit + #13;   (concatenate output)
            end;
        end;
    end;

  Result := rString;   (return – if uses Result directly, give the mark)
end;
```

```
// Question 4.6
function TStockManager.findStockITem(description : String): TStockItem;   (method
                          header correct)
var
  loop : integer;
  rItem : TStockItem;
begin
  rItem := nil;   (return nil if not found)
  for loop := 1 to counter do  (for loop)
    begin
      if (arrStock[loop].getDescription = description) then  (compare to search string)
        begin
          rItem := arrStock[loop];   (return object)
        end;
    end;

  Result := rItem;
end;
```

[35]

```
// Question 6.1
procedure TStockManager.updateStockLevels;   (method header correct)
var
  infile : textfile;
  line : String;
  item, change : String;
  amount : real;
  sItem : TStockItem;
begin
  if (FileExists('stockusage.txt')) then
    begin
      AssignFile(infile, 'stockusage.txt');
      Reset(infile);   (open file for reading)

      while NOT(EOF(infile)) do  (indefinite loop)   (condition)
        begin
          Readln(infile, line);   (read next line)
          item := Trim(Copy(line, 1, Pos(':', line) - 1));
          Delete(line, 1, Pos(':', line));   (parse text)

          change := Trim(line);

          sItem := findStockITem(item);

          if ((Length(change) >= 5) AND (Copy(change, 1, 5) = 'used ')) then
                          (check for used)
            begin
              Delete(change, 1, Pos(' ', change));
              amount := StrToFloat(change);
              sItem.changeQuantity(-amount);   (reduce)
            end
          else if ((Length(change) >= 7) AND (Copy(change, 1, 7) = 'bought '))
                          then  (bought)
            begin
              Delete(change, 1, Pos(' ', change));
              amount := StrToFloat(change);
              sItem.changeQuantity(amount);   (increase)
            end
          else
            begin
```

```
              amount := StrToFloat(change);
              sItem.setQuantity(amount);   (otherwise set level)
            end
        end;

      CloseFile(infile);
    end
  else
    begin
      ShowMessage('File not found');
    end;
end;

end.
```

[10]

QUESTIONS 5 AND 6.2

```
// Question 5.1
unit uRestaurantDriver;   (class header is correct)

interface

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

type
  TfrmRestaurantDriver = class(TForm)
    rchOutput: TRichEdit;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmRestaurantDriver: TfrmRestaurantDriver;
  sm : TStockManager; // Question 5.2

implementation

{$R *.dfm}

procedure TfrmRestaurantDriver.FormActivate(Sender: TObject);
begin
  sm := TStockManager.Create;   (instantiate a stock manager)

  // Question 5.3
  rchOutput.Lines.Add('=== STOCK LIST ===');
  rchOutput.Lines.Add(sm.getStockList);   (stock list)
  rchOutput.Lines.Add('=== ORDER LIST ===');   (both headings)
  rchOutput.Lines.Add(sm.getOrderingList);   (order list)
```

[5]

```
  // Question 6.2
  sm.updateStockLevels;   (perform stocktake)
  rchOutput.Lines.Add('=== STOCK LIST ===');   (new output)
  rchOutput.Lines.Add(sm.getStockList);
```

```
  rchOutput.Lines.Add('=== ORDER LIST ===');
  rchOutput.Lines.Add(sm.getOrderingList);
end;

