Contents

List of Figures

List of Tables

List of Algorithms

Chapter 4

Preliminaries

In this chapter, we will establish all the conventions that will be used throughout his text and explain some basic terms.

4.1 Glossary

binary number A binary number is simply a number of base 2. For example, the number 139 is represented by the binary number $1000\ 1011_2$, since

$$
\begin{aligned}
&1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \\
&2^7 + 2^3 + 2^1 + 2^0 = \\
&128 + 8 + 2 + 1 = 139
\end{aligned}
$$

Exercise 4.1

Rewrite the following numbers on binary form:

(a) 4
(b) 50
(c) 500

Exercise 4.2

Convert the following binary numbers to ordinary numbers of base 10:

(a) 1111_2
(b) $1000\ 0000_2$
(c) $1111\ 1110_2$

bit The separate digits of a binary numbers are often referred to as bits. An n-bit number is simply a number with n binary digits.

Exercise 4.3

What is the maximum value of a n-bit number? How many different possible values can a n-bit number have?

toggled bit If we say that a bit is *toggled*, we mean that the bit has a value of 1.

cleared bit And controversially, if a bit has the value 0, then we say that the bit is *cleared*.

bit counting order The bit that has the lowest value is referred to as bit 0[1]. In the binary number 0100_2, bit 0 is cleared(it has the value 0), bit 2 is toggled (it has the value 1) and the last bit 3 is cleared.

lowest bit Bit 0 will also commonly be referred to as the lowest bit.

highest bit Bit $n-1$ of a n-bit number. In order words, the last bit of a binary number. In the number 100_2 the highest bit toggled and the 2 lowest bits are cleared.

Exercise 4.4

What are the values of bits 0, 3 and the highest bit in the number 01001_2?

byte A byte is simply a binary number with 8 digits/bits. The max value of a byte is $2^8 - 1 = 255$ and there are $2^8 = 256$ different values that a byte can have. We will often be referring to bytes throughout the entire text, since, as we soon shall see, an image represented digitally is just a sequence of bytes.

ASCII A very common text encoding that will be used a lot in this text. ASCII is a 7-bit encoding and therefore covers 128 different characters. But ASCII values are for the sake of convenience always stored in 8-bit bytes. The entire ASCII table is given in table 4.1[6]. Characters 33–126 are printable characters. Characters 0–32 are on the other hand control characters. These are used to affect how the text is processed. HT (9) is for an ordinary tab while SP(32) represents space. However, many of these control characters are obsolete and are practically never used. The control codes that are actually used in modern files are NUL, HT, LF,CR and SP [7].

The usage of the two special codes CR and LF is something that we need to further discuss. They are used to start a new line in text. But representing newlines turns out to be a surprisingly complex issue. In Windows based operating systems, newlines are represented by a CRLF; that is, a carriage return, CR, followed by a linefeed, LF. Unix based operating systems, like Linux and Mac OS X, simply uses a line feed, LF, to represent newlines. But for Mac OS version 9 and lower, CR was used[8, 9, 10, 11, 12].

If a text file is to be used on only one operating system this will never pose a problem. But if the file is to be shared between computers running on different operating systems, it will. However, this is something that is almost never noticeable to the common user. It is really only of importance to programmers who want their software to flawlessly run on different operating systems.

[1] This convention is mainly used because us programmers like to start counting from 0 rather than 1

Table 4.1 – The ASCII table

Value	Code(Character)	Code Description
0	NUL	Null Character
1	SOH	Start of Heading
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Enquiry
6	ACK	Acknowledge
7	BEL	Bell(makes a sound)
8	BS	Backspace
9	HT	Horizontal tab(ordinary tab)
10	LF	Line Feed
11	VT	Vertical tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledge
22	SYN	Synchronous Idle
23	ETB	End of Transmission Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator
32	SP	Space
33	!	Exclamation Point
34	"	Quotation Mark
35	#	Number Sign
36	$	Dollar Sign
37	%	Percentage Sign
38	&	Ampersand
39	'	Apostrophe
40	(	Left Parenthesis
41	)	Right Parenthesis
42	*	Asterisk
43	+	Plus Sign
44	,	Comma
45	-	Minus Sign, hyphen

Table 4.1 – (continued)

