Development of a verified lossless compression algorithm with Dafny: Run-Length Encoding

Pedro M. Esteves, January, 2021

MIEIC/MFES

1. Introduction **11**

2. Specification **12**

3. Implementation and verification **16**

3.1. Data representation **16**

3.2. constructor **16**

3.3. repeatOccurences **16**

3.4. compress and helpCompress **18**

3.5. decompress and helpDecompress **19**

4. Static testing **20**

5. Putting it all together **21**

1. Introduction

Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data.

In this tutorial, we show how to use Dafny to develop a verified (proved correct) implementation of the Run-Length Encoding, a form of lossless compression in which sequences of the same value are stored as a single value, preceded by an escape character and followed by the number of occurrences of that character in the original string.

2. Specification

We start by specifying in Dafny the syntax and semantics (contracts) of the services provided by the class. Since we are dealing with files, we have defined some new types, such as the byte type (an integer between 0 and 255), to allow us to manipulate the data from files. The byte type is used mainly on the IO layer since the C# IO operations read and write arrays of bytes. Before the compression/decompression of data, these arrays are converted into strings (the same as seq<char> in Dafny), just to make the compression easier, since Dafny has some problems dealing with user-defined types.

After compiling the program with **dafny compression.dfy Io.dfy IoNative.cs**, the user should run the program from the command line as **./compression 0 SourceFile DestFile**, if he wants to compress the file, or **./compression 1 SourceFile DestFile**, if he wants to decompress the file.

File compression.dfy (the other files are on the last section of this report)

```
include "Io.dfy" // This file contains all the IO methods


method ArrayFromSeq<A>(s: seq<A>) returns (a: array<A>)
  ensures a[..] == s


method GetStringFromByteArray(b: array?<byte>) returns (s: string)
    ensures b == null ==> s == ""
    ensures b != null ==> b.Length == |s|
    ensures forall i :: 0 <= i < |s| ==> b[i] as char == s[i] && 0 <= s[i] as int < 256


method GetByteArrayFromString(s: string) returns (b: array?<byte>)
    requires forall i :: 0 <= i < |s| ==> 0 as char <= s[i] < 256 as char
    ensures |s| == 0 ==> b == null
    ensures |s| > 0 ==> b != null && b.Length == |s|
    ensures forall i :: 0 <= i < |s| ==> b[i] as char == s[i]


method {:verify false} callCompression (ghost env:HostEnvironment, src_name:array<char>,
src:FileStream, dst:FileStream, isCompression: bool) returns (success:bool)
    requires env.Valid() && env.ok.ok();
        requires src_name[..] == src.Name();
        requires src.Name() in env.files.state() && dst.Name() in env.files.state();
        requires env == src.env == dst.env;
        requires env.ok == src.env.ok == dst.env.ok;
        requires env.files == src.env.files == dst.env.files;
        requires src.IsOpen() && dst.IsOpen();
        requires src != dst;
        requires env.files.state()[dst.Name()] == [];
    modifies env, env.files, env.ok, src, dst, src.env, src.env.ok, src.env.files;


method {:main} Main(ghost env:HostEnvironment)
  requires env.Valid() && env.ok.ok();
```

```
  modifies env, env.files, env.ok;


function method NumDigits(n: int) : int
    decreases n
    requires n >= 0


function method ToString(n: int) : string
    decreases n
    requires n >= 0
    ensures
       var s := ToString(n);
       |s| == NumDigits(n) && forall i :: 0 <= i < |s| ==> '0' <= s[i] <= '9' && 0 <= s[i]
as int < 256


predicate method IsInt(c: char)
    ensures '0' <= c <= '9' <==> IsInt(c)


predicate method IsAlphaChar(c: char)
    ensures 'A' <= c <= 'Z' || 'a' <= c <= 'z' <==> IsAlphaChar(c)


function method GetInt(s: string, n: int) : string
    decreases |s| - n
    requires 0 <= n <= |s|
    requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
    ensures
        var integerString := GetInt(s, n);
           (|integerString| != 0 ==> forall i :: 0 <= i < |integerString| ==> '0' <=
integerString[i] <= '9' && 0 <= integerString[i] as int < 256)
            && (|s| >= n + |integerString|)


function method ParseInt(s: string, i: int) : int
    decreases i
    requires 0 <= i < |s|
    requires forall j :: 0 <= j <= i ==> '0' <= s[j] <= '9'
    ensures ParseInt(s, i) >= 0
```

```
function method RepeatChar(c: char, occ: int) : string
    decreases occ
    requires occ >= 0
    requires 0 <= c as int < 256
    ensures
        var s := RepeatChar(c, occ);
        |s| == occ && forall i :: 0 <= i < occ ==> s[i] == c && 0 <= s[i] as int < 256


class Compression {
    constructor ()


    function method repeatOccurences(cur_char: char, occ: int) : string
        requires occ > 0
        requires 0 <= cur_char as int < 256
        ensures
            var s := repeatOccurences(cur_char, occ);
            (occ <= 3 ==> |s| == occ)
            && (occ > 3 ==> |s| == 2 + |ToString(occ)|)
            && forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256


    function method helpCompress(s: string, cur_char: char, occ: int, index: int) : string
        decreases |s| - index
        requires 1 <= occ <= |s|
        requires 1 <= index <= |s| && 0 < occ <= index
        requires forall i :: index - occ <= i < index ==>  s[i] == cur_char
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        requires 0 <= cur_char as int < 256
        ensures 0 < |helpCompress(s, cur_char, occ, index)|
        ensures
           var cmp := helpCompress(s, cur_char, occ, index);
           (forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256)
            && (index >= |s| ==> (occ <= 3 ==> |cmp| == occ) && (occ > 3 ==> |cmp| == 2 +
|ToString(occ)|))


    function method compress(s: string) : string
```

```
        requires |s| > 0
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        ensures 0 < |compress(s)|
        ensures
            var cmp := compress(s);
            forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256


      function method helpDecompress(s: string, fnd_esc: bool, fnd_ch: bool, ch: char,
index: int) : string
        decreases |s| - index
        requires |s| > 0
        requires 1 <= index <= |s|
        requires fnd_ch ==> s[index-1] == ch && index >= 2
        requires fnd_esc ==> if fnd_ch then s[index - 2] == '\0' else s[index - 1] == '\0'
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        ensures
            var dcmp := helpDecompress(s, fnd_esc, fnd_ch, ch, index);
            forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256


    function method decompress(s: string) : string
        requires |s| > 0
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        ensures
            var dcmp := decompress(s);
            forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256
}
```

