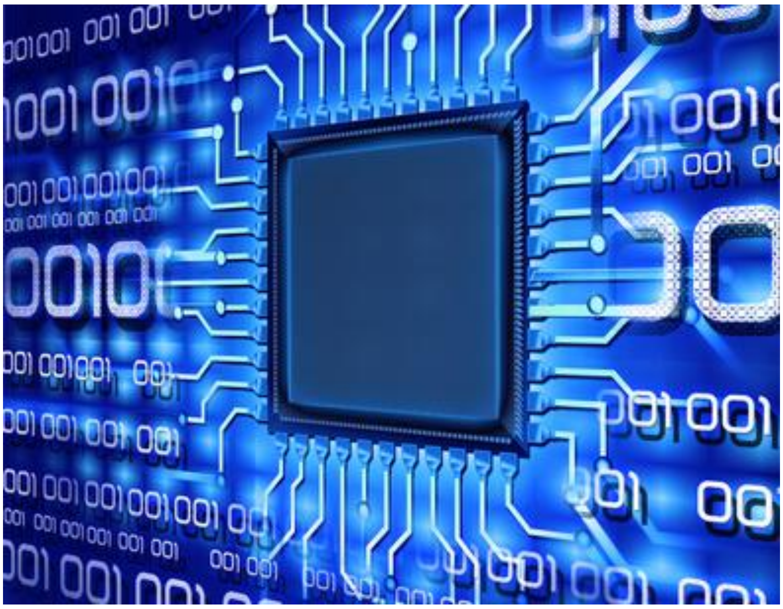
artem@artemlos.net

Algorithms via C#

Course reference



Written by Artem Los.

www.artemlos.net

This document contains descriptions of algorithms that are to be examined during the course.

Artem Los | LEARNING

This page is intentionally left blank.

Table of contents

Displaying a certain series using modulo arithmetic

Introductory problem

Consider following: *You want to display each seventh capital letter in the English alphabet*.

Obviously, we will need to work with modulo arithmetic, to solve this task. First, however, let us split up this problem into two parts: the technical side and the mathematical side.

Technical aspect

This problem mainly requires following knowledge:

1. Write text on the screen.
2. Create a for loop.
3. Create an if statement.
4. Get a letter, given an ASCII code, and vice versa.[1]

Probably, you are already familiar with 1-3, but converting an ASCII code to a char[2] might be something new. So if you want to display ‘a’, you first look up what char code ‘a’ has, which is 97, and then you convert this value (remember, whole numbers are represented by 32 bit integer – *int*).

```
char code = (char)97;
Console.WriteLine(code);
```

Remember, you can also put *(char)97* directly into *Console.WriteLine*.

```
Console.WriteLine((char)97);
```

The modulo sign is indicated by a ‘%’, in form *a % b*, and you say *a mod b*.

```
if (n % 9 == 0)
{
    // if n is a multiple of 9
}
```

Mathematical aspect

Each seventh letter means starting at 7, 14, 21, 28, 7N, i.e. all multiples of 7. A multiple of a number is when the remainder is equal to 0.

$$7N \equiv 0\ (mod\ 7)$$

In fact, the only thing that needs to be changed, in order to display each multiple + 1 is the remainder.

$$7N \equiv 1\ (mod\ 7)$$

If you want to generate the remainder quicker, please use the function below:

$$f(x, y) = x - y\left\lfloor\frac{x}{y}\right\rfloor$$

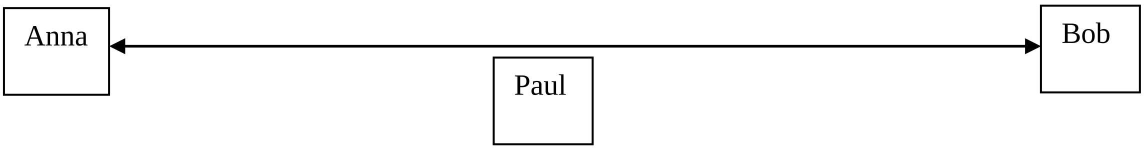
$$where\ x\ is\ the\ number\ and\ y\ is\ the\ divisor$$

So, how would you solve this problem?