end.
```

[2]

OUTPUT

SECTION A STRUCTURED QUERY LANGUAGE

QUESTION 1.1

waiterID	waiterName	waiterPhone
1	Erin	083 276 2000
13	Esme	084 112 4000
5	Hayden	079 073 0000
14	Jacques	073 127 6000
12	Jen	083 347 9000
7	Kwame	082 210 0000
2	Matthew	071 584 3000
8	Morgan	076 017 4000
3	Nomfundo	087 654 3000
10	Ramola	085 551 2000
6	Ruth	078 765 5000
4	Simphiwe	072 111 2000
11	Tendai	087 612 3000
9	Vashni	077 517 8000

QUESTION 1.2

tableID	tableGuests
6	10
8	11
9	12
10	1
35	1
48	1
50	1
51	1
54	1
57	1
58	1
68	1
70	1

QUESTION 1.3

menuDescription
Hamburger and Chips
Cheese Burger and Chips
Bacon Burger and Chips
Veggie Burger and Chips
Chips - Large Plate
Bacon and Egg Burger with Chips
Giant Burger with Chips

QUESTION 1.4

menuDescription	profit
Cola	4.46
Fresh Juice - Orange	8.23
Fresh Juice - Apple	7.73
Homemade Lemonade	10.77
Water - Sparkling (500ml)	7.00
Water - Still (500ml)	7.00

QUESTION 1.5

(No output)

QUESTION 1.6

menuDescription	MarkUp
Homemade Lemonade	207.915058
Fresh Juice - Orange	195.023697
Chips - Large Plate	182.075472
Water - Still (500ml)	177.215190
Water - Sparkling (500ml)	177.215190
Salad - Pasta	172.298006
Fresh Juice - Apple	163.771186
Salad - Potato	111.147274
Salad - Chicken	109.133217
Cola	98.454746
Salad - Greek	83.912119
Cheese Burger	72.167488
Cheese Burger and Chips	61.610032
Veggie Burger	61.455526
Giant Burger with Chips	60.508701
Bacon Burger	58.468862
Bacon Burger and Chips	53.099455
Hamburger and Chips	50.646552
Bacon and Egg Burger with Chips	50.451807
Hamburger	50.050100
Veggie Burger and Chips	48.223122
Giant Burger	26.658803

QUESTION 1.7

menuDescription	Quantity
Bacon and Egg Burger with Chips	27
Bacon Burger	31
Bacon Burger and Chips	47
Cheese Burger	26
Cheese Burger and Chips	27
Chips - Large Plate	31
Cola	19
Fresh Juice - Apple	21
Fresh Juice - Orange	26
Giant Burger	25

menuDescription	Quantity
Giant Burger with Chips	33
Hamburger	24
Hamburger and Chips	32
Homemade Lemonade	46
Salad - Chicken	37
Salad - Greek	23
Salad - Pasta	28
Salad - Potato	25
Veggie Burger	14
Veggie Burger and Chips	38
Water - Sparkling (500ml)	38
Water - Still (500ml)	24

QUESTION 1.8

waiterName	tablesServed	amountPaid
Erin	5	372
Esme	2	260
Hayden	9	303.33333
Jacques	6	330
Jen	3	203.33333
Kwame	6	225
Matthew	6	396.66667
Morgan	3	146.66667
Nomfundo	7	362.85714
Ramola	3	190
Ruth	5	292
Simphiwe	2	210
Tendai	4	342.5
Vashni	8	295

QUESTION 1.9

waiterName	Prize
Erin	40
Hayden	60
Jacques	120
Jen	20
Kwame	20
Matthew	60
Nomfundo	90
Ramola	10
Ruth	30
Simphiwe	40
Tendai	50
Vashni	40

SECTION B OBJECT ORIENTED PROGRAMMING

FINAL OUTPUT:

```
STOCK LIST
Plates: 76.0 plates
Salt: 7.6 kg
Whole-wheat bread rolls: 15.0 rolls
Knives: 105.0 knives
Tomatoes: 5.3 kg
Forks: 78.0 forks
White bread rolls: 52.0 rolls
Patties - Beef: 143.0 patties
Mayonnaise: 7.0 litres
Tomato Sauce: 15.0 litres
Lettuce: 2.0 heads
Chilli Sauce: 3.0 litres
Eggs: 3.0 dozen
Bacon: 0.5 kg
Spoons: 98.0 spoons
Avocado: 0.0 avocados
Mushrooms: 0.7 kilograms
Pineapples: 17.0 pineapples
Cheese: 1.3 kg
Salad dressing (French): 2.0 bottles
Salad dressing (Italian): 6.0 bottles
Onions: 17.0 onions
Patties - Vegetable: 23.0 patties
Pickles: 3.0 bottles
Olives: 4.0 bottles
Fresh orange juice: 6.7 litres
Fresh apple juice: 5.4 litres
Cola: 58.0 cans
Lemonade: 66.0 cans
Sparkling Apple: 40.0 cans
Patties - Chicken: 122.0 patties

ORDER LIST
Whole-wheat bread rolls: 33.0 rolls
Lettuce: 18.0 heads
Bacon: 4.5 kg
Avocado: 25.0 avocados
Cheese: 8.7 kg
Salad dressing (French): 6.0 bottles
Onions: 13.0 onions

PERFORMING STOCK TAKE...
STOCK LIST
Plates: 88.0 plates
Salt: 6.1 kg
Whole-wheat bread rolls: 51.0 rolls
Knives: 115.0 knives
Tomatoes: 5.3 kg
Forks: 78.0 forks
White bread rolls: 52.0 rolls
Patties - Beef: 23.0 patties
Mayonnaise: 7.0 litres
Tomato Sauce: 10.5 litres
Lettuce: 2.0 heads
Chilli Sauce: 3.0 litres
Eggs: 3.0 dozen
Bacon: 8.0 kg
Spoons: 98.0 spoons
```

```
Avocado: 12.0 avocados
Mushrooms: 15.45 kilograms
Pineapples: 15.0 pineapples
Cheese: 3.8 kg
Salad dressing (French): 2.0 bottles
Salad dressing (Italian): 6.0 bottles
Onions: 41.0 onions
Patties - Vegetable: 3.0 patties
Pickles: 1.0 bottles
Olives: 7.0 bottles
Fresh orange juice: 4.5 litres
Fresh apple juice: 1.75 litres
Cola: 106.0 cans
Lemonade: 17.0 cans
Sparkling Apple: 40.0 cans
Patties - Chicken: 288.0 patties

ORDER LIST
Patties - Beef: 277.0 patties
Lettuce: 18.0 heads
Salad dressing (French): 6.0 bottles
Patties - Vegetable: 33.0 patties
Fresh orange juice: 15.5 litres
Fresh apple juice: 18.25 litres
Lemonade: 103.0 cans
```

80 marks

Total: 120 marks