3. Implementation and verification

3.1. Data representation

Since this program needs to read and write to files that the user specifies as command-line arguments, we store the files as FileStream variables (a class defined by the C# IO layer). If the source file exists and the destination file does not exist, we read the data from the first one to arrays of bytes (array<byte>). After that, we convert these arrays to strings (seq<char> or string in Dafny), and we call the compression methods with these strings. Finally, we convert these strings to arrays of bytes and write them on the destination files.

Since we only call the compression method once per execution, the Compression class neither needs attributes nor a class invariant.

3.2. constructor

Unlike other OOP languages, Dafny requires the programmer to define the constructor in order to instantiate a class, so we only use it to instantiate the Compression class. This implementation could also be done using separated methods, but we have decided to group the methods in a single class, using an object-oriented approach.

```
constructor () {}
```

3.3. repeatOccurences

This is an auxiliary method, used to compress a sequence of equal characters. The sequence is only compressed if the number of occurrences of the given character is greater than 3. For example, the string “AAAA” will be compressed into “\0A4”, which has 1 character less than the original string (‘\0’ is the NULL character, usually used to point to the end of a string, which means that these files should only be used with the developed algorithm).

```
function method repeatOccurences(cur_char: char, occ: int) : string
    requires occ > 0
    requires 0 <= cur_char as int < 256
    ensures
       var s := repeatOccurences(cur_char, occ);
       (occ <= 3 ==> |s| == occ)
       && (occ > 3 ==> |s| == 2 + |ToString(occ)|)
       && forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
{
    if occ <= 3 then
       RepeatChar(cur_char, occ)
    else
       ['\0'] + [cur_char] + ToString(occ)
}
```

This method also uses two other function methods defined outside the class: RepeatChar and ToString.