[1] The ASCII codes are situated in the Appendix A.
[2] Char is a shorter form for character.

Encryption

An interesting concept that will be discussed in this section is how something can be encrypted, and later on decrypted back. The main idea behind a *cipher* is to make secret information readable for only one person – the receiver, and no one else, while transmitting the message.



Basically, Anna and Bob want to communicate with each other, without Paul knowing what they discuss. First of all, Anna and Bob will need to establish a *key phase*, and make sure they are the ones who know it. Secondly, they are to choose an algorithm, by which they will encrypt and decrypt messages.

Caesar cipher

Let us as start with a classic example of a symmetrical[3] cipher – Caesar cipher. This is probably one of the oldest ciphers, and is actually quite simple to understand and use.

First step is to decide a *shift*, i.e., the amount of times we will shift each letter in the alphabet, in our case it is 3.

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 *the original alphabet*

D E F G H I J K L M N O P Q R S T U V W X Y Z A B C *the 'shifted' alphabet*

Say Anna wants to say SEE YOU IN THE GARDEN AFTER NOON, but she does not want her old admirer know that, so she uses the *shifted alphabet* to replace each letter by the corresponding *shifted* letter. She will get something like:

VHH BRX LQ WKH JDUGHQ DIWHU QRRQ

Now she can be safe, that her ex-boyfriend Paul want now her secrets. When Bob receives the message, he will consider, that she used 3 letter shift, and do the same procedure, but vice versa. Now, we start from the shifted alphabet, and work our way back.

SEE YOU IN THE GARDEN AFTER NOON

They will now be happy together; Paul will need to do his maths homework!

Crypto analysis

Paul went home, and was so disappointed with his unsuccessful try to decode their conversation. However, his math skills lead to a solution. He discovered that the encrypted message is actually similar to the original one, but it is written using another alphabet. Paul got an idea to do a frequency analysis of a random English article, and compare it to the shifted alphabet.

[3] Symmetrical cipher is an encryption algorithm that uses a shared secret key, when encoding and decoding a message stream.

The English article				The encoded massage			
No.	Substring	Frequency (in %)	Frequency	No.	Substring	Frequency (in %)	Frequency
1	E	12.6236	217	1	H	19.2308	5
2	A	9.0750	156	2	Q	15.3846	4
3	R	8.4933	146	3	R	11.5385	3
4	T	8.2024	141	4	D	7.6923	2
5	S	7.9116	136	5	U	7.6923	2
6	N	7.3880	127	6	W	7.6923	2
7	I	6.9808	120	7	B	3.8462	1
8	O	6.1664	106	8	G	3.8462	1
9	D	3.6649	63	9	I	3.8462	1
10	H	3.6067	62	10	J	3.8462	1
11	L	3.1995	55	11	K	3.8462	1
12	C	3.1414	54	12	L	3.8462	1
13	U	2.6178	45	13	V	3.8462	1
14	F	2.5015	43	14	X	3.8462	1
15	M	2.5015	43				
16	Y	2.1524	37				
17	P	1.9197	33				
18	G	1.8034	31				
19	W	1.5707	27				
20	V	1.4543	25				
21	B	1.3962	24				
22	K	0.8726	15				
23	Q	0.3490	6				
24	X	0.1745	3				
25	J	0.1163	2				
26	Z	0.1163	2				

Clearly we see that E corresponds to H, so we can now calculate the difference, which gives us 3. Finally, he can decrypt all messages between Anna and Bob!

Computer instruction

```
static void Main(string[] args)
{
    string alphabet, text, action, result = ""; // default declarations
    int secretNum, index = 0;

    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; // original alphabet

    Console.WriteLine("(1) Please enter any text!");
    text = Console.ReadLine().ToUpper(); // make sure there are only capital letters.

    Console.WriteLine("(2) Please enter the secret digit:");
    secretNum = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Do you want to encrypt [e] or decrypt [d]?"); // select mode

    action = Console.ReadLine();

    for (int i = 0; i < text.Length; i++)
    {
        if (action == "e") // encode
        {
            index = mod(alphabet.IndexOf(text[i]) + secretNum, 26);
        }
        else // decode
        {
            index = mod(alphabet.IndexOf(text[i]) - secretNum, 26);
        }
        result += alphabet[index];
    }

    Console.WriteLine(result);
    Console.ReadLine();
}
```

Please also include the ‘mod’ function, located in Appendix A.

