Huffman code

Submission deadline:	**2022-03-27 23:59:59**
Late submission with malus:	**2022-05-15 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**5.0000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	6 / 30 Free retries + 20 Penalized retries (-2 % penalty each retry)
Advices:	5 / 3 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop two functions do compress/decompress files in UTF-8 using Huffman coding. There are three difficulty levels in this problem:

- a mandatory part that requires a working Huffman decompressor (i.e. compression is not required), moreover, the decompressed files consist purely of ASCII characters (0-127),
- an optional part that supports the decompression (i.e., compression is still not required), however UTF-8 characters must be supported in the compressed/decompressed files,
- the bonus part with the support for the compression.

Huffman code is a compression technique based on statistic and information theory. It is based on an observation that the frequency of characters is not evenly distributed. For instance, spaces are very frequent in text files. On the other hand, some characters such as Czech accent 'ř' are rare even in Czech texts. Huffman coding analyzes the data to compress and establishes a frequency for each input character. Based on the frequency, the input characters are re-coded. Typically, the length of the code varies between 1 bit and 10-20 bits. The frequent characters are assigned short codes whereas rare are assigned long codes. Thus, the overall length of the input is decreased.

The basic idea is simple, however, the variable code length introduces some technical problems. First, the Huffman code decoder needs to know how many bits to read in order to decode exactly one character. If the input codes are fixed (such as ASCII where 1 character = 8 bits = 1 byte), the problem is trivial. If the input data is in UTF-8 code (i.e. 1 character is 1 to 4 bytes), the input decoder must examine the bytes and assemble the bytes accordingly. This becomes even more difficult with Huffman code, where the code lengths are not byte-aligned and vary. Huffman code is developed as a prefix-free code. It means that a bit sequence which forms some character is not a prefix of any other bit sequence in the code. For instance, if space character is coded as a sequence of two zero bits 00, then bit sequences 001, 000, 0001, 0010, ... cannot be used as codes for other characters. This property guarantees unambiguous decoding.

The second problem is padding at the end of the file (and the related problem - detecting the last character to decode). Since the codes may be of arbitrary length in bits, the total number of bits in the file may be arbitrary. However, our files must be byte-aligned. Thus, if the total number of bits is not a multiple of 8, we shall add some padding bits at the end of the file. It is not a problem. However, the extra added bits may be interpreted as extra characters by the decoder. Suppose we had to pad the input with 5 zero bits, moreover, we code space as two zero bits. The decoded would decode the extra 5 zeros as two extra spaces. To avoid this, we code the data in chunks and we include chunk lengths in the compressed files. In our implementation we add an extra bit at the beginning of each chunk:

- a single bit set to 1 starts a chunk of length 4096 characters (coded using Huffman code),
- a single bit set to 0 is then followed by 12 bits representing the number of characters coded by the chunk (i.e. the length of the chunk is between 0 and 4095 characters). Following the chunk length, there are the compressed characters. The chunk starting with zero bit must be the last chunk in the file to clearly indicate where to stop the decompression. The last chunk may be 0 characters long if the number of characters in the file was a multiple of 4096.

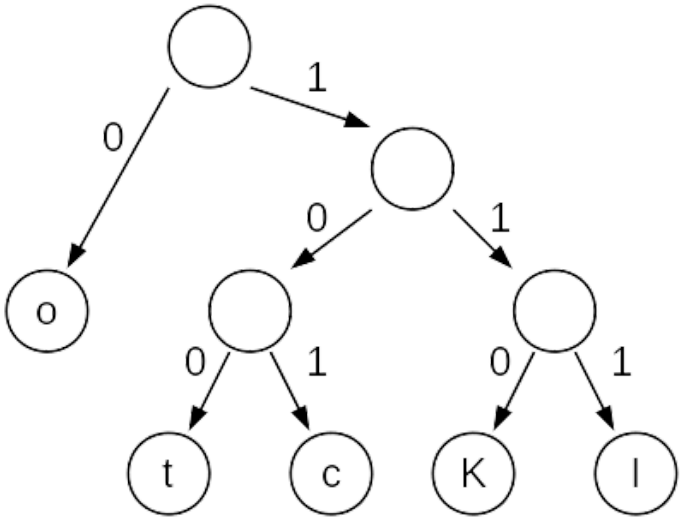
For example, an input file with 15800 characters will form the following chunks:

```
1 <bits coding characters 0 to 4095>
1 <bits coding characters 4096 to 8191>
1 <bits coding characters 8192 to 12287>
0 110110111000 <bits coding characters 12288 to 15799>
<(optional) bits padding the last byte>
```

