# Unit 0 – Solutions

***Solutions – Pages 8 and 9***

**1.** Write assignment statements in both VB and C# for each of the following situations. (For your convenience, a table is given that shows some of the most commonly used operators in VB and C#.)

| | ***Arithmetic Operators*** | | | | | | | ***Relational (Comparison) Operators*** | | | | | | ***Conditional Operators*** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| VB | + | – | * | / | \ | **Mod** | ^ | = | < | > | <= | >= | < > | **And** | **Or** | **Not** |
| C# | + | – | * | / | / | % | N/A | = = | < | > | <= | >= | != | && | \|\| | ! |

| ***Task to Complete*** | ***VB Assignment Statement*** | ***C# Assignment Statement*** |
|---|---|---|
| **(a)** Increase by 1 the count of the number of vowels found in a string. | `NumVowels = NumVowels + 1` | `numVowels = numVowels + 1;`<br>***Shortcuts for this in C/C#/C++/Java***<br>`numVowels++;`<br>or<br>`++numVowels;` |
| **(b)** Decrease the bank balance by the amount of money withdrawn. | `Balance = Balance - Withdrawal` | `balance = balance – withdrawal;`<br>***Shortcuts for this in C/C#/C++/Java***<br>`balance -= withdrawal;` |
| **(c)** Calculate the number of whole hours in a given number of seconds. | `Hours = Seconds \ 3600` | `hours = seconds / 3600;` |
| **(d)** Calculate the number of seconds remaining once all whole hours have been removed. | `Seconds = Seconds Mod 3600` | `seconds = seconds % 3600;`<br>***Shortcuts for this in C/C#/C++/Java***<br>`seconds %= 3600;` |
| **(e)** Copy the value of the variable “Position” to the variable “NewPosition.” | `NewPosition = Position` | `newPosition = position;` |
| **(f)** Change the value of the variable “Customer” to “Chris Rock.” | `Customer = "Chris Rock"` | `customer = "Chris Rock";` |
| **(g)** Triple the amount of money won by being a contestant on Jeopardy. | `Winnings = Winnings * 3` | `winnings = winnings * 3;`<br>***Shortcuts for this in C/C#/C++/Java***<br>`winnings *= 3;` |

**2.** Write assignment statements in both VB and Java for each of the following formulas. ***Remember to use MEANINGFUL variable names!*** (You'll have to do some research to find out how to use the square root and the power functions in C#.)

| ***Formula*** | ***VB Assignment Statement*** | ***C# Assignment Statement*** |
|---|---|---|
| **(a)** $F = ma$ | `Force=Mass*Acceleration` | `//Newton's Second Law`<br>`force=mass*acceleration;` |
| **(b)** $F_G = -\frac{Gm_1m_2}{r^2}$ | `GravForce=-G*Mass1*Mass2/Distance^2` | `//Newton's Law of Gravitation`<br>`//G = Universal Gravitational Constant`<br>`gravForce=-G*mass1*mass2/Math.Pow(distance,2);` |
| **(c)** $A = \frac{bh}{2}$ | `TriangleArea=Base*Height/2` | `triangleArea=base*height/2;` |
| **(d)** $A = \frac{h(a+b)}{2}$ | `TrapezoidArea=Height*(A+B)/2` | `trapezoidArea=height*(a+b)/2;` |
| **(e)** $E_0 = m_0c^2$ | `RestEnergy=RestMass*LightSpeed^2` | `//Equivalence of mass and energy (Einstein)`<br>`restEnergy=restMass*Math.Pow(LIGHT_SPEED,2);` |
| **(f)** $A = \pi r^2$ | `CircleArea=Pi*Radius^2` | `circleArea=PI*Math.Pow(radius,2);` |
| **(g)** $V = \frac{4}{3}\pi r^3$ | `SphereVolume=4/3*Pi*Radius^3` | `sphereVolume=4/3*PI*Math.Pow(radius,3);` |
| **(h)** $c = \sqrt{a^2 + b^2}$ | `Hypotenuse=Sqr(A^2+B^2)` | `//Pythagorean Theorem`<br>`hypotenuse=Math.Sqrt(Math.Pow(a,2)+ Math.Pow(b,2));` |
| **(i)** $m = \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}$ | `Mass=RestMass/Sqr(1- _`<br>`Velocity^2/LightSpeed^2)` | `/* Mass of a moving body as measured from an`<br>`inertial frame of reference (the "rest frame")`<br>`with respect to which the body moves away with`<br>`velocity 'velocity' (Einstein) */`<br>`mass=restMass/Math.Sqrt (1-`<br>`Math.Pow(velocity,2)/`<br>`Math.Pow (LIGHT_SPEED,2));` |

