# CPSC1012 – Extra Practice Problems

Sequence

The number of floor tiles required for a rectangular room is calculated by dividing the area of the room by the area required for each floor tile plus the required grout lines. Develop a program that will display the number of boxes of 30cm x 30cm floor tiles, at 10 tiles per box (plus one box for scrap and damaged tiles) given the input of the length and width of the room. The width of the grout line between each tile, and between the tile and each of the walls, is 4mm (see the diagram below).
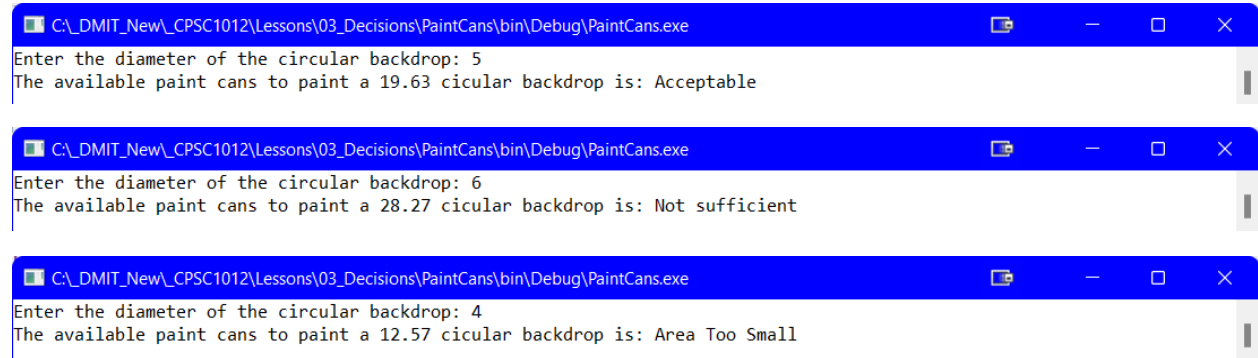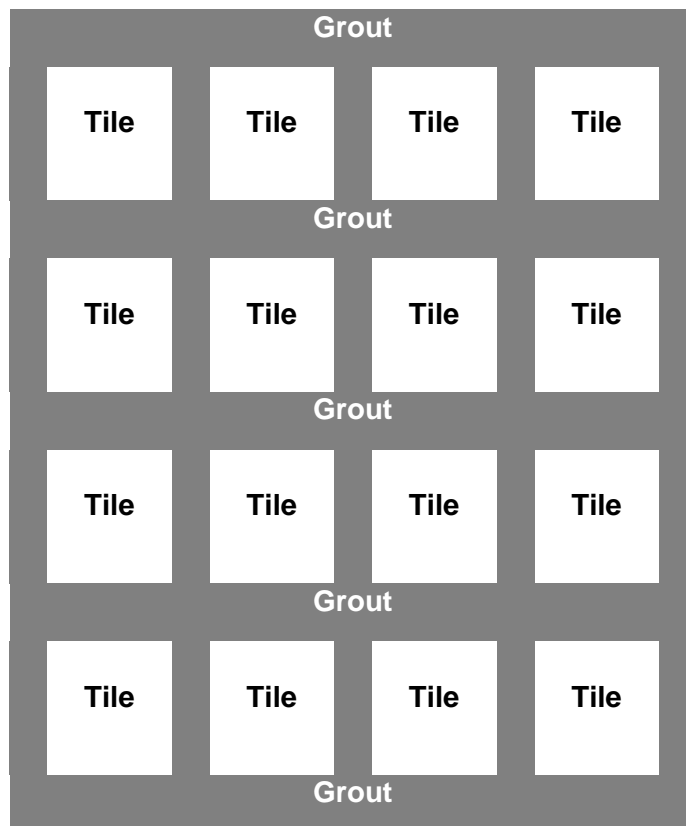


```
Enter width of the room (in metres): 3
Enter length of the room (in metres): 5
Room area = 15 sq m, # of Tiles = 162, therefore # of boxes = 18
```

Decisions

Create a program that determine the paint will be sufficient to paint a circular backdrop. There are two 4.5 L paint cans which will cover 27 m^2 of which you must use at least 1 ¾ cans of paint. The user will enter the diameter of the backdrop. The program will return a message whether the amount of paint is either "Not Sufficient", "Area Too Small", or "Acceptable".

Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\03_Decisions\PaintCans\bin\Debug\PaintCans.exe
Enter the diameter of the circular backdrop: 5
The available paint cans to paint a 19.63 cicular backdrop is: Acceptable
```

```
C:\_DMIT_New\_CPSC1012\Lessons\03_Decisions\PaintCans\bin\Debug\PaintCans.exe
Enter the diameter of the circular backdrop: 6
The available paint cans to paint a 28.27 cicular backdrop is: Not sufficient
```
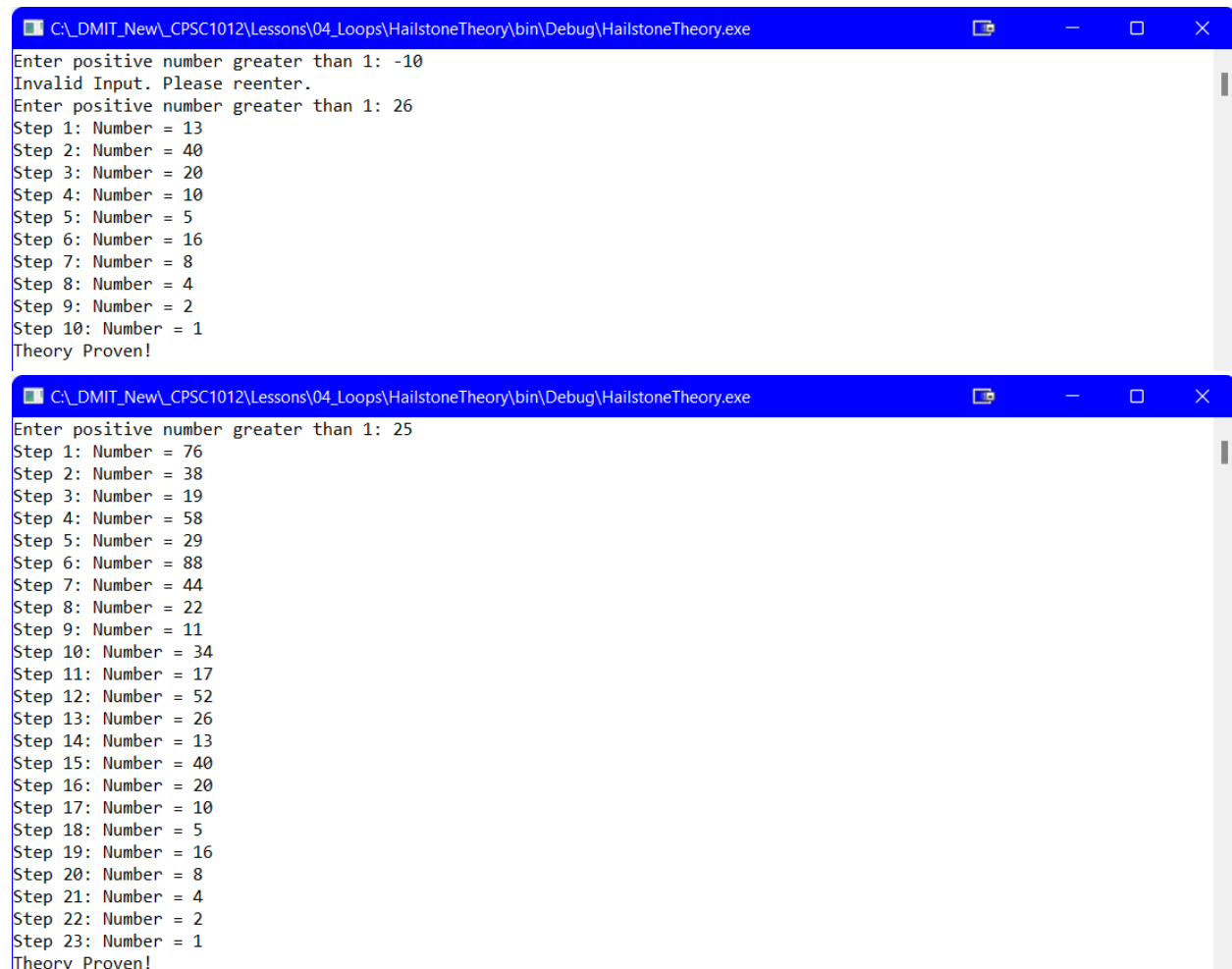
```
C:\_DMIT_New\_CPSC1012\Lessons\03_Decisions\PaintCans\bin\Debug\PaintCans.exe
Enter the diameter of the circular backdrop: 4
The available paint cans to paint a 12.57 cicular backdrop is: Area Too Small
```

Loops 1

Develop the logic for a program that will show the "Hailstone" theory. Allow the user to enter in a positive integer number greater than one and display all the numbers calculated using the algorithm described below. Stop when the calculated number is 1. Ensure that the user enters a valid number before allowing processing to continue. When the user enters a valid number perform the logic to reduce the number to 1 and finally display the message: "Theory Proven!"

The "Hailstone" theory states that any positive integer number will resolve to 1 if we treat this number in the following fashion: if the number is even, we divide the number by 2; if the number is odd, we multiply the number by 3 and increment the result by 1. We have no way of knowing the number of calculations required to reduce the number to 1, but the theory states that the number will always become 1.

The completed programs execution should match the following sample output:

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\HailstoneTheory\bin\Debug\HailstoneTheory.exe      —  □  ✕
Enter positive number greater than 1: -10
Invalid Input. Please reenter.
Enter positive number greater than 1: 26
Step 1: Number = 13
Step 2: Number = 40
Step 3: Number = 20
Step 4: Number = 10
Step 5: Number = 5
Step 6: Number = 16
Step 7: Number = 8
Step 8: Number = 4
Step 9: Number = 2
Step 10: Number = 1
Theory Proven!
```