We will demonstrate the coding process in examples:

```
the input file consists of 7 characters (the word is marry-go-around in Czech):
Kolotoc

An example Huffman code for this input is:
K        110
o        0
l        111
t        100
c        101


Thus, the input is coded as a bit sequence:
K   o l   o t   o c
110 0 111 0 100 0 101

Such input would form just one chunk (subsequently the last chunk),
thus the bit representation of the characters would be preceded  by the
last chunk indicator (bit 0) and the number of characters (7 characters,
as a 12 bit number, i.e., 000000000111):

0 000000000111 110 0 111 0 100 0 101
```

The compression decreased the input size from 7 bytes to 4 bytes, thus it saves approx. 50% of the size. However, to decompress the file, the decoder needs to know the (de)coding tables. We may save the coding table at the beginning of the compressed file. To save the coding table efficiently, we will keep the coding table in the form of a complete binary tree. Our sample coding table would be represented as:



The tree can be serialized into the file using pre-order notation. We traverse the tree in a pre-order manner. When we visit an inner node, we write zero bit to the output. When we visit a leaf, we write bit 1 to the output and we add the code of the character being coded by the leaf. The characters is stored in 8 bits (for ASCII characters), or in 8 to 32 bits (UTF-8 sequences). Note that the codes for ASCII characters 0-127 and the UTF-8 sequences for characters 0-127 are the same. The difference comes with characters not included in the ASCII set. In our example the characters are all ASCII characters, thus they are coded using 8 bits. Thus the sample Huffman tree will be coded as:

```
Pre-order tree traversal:
0 1 'o' 0 0 1 't' 1 'c' 0 1 'K' 1 'l' <chunks>

Characters replaced with their corresponding UTF-8 codes:
0 1 01101111 0 0 1 01110100 1 01100011 0 1 01001011 1 01101100 <chunks>

The same with the chunks from the previous paragraph:
0 1 01101111 0 0 1 01110100 1 01100011 0 1 01001011 1 01101100
0 000000000111  110 0 111 0 100 0 101

Bits rearranged into bytes:
01011011 11001011 10100101 10001101 01001011 10110110
00000000 00011111 00111010 00101xxx

Zero padding in the last byte:
01011011 11001011 10100101 10001101 01001011 10110110
00000000 00011111 00111010 00101000

The same as bytes in hexadecimal (10 bytes, example file test0.huf):
5b       cb       a5       8d       4b       b6
00       1f       3a       28
```

Thus the original text (7 bytes) was compressed into 10 bytes. The increase in size is due to the size of the coding tree. If the input were longer, the extra size of the serialized tree would be amortized.

When decompressing the file, your routine must first read the serialized tree. To restore the tree into memory, read the individual bits. If 0 bit is read, create an intermediate tree node and recursively call function to read the left and right subtree. If 1 bit is read, read the following ASCII/UTF-8 code and create the corresponding leaf node.

Huffman tree can be used even to decompress the file. Once the tree is read from the file and restored in memory, the decompression routine needs to read 0/1 bits from the rest of the file and traverse the tree accordingly. If 0 bit is read, the decompression moves to the left son, bit 1 moves to the right son. When a leaf is reached, the corresponding character is appended to the output and the tree traversal starts again from the root. Repeat until the end of the last chunk.

The task is to implement the two functions `compressFile` and `decompressFile` as shown in the attached archive. Both functions take 2 parameters - the names of the input and output files. Both functions read the input, compress/decompress the contents and write the result into the output. The return value is success/failure indication. Both functions return `true` if the required compression/decompression went ok. If something was wrong during the compression/decompression, the return value must be `false`. There are many reasons for the functions to fail: input file does not exist / input cannot be read / output cannot be written / invalid file contents (UTF-8 codes were invalid) / ...

The decompression function is reasonably short (the reference is approx 200 source lines, including some debug routines). The decompression function is mandatory. If not present, your program will not be awarded any points. On the other hand, the compression is more laborious. The reference with both compression/decompression implementation is approx. 500 source lines. This is more than we assume reasonable for a homework, thus the compression routine is left as a bonus for those who like the problem. There is an explanation of the Huffman code construction on **Wikipedia**. If you decide to skip the compression, please leave the compression function in your source and let it return `false`.

The input test (and thus the characters in the serialized Huffman tree) are coded using UTF-8 encoding. The mandatory tests use characters in the range 0-127, thus the UTF-8 code sequences are always 1 byte (8 bits) long, making the processing significantly easier. If you do not want to deal with the UTF-8 coding, implement a program that reads fixed 8 bits/character. Such solution passes the mandatory tests, thus will be awarded some points (less than 100%).

STL library is available for your implementation. The list of available headers (libraries) is shown below. Your implementation can use the STL libraries, however, it does not have to. If implementing the decompression only, STL is not likely to be very useful. The STL containers are intended for the compression implementation.

Notes:

- Pay a special attention to the file I/O. The testing environment tests your implementation, it tries nonexistent files, unreadable files, files with invalid contents, ...
- UTF-8 coding was outlined n PA1. You may need to read more about UTF-8, a good explanation is available on **Wikipedia**.
- Optional and bonus tests: the testing environment tests input files where the UTF-8 codes are broken (e.g. there are missing some bytes in the multibyte sequences). This applies to both compression (input text file is broken) and decompression (UTF-8 codes in the serialized tree are broken). In these cases, the expected behavior is to return false.
- You may use either C or C++ file interface, the choice is free.

