Tekst gebruiken in code

Zie Scherp Scherper - Hoofdstuk 3

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- het verschil tussen **`char`** en **`string`** uitleggen
- **ASCII** en **UNICODE** situeren en weten hoe .NET tekst bewaart
- **escape characters** (`\n`, `\t`, `\"`, …) en het **verbatim** teken `@` gebruiken
- tekst samenvoegen en formatteren met **string interpolation** (`$`)
- speciale **UNICODE-tekens** op het scherm tonen
- info over de machine ophalen via de **`Environment`**-bibliotheek

Alles is een char

- Elk teken dat je intypt is een **`char`**
- Je toetsenbord toont maar een fractie van alle mogelijke tekens (vergelijk met een Spaans, Tunesisch of Chinees toetsenbord)

Voor we tekst inlezen, moeten we begrijpen **hoe** een computer al die tekens voorstelt.

Van ASCII naar UNICODE

- **ASCII**: de oude standaard, maar 128 tekens (7 bits)
- **UNICODE**: de opvolger, ruimte voor meer dan 1 miljoen tekens
- De eerste 128 UNICODE-tekens zijn identiek aan ASCII (compatibel)
- .NET bewaart een `string` intern als **UTF-16**

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[END OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

De eerste 128 tekens (bron: Wikipedia).

> **Tip**
>
> De eerste 32 tekens zijn “onzichtbare” stuurtekens uit het Telex-tijdperk (*line feed*, *bell*, …).

Char

```
1 char eenLetter = 'X';
2 Console.WriteLine(eenLetter);
```

- Één enkel karakter, tussen **apostrofs** (`'`)
- **Intern bewaard als een 16-bit getal** (de UNICODE-waarde)

> **Belangrijk**
>
> `char eenGetal = '7';` bewaart het teken `7`, niet het getal 7. Wil je ermee rekenen, dan moet je eerst converteren (hoofdstuk 4).

String

> Een **`string`** is een reeks van 0, 1 of meer **`char`**-elementen.

- Tekst staat tussen **dubbele aanhalingstekens** (`"`)
- Een `char` gebruikt een apostrof, een `string` aanhalingstekens

> **Tip**
>
> In hoofdstuk 8 ontdek je dat een `string` eigenlijk een **array** van chars is.

Drie keer “1”, drie types

```
1 char eenKarakter = '1';
2 string eenString = "1";
3 int eenGetal = 1;
4
5 Console.WriteLine(eenKarakter);
6 Console.WriteLine(eenString);
7 Console.WriteLine(eenGetal);
```

Driemaal een `1` onder elkaar, maar telkens een ander **type**. Boeiend programma, hoor.

Escape characters

Niet de boeiendste materie (geen opvolger van Prison Break), maar wie ze beheerst, tovert mooiere tekst op het scherm.

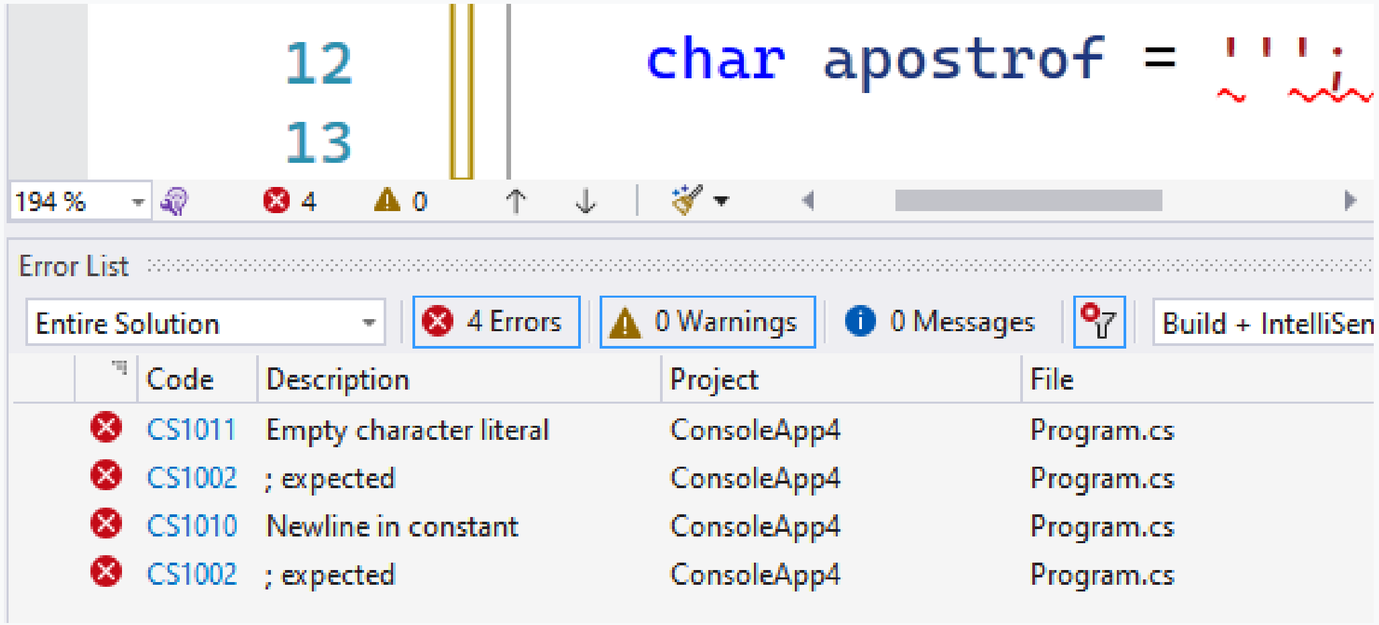
Sommige tekens hebben een **speciale functie** (bv. `"` begrenst een string). Met een **backslash** (`\`) zeg je: behandel het volgende teken letterlijk.

