Der ASCII-Code

Der ASCII-Code ist eine der wichtigsten Grundlagen der Informationstechnik. Er beschreibt, in welche Folge von Bits ein Zeichen der Tastatur übertragen wird. ASCII ist die Abkürzung für American Standard Code of Information Industry.

ASCII ist eine Kodierung der Zeichensymbole mit je 7 Bit.

Z.B. ist das große "A" die Bitfolge "1000001". Mit 7 Bit lassen sich $2^7 = 128$ Zeichen kodieren. Für alle Schriftzeichen dieser Welt ist das viel zu wenig. In den heute üblichen **Zeichencodierungen iso-8859-1 mit 8 Bit** oder **Unicode mit 16 Bit** ist der ASCII-Code aber immer als die ersten 128 Zeichen enthalten.

Hier sind ein paar Beispiele für Kodierungen:

	Zeichen	Bitfolge	Code
Im einem 8 Bit Code	A	100 0001	ascii
	@	100 0000	ascii
	ä	1110 0100	iso-8859-1
	ç	1110 0111	iso-8859-1
Im einem 16 Bit Code	A	0000 0000 0100 0001	Unicode
	ﻚ	0000 0100 0001 0110	Unicode
	€	0010 0000 1010 1100	Unicode
	ﻷ	1111 1110 1111 1100	Unicode

Dier ersten 32 Zeichen des ASCII

00	\0	NUL	Null
01		SOH	Start of Heading
02		STX	Start of Text
03		ETX	End of Text
04		EOT	End of Transmission
05		ENQ	Enquiry
06		ACK	Acknowledge
07	\a	BEL	Bell (Signalton)
08	\b	BS	Backspace
09	\t	HT	Horizontal Tabulator
10	\n	LF	Line Feed (neue Zeile)
11		VT	Vertical Tab
12		FF	Form Feed
13	\r	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

Danach kommen Zeichen der Tastatur

	32	0100000	@	64	1000000	`	96	1100000
!	33	0100001	A	65	1000001	a	97	1100001
"	34	0100010	B	66	1000010	b	98	1100010
#	35	0100011	C	67	1000011	c	99	1100011
$	36	0100100	D	68	1000100	d	100	1100100
%	37	0100101	E	69	1000101	e	101	1100101
&	38	0100110	F	70	1000110	f	102	1100110
'	39	0100111	G	71	1000111	g	103	1100111
(	40	0101000	H	72	1001000	h	104	1101000
)	41	0101001	I	73	1001001	i	105	1101001
*	42	0101010	J	74	1001010	j	106	1101010
+	43	0101011	K	75	1001011	k	107	1101011
,	44	0101100	L	76	1001100	l	108	1101100
-	45	0101101	M	77	1001101	m	109	1101101
.	46	0101110	N	78	1001110	n	110	1101110
/	47	0101111	O	79	1001111	o	111	1101111
0	48	0110000	P	80	1010000	p	112	1110000
1	49	0110001	Q	81	1010001	q	113	1110001
2	50	0110010	R	82	1010010	r	114	1110010
3	51	0110011	S	83	1010011	s	115	1110011
4	52	0110100	T	84	1010100	t	116	1110100
5	53	0110101	U	85	1010101	u	117	1110101
6	54	0110110	V	86	1010110	v	118	1110110
7	55	0110111	W	87	1010111	w	119	1110111
8	56	0111000	X	88	1011000	x	120	1111000
9	57	0111001	Y	89	1011001	y	121	1111001
:	58	0111010	Z	90	1011010	z	122	1111010
;	59	0111011	[	91	1011011	{	123	1111011
<	60	0111100	\	92	1011100	|	124	1111100
=	61	0111101	]	93	1011101	}	125	1111101
>	62	0111110	^	94	1011110	~	126	1111110
?	63	0111111	_	95	1011111		127	1111111

Das letzte Zeichen mit der Nummer 127 ist die DEL-Taste (Entf) zum Löschen eines Zeichens.

Aufbau eines Programms in C++ und in C#

C++	C#
Am Beginn werden hinter **#include** die zu ladenden Bibliotheken angegeben: "stdafx.h" ist eine Windowseigene Bibliothek und nur doert nötig. <iostream> lädt die Standard-Eingabe und Ausgabe Funktionen. Der **Namespace** ist eine logische Gruppe. namespace std bedeuted, dass jede Funktion ein vorausgestelltes "std::" erhält. Dieser Namensbereich ist vordefiniert. Das eigentliche Programm befindet sich in der **Hauptfunktion tmain().** Sie hat als Rückgabe einen Wert vom Typ "int" und deshalb als letzte Anweisung return 0; Die Argumente "int argc, _TCHAR* argv[]" werden nur benützt, wenn man beim Programmstart der exe-Datei weitere Parameter angibt.	Am Beginn werden hinter **using** die zu ladenden Bibliotheken angegeben: "System" enthält die Standard-Eingabe und Ausgabe Funktionen. Der **Namespace** ist eine logische Gruppe. Er hat den Namen des Projekts. Bei C# stehen Funktionen immer in einer Klasse. Bei Konsolenanwendungen heißt diese Klasse "Program". Das eigentliche Programm befindet sich in der **Methode Main() der Klasse Program**. Die Methode Main() muss statisch sein, das heißt sie kann ohne Objektbildung Program.Main() aufgerufen werden. Sie ist vom Typ "void" und hat keinen Return-Wert. Die Argumente " `string[] args`" werden nur benützt, wenn man beim Programmstart der exe-Datei weitere Parameter angibt.