- There are pairs of input (.orig)/compressed (.huf) files in the attached archive. File test5.huf is invalid, decompression shall fail. Therefore, there is no test5.orig in the attached archive.
- There exists more than one Huffman code for an input file. For instance, if we swapped 0 and 1 bits in the sample above, the resulting code would be also a Huffman code that could be used for the compression. Similarly, if we swapped 0 and 1 bits in some subtrees, we would get further different valid Huffman codes. This applies specifically to the compression function - if implemented, your compression function may result in a compressed file different from the samples provided, yet the implementation may be correct. A good guideline is the length of the compressed file, the length shall match. The testing environment tries to decompress (using the reference decompression routine) the file previously compressed (using your compression routine) and compares the uncompressed result.
- The **compression** method is not exactly specified for empty input files, or for uncompressed input files that consist of just one character (e.g., 100 times repeated character 'a'). If the input file is empty, return false (this is specific for the compression, indeed, decompression shall return false too as there would be the last chunk missing). When calling the compression function, the testing environment always uses an input file that contains at least two different characters.
- This homework solution may be used in the code review. However, the code review expects C++ specific constructs in the source (classes, ...). Do not submit this homework source for code review if your implementation uses only C-like constructs; such solution would be awarded only a few points. A final note: classes may be surprisingly useful in this homework.

Sample data: **Download**

Reference

- **Evaluator: computer**
  - Program compiled
  - Test 'Zakladni test se soubory dle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.004 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Mezni hodnoty (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.001 s (limit: 3.996 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Nahodny test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.035 s (limit: 3.995 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Nespravne vstupy (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.001 s (limit: 3.960 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test osetreni I/O chyb (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.293 s (limit: 3.959 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test dekomprese (UTF-8, nahodny a nespravny vstup)': success
    - result: 100.00 %, required: 75.00 %
    - Total run time: 0.085 s (limit: 4.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Test komprimace (nahodny a nespravny vstup, I/O)': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.481 s (limit: 4.000 s)
    - Bonus test - success, evaluation: 130.00 %
  - Overall ratio: 130.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.30)
- Total percent: 130.00 %
- Early submission bonus: 0.50
- Total points: 1.30 * (5.00 + 0.50) = 7.15

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**32**	--	--	--
Lines of code:	**448**	**14.00 ± 9.19**	**37**	`CContext::ReadUtf`
Cyclomatic complexity:	**140**	**4.38 ± 3.30**	**13**	`CContext::ReadUtf`

6 **2022-03-26 23:10:07** **Download**

Submission status: Evaluated
Evaluation: 5.0000

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample files': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.007 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.002 s (limit: 3.993 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.031 s (limit: 3.991 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Invalid input (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.002 s (limit: 3.960 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test I/O failures (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.376 s (limit: 3.958 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test dekomprese (UTF-8, random and invalid input)': success
    - result: 100.00 %, required: 75.00 %

- Total run time: 0.111 s (limit: 4.000 s)
- Optional test success, evaluation: 100.00 %

- Test 'Test compression (random and invalid input, I/O)': failed
  - result: 0.00 %, required: 100.00 %
  - Total run time: 0.035 s (limit: 4.000 s)
  - Bonus test - failed, evaluation: No bonus awarded
  - ☑ Failed (invalid output)

    ```
    compressFile: student reports false, ref true
    ```

    ☑ Input data **[117 B / 117 B]**

    ```
    %{ix~>- xx"~{{~ m>Tq~ 0{~>%i> l`l10 [~[>Rm{ ~xiu xKux% `%>%~x 0x~~) l"l~%|% ~~[s0v" u%X~F
    x^`x{-1 [)u| `1{x1x v~~$~-
    ```

  - ☑ Failed (invalid output)

    ```
    compressFile: student reports false, ref true
    ```

    ☑ Input data **[119 B / 119 B]**

    ```
    p|F# ICfZ\ '"%)2 &&,,F)Fg f=`jf *,&K7 &j,hf G&QS37 2L|}W G&&Gd&F& Ch=Q#F *f,,
    *6G&'} Sf&`& VLjGif *SMj &FF,Fj' ,*,j `,F
    ```

  - ☑ Failed (invalid output)

    ```
    compressFile: student reports false, ref true
    ```

    ☑ Input data **[134 B / 134 B]**

    ```
    .-Wgkk1 -.W~ --vC-y0W JyyCk~= .u-^- W@S-~00- ~5-ZJ)W #S~>k JWtW =gC-~k :>11-
    0^W-0k-~ -vmS 1)l-,> C>kn.J *i}:~1n Ck->>0>- W01- --0:=10
    ```