***Note***

1. VB requires the ***statement continuation characters*** (space followed by an underscore) to continue a long statement on the next line. Since C# statements end with a semi-colon, a long statement can be continued on the next line without the use of any special symbols.
2. By convention, ***constant names*** are written in "`ALL_CAPS`." Underscores are used to increase the readability of constant names that contain two or more words. (**e.g.** `LIGHT_SPEED`)
3. By convention, ***variable names*** are written in "`lowerCamelCase`" or "`UpperCamelCase`." (**e.g.** `restMass` or `RestMass`)
4. By convention, ***class names*** (to be explained later) are written in "`UpperCamelCase`." (**e.g.** `Math`)

***Solutions – Page 14, 15***

**1.** Complete a memory map for each loop. In addition, state the ***purpose*** of each loop.

### Loop

```
sum = 0;
count = 0;
for (int x=1000; x>=0; x-=100)
{
   sum += x;
   count++;
}
average = sum/count;
```

### Memory Map

| | x | sum | count |
|---|---|---|---|
| Values Before Entering Loop → | - | 0 | 0 |
| | 1000 | 1000 | 1 |
| | 900 | 1900 | 2 |
| | 800 | 2700 | 3 |
| | 700 | 3400 | 4 |
| | 600 | 4000 | 5 |
| | 500 | 4500 | 6 |
| | 400 | 4900 | 7 |
| | 300 | 5200 | 8 |
| | 200 | 5400 | 9 |
| | 100 | 5500 | 10 |
| | 0 | 5500 | 11 |
| Values After Exiting Loop → | - | 5500 | 11 |

### Problem Solved

The purpose of the given "**for**" loop is to

calculate the average of the values 0, 100, 200, 300, …, 1000. The average turns out to be 5500/11=500.

### Loop

```
int a = 356;
int b = 512;
do
{
   int remainder = a % b;
   a = b;
   b = remainder;
}while (b != 0);
gcd = a;
```

### Memory Map

| | a | b | remainder |
|---|---|---|---|
| Values Before Entering Loop → | 356 | 512 | – |
| | 512 | 356 | 356 |
| | 356 | 156 | 156 |
| | 156 | 44 | 44 |
| | 44 | 24 | 24 |
| | 24 | 20 | 20 |
| | 20 | 4 | 4 |
| | | | |
| | | | |
| | | | |
| | | | |
| Values After Exiting Loop → | 4 | 0 | – |

### Problem Solved

The purpose of the given "**do**" loop is to

use the Euclidean algorithm to calculate the greatest common divisor of 356 and 512, which turns out to be 4.

### Loop

```
/*The memory map at the
  right assumes that 'num'
  is an 'int' variable. */
num=1023;
numDivisionsByTwo = 0;
num/=2;
while (num > 0)
{
   numDivisionsByTwo++;
   num/=2;
}
```

### Memory Map

| | num | numDivisionsByTwo |
|---|---|---|
| Values Before Entering Loop → | 511 | 0 |
| | 255 | 1 |
| | 127 | 2 |
| | 63 | 3 |
| | 31 | 4 |
| | 15 | 5 |
| | 7 | 6 |
| | 3 | 7 |
| | 1 | 8 |
| | 0 | 9 |
| | | |
| Values After Exiting Loop → | 0 | 9 |

### Problem Solved

The purpose of the given "**do**" loop is to

determine how many times 2 divides into 'num." In this case, it turns out that 2 divides into 1023 nine times.