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\HailstoneTheory\bin\Debug\HailstoneTheory.exe      —  □  ✕
Enter positive number greater than 1: 25
Step 1: Number = 76
Step 2: Number = 38
Step 3: Number = 19
Step 4: Number = 58
Step 5: Number = 29
Step 6: Number = 88
Step 7: Number = 44
Step 8: Number = 22
Step 9: Number = 11
Step 10: Number = 34
Step 11: Number = 17
Step 12: Number = 52
Step 13: Number = 26
Step 14: Number = 13
Step 15: Number = 40
Step 16: Number = 20
Step 17: Number = 10
Step 18: Number = 5
Step 19: Number = 16
Step 20: Number = 8
Step 21: Number = 4
Step 22: Number = 2
Step 23: Number = 1
Theory Proven!
```

Hint: Use the Modulus Operation

Loops 2

Create a program that will display a diamond shape based on the size inputted by the user. The user will input an odd number between 5 and 55 inclusive; the only validation is to make sure the inputted number meets that criterion (HINT: use a do loop). For example, if the user inputs the value of 5, the following will be displayed (HINT: use nested loops):

```
     +

   +   +

 +       +

   +   +

     +
```

Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\OpenDiamond\bin\Debug\OpenDiamond.exe
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 4
Not an odd number between 5 and 55 inclusive ... try again
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 16
Not an odd number between 5 and 55 inclusive ... try again
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 5
  +
 + +
+   +
 + +
  +
```

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\OpenDiamond\bin\Debug\OpenDiamond.exe
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 15
       +
      + +
     +   +
    +     +
   +       +
  +         +
 +           +
+             +
 +           +
  +         +
   +       +
    +     +
     +   +
      + +
       +
```

Loops 3

Create a copy of the solution from **Loops 2** and modify it to display a filled diamond.

Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\FilledDiamond\bin\Debug\FilledDiamond.exe
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 5
  +
 +++
+++++
 +++
  +
```

```
C:\_DMIT_New\_CPSC1012\Lessons\04_Loops\FilledDiamond\bin\Debug\FilledDiamond.exe
Enter the size of the diamond as an odd number between 5 and 55 inclusive: 15
       +
      +++
     +++++
    +++++++
   +++++++++
  +++++++++++
 +++++++++++++
+++++++++++++++
 +++++++++++++
  +++++++++++
   +++++++++
    +++++++
     +++++
      +++
       +
```
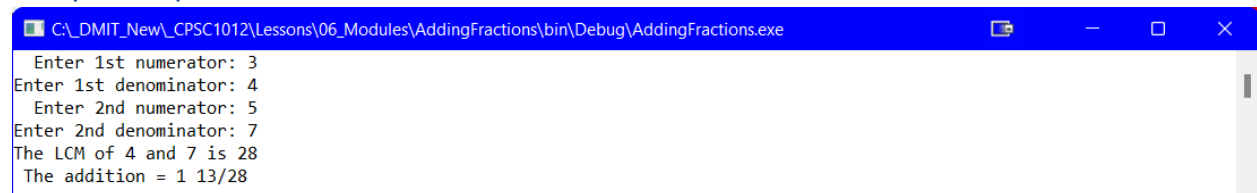
Methods 1

