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 mark sheets 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' mark sheets 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: ______________________________
### IT PRACTICAL EXAMINATION 2012

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

<table>
<thead>
<tr>
<th>Candidate:</th>
<th>Marker:</th>
<th>Checksum: 0000-0000-0120</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>Total Pupil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 1.1
SELECT * FROM tblWaiters ORDER BY waiterName;

#### 1.2
SELECT tableID, tableGuests FROM tblTables WHERE tblGuests = 1 OR tblGuests >= 10;

#### 1.3
SELECT menuDescription FROM tblMenuItems WHERE menuDescription LIKE '*chips*';

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

#### 1.5
INSERT INTO tblWaiters (waiterName, waiterPhone) VALUES ('Busi', '083 469 9000');

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

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

#### 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;

#### 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;

#### 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;

#### 3.1
Class header is correct with extend

#### 3.2
Properties: both private, both double, both named appropriately

#### 3.3
Constructor: header is correct (-1 per error to a max of 2), calls parent constructor, assignments are correct.

#### 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.
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 Language

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

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

    public StockItem (String description, double quantity, String units) {
        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)
    }
}
```
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 ();
    }
}

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 ")")
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
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
```java
for (int i = 0; i < stockcount; i++)
{
    if (inventory[i].getDescription().equalsIgnoreCase(description))
    {
        return inventory[i];
    }
}
return null;
```

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

        String line = br.readLine();
        while (line != null)
        {
            String[] bits = line.split(":");
            String item = bits[0].trim();
            String change = bits[1].trim();
            StockItem temp = findStockItem(item);
            if (temp != null)
            {
                if (change.length() >= 5 && change.substring(0, 5).equalsIgnoreCase("used "))
                {
                    String amount = change.substring(5);
                    double amt = Double.parseDouble(amount);
                    temp.changeQuantity(-amt);
                }
                else if (change.length() >= 7 && change.substring(0, 7).equalsIgnoreCase("bought "))
                {
                    String amount = change.substring(7);
                    double amt = Double.parseDouble(amount);
                    temp.changeQuantity(amt);
                }
                else
                {
                    double amt = Double.parseDouble(change);
                    temp.setQuantity(amt);
                }
            }
            line = br.readLine();
        }
    }
    catch (IOException ioe)
    {
        System.out.println(ioe);
    }
}
```
QUESTIONS 5 AND 6.2

// Question 5.1
public class RestaurantDriver {
    public static void main (String[] args) {
        // Question 5.2
        StockManager sm = new StockManager ();
        // Question 5.3
        System.out.println ("STOCK LIST");
        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");
        System.out.println (sm.getStockList ());
        System.out.println ("ORDER LIST");
        System.out.println (sm.getOrderingList ());
    }
}
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]
QUESTION 4 AND 6.1

unit uStockManager;

interface

uses uStockItem, uStockItemFood, classes, SysUtils, Dialogs;

// Question 4.1
type TStockManager = class (class definition)
  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));
      end;
    end;
  end;
end;
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;

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

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

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

  // 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.
## OUTPUT

### SECTION A  STRUCTURED QUERY LANGUAGE

#### QUESTION 1.1

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

#### QUESTION 1.2

<table>
<thead>
<tr>
<th>tableID</th>
<th>tableGuests</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td>54</td>
<td>1</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
</tr>
<tr>
<td>58</td>
<td>1</td>
</tr>
<tr>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
</tr>
</tbody>
</table>

#### QUESTION 1.3

<table>
<thead>
<tr>
<th>menuDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamburger and Chips</td>
</tr>
<tr>
<td>Cheese Burger and Chips</td>
</tr>
<tr>
<td>Bacon Burger and Chips</td>
</tr>
<tr>
<td>Veggie Burger and Chips</td>
</tr>
<tr>
<td>Chips - Large Plate</td>
</tr>
<tr>
<td>Bacon and Egg Burger with Chips</td>
</tr>
<tr>
<td>Giant Burger with Chips</td>
</tr>
</tbody>
</table>
QUESTION 1.4

<table>
<thead>
<tr>
<th>menuDescription</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cola</td>
<td>4.46</td>
</tr>
<tr>
<td>Fresh Juice - Orange</td>
<td>8.23</td>
</tr>
<tr>
<td>Fresh Juice - Apple</td>
<td>7.73</td>
</tr>
<tr>
<td>Homemade Lemonade</td>
<td>10.77</td>
</tr>
<tr>
<td>Water - Sparkling (500ml)</td>
<td>7.00</td>
</tr>
<tr>
<td>Water - Still (500ml)</td>
<td>7.00</td>
</tr>
</tbody>
</table>

QUESTION 1.5

(No output)

