What is coding — informally?

Definition (Coding)
Coding is algorithmic rewriting of data, while achieving desirable objectives.
Three main types of objectives

1. Reducing the cost of storage and transmission time required (data compression);
2. Preventing data loss due to random causes, such as defects in physical media (error correcting codes);
3. Achieving privacy of communications (data encryption).
Without data compression we would not have

1. high-definition digital streaming video;
2. cellular telephony;
3. high-definition digital cinema.
“Before we proceed any further, hear me speak. Speak, speak. You are all resolved rather to die than to famish? Resolved. resolved. First, you know Caius Marcius is chief enemy to the people. We know’t, we know’t. Let us kill him, and we’ll have corn at our own price. Is’t a verdict? No more talking on’t; let it be done: away, away! One word, good citizens... [total of 141,830 characters]”
An early printed scientific paper

Figure: Nikolaus Kopernikus, 1543, “Polish astronomer” (Encyclopedia Britannica)
Character encoding

- Characters are numbered.
- The oldest standard is ASCII (American Standard Code for Information Interchange).
- ASCII is pronounced “as key”.
- Encodes characters as numbers 0–255.
- Successor to ASCII is Unicode. It aims at encoding all characters in all human alphabets.
- Only ASCII is used in the current talk.
The term *bit* stands for a *binary digit* and it is either 0 or 1.

Bits are digits of base-2 (binary) representation of numbers, e.g. 6 in decimal is 101 in binary.

\[(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 1 = (6)_{10}\]

It is a *normalized unit of information*; we may think that information comes as a sequence of answers to Yes/No questions. A fraction of a bit makes sense, statistically speaking (e.g. tossing a biased coin).

A byte is a sequence of 8 bits. A byte can also be thought of as an unsigned integer in the range 0–255.

\[(10000100)_2 = 1 \cdot 2^7 + 1 \cdot 2^2 = 128 + 4 = (123)_{10}\]
Hexadecimal and Octal notation for integers

- 1 byte = 8 bits = 2 groups of 4 bits; \(2^4 = 16\) “hexadecimal” = base 16
- 16 digits required for hexadecimal: 0–9, A,B,C,D,E,F (letters used as digits)

\[
(1010\ 0100)_2 = (8 + 2) \cdot 16 + 4 = (A4)_{16} = \backslash xA4
\]

- Octal notation uses groups of 3 bits. 8 digits required: 0–7.

\[
(10\ 100\ 100)_2 = (244)_8 = \backslash o244
\]

- NOTE: 8 is not divisible by 3, the highest octal digit of a byte is 3.
## ASCII codes