Program to demonstrate how to add two fractions. The program will use the following methods:

- `static int GetGreatestCommonDemoninator(int denominator1, int denominator2)`
  returns the GCD of the two numbers in a loop while denominator1 != denominator2
  denominator1 > denominator2 ==> denominator1 = denominator1 - denominator2
  denominator2 > denominator1 ==> denominator2 = denominator2 - denominator1
- `static int GetLeastCommonMulitple(int denominator1, int denominator2)`
  uses the method above in the formula below:
  (denominator1 * denominator2) / GetGreatestCommonDemoninator(denominator1, denominator2)
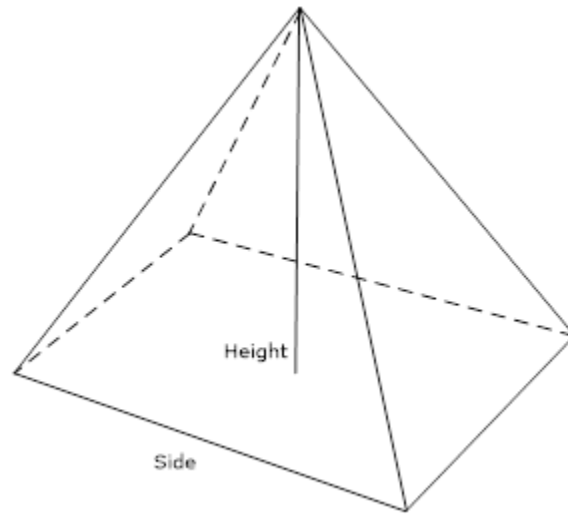
Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\06_Modules\AddingFractions\bin\Debug\AddingFractions.exe                    ☐    —    ☐    ✕
  Enter 1st numerator: 3
Enter 1st denominator: 4
  Enter 2nd numerator: 5
Enter 2nd denominator: 7
The LCM of 4 and 7 is 28
 The addition = 1 13/28
```

Methods 2

Create a program that will calculate the surface area and volume of a square pyramid:

The formulas are:

$$Volume = side^2 \times \frac{height}{3}$$

$$Surface\ Area = side^2 + 2 \times side \times \sqrt{\frac{side^2}{4} + height^2}$$

Use the following methods in your solution:

```
static double GetSafeDouble(string prompt)

static double CalculateSurfaceArea(double side, double height)

static double CalculateVolume(double side, double height)
```

Display your final answers rounded to 4 decimal places.

Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\06_Modules\SquarePyramid\bin\Debug\SquarePyramid.exe
Enter the base side of the pyramid: 5
  Enter the height of the pyramid: 6
Volume = 50.0000, Surface Area = 90.0000
```

```
Select C:\_DMIT_New\_CPSC1012\Lessons\06_Modules\SquarePyramid\bin\Debug\SquarePyramid.exe
Enter the base side of the pyramid: 4.5
  Enter the height of the pyramid: 7.8
Volume = 52.6500, Surface Area = 93.3123
```