Value	Code(Character)	Code Description
46	.	Period , Dot
47	/	Forward Slash
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	Colon
59	;	Semicolon
60	<	Less-than Sign
61	=	Equals Sign
62	>	Greater-than Sign
63	?	Question Mark
64	@	At Sign
65	A	A
66	B	B
67	C	C
68	D	D
69	E	E
70	F	F
71	G	G
72	H	H
73	I	I
74	J	J
75	K	K
76	L	L
77	M	M
78	N	N
79	O	O
80	P	P
81	Q	Q
82	R	R
83	S	S
84	T	T
85	U	U
86	V	V
87	W	W
88	X	X
89	Y	Y
90	Z	Z
91	[	Left Bracket

Table 4.1 – (continued)

Value	Code(Character)	Code Description
92		Backward Slash
93	]	Right Bracket
94	∧	Caret
95	_	Underscore
96	`	Grave Accent
97	a	a
98	b	b
99	c	c
100	d	d
101	e	e
102	f	f
103	g	g
104	h	h
105	i	i
106	j	j
107	k	k
108	l	l
109	m	m
110	n	n
111	o	o
112	p	p
113	q	q
114	r	r
115	s	s
116	t	t
117	u	u
118	v	v
119	w	w
120	x	x
121	y	y
122	z	z
123	{	Left Bracket
124	|	Vertical Line
125	}	Right Bracket
126	~	Tilde
127	DEL	Delete

Exercise 4.5

Convert the following ASCII characters to their corresponding ASCII values:

(a) A
(b) n
(c) <

Exercise 4.6

Convert the following ASCII values to their corresponding characters:

(a) 35
(b) 122
(c) 63

Exercise 4.7

What is always true for the ASCII values of uppercase characters(A–Z)?

(Hint: write them out binary)

Exercise 4.8

How do you convert an uppercase ASCII value to its corresponding lower case value? How do you do the reverse transformation, lowercase to uppercase?

(Hint: what number has to be added or subtracted?)

Hexadecimal We will also be using the hexadecimal numeral system in this text. In hexadecimal the numbers 0–9 are given their usual values, while the letters A–F are assigned to the values 10-15, so that the hexadecimal number $D3_{16}$ has the value

$$13 \cdot 16^1 + 3 \cdot 16^0 = 13 \cdot 16 + 3 = 208 + 3 = 211$$

Exercise 4.9

Convert the following hexadecimal numbers to ordinary numbers of base 10:

(a) 23_{16}
(b) FF_{16}
(c) AA_{16}

Exercise 4.10

Convert the following numbers to hexadecimal:

(a) 3
(b) 46
(c) 189

string A string is simply a sequence of letters in some encoding. The most commonly used encoding in this text will be ASCII [6].

C string String as they are represented in the C programming language. In this language, strings are always terminated by the NUL character[13]. The NULL (alternative spelling of NUL) character has, as familiar, a value of 0. This means that the string “eric” will be represented by the sequence of bytes 101, 114, 105, 99, 0 in the C programming language. This is important to know, because in some image formats strings are stored as C strings.

Why the NULL character at the end of the string even is necessary is for rather complex reasons that we will not treat in this text.

file We are going to be talking a lot about files in this text, so it is important that we as early as possible establish a strict definition for what a file is. A file is just a sequence of bytes. A perfectly valid file could for example consist of the numbers $101, 114, 105, 99$. This a file that consists of the single string "eric", where the letters use ASCII encoding. However, if you opened this file in a text editor, say notepad, you would only see the letters "eric" and not the numbers that represent the letters. This is because a text editor is programmed to see all the bytes in a file as text. If you on the other hand opened this file in a hex viewer, you would see the file for what it truly is: a sequence of numbers[2].

But since a byte can only have 256 different values, the reader may wonder how larger numbers are stored in a file. This is simply done by combining bytes. Two bytes in a sequence becomes a 16-bit number with a maximum value of $2^{16} - 1$. In the same fashion even larger numbers can be stored.

offset When we are talking about an offset we are referring to a position in a file. The offset is zero based. When we are talking about the number at offset 0 in the file $13, 2, 1$ we are talking about the number 13. In the same file at the offset 2, the number 1 can be found.

render When a program displays an image on the screen this process is known as *rendering*. To render an image is to display an image on the screen.

display Synonym for render.