  - ☑ Failed (invalid output)

    ```
    compressFile: student reports false, ref true
    ```

    ☑ Input data **[211 B / 211 B]**

    ```
    B9Y--,Y YY9c9 ,YG|{cY Y9@GY## BhYYz YHYGY H~B@Y Sfc~,] *NH9 YhC@ >#)cB+Y f|h@H99# H~@', z~HS*Y
    J#YNY YrRGYB9 |hY+@ 9B}") |@N,9!?h Y!!GY Yc)#Y9 9UU9Bz@ <|!RczY uYm9Y>-@ 9z>ziY!
    H@YY9 U+Y+ ~u9ciY9 mYmS GYY|9Y |Shc
    ```

  - ☑ Failed (invalid output)

    ```
    compressFile: student reports false, ref true
    ```

    ☑ Input data **[7779 B / 7779 B]**

    ```
    R3*-v c?W[kCOcv 5@(M@Cg @D*?@@rOW C'H@ @CgO@cR eWWW3W S)C@; z(6D @zBWk- kCBOkc
    -r[-* sCD@#cs @OM7HL LHCv*De@ cBL@@B e'E@M? @MCc@ u?|0?kW Hu'@C*S'k? LD3@@vO Ss|3 Zv;hS WBcZkM M-D@*@
    B3hC@ @@MC@-@k BLkv@-3I OMBD@D @|?| c)D~3W@ c@k7c @-@crCq ,=C5@O3 LD?I
    BW[*? ]D@Sg @D@kS#R??@ D(3cZM 3@O@D@BE @C@C@ SW*5 k]7@k L@?Bcc ?Be@@?E' 3*3@O *Wkc
    HI@@"cLB 30D?*C ?w,B@-? zH@C W@@O3 DEHOC@*C D'@-? cW-?k B@kr@c k]MCL?
    H@@k W@|s?uk|@ -DC|@3 @CF@H csW@(q Wk?kC? B@@3B?@ 533hL@ L?*CsE 4D@kB3kc ?E@MD
    @?B*;?3' ?@7DcD;M? -CHckDEH @rCHS3H D?k@ uk@W@IOL Wz@k6c3@ ksg~,@
    L3rBk *zEk;H @E-vD 3u@EO D-W3@L? *?]] DMDv~B F3r3kk3 ?SecC?D ccE3@ @3SDk MBD'@R@ WMIC@` @[kD@5|
    c-@OF k3sgW@; |z@@xWD E3ExHk Cv3OBLW `kH] H@(M*k w?Bg5 ?@CMER@ O,@F], *s33? x3OB6 `HkD
    rBBo[ O(zMC cMM*ZO@@ MIDC |C*@S*( k?v?cu M@5c,g @?k|D~ O@gk3k D3DIgwL Wk3C3F@C
    CD'O*B gj@'D @C]W'3'C @@Cg W53@ ]aO-MC BMk5gR Wcg*B3I kM*R@k @3Ik'k CB@3c ?B@e53;L BC'?*v
    ?kB;kM kCZCS-L? 3ksC|Og kqD3? 5;CCgcz @EHk *?3MB 7(xHE5 Dk-@BL@ ?@?)W gS@#v|
    @3-*'@@ ~*6RS BxI3WBW De@'@k DD-O ?cCWk @HLMR 0B@~W @kv' W-LD@c 4@3@7@ *BC*c ]I"c]|k ED]g3 WkC@B@> k*k@@
    BC-I3 O'c5 B~@* B-*DH vcr@]k?k 3Dx*@*@O3 *WMD@*--L ?CO0ky 6g0|WzD *@-EI s??-
    @O@@@ MEgO5C @@@eL *BLWxCRE/ @R*'- xEW@Dk DLk]@ |Ekz@@q@ WR@HW @xDCD@ CgCR @?5Hu* vS@I*Hu
    gM363E# CZc5@ W@@@cv |kO@ HH@k5 k@B@'? Bk3@(3@ CgCkCHD? cRvO3 x[c@*5 ?3MO-Hc 3M?;@@
    BL3SOW kL@k MLC~ -C|ozg MBLM* ;k;k;Lz@| C@@@I( (-C*3 ?-k;@S @D@E*c@3W @36'5?0FD gEkScO5OD LLDv3I3 Cg3k
    C@?WvCco Dk@@@@M ~3xk@ CO?W? @kDDWCH= -cR6k ]|C*c z*DR|L vBDkHO WkDkg ,Ck3?C -3(@36