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>O</td>
<td>#32; Space</td>
<td>64</td>
<td>40</td>
<td>H</td>
<td>#64; B</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>P</td>
<td>#33; !</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>#65; C</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>Q</td>
<td>#34; &quot;</td>
<td>66</td>
<td>42</td>
<td>D</td>
<td>#66; E</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>R</td>
<td>#35; #</td>
<td>67</td>
<td>43</td>
<td>F</td>
<td>#67; G</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>S</td>
<td>#36; $</td>
<td>68</td>
<td>44</td>
<td>G</td>
<td>#68; H</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>T</td>
<td>#37; %</td>
<td>69</td>
<td>45</td>
<td>H</td>
<td>#69; I</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>U</td>
<td>#38; &amp;</td>
<td>70</td>
<td>46</td>
<td>I</td>
<td>#70; J</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>V</td>
<td>#39; '</td>
<td>71</td>
<td>47</td>
<td>J</td>
<td>#71; K</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>W</td>
<td>#40; (</td>
<td>72</td>
<td>48</td>
<td>K</td>
<td>#72; L</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>X</td>
<td>#41; )</td>
<td>73</td>
<td>49</td>
<td>L</td>
<td>#73; M</td>
</tr>
<tr>
<td>10</td>
<td>A12</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>Y</td>
<td>#42; *</td>
<td>74</td>
<td>4A</td>
<td>M</td>
<td>#74; N</td>
</tr>
<tr>
<td>11</td>
<td>B13</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>Z</td>
<td>#43; +</td>
<td>75</td>
<td>4B</td>
<td>N</td>
<td>#75; O</td>
</tr>
<tr>
<td>12</td>
<td>C14</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>a</td>
<td>#44; :</td>
<td>76</td>
<td>4C</td>
<td>o</td>
<td>#76; p</td>
</tr>
<tr>
<td>13</td>
<td>D15</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>b</td>
<td>#45; -</td>
<td>77</td>
<td>4D</td>
<td>p</td>
<td>#77; q</td>
</tr>
<tr>
<td>14</td>
<td>E16</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>c</td>
<td>#46; .</td>
<td>78</td>
<td>4E</td>
<td>q</td>
<td>#78; r</td>
</tr>
<tr>
<td>15</td>
<td>F17</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>d</td>
<td>#47; /</td>
<td>79</td>
<td>4F</td>
<td>r</td>
<td>#79; s</td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>e</td>
<td>#48; 0</td>
<td>80</td>
<td>50</td>
<td>s</td>
<td>#80; t</td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>f</td>
<td>#49; 1</td>
<td>81</td>
<td>51</td>
<td>t</td>
<td>#81; u</td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>g</td>
<td>#50; 2</td>
<td>82</td>
<td>52</td>
<td>u</td>
<td>#82; v</td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>h</td>
<td>#51; 3</td>
<td>83</td>
<td>53</td>
<td>v</td>
<td>#83; w</td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>i</td>
<td>#52; 4</td>
<td>84</td>
<td>54</td>
<td>w</td>
<td>#84; x</td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>j</td>
<td>#53; 5</td>
<td>85</td>
<td>55</td>
<td>x</td>
<td>#85; y</td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>k</td>
<td>#54; 6</td>
<td>86</td>
<td>56</td>
<td>y</td>
<td>#86; z</td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>l</td>
<td>#55; 7</td>
<td>87</td>
<td>57</td>
<td>z</td>
<td>#87;</td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>A</td>
<td>#56; 8</td>
<td>88</td>
<td>58</td>
<td>A</td>
<td>#58;</td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>B</td>
<td>#57; 9</td>
<td>89</td>
<td>59</td>
<td>B</td>
<td>#59;</td>
</tr>
<tr>
<td>26</td>
<td>102</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>C</td>
<td>#58; :</td>
<td>90</td>
<td>5A</td>
<td>C</td>
<td>#58;</td>
</tr>
<tr>
<td>27</td>
<td>103</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>D</td>
<td>#59; ;</td>
<td>91</td>
<td>5B</td>
<td>D</td>
<td>#59;</td>
</tr>
<tr>
<td>28</td>
<td>104</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>E</td>
<td>#60; &lt;</td>
<td>92</td>
<td>5C</td>
<td>E</td>
<td>#60;</td>
</tr>
<tr>
<td>29</td>
<td>105</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>F</td>
<td>#61; =</td>
<td>93</td>
<td>5D</td>
<td>F</td>
<td>#61;</td>
</tr>
<tr>
<td>30</td>
<td>106</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>G</td>
<td>#62; &gt;</td>
<td>94</td>
<td>5E</td>
<td>G</td>
<td>#62;</td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>H</td>
<td>#63; ?</td>
<td>95</td>
<td>5F</td>
<td>H</td>
<td>#63;</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
## Extended ASCII codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Ç</td>
</tr>
<tr>
<td>129</td>
<td>ü</td>
</tr>
<tr>
<td>130</td>
<td>é</td>
</tr>
<tr>
<td>131</td>
<td>â</td>
</tr>
<tr>
<td>132</td>
<td>à</td>
</tr>
<tr>
<td>133</td>
<td>à</td>
</tr>
<tr>
<td>134</td>
<td>å</td>
</tr>
<tr>
<td>135</td>
<td>ç</td>
</tr>
<tr>
<td>136</td>
<td>è</td>
</tr>
<tr>
<td>137</td>
<td>ë</td>
</tr>
<tr>
<td>138</td>
<td>ë</td>
</tr>
<tr>
<td>139</td>
<td>ï</td>
</tr>
<tr>
<td>140</td>
<td>ï</td>
</tr>
<tr>
<td>141</td>
<td>ï</td>
</tr>
<tr>
<td>142</td>
<td>Å</td>
</tr>
<tr>
<td>143</td>
<td>Å</td>
</tr>
</tbody>
</table>