**2.** ***On paper***, write C# loops to perform each of the following tasks. Do not use a computer for this question except for verifying that your code is correct.

**(a)** Add up the numbers 1 + 2 + 3 + 4 + ... until the **Sum** > **100**.

```
int sum = 0, i = 1;

while (sum <= 100)
{
   sum += i;
   i++;
}
```

**(b)** Determine how many numbers 2 +4 + 6 + 8 + ... are needed to give a **Sum** > **1000**.

```
int sum = 0, i = 2, howMany=0;

while (sum <= 1000)
{
   sum += i;
   howMany++;
   i+=2;
}
```

**(c)** Determine the sum of all powers of 2 (i.e. 1, 2, 4, 8, 16, 32, …) that are less than 1000000.

```
int sum = 0, i=0, powerOf2;

do
{
   powerOf2=Math.Pow(2,i);
   sum += powerOf2;
   i++;
}while (powerOf2 <= 1000000);
```

**(d)** Output the smallest number (other than 1) that divides evenly into 2701.

```
int i = 2;

while (2701 % i != 0)
{
   i++;
}
```

**3.** The ancient Greek civilization had a keen interest …

**(a)** A proper divisor of an integer is any divisor of the integer except itself.

**(b)** See http://en.wikipedia.org/wiki/Perfect_number

**(c)** Use a for loop to search for divisors of the given number.
The loop counter variable "i" ranges from 2 to half the given number. The variable "i" is the candidate divisor.

If "i" is a divisor of the given number, add it to the sum. Otherwise, do nothing.

After the loop has finished calculating the sum, compare the sum to the given number.
If the sum is equal to the number, the number is perfect.
Otherwise the number is not perfect.

**(d)**

```
//Is 'number' perfect? The value of 'number' is set earlier in the code.
int sum = 0;

for (int i = 2; i <= number/2; i++)
{
    if (number % i == 0)
        sum += i;
}

if (sum == number)
    label1.Text = "Perfect!";
else
    label1.Text = "NOT perfect!";
```

***Solutions – Page 21***

**1.** Create a memory map for each code segment. In addition, determine the problem that is solved in each case. (Some variables have intentionally been given silly names to disguise their purpose.)

<table>
<tr><th>Code Segment</th><th>Memory Map (Trace Chart)</th><th>Problem Solved?</th></tr>
<tr>
<td>

```
int harinder=0;
string c="";
string s="aeiouAEIOU";
string a="Laziness is for fools!";
for (int i=0; i<a.Length; i++){
   c=a.Substring(i,1);
   if (s.IndexOf(c)>=0)
      harinder++;
}
```

