1 Advanced — Reducing Fractions

Write a program that reads the numerator and denominator of a fraction and displays this fraction reduced to lowest terms. You may assume that both the numerator and the denominator are positive integers less than 10,000. Don't worry about formatting your output — just make sure it is clear what the numerator and denominator are.

Example 1:

```
Enter numerator: 9
Enter denominator: 15

3/5
```

Example 2:

```
Enter numerator: 9
Enter denominator: 3

3/1
```

Example 3:

```
Enter numerator: 24
Enter denominator: 60

2/5
```

```
/* Advanced 1 - Reducing Fractions
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Fractions.Advanced
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter numerator: ");
            int num = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter denominator: ");
            int denom = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine();
            for (int divisor = Math.Min(num, denom); divisor >= 1; divisor--)
            {
                if (num % divisor == 0 && denom % divisor == 0)
                {
                    Console.WriteLine((num / divisor).ToString() + "/"
                        + (denom / divisor).ToString());
                    Console.ReadLine();
                    return;
                }
            }
        }
    }
}
```

1 Advanced — Reducing Fractions
Test Cases

Test Case 1:

```
Enter numerator: 1234
Enter denominator: 5678

617/2839
```

Test Case 2:

```
Enter numerator: 9999
Enter denominator: 9990

1111/1110
```

Test Case 3:

```
Enter numerator: 9999
Enter denominator: 1111

9/1
```

Second Submission: Do the above tests, plus the following:

Test Case 4:

```
Enter numerator: 1000
Enter denominator: 9000

1/9
```

Test Case 5:

```
Enter numerator: 109
Enter denominator: 127

109/127
```

2 Advanced Float your Boat

A boat floats because it displaces water. The weight of the water displaced by a boat is equal to the weight of the boat and the passengers.

Your task is to determine the minimum size of a rectangular boat in whole inches given the weight of the occupants. All width, depth, and length must be integer values of inches. Assume the width of the boat is one third of the length, the depth of the boat is one half of the width. One foot of the depth must be above the water line. Also assume that the weight of the boat is 100 lbs independent of size.

Prompt for the total weight in pounds of the prospective occupants and output the height, width, length in inches.

The weight of water per cubic foot at 70 degrees Fahrenheit is 62.3 lbs.

Sample 1:

total occupants weight: 300 Output: 16 32 96

Sample 2:

total occupants weight: 500 Output: 17 34 102

2A Float your Boat Sample Code

```
#define WaterWeight 62.30

int _tmain(int argc, _TCHAR* argv[])
{
	double weight, volume;
	int depth, width, length;
	cout<<"\nenter total weight of passengers: ";
	cin>>weight;
	//weight of boat needs to be added to occupant weight
	weight = weight + 100;
	//volume of water that needs to be displaced
	volume = (weight/WaterWeight)*12*12*12;
	int i;
	for(i = 13; i < 200; i++){
		if( (i-12)*(2*i)*(6*i) >= volume) {depth = i;break;}
	}

	cout<< "Dimensions are " << depth << " " <<2*depth << " " << 6* depth ;

	return 0;
}
```

2A float your boat Test Cases

First testing: Do the test cases listed below. The rounding off to the first decimal place is fine. Formatting is not important.

Test case	Inputs	Outputs		
1	Total occupants weight: 150	depth: 15	width: 30	length: 90
2	Total occupants weight: 550	depth: 18	width: 36	length: 108
3	Total occupants weight: 1000	depth: 20	width: 40	length: 120

Second testing: Repeat the test cases above and do the additional ones below

Test case	Inputs	Outputs		
4	Total occupants weight: 210	depth: 16	width: 32	length: 96
5	Total occupants weight: 300	depth: 16	width: 32	length: 96

3 Advanced — Number Spiral

We can assign all of the ordered pairs of natural numbers a unique natural number by placing the natural numbers on the x-y coordinate system as follows. We first place 0 at location $(0, 0)$. We then place 1 above this location at $(0, 1)$, and continue around $(0, 0)$ clockwise, placing 2 at $(1, 1)$, 3 at $(1, 0)$, etc., until we place 8 at $(-1, 1)$. We then go to $(0, 2)$ and place 9, and continue spiraling around like this. A portion of the coordinate system looks like this:

```
46 47 48 25 26 27 28
45 23 24  9 10 11 29
44 22  8  1  2 12 30
43 21  7  0  3 13 31
42 20  6  5  4 14 32
41 19 18 17 16 15 33
40 39 38 37 36 35 34
```

Write a program that takes a nonnegative integer n as input and displays the x and y coordinates of n in this spiral. You may assume that n is no more than 1,000,000.