4.2 Pseudocode Conventions

Instead of showing code examples in some random programming language, we will be using pseudocode to explain the algorithms in this book. This will keep things as general as possible, and not force the reader into knowing a specific programming language before reading this text.

The pseudocode will be kept as traditional as possible, but we will still need to establish several conventions for it, which is what we are going to do for the rest of the chapter.

4.2.1 Boolean Operators

To signify the Boolean operators, or logical operators as we will often also refer them to, we will be using the following symbols:

$\neg$ logical *not*(table 4.2d)

$\wedge$ logical *and*(table 4.2a)

$\vee$ logical *or*(table 4.2b)

Logical truth is represented by T, and falseness is represented by F.

[2]Do note that in a hex viewer these numbers are, as is implied by the name, shown as hexadecimal numbers

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

(a) Logical and

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

(b) Logical or

p	q	$p \otimes q$
T	T	F
T	F	T
F	T	T
F	F	F

(c) Logical exclusive or

p	$\neg p$
T	F
F	T

(d) Logical not

Table 4.2 – Logical truth tables

4.2.2 Bitwise Operators

Notation

The bitwise operators will also be used in this text. We will use the notation introduced in the C programming language[13] to represent them in pseudocode:

$\&$ Bitwise and

$|$ Bitwise or

$\otimes$ Bitwise xor

$\sim$ Bitwise not

$\ll$ Left bit shift

$\gg$ Right bit shift

Notice that we are using $\otimes$ for representing bitwise xor rather than the traditional C notation $\wedge$. This is due to the fact that we would otherwise confuse it with logical and, $\wedge$.

What follows is a short introduction to the very simple bitwise operators.

Bitwise and, or, and xor

Bitwise and is just like logical and, except for the fact that it operates on the bit level. Let us for demonstrative consider the result 22 & 12. Since bitwise and operates on the bit level we first must convert the two numbers to binary: 10110_2 & 01100_2. Then the calculation is simply done like this:

$$\begin{array}{rr} & 10110 \\ \& & 01100 \\ \hline & 00100 \end{array}$$

So as you can see, the bitwise operators do Boolean logic on the bit level.

Exercise 4.11

(a) 2 & 1

(b) 255 & 23

(c) $26 \,\&\, 12$

Bitwise or is in the same way logical or on the bit level. Let us perform the former calculation using bitwise or:

$$\begin{array}{rr} & 10110 \\ | & 01100 \\ \hline & 11110 \end{array}$$

Exercise 4.12

(a) $172 \mid 52$

(b) $3 \mid 3$

(c) $240 \mid 15$

Bitwise xor on the other hand, operates on bits by using logical exclusive or. The truth table of logical exclusive or is given in table 4.2c. Using this table, we can easily understand how bitwise xor works:

$$\begin{array}{rr} & 10110 \\ \otimes & 01100 \\ \hline & 11010 \end{array}$$

Exercise 4.13

(a) $10 \otimes 10$

(b) $12 \otimes 7$

(c) $48 \otimes 16$

Bitwise not

When dealing with bitwise not, it is important that we consider the size of the numbers that we are performing the operation on. If for example $b = 10$ and the variable b is of type byte, then it *must* be of length 8 bits: $b = 0000\ 1010_2$. What bitwise not does, is that it inverts the number so that at all toggled bits get cleared, and all cleared bits get toggled, so $\sim b = 1111\ 0101_2$.

Now you should see why it was important that we considered the size of the number. Had the variable b been of size 4 bits, then $b = 1010_2$ and then the end result of the operation $\sim b$ would have been 0101_2 instead of $1111\ 0101_2$.

Exercise 4.14

What are the values of the following expressions, if all the numbers are bytes?

(a) ~ 11

(b) $(\sim 4) \otimes 4$

(c) $(\sim b) \otimes b$, for any byte b

(d) $(\sim b) \mid b$, for any byte b

(e) $(\sim b) \,\&\, b$, for any byte b

Bitwise shifting

It is also in bitwise shifting important that we consider the size of the numbers. Bitwise shifting is actually very simple: all the operation $b \ll n$ really does, is that it shifts the bit pattern in the number b n steps to the left. For the 4-bit number 0011_2, this means that $0011_2 \ll 2 = 1100_2$. But what would have happened if the bit pattern was shifted 3 steps? Then one bit is going fall of the bit boundary and disappear, so $0011 \ll 3 = 1000_2$.