The RepeatChar function method is recursive, and, as the name suggests, repeats one single char.

```
function method RepeatChar(c: char, occ: int) : string
    decreases occ
    requires occ >= 0
    requires 0 <= c as int < 256
    ensures
```

```
        var s := RepeatChar(c, occ);
        |s| == occ && forall i :: 0 <= i < occ ==> s[i] == c && 0 <= s[i] as int < 256
{
    if occ == 0 then "" else [c] + RepeatChar(c, occ - 1)
}
```

The ToString method is also a recursive method that converts an integer into a string. In its post-conditions, it calls another function method, NumDigits, that counts the number of digits of an integer.

```
function method NumDigits(n: int) : int
    decreases n
    requires n >= 0
{
    if n <= 9 then 1 else 1 + NumDigits(n / 10)
}

function method ToString(n: int) : string
    decreases n
    requires n >= 0
    ensures
        var s := ToString(n);
         |s| == NumDigits(n) && forall i :: 0 <= i < |s| ==> '0' <= s[i] <= '9' && 0 <= s[i] as int < 256
{
    if n <= 9 then [n as char + '0']
    else ToString(n / 10) + [(n % 10) as char + '0']
}
```

3.4. compress and helpCompress

The compress method is one of the most important methods of the Compression class. This method executes the Run-Length Encoding algorithm with the help of the helpCompress method.

```
function method compress(s: string) : string
   requires |s| > 0
   requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
   ensures 0 < |compress(s)|
   ensures
     var cmp := compress(s);
```

```
    forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256
{
   helpCompress(s, s[0], 1, 1)
}
```

As mentioned earlier, the compression of files will compress only sequences with the same character, inserting an escape character, followed by the repeated character and ending with the number of occurrences of that character. For this purpose, we chose the character '\0', the NULL character, since this often marks the end of a string and never appears inside a text.

The compress method calls the helpCompress, a recursive function method that counts the number of repetitions of a character, and decides whether it should be repeated or compressed in the new string.

```
function method helpCompress(s: string, cur_char: char, occ: int, index: int) : string
   decreases |s| - index
   requires 1 <= occ <= |s|
   requires 1 <= index <= |s| && 0 < occ <= index
   requires forall i :: index - occ <= i < index ==>  s[i] == cur_char
   requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
   requires 0 <= cur_char as int < 256
   ensures 0 < |helpCompress(s, cur_char, occ, index)|
   ensures
      var cmp := helpCompress(s, cur_char, occ, index);
      (forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256)
       && (index >= |s| ==> (occ <= 3 ==> |cmp| == occ) && (occ > 3 ==> |cmp| == 2 +
|ToString(occ)|))
{
    if index >= |s| then
       repeatOccurences(cur_char, occ)
    else if s[index] == cur_char then
       helpCompress(s, cur_char, occ + 1, index + 1)
    else
       repeatOccurences(cur_char, occ) + helpCompress(s, s[index], 1, index + 1)
}
```

3.5. decompress and helpDecompress

The other important method that the Compression class defines is the decompress method. Opposing to the compress method, it decompresses a string with the same logic. When it finds an escape character, it knows that

the next character is repeated a certain number of times. As compress, it uses an auxiliary recursive method, helpDecompress, that decompresses the given string.

```
function method decompress(s: string) : string
   requires |s| > 0
   requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
   ensures
      var dcmp := decompress(s);
      forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256
{
   helpDecompress(s, s[0] == '\0', false, '\0', 1)
}


function method helpDecompress(s: string, fnd_esc: bool, fnd_ch: bool, ch: char, index:
int) : string
   decreases |s| - index
   requires |s| > 0
   requires 1 <= index <= |s|
   requires fnd_ch ==> s[index-1] == ch && index >= 2
   requires fnd_esc ==> if fnd_ch then s[index - 2] == '\0' else s[index - 1] == '\0'
   requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
   ensures
      var dcmp := helpDecompress(s, fnd_esc, fnd_ch, ch, index);
      forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256
{
   if index >= |s| then
      if fnd_esc then
         if fnd_ch then ['\0'] + [ch] else ['\0']
      else ""
   else
      if fnd_esc then
         if fnd_ch then
            var integer := GetInt(s, index);
            if |integer| > 0 then
                var occ := ParseInt(integer, |integer| - 1);
                if occ > 3 then // If the number is 3 or less, the char won't be repeated
```

```
                    RepeatChar(ch, occ) + helpDecompress(s, false, false, '\0', index +
|integer|)

             else ['\0'] + [ch] + [s[index]] + helpDecompress(s, false, false, '\0', index
+ |integer|)

           else ['\0'] + [ch] + helpDecompress(s, false, false, '\0', index + 1)

        else helpDecompress(s, true, true, s[index], index + 1)

     else

        if s[index] == '\0' then

           helpDecompress(s, true, false, '\0', index + 1)

        else [s[index]] + helpDecompress(s, false, false, '\0', index + 1)
}
```