    D;DOc] *D3kk #B]c[|k- @3c??g vDEI@7 3ek@4I @-]-Z LLI@3W k3?k@;*v 5O@C]H?k |B]CH HkL*
    $S@c@@D c*k@@k [3Bk7?M D@Bk 5kkOB(C 3*3(3O @?305- ZDHLW z5q@* kOL-EDH @CIc6M
    C3sOk3 CL@|' D@@OREM C@IkEMCCg- D*OHHD kC*5@Cg @]@I, IS?M~ rkBk@ @kk?3> D@@J Oc3@W@ @@>5-Okv cD;3e*
    kR?OC B5c3RZM[s v3*B@#@ @g@Ew @c@x3|3M3 vH?7@k3W eguLgx 'O)LC@ ?@Zk 3@63k5 @3@@cC
    BZ?j7,D WO@;B@Cu [MELzgc *vz*@ ,u*#ZvW Cvuc;3@ x@Zs5 HC-ek? 5LO'? WZe7]
    @C@gC- Cg-Lc BcM*IW @*c@3jc@ LBE?k E*-3S3 ccs-kM @LD*q*@ @3BwC@'
    g]DO?-k ?JW]v* B@@WH cgLW @z)kO @LH'6Kkkc c@?-k @*'kz[ 3O7@@BD WW0kcC@ DC'EgZ
    ?3M54H @CL3L3$ @OzOHD @5-33k *3[-|C( 3LDM~WR kL@WBC HLxkO? ?zDc -?k*@ w37H *@gcC HHBk ?u6B@sC R*O@@
    @@Dr' *M*W- (@@s@ MkWW@ WkL"3 @kLq@ BsC-Zxk *kckg Cc@B |F3@ 0E],W? *kC5kW@ @E@?c| @@ck
    W7@W @@HWkM OL@k*W k*3@M;? Ck@~~ 0?xHEC u)4k| BWMkW (k@3k D?O@L~ @kM@gOC *@FD B@kuD5
    ueC@ ~5@B@gs@ CCW?DH D@3@k3O k@=3Bxe cI~k]L Cevz@[ L|@k- kc|v ~?E@@O uL@LC-B @@BD @@W*u *@@I?@@g 3?LWg
    DBH3WO Bk;@]H BC3(BWM @BDkvH L?@3k ;vkI *-3R|5@ @skB@ M~3@CkB- H*u@O kw@I
    >a?[WC3 @?30Lkc CL33kO @D0)kIWD D8W3@"  W0L@cc0 c-HkMB@w 3x3-5 ]B@kxD @DWW3 #BBD*WW
    ]?|@*W @?3B kW~D] kB3[ @@"@DB OMCeM B?@x OC@]B@ -CB3Z]I- Cg~sk k@?Mk Dc@c3OD
    eB@Ix@ @;;?H@c @3@D vOOk;k 3?BL?C BO@CO ek@-k sMHF Be*BS@]) xvcM0@k
    Bk'OsCM kx@3c CBc-Z gc[@v cEc]@?- LEDO O(-C? CELDCO @@@'DW-M @]cIL- cc>Z3@k
    @@3kkc @33;D B5OWvDe H(-C@] ECDg@5@K ge~BkMR DD[B3 EWHD *cMw@3 cBB@D
    Dzkg@O H@cC@cZ @CI?0=x k3Hj Le@BIc @=C; kO3D @F@3**v@ 'WI@R@ BD-k MB]37r#*W
    ck??kB@ vO@kBRg@ SHB|@ BgD*Bk3De MD'kB@ W?ZBZu 3@ejDx M5sELk@ Sz?@k S@-@C
    @kk@3I *CWDcRs M@c@ C`RLC3 ?O;vq~ v*@W 5C@O?vE6 @v[=3Lk CD33I gBHgH| L#E@gcC u;g3DD3 *kkO-E@ @MS?@[
    ~vcBw @@@'7W @*;BsM IBO0C -kOLzOC@ *cCE5*@ 3?gL*C @c4@LH? F3S[CC'g Ee*3eS 35OELH@ g-'F@@ cEB@k
    Wk?KkD ?,k;R~c SHI*@W@M3 MO"RR@7W@ ws0WJuEB Cc3c*@W# 535D-SC BB@|*B H@3CHcC B*C5 ?DM[3k B3v@O=-c ?B@OWW@
    @gW@DM3 CI-M- *Mcs ch@EL WBkC|*k B?*@kk x@kvC(3 k@cH ??OM g@@@vc- k[ZR?k kg5?k( 0SrMvW B3?*@ EDk~ *LCICB
    @3E5 cL@@DcL ]kI3(Z Dg3Wr@ ?vEM#H MW@@3L CCD3=H3 ;D@6@M] CBCDB? DBk*0