And bitwise right shifting works in pretty much the same way, expect for the fact that the bit shifting is done to the right instead of the left, so $0110_2 \gg 2 = 0001_2$.

Exercise 4.15

(a) $1 \ll 0$

(b) $1 \ll 1$

(c) $1 \ll 2$

(d) $3 \ll 1$

(e) $3 \ll 2$

(f) $3 \ll 3$

Can you express the operation $N \ll S$, if the condition $S \geq 0$ holds, in terms of the arithmetic operators? Exponentiation counts as an arithmetic operator in this exercise. Also, you can ignore the possibility of bits falling of the bit boundary in this exercise.

4.2.3 Typographical Conventions

keywords will use a **bold** font.

functions will be signified by small caps.

variables can noticed by their *cursive slant*.

4.2.4 Syntax

In this section we will discuss the basic syntax of the pseudocode.

The start of a comment is indicated by the symbol ▷.

To assign the value n to the variable var, the we use the notation $var \leftarrow n$

To store a sequence of values we will use arrays. If for example the array a contains the values the $3, 1, 2$ then to access the first value of this array, 3, the syntax $a[0]$ is used. In general, to access the n:th value of an array you do $a[n-1]$, since the indexes of arrays are zero-based.

To to go through each value in the array a, the syntax demonstrated in algorithm 4.1 is used.

Algorithm 4.1 The for each control structure

```
▷Go through every value v in the array a. The variable v is assigned every
element in the array a in order.
for each v in a do
    ▷Do something with a here.
end for each
```

In algorithm 4.2 the control structure repeat is demonstrated, which is used unconditionally looping a number of times. Prematurely terminating a loop is done with the **break** statement.

Algorithm 4.2 The repeat control structure

```
repeat n do
    actions                                   ▷actions are repeated n times.
end repeat
```

For functions, we will be using the traditional syntax; FUNC(a, b, c) means that we are calling the function FUNC with the arguments a, b and c and that the value of this expression is the return value of the function. To return a value from a function the **return** statement is used.

The function syntax is demonstrated in algorithm 4.3. In this function, Euclid's algorithm is used to calculate the greatest common divisor of two given numbers. [14, 15].

Algorithm 4.3 Euclid's algorithm

```
procedure EUCLID(a, b)
    r ← a mod b
    while r ≠ 0 do
        a ← b
        b ← r
        r ← a mod b
    end while
    return b
end procedure
```

4.2.5 Functions

We will be dealing with files in many of these algorithms, so we will need to introduce several functions for handling file operations.

READBYTE() It is assumed from the beginning of the algorithm that a file has already been opened for reading. This function reads a byte from that file.

WRITEBYTE(*byte*) At the beginning of every algorithm, we also assume that there is a file opened for output. This function writes a byte to that file.

ENDOFFILEREACHED() True if the end the file we are reading from have been reached.

Exercise 4.16

Make a function GETBITS(b, *start*, *end*) that extracts the bit pattern of a number b from a starting position to a ending position. These positions are be zero-based. Example:

$$\text{GETBITS}(80, 4, 6) = 5,$$

because 80 is represented by the binary number $0101\ 0000_2$ and from the positions 4 to 6 there is a bit pattern that has a value of $5(101_2)$, which is what this function was supposed to extract.

Hint: Use the bitwise operators.

4.3 Answers to the exercises

Answer of exercise 4.1

(a) 100_2

(b) $11\ 0010_2$

(c) $1\ 1111\ 0100_2$

Answer of exercise 4.2

(a) 15

(b) 128

(c) 254

Answer of exercise 4.3

The maximum value of a n-bit number is $2^n - 1$.

Since every binary digit only has two possible values, an n-bit number has 2^n possible values.

Answer of exercise 4.4

1, 1 and 0. Or: toggled, toggled, cleared.

Answer of exercise 4.5

(a) 65

(b) 110

(c) 60

Answer of exercise 4.6

(a) #

(b) z

(c) ?

Answer of exercise 4.7

The 6:th bit is always toggled. This is because the lowest uppercase character, A, has the value 65 and the sixth bit has the value 64.

Answer of exercise 4.8

You convert an uppercase ASCII value to lowercase by adding 32 to it, since a $-$ A $= 97 - 65 = 32$. To do the reverse transformation, you subtract 32 from the value.