Methods 3

Below is a floor layout of a community hall. It has a large central fireplace and four circular columns that support the vaulted ceiling. A program was developed to determine the amount of flooring required to finish the floor. All areas should be calculated to three decimal places.
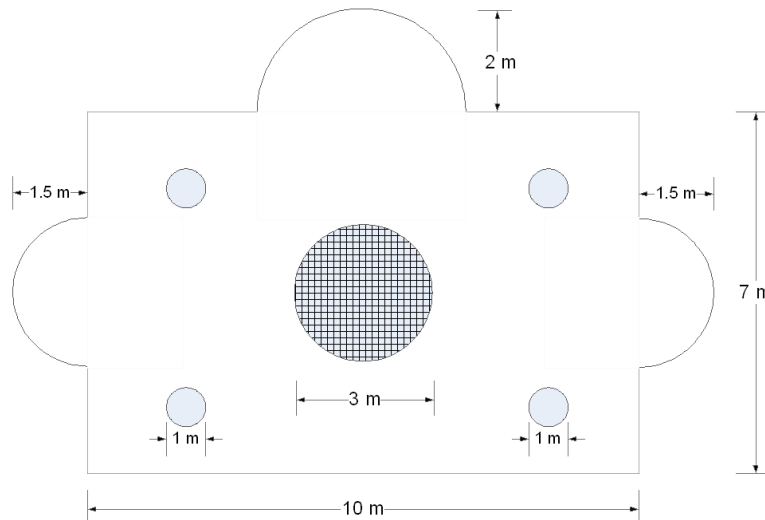
Must use the following methods:

static double GetSafeDouble(string prompt)

static double CircleArea(double radius)

       Remember to use **Math.PI** in the method above.

Sample output:

```
C:\_DMIT_New\_CPSC1012\Lessons\06_Modules\FloorPlan\bin\Debug\FloorPlan.exe
        Enter the length of the room: 10
        Enter the width of the room: 7
  Enter the radius of 1st alcove or 0: 1.5
  Enter the radius of 2nd alcove or 0: 2
  Enter the radius of 3rd alcove or 0: 1.5
    Enter the diameter of the pillars: 1
   Enter the diameter of the fireplace: 3
Floor area: 73.143
```

List<T> with File I/O

A variety store wholesale business requires an inventory system that will update its inventory when orders are filled, and bills are produced. To make better decisions regarding which products should be promoted and which products should possibly be dropped from the inventory, it would also like the system to produce an inventory report when requested indicating the quantity sold of each item in stock.

The inventory is loaded from a file and stored into a `List<Product>`. A `Product` has the following information:

- `ProductNumber (string)`
- `Description (string)`
- `Price (double)`
- `QuantityOnHand (int)`
- `QuantitySold (int)` --> initially this is set to zero

The Customer order information is loaded from a file and stored in a `List<CustomerOrder>`. A `CustomerOrder` has the following information:

- `CustomerName (string)`
- `ItemNumber (string)`
- `QuantityOrdered (int)`

Create the `Product` and `CustomerOrder` classes using the information above.

Create the following methods:

- `static void LoadProdcut(List<Product> products)` --> read from file and store in `List<Product>`
- `static void LoadCustomerOrder(List<CustomerOrder> customerOrders)` --> read from file and store in `List<CustomerOrder>`
- `static int LocateProduct(List<Product> products, string productNumber)` --> returns the location in `List<Product>` or `-1` if not found
- `static void UpdateProduct(List<Product> products, List<CustomerOrder> customerOrders)` --> QuantityOnHand reduced & QuantitySold updated by QuantityOrdered
- `static void DisplayReport(List<Product> products)` --> displays the `List<Product>` after all the orders are processed.
- `static void SaveReport(List<Product> products)` --> write data to a new file named `InventoryReport.csv`
- Any other method that makes your program better

Business Rules:

1. An order cannot be filled if the `QuantityOnHand` is zero
2. An order can be partially filled, i.e., if the `QuantityOnHand` is not zero and less than the `QuantityOrdered`