4. Static testing

For a sanity check of the specification, we write a simple test scenario. Although this implementation works with real files, Dafny can't statically check the decompress method. This means that probably its pre or post-conditions are not complete, which can lead to an error in some edge cases.

```
method testCompression() {

    var c := new Compression();

    var s := "AAAABBBBCCCC";

    s := c.compress(s);

    assert s == "\0A4\0B4\0C4";

    s := c.decompress(s);

    assert s == "AAAABBBBCCCC";
}
```

5. Putting it all together

File IoNative.cs

```
/* Rui Maranhao -- rui@computer.org */


using System;

using System.Numerics;

using System.Diagnostics;

using System.Threading;

using System.Collections.Concurrent;

using System.Collections.Generic;

using FStream = System.IO.FileStream;
```

```
namespace @__default {
    public partial class HostConstants
    {
        public static void NumCommandLineArgs(out uint n)
        {
            n = (uint)System.Environment.GetCommandLineArgs().Length;
        }
        public static void GetCommandLineArg(ulong i, out char[] arg)
        {
            arg = System.Environment.GetCommandLineArgs()[i].ToCharArray();
        }
    }

    public partial class FileStream
    {
        internal FStream fstream;
        internal FileStream(FStream fstream) { this.fstream = fstream; }

        public static void FileExists(char[] name, out bool result)
        {
              result = System.IO.File.Exists(new string(name));
        }
        public static void FileLength(char[] name, out bool success, out int len)
        {
              len = 42;
              try {
                    System.IO.FileInfo fi = new System.IO.FileInfo(new string(name));
                    if (fi.Length < System.Int32.MaxValue) { // We only support small files
                          len = (int)fi.Length;
                          success = true;
                    } else {
                          success = false;
                    }
              } catch (Exception e) {
```

```
                    System.Console.Error.WriteLine(e);
                    success = false;
                }
            }
            public static void Open(char[] name, out bool ok, out FileStream f)
            {
                try {
                    f = new FileStream(new FStream(new string(name),
System.IO.FileMode.OpenOrCreate, System.IO.FileAccess.ReadWrite));
                    ok = true;
                } catch (Exception e) {
                    System.Console.Error.WriteLine(e);
                    f = null;
                    ok = false;
                }
            }
            public void Close(out bool ok)
            {
                try {
                    fstream.Close();
                    ok = true;
                } catch (Exception e) {
                    System.Console.Error.WriteLine(e);
                    ok = false;
                }
            }
           public void Read(int file_offset, byte[] buffer, int start, int num_bytes, out bool ok)
            {
                try {
                    fstream.Seek(file_offset, System.IO.SeekOrigin.Begin);
                    fstream.Read(buffer, start, num_bytes);
                    ok = true;
                } catch (Exception e) {
                    System.Console.Error.WriteLine(e);
                    ok = false;
```

```
            }
        }
         public void Write(int file_offset, byte[] buffer, int start, int num_bytes, out bool
ok)
        {
            try {
                fstream.Seek(file_offset, System.IO.SeekOrigin.Begin);
                fstream.Write(buffer, start, num_bytes);
                ok = true;
            } catch (Exception e) {
                System.Console.Error.WriteLine(e);
                ok = false;
            }
        }
    }
}
```