Answer of exercise 4.9

(a) 35

(b) 255

(c) 170

Answer of exercise 4.10

(a) 03_{16}
(b) $2E_{16}$
(c) BD_{16}

Answer of exercise 4.11

(a) 0
(b) 23
(c) 8

Answer of exercise 4.12

(a) 188
(b) 3
(c) 255

Answer of exercise 4.13

(a) 0
(b) 11
(c) 32

Answer of exercise 4.14

(a) 244
(b) 255
(c) 255
(d) 255
(e) 0

Answer of exercise 4.15

(a) 1
(b) 2
(c) 4
(d) 6
(e) 12
(f) 24

The operation $N \ll S$ is equivalent to $N \cdot 2^S$.

Answer of exercise 4.16

procedure GETBITS($b, start, end$)
  ▷Calculate the length of the bit pattern
  $len \leftarrow end - start + 1$
  return $(b \gg start)$ & $(\sim(\sim 0 \ll len))$
end procedure

The answer is given above. We will now explain this function.

If given the input GETBITS$(80, 4, 6)$ how may we calculate the value 5 from this?

80 is represented by the bit pattern $0101\ 0000_2$. First, we will right shift down the pattern $start = 4$ steps, $0101\ 0000_2 \gg start$, resulting in the bit pattern $0000\ 0101_2$. Now all that remains to be done is that we need to figure out how construct the bit pattern $0000\ 0111_2$ from the input values. Once we have figured out how to make this pattern, we can calculate the proper result:

$$0000\ 0101_2\ \&\ 0000\ 0111_2 = 101_2 = 5$$

Using bitwise and in this way to extract bit patterns is a common idiom in the C programming language.

The bit pattern we want to construct is $0000\ 0111_2$. We want a sequence of 3 toggled bits from the lowest bit. We can trivially calculate the length of this pattern as

$$end - start + 1 = 6 - 4 + 1 = 3$$

$+1$ is necessary because the bit positions are zero based.

We now have the length of the pattern. Now we need to figure out how to construct it. The operation ~ 0, assuming we are dealing with bytes, gets you the pattern $1111\ 1111_2$. Then by shifting this pattern *len* steps to the left, $3 \ll 1111\ 1111_2$, we end up with the value 1111 1000. Now, by simply using the bitwise not operation again, $\sim 1111\ 1000_2$, we end up with the desired pattern $0000\ 0111_2$.

Bibliography

[1] Standard of the Camera & Imaging Products Association. *CIPA DC- 008-Translation- 2010, Exchangeable image file format for digital still cameras: Exif Version 2.3*. Apr. 26, 2010. URL: http://www.cipa.jp/english/hyoujunka/kikaku/pdf/DC-008-2010_E.pdf.

[2] J. D. Murray and W. VanRyper. *Encyclopedia of graphics file formats*. 2nd ed. O'Reilly Series. O'Reilly & Associates, 1996. ISBN: 9781565921610.

[3] ImageMagick Studio LLC. *Convert, Edit, and Compose Images*. Dec. 22, 2011. URL: http://www.imagemagick.org.

[4] Netscape. *Data Formats*. Dec. 22, 2011. URL: http://www.dmoz.org/Computers/Data_Formats/.

[5] Greg Roelofs. *PNG: The Definitive Guide*. O'Reilly, 1999. URL: http://www.libpng.org/pub/png/book/.

[6] V. G. Cerf. *ASCII format for network interchange*. RFC 20. Internet Engineering Task Force, Oct. 1969. URL: http://tools.ietf.org/html/rfc20.

[7] A.K. Maini. *Digital electronics: principles, devices and applications*. J. Wiley, 2007. ISBN: 9780470032145.

[8] Daniel Robbins. *Common threads: Sed by example, Part 3*. IBM developerWorks. Nov. 1, 2000. URL: http://www.ibm.com/developerworks/linux/library/l-sed3/index.html.

[9] Xavier Noria. *Understanding Newlines*. O'Reilly. Aug. 17, 2006. URL: http://onlamp.com/pub/a/onlamp/2006/08/17/understanding-newlines.html.

[10] RFC Editor. *The End-of-Line Story*. Apr. 18, 2004. URL: http://www.rfc-editor.org/EOLstory.txt.