Example 1:

```
Enter n: 0
0 is at location (0, 0)
```

Example 2:

```
Enter n: 48
48 is at location (-1, 3)
```

```
/* 3 Advanced - Number Spiral
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Number.Spiral.Advanced
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter n: ");
            int n = Convert.ToInt32(Console.ReadLine());
            if (n == 0)
            {
                Console.WriteLine(n + " is at location (0, 0)");
            }
            else if (n == 1)
            {
                Console.WriteLine(n + " is at location (0, 1)");
            }
            else
            {
                int x = 0;
                int y = 1;
                for (int i = 1; i < n; i++)
                {
                    if (x == -1 && y > 0)
                    {
                        x++;
                        y++;
                    }
                    else if (y > 0 && x >= -y && x < y)
                    {
                        x++;
                    }
                    else if (x > 0 && y <= x && y > -x)
                    {
                        y--;
                    }
                    else if (y < 0 && x <= -y && x > y)
                    {
                        x--;
                    }
                    else if (x < 0 && y >= x && y < -x)
                    {
                        y++;
                    }
                }
                Console.WriteLine(n + " is at location (" + x + ", " + y + ")");
            }
            Console.ReadLine();
        }
    }
}
```

3 Advanced — Number Spiral
Test Cases

Test Case 1:

```
Enter n: 0
0 is at location (0, 0)
```

Test Case 2:

```
Enter n: 1
1 is at location (0, 1)
```

Test Case 3:

```
Enter n: 1000000
1000000 is at location (1, -500)
```

Second Submission: Do the above tests, plus the following:

Test Case 4:

```
Enter n: 8
8 is at location (-1, 1)
```

Test Case 5:

```
Enter n: 234
234 is at location (8, 7)
```

4A Rabbits For Profit

Rabbits can produce 4-12 baby rabbits every 30 days. Assume an average of 6 baby rabbits, half female are born each month from each pair. Rabbits mature between 6 and 12 months after birth depending on breed. That is, the rabbits born at the end of month 1 will have babies at the end of month 7 if they mature at 6 months and at the end of month 13 if 12 months.

This problem is to simulate raising rabbits for profit. The initial inputs are the initial number of breeding pairs, the normal length before maturity, and the length of time in months for the simulation.

Additional inputs are the number of pairs of adult rabbits to be harvested at the end of each month and when that harvesting starts. If, at the end of the month, there are less adults than the number to be harvested, then only the adults will be harvested.

Assume no deaths except for harvesting during the specified time.

At the end of the simulation, display the number of breeding pairs and the number of juveniles.

Example 1

Initial pairs: 3 months to maturity: 7 number to harvest: 0 month to start: 10 length: 12

Output: number adults: 42 number juveniles: 234

Example 2

Initial pairs: 3 months to maturity: 8 number to harvest: 20 month to start: 12 length: 17

Output: number adults: 18 number juveniles: 990

4A Rabbits for Profit Solution

```
int length, harvest, start, maturity;
int babies[15];
int pairs, sum;
int newbabies;
int i,j;

cout<<"\nenter number of breeding pairs: ";
cin>>pairs;
cout<<"\nenter the number of months before breeding: ";
cin>> maturity;
cout<<"\nenter the number of pairs to harvest: ";
cin>>harvest;
cout<<"\nenter the month to start harvest: ";
cin>>start;
cout<<"\nenter number of months: ";
cin>>length;
for(j=0;j<15;j++) babies[j]=0;

for (i=0; i<length; i++) {
       newbabies = 3*pairs;
       if(i>=start) pairs = pairs - harvest;
       if(pairs < 0) pairs = 0;
       pairs = pairs + babies[maturity-1];
       sum = 0;
       for (j=maturity-1; j>0; j--) {babies[j] = babies[j-1]; sum=sum+babies[j]; }
       babies[0] = newbabies; sum=sum+babies[0];
       cout<< "\nAfter "<< i+1 <<" months: "<< 2*pairs
              <<"  adults  "<<2*sum <<"  babies "<< newbabies*2 << " newbabies";
}
```

4A Rabbits for Profit Test Cases

First testing: Do the test cases listed below. The results are all integer. Formatting is not important.