Source: [www.LookupTables.com](http://www.LookupTables.com)
Reading some characters of “Coriolanus” in MATLAB

```matlab
%% File: p1.m
% Read first 10 characters from Coriolanus
f=fopen( 'TextCompression/coriolan.txt', 'r' );
A=fread(f,10,'uint8');
fclose(f);
A'       % Print in decimal
dec2hex(A) % Print in hexadecimal
dec2bin(A) % Print in binary
```
Output I

```matlab
>> p1
ans =
    66   101   102   111   114   101    32   119

ans =
   42
   65
   66
   6F
```
Output II

72  
65  
20  
77  
65  
20  

ans =

1000010  
1100101  
1100110  
1101111
Definition of coding
Compression of plain (ASCII) text
Shannon lower bound

Output III

1110010
1100101
0100000
1110111
1100101
0100000
An important paradigm of data compression

Data Compression = Data Modeling + Source Coding
A data model of “Coriolanus”

Example (Data Modeling)

Out model of “Coriolanus” is a random data model: Every character is drawn at random from the alphabet, according to the probability distribution

\[ P = (p_1, p_2, \ldots, p_n) \]

Numbers \( p_j \) are considered parameters, to be estimated.

George Box, a statistician, \textit{circa} 1979

All models are wrong but some are useful.
Arithmetic coding as a source coding approach

Example (Source Coding)

Our source coding technique is **arithmetic coding**, to be explained later. “Coriolanus” will be encoded as a single **binary fraction** with approximately 640,000 binary digits (bits) after the binary point!
The alphabet and "table" of *Coriolanus* in MATLAB

```matlab
1  % File: p2.m
2  %% Read all characters of Coriolanus as ASCII codes
3  f=fopen('TextCompression/coriolan.txt','r');
4  A=fread(f,inf,'uint8');
5  fclose(f);
6  %% Conduct frequency analysis (build "table")
7  % C is the alphabet: as per documentation
8  % of unique, IA and IC are index arrays s.t.
9  % C = A(IA) and A = C(IC)
10  [C, IA, IC] = unique(A);
11  table = histcounts(IC, 1: (length(C)+1));
12  %% Bar plots of alphabet and table
13  bar(C); bar(table)
```
The alphabet and “table” bar plots
Entropy of a probability distribution

**Definition (Shannon Entropy)**

Let \( P = (p_1, p_2, \ldots, p_n) \) be a probability distribution on numbers 1–\( n \). The number \( H(P) \) defined by the formula:

\[
H(P) = \sum_{j=1}^{n} (- \log_2 p_j) \cdot p_j
\]

is called the **Shannon entropy** of the distribution \( P \).
Entropy of the “Coriolanus” distribution

```matlab
1 %% File: p3.m
2 %% Read data, find table
3 p2;
4 %% Find entropy
5 p = table / sum(table);
6 entropy = sum(-p.*log2(p))

>> p3

entropy =

   4.5502

>>
```
Entropy as mean number of bits per symbol

If entropy is $h = 4.5502$ bits, we expect $h$ bits to be issued in the compressed message, per every symbol of the original message. Since ASCII encoding uses 1 byte (=8 bits), per symbol, we expect compression ratio to be:

$$\frac{8}{4.5502} = 1.7582$$
Calculation of arithmetic code in MATLAB

%% File: p4.m
%% Read text file into array A, compute table, alphabet, index array IC
p2;
%% Use arithmetic coding to arithmetically encode the sequence of indices
code=arithenco(IC,table);
compression_ratio=(8*length(A))/length(code)

>> p4

compression_ratio =

1.7581
What is arithmetic code?

**Arithmetic code** is a binary code which represents digits of a real number \( x \in [0, 1] \) uniquely assigned to the original message (Shakespeare’s play “Coriolanus”). The number has approximately

\[
H(P) \cdot N = 645354
\]

binary digits after the binary point:

\[
x = 0.d_1 d_2 \ldots
\]

where \( d_j \in \{0, 1\} \).
Commentary on arithmetic coding

- The idea of arithmetic coding comes from Shannon-Fano-Elias coding.
- The algorithm was implemented in a practical manner by IBM around 1980.
- IBM obtained a patent for arithmetic coding and defended it vigorously for approximately 20 years. The value of the algorithm as intellectual property was estimated at tens of billions of dollars.
- Now there are many freely available implementations of arithmetic coding, including the MATLAB implementation.
The principle of arithmetic coding I

1. Let \( m = (a_0, a_1, \ldots, a_N) \) be a message with \( a_j \in \{1, 2, \ldots, n\} \).

2. Let \( P = (p_1, p_2, \ldots, p_n) \) be a relative frequency table of the symbols in a message.

3. Every partial message \( m_K = (a_0, a_1, \ldots, a_K) \) is assigned a semi-open subinterval of \( I = [0, 1) \).

4. \( m_0 \) (the empty message) is assigned \( I_0 = I \).