[11] David Tancig. *Apart No More—Sharing Files between Linux and Windows Partitions*. Beedle & Associates, Inc. 2001.

[12] Dialogic Corporation. *Creating Telephony Applications for Both Windows and Linux: Principles and Practice*. Dialogic Research, Inc. 2008.

[13] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. 2nd ed. Prentice-Hall software series. Prentice Hall, 1988. ISBN: 9780131103627.

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. MIT Press, 2009. ISBN: 9780262033848.

[15] Eric W Weisstein. *Euclidean Algorithm*. Dec. 28, 2011. URL: http://mathworld.wolfram.com/EuclideanAlgorithm.html.

[16] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. 1st ed. Addison-Wesley Publishing Company, 1993. Chap. 5. URL: `http://fly.cc.fer.hr/~unreal/theredbook/chapter05.html`.

[17] Stefan Ohlsson, John S. Webb, and Bo Westerlund. *Digital Bild - Kreativt bildskapande med dator*. Bonniers, 1999.

[18] János D. Schanda. "Colorimetry". In: *Handbook of Applied Photometry*. Springer, 1997. Chap. 9.

[19] Symon D'o. Cotton. *Colour, Colour Spaces and the Human Visual System*. 1995.

[20] Danny Pascale. *A review of RGB color spaces*. Tech. rep. Montreal, Canada: The BabelColor Company, Oct. 2003.

[21] D. Hearn and M.P. Baker. *Computer graphics, C version*. Prentice Hall, 1997. ISBN: 9780135309247.

[22] Thomas Porter and Tom Duff. "Compositing digital images". In: *ACM Siggraph Computer Graphics* 18 (1984), pp. 253–259. DOI: `10.1145/964965.808606`.

[23] J. Niederst and J.N. Robbins. *Web design in a nutshell: a desktop quick reference*. In a Nutshell Series. O'Reilly, 1999. ISBN: 9781565925151.

[24] M.K. Sitts and Northeast Document Conservation Center. *Handbook for digital projects: a management tool for preservation and access*. Northeast Document Conservation Center, 2000.

[25] Mark Roth. *Some women may see 100 million colors, thanks to their genes*. Sept. 13, 2006. URL: `http://www.post-gazette.com/pg/06256/721190-114.stm`.

[26] *Truevision TGA^{TM} File Format Specification*. Truevision, Inc. Jan. 1991.

[27] David Salomon. *Data Compression: The Complete Reference*. 3rd ed. 2004. ISBN: 0-387-40697-2.

[28] Mark Nelson and J.L. Gailly. *The data compression book*. 2nd ed. M&T Books, 1996. ISBN: 9781558514348.

[29] Timothy Bell, Ian H. Witten, and John G. Cleary. "Modeling for text compression". In: *ACM Comput. Surv.* 21 (4 Dec. 1989), pp. 557–591. ISSN: 0360-0300.

[30] Jean loup Gailly. *compression-faq/part1*. Sept. 5, 1999. URL: `http://www.faqs.org/faqs/compression-faq/part1/`.

[31] A. Nagarajan and Dr. K. Alagarsamy. "An Enhanced Approach in Run Length Encoding Scheme". In: *International Journal of Engineering Trends and Technology* (July 2011).

[32] Apple Inc. *Technical Note TN1023*. Feb. 1, 1996. URL: `http://web.archive.org/web/20080705155158/http://developer.apple.com/technotes/tn/tn1023.html`.

[33] Inc Apple Computer. *Inside Macintosh: Operating system utilities*. Apple technical library. Addison-Wesley Pub. Co., 1994. ISBN: 9780201622706.

[34] Mark R. Nelson. "LZW data compression". In: *Dr. Dobb's Journal* 14 (10 Oct. 1989), pp. 29–36. ISSN: 1044-789X. URL: `http://marknelson.us/1989/10/01/lzw-data-compression/`.

[35] T. A. Welch. "A Technique for High-Performance Data Compression". In: *Computer* 17 (6 June 1984), pp. 8–19. ISSN: 0018-9162. DOI: 10.1109/MC.1984.1659158.

[36] Mark Nelson. *LZW Revisited*. Dec. 31, 2011. URL: http://marknelson.us/2011/11/08/lzw-revisited/.

[37] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *IEEE Transactions On Information Theory* 23.3 (1977), pp. 337–343.