Sample Data

ProducList.csv	CustomerOrder.csv
A1100-A,Zip Ties Small,1.5,100,0 A1100-B,Zip Ties Medium,2.5,200,0 A1100-C,Zip Ties Large,4.25,150,0 B1001-A,Duct Tape Green,5.25,75,0 B1001-B,Duct Tape Silver,5.25,120,0 C1250-1,Teflon Tape,1.75,50,0 D1250-2,Electrical Tape,2.25,50,0 F2501-C,Finishing Nails,5.45,125,0 F2502-C,#8 Wood Screws,3.75,25,0 F2502-D,#10 Wood Screws,3.99,20,0 G4501-1,Stove Bolt,2.25,10,0 E1000-1,Ball Pean Hammer,7.50,12,0 E1001-1,Claw Hammer,7.75,10,0 E1002-1,Finishing Hammer,6.50,10,0 AB-1001,2x4 8ft Pine,12.50,20,0 AB-1002,2x4 10ft Pine,15.75,10,0 AB-1003,2x4 12ft Pine,18.50,5,0 AC-1001,4x8 Plywood Std,27.75,25,0 AC-1002,4x8 Plywood G1S,37.50,45,0 AC-1003,4x8 Plywood G2S,55.25,20,0 AD-1001,4x4 8ft Fence Post,7.75,45,0 AD-1002,4x4 10Ft Fence Post,9.95,30,0 AD-1004,Cedar Fence Board 6ft,5.85,100,0 AD-1004,Composite Fence Board 6ft,12.95,30,0 ZZ-9999,Specialty Ball Cap,2.25,5,0	Ziggy Stardust,E1001-1,5 Clarke Kent,A1100-C,50 Max Planck,AD-1004,10 Fred Flinstone,E1001-1,1 Barney Rubble,B1001-B,1 Obi-Wan Kenobi,ZZ-9999,1 Billy Bud,ZZ-9999,1 Mark Farner,B1001-A,2 George Reede,B1001-A,3 Francis Feldman,ZZ-9999,3 Zack Smith,A1100-C,20 Susan Dey,AD-1001,10 Susan Dey,AD-1004,50 Susan Dey,G4501-1,100 Gordon Jump,C1250-1,5 Jimmy Smalls,D1250-2,5 Betty Rubble,F2502-D,25 Wilma Flinstone,E1001-1,2 Frank Warner,A1100-A,50 Anne Franks,B1001-A,2 Grace Slick,F2502-C,25 Grace Slick,F2501-C,20 George Plimpton,AD-1002,6 George Plimpton,AD-1004,30 Cat Stevens,E1000-1,2 Ziggy Stardust,G4501-1,10 Anon Amous,A1100-C,10 Bill Board,AC-1001,6 Becky Board,AC-1003,4 George Jetson,B1001-B,5 Yosemite Sam,E1002-1,1 Alice Wonder,C1250-1,3 Adam West,E1001-1,4 Burt Ward,E1002-1,4 Albert Flasher,F2502-C,50 Albert Flasher,F2502-D,20 Cam Frost,ZZ-9999,1 Jack Adamson,E1002-1,1 Mable Jeffries,E1000-1,5 Dave Woods,AB-1001,30

Sample Output

```
C:\_DMIT_New\_CPSC1012\Lessons\08_Advanced_File_IO\ProductInventoryReport\bin\Debug\ProductInventoryReport.exe

Products loaded successfully ...
Customer Orders loaded successfully ...
Product Number   Description            Price    QOH  Sold
A1100-A          Zip Ties Small         $1.50     50    50
A1100-B          Zip Ties Medium        $2.50    200     0
A1100-C          Zip Ties Large         $4.25     70    80
B1001-A          Duct Tape Green        $5.25     68     7
B1001-B          Duct Tape Silver       $5.25    114     6
C1250-1          Teflon Tape            $1.75     42     8
D1250-2          Electrical Tape        $2.25     45     5
F2501-C          Finishing Nails        $5.45    105    20
F2502-C          #8 Wood Screws         $3.75      0    25
F2502-D          #10 Wood Screws        $3.99      0    20
G4501-1          Stove Bolt             $2.25      0    10
E1000-1          Ball Pean Hammer       $7.50      5     7
E1001-1          Claw Hammer            $7.75      0    10
E1002-1          Finishing Hammer       $6.50      4     6
AB-1001          2x4 8ft Pine          $12.50      0    20
AB-1002          2x4 10ft Pine         $15.75     10     0
AB-1003          2x4 12ft Pine         $18.50      5     0
AC-1001          4x8 Plywood Std       $27.75     19     6
AC-1002          4x8 Plywood G1S       $37.50     45     0
AC-1003          4x8 Plywood G2S       $55.25     16     4
AD-1001          4x4 8ft Fence Post     $7.75     35    10
AD-1002          4x4 10Ft Fence Post    $9.95     24     6
AD-1004          Cedar Fence Board 6ft  $5.85     10    90
AD-1004          Composite Fence Board 6ft $12.95  30    0
ZZ-9999          Specialty Ball Cap     $2.25      0     5
InventoryReport.csv created successfully ...
```