5. If \( m_{K-1} \) is assigned interval \( I_{K-1} \), interval \( I_K \subseteq I_{K-1} \) and the length of \( I_K \) is proportional to \( p_{a_K} \) (the probability of the \( K \)-th symbol).
The principle of arithmetic coding II

6  As the length of the interval

\[ |I_N| = \prod_{j=0}^{N} p_{a_j} \]

\[ -\log_2 |I_N| = \sum_{j=0}^{N} (-\log p_{a_j}) = N \cdot \sum_{l=1}^{n} (-\log p_l) \frac{N_l}{N} \]

\[ = N \cdot \sum_{l=1}^{n} (-\log p_l) p_l = N \cdot H(P). \]

where \( N_l \) is the number of times \( l \) appears in the message, and \( N \) is the length of the message.

7  We select a number in \( I_N \) with the smallest number of binary digits, approximately \( N \cdot H(P) \).
Useful definitions I

- An *alphabet* is any finite set $A$.
- A *message* $m = a_0 a_1 \ldots a_n$ (or $m = (a_0, a_1, \ldots, a_n)$) over the alphabet $A$ is any finite sequence of elements of $A$. The set of all messages is denoted $A^+$. We have this disjoint union of Cartesian powers of the alphabet:

\[ A^+ = \bigcup_{n=0}^{\infty} \bigtimes_{j=1}^{n} A \]

- A *code* is a map $C : A^+ \rightarrow B^+$. 
A symbol code is a map $C : A \rightarrow B^+$, i.e. a map that maps symbols to one alphabet $A$ to messages (or blocks of symbols) in another alphabet $B$. The symbol code extends to $C : A^+ \rightarrow B^+$ of all messages over alphabet $A$ to all messages over alphabet $B$, by concatenation:

$$C(a_0a_1 \ldots a_n) = C(a_0)C(a_1) \ldots C(a_n)$$

The length of a message is its number of symbols; The function $\ell : A^+ \rightarrow \{0, 1, \ldots\}$ maps messages to their lengths.

A binary (symbol) code is a symbols code $C : A \rightarrow \{0, 1\}$.

A code $C : A^+ \rightarrow B^+$ is lossless, if it is 1:1 as a map.

A symbol code $C : A \rightarrow B^+$ is called uniform iff $\ell \circ C = \text{const..}$
Morse code — an old, binary, non-uniform code with some compression

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

1
2
3
4
5
6
7
8
9
0

Figure: Samuel Morse, 1840, "American inventor and painter"
Problem (Coriolanus to Morse Code)

Write a program which will encode Shakespeare’s “Coriolanus” in Morse code. Compare the output length to that of the original message (141,830 characters). Also, estimate the duration in “units” of the transmission of “Coriolanus”, assuming that the “unit” is 1/4 second.

Speed of Morse Code “Benchmark”

“High-speed telegraphy contests are held; according to the Guinness Book of Records in June 2005 at the International Amateur Radio Union’s 6th World Championship in High Speed Telegraphy in Primorsko, Bulgaria, Andrei Bindasov of Belarus transmitted 230 morse code marks of mixed text in one minute.”
The expected length of the code

Definition (Expected length of a binary symbol code)

Given a probability distribution $P : A \rightarrow [0, 1]$ on the alphabet $A$, the expected length of a binary symbol code $C : A \rightarrow \{0, 1\}^+$ is defined as:

$$\mathbb{E}(\ell \circ C) = \sum_{a \in A} \ell(C(a)) \cdot P(a).$$
Theorem (Shannon source coding theorem — Shannon, 1948)

If a binary symbol code $C : A \rightarrow \{0, 1\}^+$ is lossless then

$$\mathbb{E}(\ell \circ C) \geq H(P)$$

where $H(P)$ is the Shannon entropy of the distribution $P$:

$$H(P) = \sum_{a \in A} (- \log_2 P(a)) \cdot P(a)$$
Kraft inequality (extended variant)