[38] Jacob Ziv and Abraham Lempel. "Compression of Individual Sequences via Variable-Rate Coding". In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 530–536.

[39] Greg Roelofs. *History of the Portable Network Graphics (PNG) Format*. Mar. 2009. URL: http://www.libpng.org/pub/png/pnghist.html.

[40] Kel D. Winters et al. *United States Patent 5532693 - Adaptive data compression system with systolic string matching logic*. July 2, 1996. URL: http://www.google.com/patents/US5532693.

[41] Igor Pavlov. *7z Format*. Dec. 31, 2011. URL: http://www.7-zip.org/7z.html.

[42] Terry A. Welch. *U.S. Patent 4,558,302*. High speed data compression and decompression apparatus and method. Dec. 1985.

[43] Ross Arnold and Tim Bell. *A corpus for the evaluation of lossless compression algorithms*. Department of Computer Science, University of Canterbury, Christchurch, NZ. 1997. URL: http://corpus.canterbury.ac.nz/research/corpus.pdf.

[44] Matt Powell. *Descriptions of the corpora*. Jan. 8, 2001. URL: http://corpus.canterbury.ac.nz/descriptions/.

[45] Michael C. Battilana. *The GIF Controversy: A Software Developer's Perspective*. June 20, 2004.

[46] Stuart Caie. *Sad day... GIF patent dead at 20*. Jan. 3, 2011. URL: http://www.kyzer.me.uk/essays/giflzw/.

[47] Compuserve Incorporated. *GIF Graphics Interchange Format: A standard defining a mechanism for the storage and transmission of bitmap-based graphics information*. Columbus, Ohio, USA. 1987.

[48] CompuServe Incorporated. *Graphics Interchange Format — Version 89a*. Columbus, Ohio, USA. 1990.

[49] Royal Frazier. *All About GIF89a*. Jan. 3, 2011. URL: http://www.etsimo.uniovi.es/gifanim/gifabout.htm.

[50] Scott Walte. *Web Scripting Secret Weapons*. Que Corporation, 1996.

[51] Ross N Williams. *A painless guide to CRC error detection algorithms*. Sept. 24, 1996. URL: http://www.repairfaq.org/filipg/LINK/F_crc_v3.html.

[52] Michael Barr. *Additive Checksums*. Dec. 1, 2007. URL: http://www.netrino.com/Embedded-Systems/How-To/Additive-Checksums.

[53] A.S. Tanenbaum. *Computer networks*. 4th ed. Prentice Hall PTR, 2003. ISBN: 9780130661029.

[54] Mark R. Nelson. "File verification using CRC". In: *Dr. Dobb's Journal* 17 (5 May 1992), pp. 64–67. ISSN: 1044-789X. URL: `http://marknelson.us/1992/05/01/file-verification-using-crc-2/`.

[55] Terry Ritter. "The great CRC mystery". In: *Dr. Dobb's Journal* 11 (2 Feb. 1986), pp. 26–34. ISSN: 1044-789X. URL: `http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM`.

[56] Martin Stigge et al. *Reversing CRC – Theory and Practice*. 2006.

[57] Michael Barr. *CRC Implementation Code in C*. Dec. 2, 2007. URL: `http://www.netrino.com/Embedded-Systems/How-To/CRC-Calculation-C-Code`.

[58] Richard Black. *Fast CRC32 in Software*. Feb. 18, 1994. URL: `http://www.cl.cam.ac.uk/research/srg/bluebook/21/crc/crc.html`.

[59] Patrick Geremia. *Cyclic redundancy check computation: an implementation using the TMS320C54x*. Texas Instruments. Apr. 1999. URL: `http://focus.ti.com/lit/an/spra530/spra530.pdf`.

[60] William H. Press et al. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press, 2007. ISBN: 9780521880688.

[61] Philip Koopman and Tridib Chakravarty. *Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks*. 2004.

[62] Greg Cook. *Catalogue of parametrised CRC algorithms*. Oct. 17, 2011. URL: `http://regregex.bbcmicro.net/crc-catalogue.htm`.

[63] Thomas Boutel et al. *Portable Network Graphics (PNG) Specification (Second Edition)*. Nov. 10, 2003. URL: `http://www.w3.org/TR/PNG/`.