Caesar cipher with a longer key

Paul is a clever guy, and his solution almost made ‘privacy’ to be questioned. However, Bob started to think of a safer way of transmitting messages, so he came to a final conclusion – he should have a longer key.

Vigenère cipher

This is basically the same as Caesar cipher, but with a longer key. The algebra looks as following:

$$C_i = (M_i + K_i) \; mod \; 26$$

In comparison to the Caesar cipher:

$$C_i = (M_i + K) \; mod \; 26$$

Now, if we decrypt, we get:

$$M_i = (C_i - K_i) \; mod \; 26$$

Usually, the key is shorter than the actual message, so when the text is to be encrypted, repeat it.

HE WENT OUT TO LOOK AT THE TREES

KE YKEY KEY KE YKEY KE YKE YKEYK

In order to make this ‘repeating’ more efficient, use modulo arithmetic (considering *key length* is 3)!

$$K_{i \; mod \; 3}$$

Computer instruction

```
static void Main(string[] args)
{
    string alphabet, text, action, result = ""; // default declarations
    int[] secretNums = new int[1024];
    int index = 0;

    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; // original alphabet

    Console.WriteLine("(1) Please enter any text!");
    text = Console.ReadLine().ToUpper(); // make sure there are only capital letters.

    Console.WriteLine("(2) Please enter the secret digits:");
    secretNums = numToArray(Convert.ToInt32(Console.ReadLine()));

    Console.WriteLine("Do you want to encrypt [e] or decrypt [d]?"); // select mode
    action = Console.ReadLine();

    for (int i = 0; i < text.Length; i++)
    {
        if (action == "e") // encode
        {
            index = mod(alphabet.IndexOf(text[i]) + secretNums[i % secretNums.Length], 26);
        }
        else // decode
        {
            index = mod(alphabet.IndexOf(text[i]) - secretNums[i % secretNums.Length], 26);
        }
        result += alphabet[index];
    }

    Console.WriteLine(result);
    Console.ReadLine();
}
```

Please also include the ‘number to integer array’ and ‘mod’ procedures, located in Appendix A.

Recursive functions

Introduction

A recursive function is a function that calls itself. An example of a recursive function is *factorial*. For the moment, let us look at a function that will calculate the sum of some positive integers.

Problem: *Calculate the sum of a specific amount of positive integers, using a recursive function.*

Sum of positive integers

Sure enough, the mathematical part looks as following:

$$S_n = S_{n-1} + n, \quad for\ n \geq 1$$

So if n is 3, the recurrence works as follows:

1) Sum of 3 integers = sum of two integers + 3
2) Sum of 2 integers = sum of one integer + 2
3) Sum of 1 integer = sum of zero integers + 1;

Therefore it $S_3 = 6$.

The code is located below:

```
static int SumOfIntegers(int n)
{
    if (n > 0)
    {
        return SumOfIntegers(n - 1) + n;
    }
    else
    {
        return 0;
    }
}
```

This code is good because it works, but it might take some time to execute it. Therefore, we might rewrite this recurrence in a *closed form* – you plug in a number into a single formula, and it outputs the value directly, without looking up the previous value (which might take time).

As we now, there is a closed form a series of natural numbers. It was Gauss[4] who came up with it in 1786.

$$S_n = \frac{n(n+1)}{2}$$

The code is as follows:

```
static int SumOfIntegers2(int n)
{
    if (n > 0)
    {
        return (n * (n + 1)) / 2;
    }
    else
    {
        return 0;
    }
}
```

Sometimes there are recurrences that don't have a closed form. If they are used a lot, we usually define them using a short notation, e.g. factorial or exponent.

[4] *It seems a lot of stuff is attributed to Gauss – either he was really smart or he had a great press agent.* (Concrete Mathematics – A Foundation for Computer Science, p.6.)