File Io.dfy

```
/*  Rui Maranhao -- rui@computer.org */

newtype {:nativeType "byte"}   byte   = b:int | 0 <= b < 256

newtype {:nativeType "ushort"} uint16 = i:int | 0 <= i < 0x10000

newtype {:nativeType "int"}    int32  = i:int | -0x80000000 <= i < 0x80000000

newtype {:nativeType "uint"}   uint32 = i:int | 0 <= i < 0x100000000

newtype {:nativeType "ulong"}  uint64 = i:int | 0 <= i < 0x10000000000000000

newtype {:nativeType "int"}     nat32 = i:int | 0 <= i < 0x80000000

class HostEnvironment

{

    ghost var constants:HostConstants?;

    ghost var ok:OkState?;

    ghost var files:FileSystemState?;


    constructor {:axiom} () //requires false;


    predicate Valid()

        reads this;

    {

        constants != null && ok != null && files != null

    }


    method {:axiom} foo()

        ensures Valid();

}


////////////////////////////////////////////////////////////////////////////

// Per-host constants

////////////////////////////////////////////////////////////////////////////
```

```
class HostConstants
{
    constructor{:axiom} () requires false;

    // result of C# System.Environment.GetCommandLineArgs(); argument 0 is name of executable
    function {:axiom} CommandLineArgs():seq<seq<char>> reads this;

    static method {:extern} NumCommandLineArgs(ghost env:HostEnvironment) returns(n:uint32)
        requires env.Valid();
        ensures  n as int == |env.constants.CommandLineArgs()|;

    static method {:extern} GetCommandLineArg(i:uint64, ghost env:HostEnvironment)
returns(arg:array<char>)
        requires env.Valid();
        requires 0 <= i as int < |env.constants.CommandLineArgs()|;
        ensures  fresh(arg);
        ensures  arg[..] == env.constants.CommandLineArgs()[i];
}

//////////////////////////////////////////////////////////////////////////////
// Failure
//////////////////////////////////////////////////////////////////////////////

// not failed; IO operations only allowed when ok() == true
class OkState
{
    constructor {:axiom} () requires false;
    function {:axiom} ok():bool reads this;
```

```
}

////////////////////////////////////////////////////////////////////////////

// File System

////////////////////////////////////////////////////////////////////////////

class FileSystemState

{

    constructor{ :axiom} () requires false;

    function {:axiom} state() : map<seq<char>,seq<byte>>    // File system maps file names
(sequences of characters) to their contents

        reads this;

}

class FileStream

{

    ghost var env:HostEnvironment;

    function {:axiom} Name():seq<char> reads this;

    function {:axiom} IsOpen():bool reads this;

    constructor {:axiom} () requires false;



        static method {:extern} FileExists(name:array<char>, ghost env:HostEnvironment?)
returns(result:bool)

        requires env != null && env.Valid();

        requires env.ok.ok();

        ensures  result <==> old(name[..]) in env.files.state();



        static method {:extern} FileLength(name:array<char>, ghost env:HostEnvironment)
returns(success:bool, len:int32)

        requires env.Valid();

        requires env.ok.ok();

        requires name[..] in env.files.state();

        modifies env.ok;

        ensures  env.ok.ok() == success;
```

```
        ensures  success ==> len as int == |env.files.state()[name[..]]|;


    static method {:extern} Open(name:array<char>, ghost env:HostEnvironment) returns(ok:bool,
f:FileStream)
        requires env.Valid();
        requires env.ok.ok();
        modifies env.ok;
        modifies env.files;
        ensures  env.ok.ok() == ok;
          ensures  ok ==> fresh(f) && f.env == env && f.IsOpen() && f.Name() == name[..] &&
// FileStream object is initialized
                      env.files.state()  ==  if  name[..]  in  old(env.files.state())  then
old(env.files.state())  // If the file exists, then the file contents are unchanged
                                              else old(env.files.state())[name[..] := []] //
Otherwise, the file now exists with no content


    method {:extern} Close() returns(ok:bool)
        requires env.Valid();
        requires env.ok.ok();
        requires IsOpen();
        modifies this;
        modifies env.ok;
        ensures  env == old(env);
        ensures  env.ok.ok() == ok;
        ensures  !IsOpen();


    method {:extern} Read(file_offset:nat32, buffer:array?<byte>, start:int32, num_bytes:int32)
returns(ok:bool)
        requires env.Valid();
        requires env.ok.ok();
        requires IsOpen();
        requires buffer != null;
```

```
        requires Name() in env.files.state();

         requires file_offset as int + num_bytes as int <= |env.files.state()[Name()]|;    // Don't read beyond the end of the file

         requires 0 <= start as int <= start as int + num_bytes as int <= buffer.Length;   // Don't write outside the buffer

        modifies this;

        modifies env.ok;

        modifies env.files;

        modifies buffer;

        ensures  env == old(env);

        ensures  env.ok.ok() == ok;

        ensures  ok ==> env.files.state() == old(env.files.state());

        ensures  Name() == old(Name());

        ensures  ok ==> IsOpen();

         ensures  ok  ==>  buffer[..]  ==  buffer[..start]  +  env.files.state()[Name()][file_offset..file_offset as int + num_bytes as int] + buffer[start as int + num_bytes as int..];



    method {:extern} Write(file_offset: nat32, buffer: array?<byte>, start: int32, num_bytes: int32) returns(ok:bool)

        requires env.Valid();

        requires env.ok.ok();

        requires IsOpen();

        requires buffer != null;

        requires Name() in env.files.state();

          requires file_offset as int <= |env.files.state()[Name()]|;  // Writes must start within existing content (no support for zero-extending the file)

         requires 0 <= start as int <= start as int + num_bytes as int <= buffer.Length;  // Don't read outside the buffer

        modifies this;

        modifies env.ok;

        modifies env.files;
```

```
        ensures  env == old(env);
        ensures  env.ok.ok() == ok;
        ensures  Name() == old(Name());
        ensures  ok ==> IsOpen();
        ensures  ok ==>
                  var old_file := old(env.files.state()[Name()]);
                                      env.files.state()  ==  old(env.files.state())[Name()  :=
old_file[..file_offset]
                                                                                + buffer[start..start as
int + num_bytes as int]
                                                                                 + if file_offset as int
+ num_bytes as int > |old_file| then []
                                                                                                  else
old_file[file_offset as int + num_bytes as int..]];

}
```