C++	

`#include <iostream>`
`using namespace std;`
 	
`int main()`
`{`
 	
`  // Programm`
 	
`return 0;`
`}`	

C++ Visual Studio	**C# Visual Studio**

`#include "stdafx.h"`
`#include <iostream>`
`using namespace std;`
 	
`int _tmain(int argc, _TCHAR* argv[])`
`{`
	`using System;`
	
	`namespace anton`
	`{`
	`class Program`
	`{`
	`static void Main(string[] args)`
	`{`

`return 0;`
`}`
	
	`}`
	
	`} // Ende der Klasse`
	`} // Ende des Namespace`

Der **Compiler** übernimmt den gesamten Programmtext und versucht ihn in die Maschinensprache zu übersetzen. Wird ein Programmierfehler (Syntaxfehler) entdeckt, liefert der Compiler eine Fehlermeldung und nimmt keine Übersetzung vor. Klappt die Übersetzung, wird der Maschinencode als neue binäre Datei gespeichert. Diese alleine ist aber noch nicht selbständig als Programm lauffähig. Der **Linker** übernimmt diese Aufgabe und produziert aus dem Objectcode ein lauffähiges Programm auf dem Betriebssystem.

Die Syntax

Unter der "**Syntax**" einer Programmiersprache versteht man alle Regeln für die Programmerstellung. Nur syntaktisch richtige Programme werden vom Compiler in die Maschinensprache übersetzt. Syntaxfehler werden vom Compiler erkannt und in einer Fehlerliste angezeigt.

Regel 1:

Schlüsselwörter sind reserviert und dürfen nicht als Variablennamen verwendet werden!

Regel 2:

Jede Anweisung wird durch ";" beendet.

Regel 3:

Programmblöcke werden in { } geschrieben. Die Programmblöcke können geschachtelt werden, sie dürfen sich aber nicht überlappen.
Vereinbarung: Für jede weitere Schachtelungstiefe wird der Text weiter eingerückt.

Regel 4:

Bei Zuweisungen steht die Variable, die den Wert erhält links und der Rechenausdruck rechts vom "="-Zeichen.

Kommentare

Der Doppel-Slash // leitet einen Kommentar bis zum Zeilenende ein
Mehrzeilige Kommentare stehen zwischen /* und */

Variablen und Datentypen

Wie in jeder höheren Programmiersprache werden symbolische Namen, sog. "Variablen" zur Ablage von Daten im Hauptspeicher verwendet. Alle Variable müssen mit ihrem Datentyp deklariert werden. Die Deklaration von Variablen kann innerhalb des Programms nach folgendem Schema erfolgen:

```
datentyp var1, var2,  ... ;
```

Erst nachdem eine Variable deklariert ist, kann ihr mit dem "=" - Zeichen ein Wert zugewiesen werden.
Für Texte verwenden wir den Typ **string**.

Es gibt Unterschiede zwischen ganzen Zahlen, Kommazahlen oder Textzeichen. Das liegt daran, dass diese verschieden verarbeitet werden müssen und unterschiedlich große Bereiche im Speicher reservieren: Die **Wertzuweisung einer Variablen** während des Programmablaufs übernimmt das **„=“ Zeichen**.

	Abkürzung	**Platzbedarf**	**Beispiel**
Textzeichen	char	1 Byte	char c = 'a';
ganze Zahlen	int	4 Byte	int a = 5;
Kommazahlen	double	8 Byte	double d = 3.5;
Text	string	je nach Zeichenzahl	string s="Anton";

Während man Strings z.B. "das ist ein Text" in doppelte Hochkomma einschließt, sind die einzelnen Zeichen immer nur in einfachen Hochkommas: 'A' oder '3'. Beachte, dass man nur mit den Datentypen für Zahlen rechnen kann!

Operatoren-Übersicht

Zuweisen	= += -= *= /= %= &= |= ^= <<= >>= ??
Vergleichen	== != < > <= >=
Berechnungen	+ - * / %
Bit Verknüpfungen	& | ^ ! ~ AND OR XOR NOT Komplement
Logische Verknüpfungen	! && || true false NOT AND OR
String-Verknüpfung	+
Increment, Decrement	++ --
Bitweises Verschieben (Shift)	<< >>