    @R[Mc|5? @-O3c *O3D0Cc @#@DD ]*@3 B@O7CB MrB(S*@W L@@EB#@ M0B*@ kMIWR LCJO@0 3-sDZ
    ```

```
@-]'ggkcH @3HLCc@ @D]5c L*|) H;@BC C@3RO@ H?0@ c*S3@ @@k@-E 6*|Ck@ Ec*~ O@3zB(?
LB@Dc? @WCWDc? kx@|3c @og@B@g@ MOL@C C-@W@@?? 3[yk @?*DDk @vDC @ Z7]u
@eCMuF@k @|EvW|@ @Ck'Dkce* D@3RzW 5E*k @@k@3Cu ~]Okk@@ 7*3WDzI gO@kL ??x*L LL33R7 k*@,@MD -OHCcc
@*@k 'gB@M7 C?DLOC@ D@k@ 6@@-,O 3DCk?@@M BBM-kk `?;'k43 @B@Hq@ Ck]k*Dk L@@@(k
@kr@BD; 3COk6L7B3 3@ckD Og-3I' E'B@c* 7@RBO *@BCW ?M;?Hs ??v@ cDD3 @E7D33@ *35I*C k3Dkk@
@kg@@ ?BM7]BSD ?85Bk@@ ?u[C5 3@k@LI* @|]W@k kBO7L ]-*Bc?@k WvWDc
kE;@c| k-LM"E@kC BFcWRS@O@ vk-DO DOkMDk OEe@F? k@3OD @k|k @'3'@L 3EECBB[@ BxSug@C
-kg*I kO*@HBC F]rL5? IDv3,@C]g v@3e vB@c3k 3@0u k3B*'v Hvv@ 3;L|H@? cDCk@c 3g?-|-k Hgu73kv7
3k'CDC3 cc?c LCk]-O C@@k@? c~Dge] @5@`Sue 3k]c-LLH M?3SDC3 W33k@6 S?B5? 3]H3@ k@Be]C
k~?I33O* ReOc ?@|@ geOEDO0@ -B(kD* @OZC@xv wM*@* @'r|B B??@B CEke~3@E- ~gC| 3DDCD 3?c0
B@CC@3 @@@kkW5 0-@DB- BLvkCkk@ D@@@Fk-? @@3cc H3w@( 3BMr@ ODDB@c (s?@D@C $IW*W
kkL-kx 3B]5|] -c3k c3;ckkS@B uWk- MO0I-]3 @O@c kLcScx u(?@'C c3k@-3~g Bk~@O@B
E|W?*k zD?*?DSC 5*qEc @?kD CWD@5@ D35e k*;@g@(@@H ?k*D@@ O@CWR k3H3O C"kBRL *kIk@
cH@Oc kFC@@x zsCk3@@ @kL33 Dc@Z5 >C@3@OD c*"kkWk ~@333gLg @CRDk |@@B?kW -S0kc@H O@C@O
c3BFkH sO?3 0-zDLkk3 BL3* k-W)D@5 rELkC ?B@~I B@C*CKg z53@ M?C3 c@DD ?W~?k ZE-J0'c g3@*I~@ M=c?(*
@5(B]M -O@DDDBS c*@zkc@ @@M@~ I@Ck?@s Z>@R?@3Dk @L--B k@@33 kC@@@ @*c5Dk E;3D[ 5kE@ ?'B@D D33@B @@*CO
C?WDk( CeWcOxkBB3 ?@J*@jcB *3@@CMx DI,'C B@LRkL3 *MkD ,cD6E?B 3gEk ;q[D '3kHg O@CL @k@Bku
W3@W* DL-DWk *BHD63 O@MBe @D*kO OWIW???e @ZCB 5|5-@ E*@@3?? @q-M BRCk @C?H"@ *e,Cc-
3kkOI (D@kEC @B@k7g IccDL LCO` LE3;CO eCDW 8LWkB@ gO@B'3? R~3CHZk gc-c3 [@3B-@BC| HR'@
*v*3*D@ DkHD3 C;?D?k RzCE@S 6kg*D '*Dk@SD D@vO@ ~ucBDr WMkC3 W@B*D k@O@gDMg S7Bg|B 5@H@? z;Mc MWIMkB
*-c;@? @*'BRx LIHC7gBc3W 33[xk3] @@Bkk5? ?k?*@ =W?CO@ @?*C" ;D@"*3Dc @vcz 3@B5 Bx@@WHBk
MxO@k? WBD0@C @@3@-* L>;?DC @OLH] [C3DEWL k@c@@D g;3* M?CC k7>c5- WWZ@*
c@*3k k'*;3 ?O@3Dk @DsWM CE0' c;~WI 3@?g;C O3@L @rB' kD~c(B3D vkzO eL,EHH@v DHCD
cW@HC' u@@*3 #C@c3kD @xB@30* DCc?C3 C6LkC] HLWB Dk3O 3>@HB@ O`IOB @F@Bk@'@c @?CR@ L@k]@g