File compression.dfy

```
include "Io.dfy"

method ArrayFromSeq<A>(s: seq<A>) returns (a: array<A>)
  ensures a[..] == s
{
  a := new A[|s|] ( i requires 0 <= i < |s| => s[i] );
}

method GetStringFromByteArray(b: array?<byte>) returns (s: string)
    ensures b == null ==> s == ""
    ensures b != null ==> b.Length == |s|
    ensures forall i :: 0 <= i < |s| ==> b[i] as char == s[i] && 0 <= s[i] as int < 256
{
    if b == null {
```

```
        return "";
    }

    s := "";

    var i := 0;
    while i < b.Length
        decreases b.Length - i
        invariant 0 <= i <= b.Length
        invariant |s| == i
        invariant forall j :: 0 <= j < i ==> s[j] == b[j] as char
    {
        s := s + [b[i] as char];
        i := i + 1;
    }
}

method GetByteArrayFromString(s: string) returns (b: array?<byte>)
    requires forall i :: 0 <= i < |s| ==> 0 as char <= s[i] < 256 as char
    ensures |s| == 0 ==> b == null
    ensures |s| > 0 ==> b != null && b.Length == |s|
    ensures forall i :: 0 <= i < |s| ==> b[i] as char == s[i]
{
    if |s| == 0 {
        return null;
    }

    b := new byte[|s|];

    var i := 0;
```

```
    while i < |s|
        decreases |s| - i
        invariant 0 <= i <= |s|
        invariant forall j :: 0 <= j < i ==> b[j] as char == s[j]
    {
        b[i] := (s[i] as int) as byte;
        i := i + 1;
    }
}



method {:verify false} callCompression(ghost env:HostEnvironment, src_name:array<char>,
src:FileStream, dst:FileStream, isCompression: bool) returns (success:bool)
    requires env.Valid() && env.ok.ok();
        requires src_name[..] == src.Name();
        requires src.Name() in env.files.state() && dst.Name() in env.files.state();
        requires env == src.env == dst.env;
        requires env.ok == src.env.ok == dst.env.ok;
        requires env.files == src.env.files == dst.env.files;
        requires src.IsOpen() && dst.IsOpen();
        requires src != dst;
        requires env.files.state()[dst.Name()] == [];
    modifies env, env.files, env.ok, src, dst, src.env, src.env.ok, src.env.files;
{
    var ok, src_len := FileStream.FileLength(src_name, env);
    if !ok {
        print "Failed to find the length of src file: ", src, "\n";
        return false;
    }
```

```
  var buffer := new byte[src_len];
  ok := src.Read(0, buffer, 0, src_len);
  if !ok {
      print "Failed to read the src file: ", src, "\n";
      return false;
  }

assert buffer[..] == old(env.files.state()[src_name[..]]);

  var cmp := new Compression();
  var buffer_string := GetStringFromByteArray(buffer);

  var str := "";
  if |buffer_string| > 0 {
      if isCompression {
          str := cmp.compress(buffer_string);
      }
      else {
          str := cmp.decompress(buffer_string);
      }
  }

  var cmp_buffer := GetByteArrayFromString(str);

  if cmp_buffer == null {
      print "Source file ", src, " is empty", "\n";
      return false;
  }

  var cmp_buff_leng : int32 := cmp_buffer.Length as int32;
```

```
    ok := dst.Write(0, cmp_buffer, 0, cmp_buff_leng);
    if !ok {
        print "Failed to write the dst file: ", dst, "\n";
        return false;
    }

  assert cmp_buffer[..] == env.files.state()[dst.Name()];

    ok := src.Close();
    if !ok {
        print "Failed to close the src file: ", src, "\n";
        return false;
    }

    ok := dst.Close();
    if !ok {
        print "Failed to close the dst file: ", dst, "\n";
        return false;
    }

    return true;
}

method {:main} Main(ghost env:HostEnvironment)
  requires env.Valid() && env.ok.ok();
  modifies env, env.files, env.ok;
{
    var num_args := HostConstants.NumCommandLineArgs(env);
    if num_args != 4 {
```

```
        print "Expected usage: compression.exe [0|1] [src] [dst]\n";
