**ValidateInput**

**Test – A = True**

return true;

**Test – A-Z = True** (No Change)

**Test – 0 = False**

if (character == '0')  
{  
 return false;  
}

return true;

**Test – 1 = False**

if ((character == '0') || (character == '1'))  
{  
 return false;  
}

**Test – 2 = False**

if ((character == '0') || (character == '1') || (character == '2'))

**Test – 3 = False**

if ((character == '0') || (character == '1') || (character == '2') ||

(character == '3'))

**Test – 5 = False**

if ((character == '0') || (character == '1') || (character == '2') ||

(character == '3') || (character == '4') || (character == '5'))

Refactor

return ((character >= 'A') && (character <= 'Z'));

**Test – !A-Z = True** (No Change)

**Convert**

**Test – A = 1 Line**

List<string> result = new List<string>();  
result.Add(string.Empty);  
return result;

**Test – A = A**

List<string> result = new List<string>();  
result.Add("A");  
return result;

**Test – B = 3 Lines**

if (character == 'A')

{

result.Add("A");  
}

else

{

result.Add(string.Empty);

result.Add(string.Empty);

result.Add(string.Empty);

}

**Test – B = ABBA**

if (character == 'A')

{

result.Add("A");  
}

else

{

result.Add(" A");

result.Add("B B");

result.Add(" A");

}

**Test – C = 5 Lines & ABCCBA**

else if (character == 'B')

{

result.Add(" A");

result.Add("B B");

result.Add(" A");

}

else

{

result.Add(" A");

result.Add(" B B");

result.Add("C C");

result.Add(" B B");

result.Add(" A");

}

**Test – D = 7 Lines & ABCDDCBA**

else if (character == 'C')

{

result.Add(" A");

result.Add(" B B");

result.Add("C C");

result.Add(" B B");

result.Add(" A");

}

else

{

result.Add(" A");

result.Add(" B B");

result.Add(" C C");

result.Add("D D");

result.Add(" C C");

result.Add(" B B");

result.Add(" A");

}

*Refactor*

List<string> result = new List<string>();  
if (character == 'A')  
{  
 result.Add(GenerateLine('A', 0, 0));  
}  
else if (character == 'B')  
{  
 result.Add(GenerateLine('A', 1, 0));  
 result.Add(GenerateLine('B', 0, 1));  
 result.Add(GenerateLine('A', 1, 0));  
}  
else if (character == 'C')  
{  
 result.Add(GenerateLine('A', 2, 0));  
 result.Add(GenerateLine('B', 1, 1));  
 result.Add(GenerateLine('C', 0, 3));  
 result.Add(GenerateLine('B', 1, 1));  
 result.Add(GenerateLine('A', 2, 0));  
}  
else  
{  
 result.Add(GenerateLine('A', 3, 0));  
 result.Add(GenerateLine('B', 2, 1));  
 result.Add(GenerateLine('C', 1, 3));  
 result.Add(GenerateLine('D', 0, 5));  
 result.Add(GenerateLine('C', 1, 3));  
 result.Add(GenerateLine('B', 2, 1));  
 result.Add(GenerateLine('A', 3, 0));  
}  
return result;

private static string GenerateLine(char character, int offset, int distance)

{

var builder = new StringBuilder();

builder.Append(new string(' ', offset));

builder.Append(character);

if (distance > 0)

{

builder.Append(new string(' ', distance));

builder.Append(character);

}

return builder.ToString();

}

*Refactor*

List<string> result = new List<string>();  
int offset = character - 'A';  
int distance = 0;  
  
while (offset >= 0)  
{  
 result.Add(GenerateLine((char)('A' + distance), offset, (distance \* 2) - 1));  
 offset--;  
 distance++;  
}  
  
distance = distance - 2;  
offset = offset + 2;  
  
while (distance >= 0)  
{  
 result.Add(GenerateLine((char)('A' + distance), offset, (distance \* 2) - 1));  
 offset++;  
 distance--;  
}  
return result;

*Refactor*

List<string> result = new List<string>();  
int offset = character - 'A';  
int distance = 0;  
int direction = 1;

while (distance >= 0)  
{  
 char currentCharacter = (char)('A' + distance);  
 result.Add(GenerateLine(currentCharacter, offset, (distance \* 2) - 1));  
 if (currentCharacter == character)  
 {  
 direction = -1;  
 }  
 offset -= direction;  
 distance += direction;  
}  
return result;

**Test – E = 9 Lines & ABCDEEDCBA** (No Change)