Test	Inputs	Outputs	
1	pairs: 3 maturity: 7 harvest: 0 start: 14 length: 14	adults: 132	juveniles: 1260
2	pairs: 2 maturity: 10 harvest: 6 start: 14 length: 12	adults: 42	juveniles: 234
3	pairs: 3 maturity: 8 harvest: 20 start: 12 length: 17	adults: 18	juveniles: 990

Second testing: Repeat the test cases above and do the additional ones below

Test case	Inputs	Outputs	
4	pairs: 2 maturity: 7 harvest: 6 start: 14 length: 4	adults: 4	juveniles: 48
5	pairs: 3 maturity: 8 harvest: 20 start: 12 length: 25	adults: 710	juveniles: 9108

5 Advanced — Green Lights

We are driving down a street with a series of traffic lights ahead. Because we know precisely the pattern of these traffic lights, we wish to compute whether we will pass each traffic light while it is either green or amber if we continue at our current speed. Write a program that takes as input a floating-point speed in miles per hour, and a positive integer giving the number of traffic lights. Then for each traffic light, it takes the following positive floating-point numbers:

- the distance to the light in miles;
- the time in seconds since it last turned from red to green;
- the time in seconds in each cycle that it is not red; and
- the time in seconds in each cycle that it is red.

These lights may be given in any order, but the first light given is labeled Light1, the second Light2, etc. If we will pass every light when it is not red, display a message to this effect. Otherwise, display the label of the first light we pass while it is red. Note that the first light we pass while it is red may not be the first light given in the input that we pass while it is red. You may assume that the total number of lights is no more than 5 and that the input will be such that you will not pass a light at exactly the moment it turns red.

Example 1:

```
Enter speed: 60
Enter the number of lights: 1
Light1:
  Enter distance: 1
  Enter time since last red: 20
  Enter length of time not red: 15
  Enter length of time red: 10
We will make all lights.
```

Example 2:

```
Enter speed: 37.5
Enter the number of lights: 3
Light1:
  Enter distance: 1.875
  Enter time since last red: 7.1
  Enter length of time not red: 6.9
  Enter length of time red: 3.1
Light2:
  Enter distance: 1.25
  Enter time since last red: 9.5
  Enter length of time not red: 8.5
  Enter length of time red: 11.5
Light3:
  Enter distance: 0.625
  Enter time since last red: 4.5
  Enter length of time not red: 9.5
  Enter length of time red: 5.5
We won't make Light2.
```

```
/* 5 Advanced - Green Lights
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Green.Light.Advanced
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter speed: ");
            double speed = Convert.ToDouble(Console.ReadLine());
            Console.Write("Enter the number of lights: ");
            int n = Convert.ToInt32(Console.ReadLine());
            int firstBad = 0;
            double badDistance = Double.PositiveInfinity;
            for (int i = 1; i <= n; i++)
            {
                Console.WriteLine("Light" + i + ":");
                Console.Write("  Enter distance: ");
                double distance = Convert.ToDouble(Console.ReadLine());
                Console.Write("  Enter time since last red: ");
                double start = Convert.ToDouble(Console.ReadLine()) / 3600;
                Console.Write("  Enter length of time not red: ");
                double notRed = Convert.ToDouble(Console.ReadLine()) / 3600;
                Console.Write("  Enter length of time red: ");
                double red = Convert.ToDouble(Console.ReadLine()) / 3600;
                if ((start + distance / speed) % (red + notRed) > notRed)
                {
                    if (distance < badDistance)
                    {
                        firstBad = i;
                        badDistance = distance;
                    }
                }
            }
            if (firstBad == 0)
            {
                Console.WriteLine("We will make all lights.");
            }
            else
            {
                Console.WriteLine("We won't make Light" + firstBad + ".");
            }
            Console.ReadLine();
        }
    }
}
```

5 Advanced — Green Lights
Test Cases

Test Case 1:

```
Enter speed: 22.5
Enter the number of lights: 1
Light1:
  Enter distance: 0.375
  Enter time since last red: 3.7
  Enter length of time not red: 8.6
  Enter length of time red: 6.4
We will make all lights.
```

Test Case 2:

```
Enter speed: 40
Enter the number of lights: 1
Light1:
  Enter distance: 1
  Enter time since last red: 1
  Enter length of time not red: 10
  Enter length of time red: 10
We won't make Light1.
```