        return;
    }

    var compression := HostConstants.GetCommandLineArg(1, env);
    if compression.Length != 1 {
          print "The first argument should be 1 for compression or 0 for decompression, but instead got: ", compression, "\n";
        return;
    }

    if !(compression[0] == '0' || compression[0] == '1') {
          print "The first argument should be 1 for compression or 0 for decompression, but instead got: ", compression, "\n";
        return;
    }

    var isCompression : bool := if compression[0] == '0' then false else true;

    var src := HostConstants.GetCommandLineArg(2, env);
    var dst := HostConstants.GetCommandLineArg(3, env);

    var src_exists := FileStream.FileExists(src, env);
    if !src_exists {
        print "Couldn't find src file: ", src, "\n";
        return;
    }

    var dst_exists := FileStream.FileExists(dst, env);
    if dst_exists {
```

```
        print "The dst file: ", dst, " already exists.  I don't dare hurt it.\n";
        return;
    }

    var ok, src_stream := FileStream.Open(src, env);
    if !ok {
        print "Failed to open src file: ", src, "\n";
        return;
    }

    var dst_stream;
    ok, dst_stream := FileStream.Open(dst, env);
    if !ok {
        print "Failed to open dst file: ", dst, "\n";
        return;
    }

    ok := callCompression(env, src, src_stream, dst_stream, isCompression);
}

function method NumDigits(n: int) : int
    decreases n
    requires n >= 0
{
    if n <= 9 then 1 else 1 + NumDigits(n / 10)
}

function method ToString(n: int) : string
    decreases n
    requires n >= 0
```

```
    ensures

        var s := ToString(n);

        |s| == NumDigits(n) && forall i :: 0 <= i < |s| ==> '0' <= s[i] <= '9' && 0 <= s[i] as
int < 256

{

    if n <= 9 then [n as char + '0']

    else ToString(n / 10) + [(n % 10) as char + '0']

}



predicate method IsInt(c: char)

    ensures '0' <= c <= '9' <==> IsInt(c)

{

    if '0' <= c <= '9' then true else false

}



predicate method IsAlphaChar(c: char)

    ensures 'A' <= c <= 'Z' || 'a' <= c <= 'z' <==> IsAlphaChar(c)

{

    if ('A' <= c <= 'Z') || ('a' <= c <= 'z') then true else false

}



function method GetInt(s: string, n: int) : string

    decreases |s| - n

    requires 0 <= n <= |s|

    requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256

    ensures

        var integerString := GetInt(s, n);

            (|integerString| != 0 ==> forall i :: 0 <= i < |integerString| ==> '0' <=
integerString[i] <= '9' && 0 <= integerString[i] as int < 256)

{
```

```
    if n == |s| then "" else
    if IsInt(s[n]) then [s[n]] + GetInt(s, n + 1)
    else ""
}

function method ParseInt(s: string, i: int) : int
    decreases i
    requires 0 <= i < |s|
    requires forall j :: 0 <= j <= i ==> '0' <= s[j] <= '9'
    ensures ParseInt(s, i) >= 0
{
    if i == 0 then (s[i] - '0') as int else ((s[i]-'0') as int) + 10 * ParseInt(s, i - 1)
}

function method RepeatChar(c: char, occ: int) : string
    decreases occ
    requires occ >= 0
    requires 0 <= c as int < 256
    ensures
        var s := RepeatChar(c, occ);
        |s| == occ && forall i :: 0 <= i < occ ==> s[i] == c && 0 <= s[i] as int < 256
{
    if occ == 0 then "" else [c] + RepeatChar(c, occ - 1)
}

class Compression {

    constructor ()
    {}
```

```
function method repeatOccurences(cur_char: char, occ: int) : string
    requires occ > 0
    requires 0 <= cur_char as int < 256
    ensures
        var s := repeatOccurences(cur_char, occ);
        (occ <= 3 ==> |s| == occ)
        &&
        (occ > 3 ==> |s| == 2 + |ToString(occ)|)
        &&
        forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
{
    if occ <= 3 then
        RepeatChar(cur_char, occ)
    else