</td>
<td>
The values of “s” and “a” do not change. Therefore, they are not included in the memory map.
<table>
<tr><th></th><th>i</th><th>c</th><th>harinder</th></tr>
<tr><td>Values Before Entering Loop →</td><td>-</td><td>""</td><td>0</td></tr>
<tr><td></td><td>0</td><td>"L"</td><td>0</td></tr>
<tr><td></td><td>1</td><td>"a"</td><td>1</td></tr>
<tr><td></td><td>2</td><td>"z"</td><td>1</td></tr>
<tr><td></td><td>3</td><td>"i"</td><td>2</td></tr>
<tr><td></td><td>4</td><td>"n"</td><td>2</td></tr>
<tr><td></td><td>5</td><td>"e"</td><td>3</td></tr>
<tr><td></td><td>6</td><td>"s"</td><td>3</td></tr>
<tr><td></td><td>7</td><td>"s"</td><td>3</td></tr>
<tr><td></td><td>8</td><td>" "</td><td>3</td></tr>
<tr><td></td><td>9</td><td>"i"</td><td>4</td></tr>
<tr><td></td><td>10</td><td>"s"</td><td>4</td></tr>
<tr><td></td><td>11</td><td>" "</td><td>4</td></tr>
<tr><td></td><td>12</td><td>"f"</td><td>4</td></tr>
<tr><td></td><td>13</td><td>"o"</td><td>5</td></tr>
<tr><td></td><td>14</td><td>"r"</td><td>5</td></tr>
<tr><td></td><td>15</td><td>" "</td><td>5</td></tr>
<tr><td></td><td>16</td><td>"f"</td><td>5</td></tr>
<tr><td></td><td>17</td><td>"o"</td><td>6</td></tr>
<tr><td></td><td>18</td><td>"o"</td><td>7</td></tr>
<tr><td></td><td>19</td><td>"l"</td><td>7</td></tr>
<tr><td></td><td>20</td><td>"s"</td><td>7</td></tr>
<tr><td></td><td>21</td><td>"!"</td><td>7</td></tr>
<tr><td>Values After Exiting Loop →</td><td>-</td><td>"!"</td><td>7</td></tr>
</table>
</td>
<td>By the time the loop has finished executing, the variable “harinder” stores<br><br>the number of vowels found in the string “a.”</td>
</tr>
<tr>
<td>

```
string c="";
string s="Einstein";
for (int i=s.Length-1; i>=0; i--)
   c=c+s.Substring(i,1);
```

</td>
<td>
<table>
<tr><th></th><th>i</th><th>c</th></tr>
<tr><td>Values Before Entering Loop →</td><td>-</td><td>""</td></tr>
<tr><td></td><td>0</td><td>"n"</td></tr>
<tr><td></td><td>1</td><td>"ni"</td></tr>
<tr><td></td><td>2</td><td>"nie"</td></tr>
<tr><td></td><td>3</td><td>"niet"</td></tr>
<tr><td></td><td>4</td><td>"niets"</td></tr>
<tr><td></td><td>5</td><td>"nietsn"</td></tr>
<tr><td></td><td>6</td><td>"nietsni"</td></tr>
<tr><td></td><td>7</td><td>"nietsniE"</td></tr>
<tr><td>Values After Exiting Loop →</td><td>-</td><td>"nietsniE"</td></tr>
</table>
</td>
<td>By the time the loop has finished executing, the string variable “c” stores<br><br>the “reverse” of string “s,” i.e. having the same characters as “s” but in the reverse order.</td>
</tr>
</table>

***Solutions – Page 22***

***Exercises***

Study the given C# declarations. Then complete the table.

```
int a;
long b=3;
short c=1;
byte d=1;
char e=(char)1024; // Force a conversion from 'int' to 'char.' The decimal value 1024 can
                   // also be specified explicitly as the hexadecimal Unicode value
                   // '\u0400' or as the hexadecimal escape sequence '\x0400'
float f=3.8e21f; // The 'f' at the end means that the value should be stored as a 'float'
double g=3.232552e152; // Unless specified otherwise, floating point values are stored as
                       // 'double' values. A 'd' can also be used to indicate that the
                       // number should be stored as a 'double' value: e.g. 2.0d
string h="If thinking makes your brain hurt it's probably due to lack of practice!";
```