Test Case 3:

```
Enter speed: 30
Enter the number of lights: 5
Light1:
  Enter distance: 1.5
  Enter time since last red: 7
  Enter length of time not red: 5
  Enter length of time red: 10
Light2:
  Enter distance: 1
  Enter time since last red: 3
  Enter length of time not red: 7
  Enter length of time red: 3
Light3:
  Enter distance: 0.75
  Enter time since last red: 5
  Enter length of time not red: 12
  Enter length of time red: 8
Light4:
  Enter distance: 0.5
  Enter time since last red: 4
  Enter length of time not red: 8
  Enter length of time red: 7
Light5:
  Enter distance: 1.25
  Enter time since last red: 15
  Enter length of time not red: 12
  Enter length of time red: 18
We won't make Light3.
```

Second Submission: Do the above tests, plus the following:

Test Case 4:

```
Enter speed: 45
Enter the number of lights: 3
Light1:
  Enter distance: 0.72
  Enter time since last red: 3.5
  Enter length of time not red: 7.6
  Enter length of time red: 3.9
Light2:
  Enter distance: .89
  Enter time since last red: 4.7
  Enter length of time not red: 10.3
  Enter length of time red: 7.9
Light3:
  Enter distance: .52
  Enter time since last red: 3.3
  Enter length of time not red: 9.2
  Enter length of time red: 10.8
We will make all lights.
```

6A Making Change in Different Coins

This problem is to calculate the minimum number of coins, chosen from given sets of coin values, needed to make specific change. Assume that there are four different, integer-valued coins available. Input is the values of the four different coins (in cents), the total bought (in dollars) and the amount paid (in dollars). The coin values will be labeled coin1, coin2, etc. The coin values might not be given in order of value. The output should be the number of each coin needed, together with the label of that coin. Assume one of the coins is always 1 cent.

Use a greedy algorithm that first chooses as many as possible of the largest coin, then uses the same strategy for the next largest coin, etc. This algorithm will not always be optimum for all sets of coin values. However, it will work for the sets of coin values used in this competition.

Examples:

1) coin1: 50 coin2: 25 coin3: 5 coin4: 1 bought: 2.87 paid: 3.00
   Output: 0 coin1 , 0 coin2, 1 coin3, 3 coin4
2) coin1: 2 coin2: 1 coin3: 25 coin4: 5 bought: 2.37 paid: 3.00
   Output: 2 coin3 , 2 coin4, 1 coin1, 1 coin2

6A making change Solution

```
{

        float purchase, tendered, change;
        int coin[4];
        int num[4];
        int order[4];
        int tempval, tempord;
        int i,j;
        for (i=0; i<4; i++){
               cout<<"\nEnter value of coin "<< i+1 <<": "; cin>>coin[i];order[i]=i;
        }
        //sort
        for(j=0;j<3;j++){
        for(i=0;i<3;i++){
               if(coin[i]<coin[i+1]){tempval = coin[i]; coin[i]=coin[i+1];
                      coin[i+1]=tempval;
               tempord=order[i]; order[i]=order[i+1]; order[i+1]=tempord;}
        }}

        cout<<"\nenter cost of purchase: ";
        cin>>purchase;
        cout<<"\nenter amount tendered: ";
        cin>>tendered;

        change = tendered - purchase;
        cout<<"\n amount to be returned: $"<<change;
        //convert dollars and cents to pennies
        change = change*100;
        cout<<"\n change ";
        for (i=0; i<4; i++) {

        num[i] = 0;
        while ( change >= coin[i]) {num[i]++; change = change - coin[i];}
        cout<<"\n "<<num[i] <<" of coin "<<order[i]+1 <<" needed, value "<<coin[i];
        }
```

6A making change Test Cases

First testing: Do the test cases listed below. The output should be integers . Formatting is not important.

Test case

1) coin1: **50** coin2: **10** coin3: **5** coin4: **1** bought: **2.27** paid: **5.00**

Output: **5** coin1 , **2** coin2, **0** coin3, **3** coin4

2) coin1: **5** coin2: **10** coin3: **1** coin4: **50** bought: **2.27** paid: **5.00**

Output: **5** coin4 , **2** coin2, **0** coin1, **3** coin3

3) coin1: **2** coin2: **1** coin3: **25** coin4: **5** bought: **0.37** paid: **1.00**

Output: **2** coin3 , **2** coin4, **1** coin1, **1** coin2

Second testing: Repeat the test cases above and do the additional ones below

Test case

4) coin1: **20** coin2: **40** coin3: **1** coin4: **5** bought: **2.12** paid: **3.00**

Output: **2** coin2 , **0** coin1, **1** coin4, **3** coin3

5) coin1: **2** coin2: **7** coin3: **25** coin4: **5** bought: **4.93** paid: **5.00**

Output: **0** coin3 , **1** coin2, **0** coin4, **0** coin1