        ['\0'] + [cur_char] + ToString(occ)
}


function method helpCompress(s: string, cur_char: char, occ: int, index: int) : string
    decreases |s| - index
    requires 1 <= occ <= |s|
    requires 1 <= index <= |s| && 0 < occ <= index
    requires forall i :: index - occ <= i < index ==>  s[i] == cur_char
    requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
    requires 0 <= cur_char as int < 256
    ensures 0 < |helpCompress(s, cur_char, occ, index)|
    ensures
        var cmp := helpCompress(s, cur_char, occ, index);
        (forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256)
        &&
```

```
            (index >= |s| ==> (occ <= 3 ==> |cmp| == occ) && (occ > 3 ==> |cmp| == 2 +
|ToString(occ)|))

    {

        if index >= |s| then

            repeatOccurences(cur_char, occ)

        else if s[index] == cur_char then

            helpCompress(s, cur_char, occ + 1, index + 1)

        else

            repeatOccurences(cur_char, occ) + helpCompress(s, s[index], 1, index + 1)

    }



    function method compress(s: string) : string

        requires |s| > 0

        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256

        ensures 0 < |compress(s)|

        ensures

            var cmp := compress(s);

            forall i :: 0 <= i < |cmp| ==> 0 <= cmp[i] as int < 256

    {

        helpCompress(s, s[0], 1, 1)

    }




     function method helpDecompress(s: string, fnd_esc: bool, fnd_ch: bool, ch: char, index:
int) : string

        decreases |s| - index

        requires |s| > 0

        requires 1 <= index <= |s|

        requires fnd_ch ==> s[index-1] == ch && index >= 2

        requires fnd_esc ==> if fnd_ch then s[index - 2] == '\0' else s[index - 1] == '\0'
```

```
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        ensures
            var dcmp := helpDecompress(s, fnd_esc, fnd_ch, ch, index);
            forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256
    {
        if index >= |s| then
            if fnd_esc then
                if fnd_ch then
                    ['\0'] + [ch]
                else
                    ['\0']
            else
                ""
        else
            if fnd_esc then
                if fnd_ch then
                    var integer := GetInt(s, index);
                    if |integer| > 0 then
                        var occ := ParseInt(integer, |integer| - 1);
                            if occ > 3 then // If the number is 3 or less, the char won't be
repeated
                             RepeatChar(ch, occ) + helpDecompress(s, false, false, '\0', index
+ |integer|)
                        else
                            ['\0'] + [ch] + [s[index]] + helpDecompress(s, false, false, '\0',
index + |integer|)
                    else
                        ['\0'] + [ch] + helpDecompress(s, false, false, '\0', index + 1)
                else
                    helpDecompress(s, true, true, s[index], index + 1)
            else
```

```
            if s[index] == '\0' then
                helpDecompress(s, true, false, '\0', index + 1)
            else
                [s[index]] + helpDecompress(s, false, false, '\0', index + 1)
    }


    function method decompress(s: string) : string
        requires |s| > 0
        requires forall i :: 0 <= i < |s| ==> 0 <= s[i] as int < 256
        ensures
            var dcmp := decompress(s);
            forall i :: 0 <= i < |dcmp| ==> 0 <= dcmp[i] as int < 256
    {
        helpDecompress(s, s[0] == '\0', false, '\0', 1)
    }
}


method testCompression() {
    var c := new Compression();

    var s := "AAAABBBBCCCC";
    s := c.compress(s);
    assert s == "\0A4\0B4\0C4";

    s := c.decompress(s);
    assert s == "AAAABBBBCCCC";
}
```