Number Games

Introduction

Number games are probably the most basic algorithms that you can deal with. In this section, we will look at three different number games: playing with a friend version (i.e. you set values, your friend guesses), challenging your *pc*, and making the *pc* challenging you.

Playing versus a friend

This game is probably the most simple you can develop, because it does not require anything more than a part which takes in a value (the original number), and a part that checks whether the number entered (the guess of the original number) is less than/bigger than/equal to the original value. This procedure can then repeat for the second user as well.

In order to construct this program, we only need to know how to:

1. Write text on the screen
2. Read text from the screen
3. Create if-statements
4. Create while-loops

As you might see, you do not need to now a specific language, because the principle of an algorithm is the same in all programming languages. We can actually write an algorithm using *pseudocode*[5].

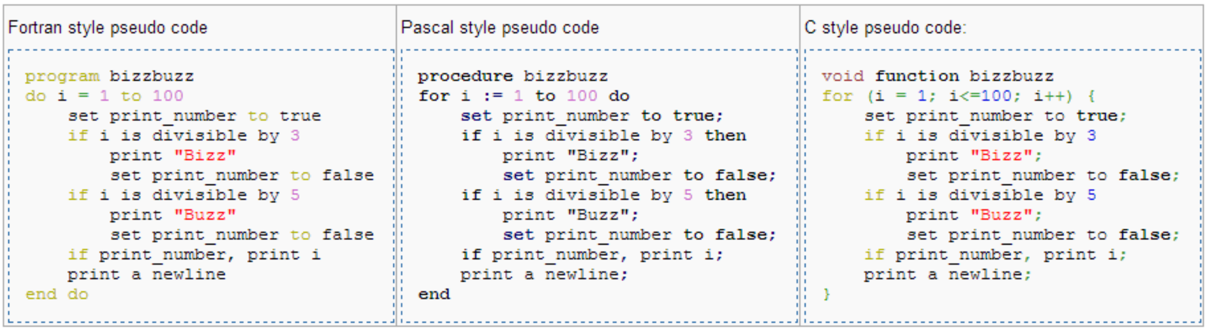
```
    static void Main(string[] args)
    {
        Console.WriteLine("Please spesify a number:");
        int originalNumber = Convert.ToInt32(Console.ReadLine());

        Console.Clear(); // clear everything from the screen

        bool continueAsking = true;
        int steps = 0;

        while (continueAsking)
        {
            int guess = Convert.ToInt32(Console.ReadLine());

            if (guess == originalNumber)
            {
                Console.WriteLine("You won by guessing " + steps + "time(s)!");
                continueAsking = false; // go out fromt the loop
            }
            else if (guess > originalNumber)
            {
                // if it is bigger, say that it should be smaller
                Console.WriteLine("It's smaller!");
            }
            else if (guess < originalNumber)
            {
                // if it is less, say that it should be bigger
                Console.WriteLine("It's bigger!");
            }

            steps++; // counts amount of guesses
        }

        Console.ReadLine(); // pause
    }
}
```

[5] Pseudocode – informal high-level description of the operating principle of a computer program or an algorithm.

Below, you see the how it might be expressed using C#, but pseudocode will make it much easier to read.

```
print “Please specify a number”

originalNumber = read
continueAsking = true

steps = 0

while as long as continueAsking is true
        guess = read
        if guess equals original number
                write “You won ” + steps + “times(s)!”
        end

        else if guess is bigger than originalNumber
                write “It’s smaller!”
        end

        else if guess is less than originalNumber
                write “It’s bigger!”.
        end

        increase step by
end

pause
```

Pseudocode is an informal language, and it is not required that the code follows certain syntax. However, there are different kinds of pseduocodes. Fortan style, Pascal style, and C style are just some of the syntax styles you might want to use.