QUESTION 1.6

<table>
<thead>
<tr>
<th>menuDescription</th>
<th>MarkUp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homemade Lemonade</td>
<td>207.915058</td>
</tr>
<tr>
<td>Fresh Juice - Orange</td>
<td>195.023697</td>
</tr>
<tr>
<td>Chips - Large Plate</td>
<td>182.075472</td>
</tr>
<tr>
<td>Water - Still (500ml)</td>
<td>177.215190</td>
</tr>
<tr>
<td>Water - Sparkling (500ml)</td>
<td>177.215190</td>
</tr>
<tr>
<td>Salad - Pasta</td>
<td>172.298006</td>
</tr>
<tr>
<td>Fresh Juice - Apple</td>
<td>163.771186</td>
</tr>
<tr>
<td>Salad - Potato</td>
<td>111.147274</td>
</tr>
<tr>
<td>Salad - Chicken</td>
<td>109.133217</td>
</tr>
<tr>
<td>Cola</td>
<td>98.454746</td>
</tr>
<tr>
<td>Salad - Greek</td>
<td>83.912119</td>
</tr>
<tr>
<td>Cheese Burger</td>
<td>72.167488</td>
</tr>
<tr>
<td>Cheese Burger and Chips</td>
<td>61.610032</td>
</tr>
<tr>
<td>Veggie Burger</td>
<td>61.455526</td>
</tr>
<tr>
<td>Giant Burger with Chips</td>
<td>60.506701</td>
</tr>
<tr>
<td>Bacon Burger</td>
<td>58.468862</td>
</tr>
<tr>
<td>Bacon Burger and Chips</td>
<td>53.099455</td>
</tr>
<tr>
<td>Hamburger and Chips</td>
<td>50.646552</td>
</tr>
<tr>
<td>Bacon and Egg Burger with Chips</td>
<td>50.451807</td>
</tr>
<tr>
<td>Hamburger</td>
<td>50.050100</td>
</tr>
<tr>
<td>Veggie Burger and Chips</td>
<td>48.223122</td>
</tr>
<tr>
<td>Giant Burger</td>
<td>26.658803</td>
</tr>
</tbody>
</table>

QUESTION 1.7

<table>
<thead>
<tr>
<th>menuDescription</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bacon and Egg Burger with Chips</td>
<td>27</td>
</tr>
<tr>
<td>Bacon Burger</td>
<td>31</td>
</tr>
<tr>
<td>Bacon Burger and Chips</td>
<td>47</td>
</tr>
<tr>
<td>Cheese Burger</td>
<td>26</td>
</tr>
<tr>
<td>Cheese Burger and Chips</td>
<td>27</td>
</tr>
<tr>
<td>Chips - Large Plate</td>
<td>31</td>
</tr>
<tr>
<td>Cola</td>
<td>19</td>
</tr>
<tr>
<td>Fresh Juice - Apple</td>
<td>21</td>
</tr>
<tr>
<td>Fresh Juice - Orange</td>
<td>26</td>
</tr>
<tr>
<td>Giant Burger</td>
<td>25</td>
</tr>
<tr>
<td>menuDescription</td>
<td>Quantity</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Giant Burger with Chips</td>
<td>33</td>
</tr>
<tr>
<td>Hamburger</td>
<td>24</td>
</tr>
<tr>
<td>Hamburger and Chips</td>
<td>32</td>
</tr>
<tr>
<td>Homemade Lemonade</td>
<td>46</td>
</tr>
<tr>
<td>Salad - Chicken</td>
<td>37</td>
</tr>
<tr>
<td>Salad - Greek</td>
<td>23</td>
</tr>
<tr>
<td>Salad - Pasta</td>
<td>28</td>
</tr>
<tr>
<td>Salad - Potato</td>
<td>25</td>
</tr>
<tr>
<td>Veggie Burger</td>
<td>14</td>
</tr>
<tr>
<td>Veggie Burger and Chips</td>
<td>38</td>
</tr>
<tr>
<td>Water - Sparkling (500ml)</td>
<td>38</td>
</tr>
<tr>
<td>Water - Still (500ml)</td>
<td>24</td>
</tr>
</tbody>
</table>

**QUESTION 1.8**

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

**QUESTION 1.9**

<table>
<thead>
<tr>
<th>waiterName</th>
<th>Prize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erin</td>
<td>40</td>
</tr>
<tr>
<td>Hayden</td>
<td>60</td>
</tr>
<tr>
<td>Jacques</td>
<td>120</td>
</tr>
<tr>
<td>Jen</td>
<td>20</td>
</tr>
<tr>
<td>Kwame</td>
<td>20</td>
</tr>
<tr>
<td>Matthew</td>
<td>60</td>
</tr>
<tr>
<td>Nomfundo</td>
<td>90</td>
</tr>
<tr>
<td>Ramola</td>
<td>10</td>
</tr>
<tr>
<td>Ruth</td>
<td>30</td>
</tr>
<tr>
<td>Simphiwe</td>
<td>40</td>
</tr>
<tr>
<td>Tendai</td>
<td>50</td>
</tr>
<tr>
<td>Vashni</td>
<td>40</td>
</tr>
</tbody>
</table>
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