| ***C# Statement*** | ***Is it Allowed? If so, explain why. If not, suggest a correction.*** |
|---|---|
| 1. `b=a;` | NOT allowed because "a" has not previously been assigned a value.<br>**Correction:** `int a=0; //Any 32-bit (or smaller) integral value //can be assigned to "a"` |
| 2. `a=b;` | NOT allowed because "int" is a smaller integral (integer) type than "long."<br>**Correction:** `a=(int)b;` |
| 3. `h=f;` | NOT allowed because "h" is of type "string" and "f" is of type "float."<br>**Correction:** `h=Convert.ToString(f);` |
| 4. `f=h;` | NOT allowed because "f" is of type "float" and "h" is of type "string."<br>**Correction:** `f=Convert.ToSingle(h); //'Single' is the .NET class for 'float'` |
| 5. `e=c;` | NOT allowed because "e" is of type "char" and "c" is of type "short."<br>**Correction:** `e=(char)b;` |
| 6. `c=e;` | NOT allowed because "c" is of type "short" and "e" is of type "char."<br>**Correction:** `c=(short)e;` |
| 7. `g=f;` | Allowed because "double" is a larger floating point type than "float." |
| 8. `f=g;` | NOT allowed because "float" is a smaller floating point type than "double."<br>**Correction:** `f=(float)g;` |
| 9. `f=d;` | Allowed because "float" is a floating point type that is large enough to accommodate "byte" integral values. (The "byte" type is a very small integral type, which means that a "byte" value can be assigned to almost any numeric type.) |
| 10. `d=f;` | NOT allowed because "float" is a much larger type than "byte."<br>**Correction:** `d=(byte)f;` |
| 11. `d=e;` | NOT allowed because "d" is of type "byte" and "e" is of type "char."<br>**Correction:** `d=(byte)e;` |
| 12. `e=d;` | NOT allowed because "e" is of type "char" and "d" is of type "byte."<br>**Correction:** `e=(char)d;` |
| 13. `f=b;` | Allowed because "float" is a floating point type that is large enough to accommodate "long" integral values. Note that ***precision*** can be lost in such an assignment because "float" values only have up to 7 significant digits while "long" values can have up to 10 digits. |

### *Using C# to Understand the Solutions on the Previous Page*

If the code is typed exactly as shown on the previous page, the following appears in the C# development environment:

| *Code in Code Editor* | *Errors in the Error List* |
|---|---|

```
b = a;
a = b;
h = f;
f = h;
e = c;
c = e;
g = f;
f = g;
f = d;
d = f;
d = e;
e = d;
f = b;
```

| | Description |
|---|---|
| 1 | Cannot implicitly convert type 'long' to 'int'. An explicit conversion exists (are you missing a cast?) |
| 2 | Cannot implicitly convert type 'float' to 'string' |
| 3 | Cannot implicitly convert type 'string' to 'float' |
| 4 | Cannot implicitly convert type 'short' to 'char'. An explicit conversion exists (are you missing a cast?) |
| 5 | Cannot implicitly convert type 'char' to 'short'. An explicit conversion exists (are you missing a cast?) |
| 6 | Cannot implicitly convert type 'double' to 'float'. An explicit conversion exists (are you missing a cast?) |
| 7 | Cannot implicitly convert type 'float' to 'byte'. An explicit conversion exists (are you missing a cast?) |
| 8 | Cannot implicitly convert type 'char' to 'byte'. An explicit conversion exists (are you missing a cast?) |
| 9 | Cannot implicitly convert type 'byte' to 'char'. An explicit conversion exists (are you missing a cast?) |

Once the code is typed as follows, that is, with all the corrections as indicated on the previous page, the errors disappear!

```
int a=0;
long b = 3;
short c = 1;
byte d = 1;
char e = (char)1024; // Force a conversion from 'int' to 'char.' The decimal value 1024 can
                     // also be specified explicitly as the hexadecimal Unicode value
                     // '\u0400' or as the hexadecimal escape sequence '\x0400'
float f = 3.8e21f;   // The 'f' at the end means that the value should be stored as a 'float'
double g = 3.232552e152; // Unless specified otherwise, floating point values are stored as
                         // 'double' values. A 'd' can also be used to indicate that the
                         // number should be stored as a 'double' value: e.g. 2.0d
string h = "If thinking makes your brain hurt it’s probably due to lack of practice!";

b = a;
a = (int)b;
h = Convert.ToString(f);
f = Convert.ToSingle(h);
e = (char)c;
c = (short)e;
g = f;
f = (float)g;
f = d;
d = (byte)f;
d = (byte)e;
e = (char)d;
f = b;
```