[64] J-L. Gailly. *ZLIB Compressed Data Format Specification version 3.3*. May 1996.

[65] Theresa Maxino. *Revisiting Fletcher and Adler Checksums*. Carnegie Mellon University.

[66] T. C. Maxino and P. J. Koopman. "The Effectiveness of Checksums for Embedded Control Networks". In: *IEEE Trans. Dependable Sec. Comput.* 6.1 (2009), pp. 59–72.

[67] Gary Stix. "Profile: David A. Huffman". In: *Scientific American* (Sept. 1991), pp. 54–58.

[68] David A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the Institute of Radio Engineers* 40.9 (Sept. 1952), pp. 1098–1101.

[69] Andy McFadden. *Shannon, Fano, and Huffman*. 1992. URL: `http://www.fadden.com/techmisc/hdc/lesson04.htm`.

[70] Matt Mahoney. *Data Compression Explained*. Dell, Inc, 2011. URL: `http://mattmahoney.net/dc/dce.html`.

[71] Debra A. Lelewer and Daniel S. Hirschberg. "Data compression". In: *ACM Comput. Surv.* 19 (3 Sept. 1987), pp. 261–296. ISSN: 0360-0300. DOI: `http://doi.acm.org/10.1145/45072.45074`. URL: `http://doi.acm.org/10.1145/45072.45074`.

[72] Eric Bodden, Malte Clasen, and Joachim Kneis. *Arithmetic Coding revealed - A guided tour from theory to praxis*. Tech. rep. 2007-5. Sable Research Group, McGill University, May 2007. URL: http://www.bodden.de/pubs/sable-tr-2007-5.pdf.

[73] R. Lewand. *Cryptological mathematics*. Classroom resource materials. Mathematical Association of America, 2000. ISBN: 9780883857199.

[74] C. E. Shannon. "A mathematical theory of communication". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5 (1 Jan. 2001), pp. 3–55. ISSN: 1559-1662. DOI: http://doi.acm.org/10.1145/584091.584093. URL: http://doi.acm.org/10.1145/584091.584093.

[75] Andy McFadden. *Ziv and Lempel*. 1992. URL: http://www.fadden.com/techmisc/hdc/lesson07.htm.

[76] Thomas D. Schneider. *Information Theory Primer*. Jan. 8, 2010. URL: http://alum.mit.edu/www/toms/paper/primer/primer.pdf.

[77] Donald E. Knuth, J.H. Morris, and Vaughan R. Pratt. "Fast pattern matching in strings". In: *SIAM Journal of Computing* 6.2 (1977), pp. 323–350.

[78] Robert S. Boyer and J. Strother Moore. "A fast string searching algorithm". In: *Commun. ACM* 20.10 (Oct. 1977), pp. 762–772. ISSN: 0001-0782. DOI: 10.1145/359842.359859. URL: http://doi.acm.org/10.1145/359842.359859.

[79] Andy McFadden. *LZSS*. 1992. URL: http://www.fadden.com/techmisc/hdc/lesson10.htm.

[80] Haruhiko Okumura. *Data Compression Algorithms of LARC and LHarc*. July 16, 1999.

[81] PKWARE. *APPNOTE.TXT - .ZIP File Format Specification*. Sept. 28, 2007. URL: http://www.pkware.com/documents/casestudies/APPNOTE.TXT.

[82] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. May 1996.

[83] Antaeus Feldspar. *An Explanation of the Deflate Algorithm*. Aug. 23, 1997. URL: http://zlib.net/feldspar.html.

[84] Greg Roelofs. *Portable Network Graphics*. Dec. 16, 2011. URL: http://www.libpng.org/pub/png/.

[85] Thomas Boutel et al. *PNG (Portable Network Graphics) Specification, Version 1.1*. Dec. 31, 1997. URL: http://www.libpng.org/pub/png/spec/1.1/PNG-Contents.html.

[86] J. Klensin. *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard). Internet Engineering Task Force, Oct. 2008. URL: http://www.ietf.org/rfc/rfc5321.txt.

[87] K. Sayood. *Lossless compression handbook*. Academic Press series in communications, networking and multimedia. Academic Press, 2003. ISBN: 9780126208610.

[88] J. Arvo. *Graphics Gems II*. Graphics gems series. Academic Press, 1994. ISBN: 9780120644810.