Fortran style pseudo code

```
program bizzbuzz
do i = 1 to 100
    set print_number to true
    if i is divisible by 3
        print "Bizz"
        set print_number to false
    if i is divisible by 5
        print "Buzz"
        set print_number to false
    if print_number, print i
    print a newline
end do
```

Pascal style pseudo code

```
procedure bizzbuzz
for i := 1 to 100 do
    set print_number to true;
    if i is divisible by 3 then
        print "Bizz";
        set print_number to false;
    if i is divisible by 5 then
        print "Buzz";
        set print_number to false;
    if print_number, print i;
    print a newline;
end
```

C style pseudo code:

```
void function bizzbuzz
for (i = 1; i<=100; i++) {
    set print_number to true;
    if i is divisible by 3
        print "Bizz";
        set print_number to false;
    if i is divisible by 5
        print "Buzz";
        set print_number to false;
    if print_number, print i;
    print a newline;
}
```

Figure 1: These examples are pseudo codes, based on different language syntaxes. Picture from http://en.wikipedia.org/wiki/Pseudocode.

Computer challenger

This is almost the same as the previous example, but we need the number to be generated for us. The friend in this case is our computer. All we have to do is to add a Random object, which will generate a random number.

```
static void Main(string[] args)
{
    Random r = new Random();
    int originalNumber = r.Next(1,100);

    Console.Clear(); // clear everything from the screen

    bool continueAsking = true;
    int steps = 0;

    Console.WriteLine("What is my number?");

    while (continueAsking)
    {
        int guess = Convert.ToInt32(Console.ReadLine());

        if (guess == originalNumber)
        {
            Console.WriteLine("You won by guessing " + steps + "time(s)!");
            continueAsking = false; // go out fromt the loop
        }
        else if (guess > originalNumber)
        {
            // if it is bigger, say that it should be smaller
            Console.WriteLine("It's smaller!");
        }
        else if (guess < originalNumber)
        {
            // if it is less, say that it should be bigger
            Console.WriteLine("It's bigger!");
        }

        steps++; // counts amount of guesses
    }

    Console.ReadLine(); // pause
}
```

Computer's number guess

This part is the difficult one, because it can be solved in several ways. One way of approaching this problem could have been to collect the most common values that a human being can pick (using statistics). However, that is not the best way – what if a computer is playing this game (one computer is guessing, another is checking). Actually, random numbers generated by a computer can also be predicted (strange ha?). A computer is generating pseudo-random numbers. Let's pretend that a computer has the power of generating truly random numbers. How would our previous program work then?

So, we need a more general solution! Say the number we are to guess is 12, and 55. The question is, how should we start to guess the number, so that it is *almost* the same amount of guesses for both numbers, $P(x) = amountOfGuesses$. First, we need to specify a limit, in this example it is 100. Secondly, we can try to divide the limit by 2^n, as $n \to +\infty$ (you'll see that we get subgroups). Let's call this number for a factor, defined as following:

$$factor = \left\lceil \frac{limit}{2^n} \right\rfloor, n \in \mathbb{Z}$$

$$2^n < limit$$

In other words, we will either add or subtract the *factor*, depending on what the user tells us to do. The *factor* + *previous guess* (*previous factor*) will continue, until we get the original number. You will see that the difference between the previous *factor* and the current *factor* gets less, as the *n* increases. Remember, 2^n cannot be bigger than the *limit*, because it means that we have asked more than 100 times, which gets a bit sophisticated (we cannot ask more than 100 times, because it means that we have asked for the same number at least twice.)

What would then the numbers that the computer would ask for 12.

1. 50
2. 25
3. 13
4. 7
5. 10
6. 12 (yes, finally!)

So $P(12) = 6$

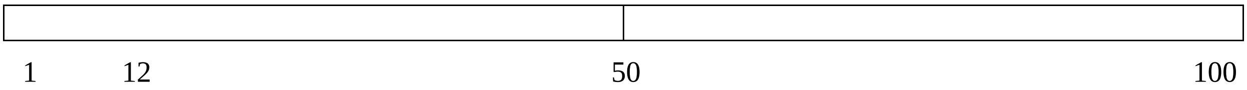
The same for 55:

1. 50
2. 75
3. 63
4. 57
5. 54
6. 56
7. 55.

So, $P(55) = 7$.

It seems that $P(x) \leq 7, 0 < x < 100$, can you prove why?