?kuBv @HBDrW@ C3@W*H@Z @7C?* C@W30-'H zMC@H@M @k3C35* Hg3;7C (@k@@@k@@ D@k0B3- sWHks kWO->
5@?@@# ccCk M0|@W k8@|D?, ?DOxv* g?0COsW M@C@@@ Cc7Mk7JB@ |SgWL3 kSDO@ 3W-Og Z@DR3? ]Dg|k?OR
'e?D#33 (@]-3@3 kv?@w W@g5 *5k@ C3-@3- Ze*Os @a'MDDg' k@L@RW[ 3E@B@0LW) g]M*kH @Lk# ?BkDv
OILg? SB*7 x0-LE MD(z; @@@kH?RO 3z3k;W@L @@u$sqk LS?k Mc|M3 D@k@5?]* @'B?- vM@@O
5@@C@3 @@LxvM c@v3, CD$Z@@ 03kC [C@@e@S =D3Ok@s Bv5@@ ?k-,IEDM @k?@Dk g5Hkkk c6-@C BOkk@L 3*DkI @k-|3CkD
RH5@ g?k@0k ck-> cc|k[ @@?Og*@ ~@Dc c@3[ CkLu@k *?E3@ Lc*@D ccD*H3W Bg](;S @gk5LZ- Z@C*@xk kDv@]5@ kC@5@s
(k?E#W? >ck'@W kBWSC] 'HCk zS|z u3kE|*SW DRL?-kC@ @B5@ECD @*-JO;? kv3k@;(
)@*u BB0H;( CO3D-@ W@-6@ Fv@H*]u3 ;5gZ3 OkE- k@kCE@ 3@WDB @@3WOB# 33W@DeO@k cBk=*@
cZ@DW H@*'?k B7-> zg*R OukI@D @;O@Dk -*,@c= *@L?E( @?D*M 3EO@D gHDcW@?L FO@@C| H3H@k kcWk* ?cMk[?B
M?M*v -kORk R5**@H k]OZ@@ @@;Rg @@30E3B0 ?3B[*C v~-RZ @*(gO?' #-,@CD @B@3D -;0@?k?
~LCL3 gqC3;;@e I73W@@ ?@Hu k@CvuCL qDqE3Rk @BDDc@H D35W @Ck*O?- gu'?k#Z MCkD@ @s@@Z 3c@3ZDC
-O@@? e3WDc'@ 5@gk*DB3C *HW0 ?Dg?-D  D3DB7Du 5;3@Bz @0Lk@W *uck@@S 3-gBo B~3sk
0c8MSB RWwS?3 ]kewk* W?@OBDC zB?-Z MC33; D~@3Lv@ WgcLv BkC@S ?-WcC@ *xL3EC
*H@|C*c- 'cL@L? *|x*? @CB*# ?@(Wu 5k5@?5 CBBc DOE?O(Ckk D@BkB? zc53*e]C Mk]@;kO*
BL*k *-kWWSM gcWHM @]kM' RD'3* @@@F@3 ICkLD@ 33B@3@E |z@]k @"D33Bg v(@k sDHD sRk#MLk kwE@B*
*u*'B@ C3?7; @5k-|BDBe W@c@~- F3*DW@ cCZk@C3 *C3r gHk~kW' ROvO@>* HE3kk3C3 uZB(
BC'ED k,kx[@c L@J3 FsC;3@@ 6D-k@ DB[H3|* vZCWWH ek;Byc s`33Dv @?@*R;; HLkDk CDc3
H(3k7 ?33@5HH c@-cHW@ @Ck-D '@Drk ?k?@kcDM DOD@] Hx@D RkD],5k 3c`DBRRc ~u?zL @g'' H@k-
okk3x WEL[ M)-; r|H]W@? CcD?OI kBeDk** @O@k @kc5@gHz k=@0D*3 MCWD3@ *k@)HIv 53|* Dk WOCL' @Ik* 7@;u ?6@B
HS-3c 3WW?$ BDW;|kL @533(B xF--kk 5cg?@? L@gc@k P@?C -3kC7?@ E|WBc LBC@3 LCL@kZWD
Zse; D3ZJk @]*35 @cuL ~@kF?R0 WO@5@u? D=rSx@ O-cgx HcESO ~Dc@k DuEocL Wx;3g7
L)@>DML E@CO k35W3 8W?C 'kE|*@BW DCkk?B kx3]k@W z@|Lo @D@M3 @KqOc u@@RBc~
3oHI5 CcqDDC @B|BHcq IesDB?HC LL3@@ u*'kCD-@ @@Mk@H k?@kW@| 3w@E@3
MB@eg kk3Os*O* 5c3C Ck-kk D@OD;c@ Dk?C*DB x@@OE7 -WDW@@ C3D-W3@M
@C7O >BC@ cB**' @|k@@F @rZc]MD BIC5** @@@37~~ |B=CkM c5k@??@ ?@@Sks] >ck3] g@|?"@3
```