Theorem (Kraft Inequality)

If $A$ is countable and $\mathcal{C} : A \rightarrow \{0, 1\}^+$ is a lossless symbol code then

$$\sum_{a \in A} 2^{-D(a)} \leq 1.$$  

where $D(a) = \ell(\mathcal{C}(a))$. 
The existence of nearly optimal codes

**Theorem (Shannon-Fano, 1948)**

*For every alphabet $A$ and a distribution function $P : A \rightarrow (0, 1]$ there exists a binary code $C : A \rightarrow \{0, 1\}^+$ such that:*

$$H(P) \leq \mathbb{E}(\ell \circ C) \leq H(P) + 1.$$
%% Specify input and output filenames
infile='coriolan.txt';
outfile='coriolan.zzz';

%%% Read uncompressed text file
fileID=fopen(infile);
A=fread(fileID,infile,'uint8');
fclose(fileID);
disp('Entering RTG DEMO text encoder...');

%%% Conduct frequency analysis (build "table")
% C is the alphabet: as per documentation of unique,
% IA and IC are index arrays C = A(IA) and A = C(IC)
[C,IA,IC]=unique(A);
table=histcounts(IC,1:(length(C)+1));

%%% Find entropy
p=table/sum(table);
entropy=sum(-p.*log2(p));
disp(['Expecting ',num2str(entropy),' bits per symbol']);

%%% Use arithmetic coding to encode the sequence
code=arithenco(IC,table);

%%% Calculate bits per symbol
bitspersymbol=length(code)/length(A);
disp(['Achieved ',num2str(bitspersymbol),' bits per symbol']);

%%% Save compressed data
f=fopen(outfile,'wb+');
fwrite(f,'_RTG');

%%% Write data to file
Full encoder in MATLAB II

27  % ----------------- Write metadata -----------------
28  % Write unencoded data length
29  fwrite(f,length(A),'uint32');
30  % Write alphabet length
31  fwrite(f,length(C),'uint8');
32  % Write alphabet
33  fwrite(f,C,'uint8');
34  % Write table
35  bits_per_table_entry=floor(log2(max(table)))+1;
36  bytes_per_table_entry=ceil(bits_per_table_entry/8);
37  if bytes_per_table_entry > 2
38      count_class='uint32';
39  elseif bytes_per_table_entry > 1
40      count_class='uint16';
41  else
42      count_class='uint8';
43  end
44  fh=str2func(count_class);
45  counts=fh(table);
46  fwrite(f,bytes_per_table_entry,'uint8');
47  fwrite(f,counts,count_class);
48  % ----------------- Write binary code -----------------
49  % Padd code to a multiple of 8 bits
50  padd=ceil(length(code)/8)*8-length(code);
51  code=[code;zeros(padd,1)];
52  % Pack code into bytes
53  packed_code=zeros(length(code)/8,1,'uint8');
Full encoder in MATLAB III

54    for i = 1:length(packed_code)
55        packed_code(i,1) = bi2de(code((1 + 8*(i-1)):(8 + 8*(i-1)))');
56    end
57    fwrite(f,length(packed_code),'uint32');
58    fwrite(f,packed_code,'uint8');
59    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60    fclose(f);
61    disp('Exiting RTG text encoder.');
Full decoder in MATLAB I

1 disp('Entering RTG text decoder...');
2 infile='coriolan.zzz';
3 outfile='coriolan_d.txt';
4
5 disp('Great! Magic number matches.');
Full decoder in MATLAB II

```matlab
27    count_class = 'uint8';
28    end
29    recovered_table = fread(f, alphabet_len, count_class);'
30    \% Read packed code
31    packed_code_len = fread(f, 1, 'uint32');
32    packed_code = fread(f, packed_code_len, 'uint8');
33    \% Unpack the code
34    code_len = packed_code_len * 8;
35    recovered_code = zeros(code_len, 1, 'double');
36    recovered_code = de2bi(packed_code, 8);'
37    recovered_code = recovered_code(:);
38    \% Recover index array
39    IC = arithdeco(recovered_code, recovered_table, len);
40    \% Apply alphabet
41    recovered = C(IC);
42    \% Write the file
43    f = fopen(outfile, 'w');
44    fwrite(f, recovered);
45    fclose(f);
46    \-------------------------------
47    disp('Exiting RTG text decoder...');
```