Probably, you would ask why we divide by 2 each time the computer makes a guess. That is done because we want to keep the same probability on both sides, i.e. 1-50 and 50-100, were there is 50% chance that the number will be in the first group, and vice versa. We continue like this, and in fact, we will always have 50% to get in any of these groups.



Please take a look at the code below:

```
/*
 * This is an example of an algorithm that will try to
 * guess the input number, "userNumber", as less as
 * possible.
 *
 * b = bigger
 * s = smaller
 * r = right guess
 *
 */
static void Main(string[] args)
{
    // max value, i.e. the range of guesses.
    int max = 100;

    // ask for the number
    Console.WriteLine("Please select a number:");
    int userNumber = Convert.ToInt32(Console.ReadLine());
    Console.Title = "Number game - " + "0 guesses. Your number is " + userNumber;
    Console.Clear();

    int previous = 0;
    int computerGuess = max / 2;
    string choice = "";

    for (int i = 0; i < max / 2; i++)
    {
        int factor = (int)Math.Round((decimal)( max / (decimal)Math.Pow(2, i+1)) );

        if (choice == "b")
        {
            computerGuess += factor;
        }
        else if (choice == "l")
        {
            computerGuess -= factor;
        }
        else if(choice == "r")
        {
            Console.WriteLine("Computers have power to read peoples' minds");
            Console.WriteLine("Amount of guesses is " + i + ".");
            Console.ReadLine();
            break;
        }

        Console.WriteLine("Is your number " + computerGuess + "?");
        choice = Console.ReadLine();

        Console.Title = "Number game - " + (i+1) + " guesses. Your number is " + userNumber;
    }
}
```

Additional Problems[6]

1) Say you have the series ‘A’ ‘E’ ‘I’ ‘M’. Consider also a function, M, which displays an element, given an index in this series, i.e. M(1) = ‘A’, M(3) = ‘I’. Describe the way you would construct this program, and hence find M(7).
2) Imagine you combine the elements of two sequences, and take only the terms that they have in common. If the first sequence is representing each second letter, and the second sequence represents each third, make a method that will represent the *new* sequence.
3) Decrypt following text given that the original language is English.

    RJ PFL TRE JVV, KYV CFEXVI KYV DVJJRXV ZJ, KYV VRJZVI ZK NZCC SV WFI PFL KF UVTIPGK ZK

4) A series of positive even numbers is given, e.g. 2, 4, 6, 8, etc.
    a. Create a program that will calculate the sum of this series using a recursive function.
    b. The same thing as in (a), but now, write this recurrence in a closed form.
5) Instead of the sum of all natural numbers, find a recurrence for the product. Is it possible to write it in a closed form?
6) The same as in (6), but instead, find the sum of all odd numbers.
7) How many slices of pizza can a person obtain by making n straight cuts? (More academically: what is the maximum number of L_n regions defined by n lines in the plane?)[7]

[6] Hopefully, you will be able to solve one of these problems!

[7] From Concrete Mathematics – A Foundation for Computer Science

Appendix A

Code snippet	Remark

`static int mod(int n, int b)`
`{`
`    return n - b * (int)Math.Floor((double)n / b);`
`}`
	Please use this snippet when working with negative numbers.[8]

`static int[] numToArray(int num)`
`{`
`    string stringNum = num.ToString();`
`    int[] result = new int[stringNum.Length];`
 	
`    for (int i = 0; i < stringNum.Length; i++)`
`    {`
`        result[i] = stringNum[i];`
`    }`
 	
`    return result;`
`}`
	Splits up a number into digits, and converts those into an integer array.

More code examples can be found at http://learning.artemlos.net/avc/.

[8] Please read more at http://blog.artemlos.net/news/the-right-modulo/.

American Standard Code for Information Interchange (ASCII)[9]

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

ASCII Name	Description	C# Escape Sequence
nul	null byte	\0
bel	bell character	\a
bs	backspace	\b
ht	horizontal tab	\t
np	formfeed	\f
nl	newline	\n
cr	carriage return	\r
vt	vertical tab	
esc	escape	
sp	space	

[9] The content at this page is based on a work at http://web.cs.mun.ca/~michael/c/ascii-table.html