- Failed (invalid output)
- Failed (invalid output)
- Failed (invalid output)

- Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00)

- Total percent: 100.00 %
- Total points: 1.00 * 5.00 = 5.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**10**	--	--	--
Lines of code:	**191**	**19.10 ± 31.85**	**111**	decompressFile
Cyclomatic complexity:	**77**	**7.70 ± 15.81**	**55**	decompressFile

5	**2022-03-14 17:38:12**	**Download**
Submission status:	Evaluated	
Evaluation:	3.7500	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample files': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.009 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.002 s (limit: 3.991 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.042 s (limit: 3.989 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Invalid input (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.003 s (limit: 3.947 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test I/O failures (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.374 s (limit: 3.944 s)
    - Mandatory test success, evaluation: 100.00 %

- Test 'Test dekomprese (UTF-8, random and invalid input)': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
  - Total run time: 0.011 s (limit: 4.000 s)
  - Optional test failed, evaluation: 75.00 %
- Test 'Test compression (random and invalid input, I/O)': Not tested
  - Bonus test - failed, evaluation: No bonus awarded
- Overall ratio: 75.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 0.75)

- Total percent: 75.00 %
- Total points: 0.75 * 5.00 = 3.75

SW metrics:		Total	Average	Maximum	Function name
	Functions:	**10**	--	--	--
	Lines of code:	**157**	**15.70 ± 22.22**	**77**	`decompressFile`
	Cyclomatic complexity:	**61**	**6.10 ± 11.03**	**39**	`decompressFile`

4	**2022-03-14 12:17:41**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample files': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.009 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.002 s (limit: 3.991 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.043 s (limit: 3.989 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Invalid input (ASCII, decompress)': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
    - Total run time: 0.006 s (limit: 3.946 s)
    - Mandatory test failed, evaluation: 0.00 %
  - Overall ratio: 0.00 % (= 1.00 * 1.00 * 1.00 * 0.00)
- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00

SW metrics:		Total	Average	Maximum	Function name
	Functions:	**10**	--	--	--
	Lines of code:	**155**	**15.50 ± 21.67**	**75**	`decompressFile`
	Cyclomatic complexity:	**57**	**5.70 ± 9.84**	**35**	`decompressFile`

3	**2022-03-14 12:13:13**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Compile in 'pedantic' mode failed (10 % penalty).
  - Test 'Basic test with sample files': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.009 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.002 s (limit: 3.991 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.039 s (limit: 3.989 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Invalid input (ASCII, decompress)': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
    - Total run time: 0.007 s (limit: 3.950 s)
    - Mandatory test failed, evaluation: 0.00 %
  - Overall ratio: 0.00 % (= (1.00 * 1.00 * 1.00 * 0.00) * 0.9)
- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00

SW metrics:		Total	Average	Maximum	Function name
	Functions:	**10**	--	--	--
	Lines of code:	**155**	**15.50 ± 21.67**	**75**	`decompressFile`
	Cyclomatic complexity:	**57**	**5.70 ± 9.84**	**35**	`decompressFile`

2	**2022-03-14 12:11:30**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**

- Compile in 'pedantic' mode failed (10 % penalty).
- Test 'Basic test with sample files': success
  - result: 100.00 %, required: 100.00 %
  - Total run time: 0.006 s (limit: 4.000 s)
  - Mandatory test success, evaluation: 100.00 %
- Test 'Borderline test (ASCII, decompress)': success
  - result: 100.00 %, required: 80.00 %
  - Total run time: 0.002 s (limit: 3.994 s)
  - Mandatory test success, evaluation: 100.00 %
- Test 'Random test (ASCII, decompress)': success
  - result: 100.00 %, required: 80.00 %
  - Total run time: 0.028 s (limit: 3.992 s)
  - Mandatory test success, evaluation: 100.00 %
- Test 'Invalid input (ASCII, decompress)': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
  - Total run time: 0.004 s (limit: 3.964 s)
  - Mandatory test failed, evaluation: 0.00 %
- Overall ratio: 0.00 % (= (1.00 * 1.00 * 1.00 * 0.00) * 0.9)

- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**10**	--	--	--
Lines of code:	**155**	**15.50 ± 21.67**	**75**	`decompressFile`
Cyclomatic complexity:	**57**	**5.70 ± 9.84**	**35**	`decompressFile`

1	**2022-03-14 12:08:21**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Compile in 'pedantic' mode failed (10 % penalty).
  - Test 'Basic test with sample files': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.009 s (limit: 4.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.003 s (limit: 3.991 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test (ASCII, decompress)': success
    - result: 100.00 %, required: 80.00 %
    - Total run time: 0.046 s (limit: 3.988 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Invalid input (ASCII, decompress)': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
    - Total run time: 0.006 s (limit: 3.942 s)
    - Mandatory test failed, evaluation: 0.00 %
  - Overall ratio: 0.00 % (= (1.00 * 1.00 * 1.00 * 0.00) * 0.9)
- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**10**	--	--	--
Lines of code:	**157**	**15.70 ± 22.22**	**77**	`decompressFile`
Cyclomatic complexity:	**57**	**5.70 ± 9.84**	**35**	`decompressFile`