Het probleem, en de redding

```
1  char weglatingsteken = '''';  // VS raakt volledig in de war
```

Escape characters to the rescue:

```
1  char weglatingsteken = '\'';
```



De belangrijkste escape characters

```
\'      apostrof
\"      aanhalingsteken
\\      backslash
\n      nieuwe lijn (enter / newline)
\t      horizontale tab
\uxxxx  teken met hexadecimale UNICODE-waarde xxxx
```

Escape in strings

```
1  string woord = "\'s avonds";
```

Code op het scherm tonen wordt al snel *Inception*:

```
1  Console.WriteLine("Console.WriteLine(\"Cool he\");");
```

Uitvoer:

```
1  Console.WriteLine("Cool he");
```

Witregels en tabs

```
1 string eenString = "Een zin.\t na een tab \nDan een nieuwe regel";
2 Console.WriteLine(eenString);
```

Uitvoer:

```
1 Een zin.         na een tab
2 Dan een nieuwe regel
```

> **Tip**
>
> `\t` springt naar de volgende vaste **tabstop**, handig om data uitgelijnd in een tabel te zetten.

Het verbatim teken @

Het apenstaartje @ voor een string zegt: **behandel alles letterlijk**, escape characters worden genegeerd.

```
1  string zonderAt = "C:\\Temp\\Myfile.txt";
2  string metAt = @"C:\Temp\Myfile.txt";
```

> **Opmerking**
>
> Aanhalingstekens moet je nog steeds escapen. Heb je een " in je tekst, dan kan @ niet helpen.

Strings samenvoegen met +

De `+`-operator plakt tekst aan elkaar (**concatenatie**), van links naar rechts:

```
1  Console.WriteLine("1" + 1 + 1);     // 111
2  Console.WriteLine(1 + 1 + "1");     // 21
3  Console.WriteLine("1" + (1 + 1));   // 12
```

> ⚠ **Waarschuwing**
>
> De **linkse operand** bepaalt het resultaat bij twijfel. Bij `"1" + 1 + 1` wordt eerst `"11"`, dan `"111"`. Bij `1 + 1 + "1"` wordt eerst `2`, dan `"21"`.

String interpolation met $

Lange `+`-zinnen worden onleesbaar. Zet een **`$`** voor de string en alles tussen **`{ }`** wordt als code uitgevoerd:

```
1 int leeftijd = 13;
2 string naam = "Finkelstein";
3 string zin = $"Ik ben {naam} en ik ben {leeftijd} jaar.";
```

Veel leesbaarder dan de `+`-variant.

> **Tip**
>
> Vanaf nu gebruik ik bijna altijd string interpolation. Het is de moderne aanpak en 99% van de tijd leesbaarder.

Expressies in interpolatie

Tussen de accolades mag **elke expressie**:

```
1  string a = $"Ik ben {leeftijd + 4} jaar.";          // 17
2  string b = $"Je naam telt {naam.Length} tekens.";   // methode-aanroep
3  Console.WriteLine($"3 maal 9 is {3 * 9}");           // 27
```

De expressie wordt eerst berekend, dan in de tekst geplaatst.

Formatteren

Plaats een dubbelpunt na de expressie om de weergave te sturen:

```
1  double number = 12.345;
2  Console.WriteLine($"{number:F2}");   // 12,35  (afgerond, niet afgekapt)
```

- `F2`: kommagetal met 2 cijfers na de komma
- `D5`: geheel getal met minstens 5 cijfers (`123` wordt `00123`)
- `C`: geldbedrag volgens de landinstellingen

Formatteren met een masker

```
1  double number = 12.3;
2  Console.WriteLine($"{number:0.00}");    // 12,30
3  Console.WriteLine($"{number:000.00}");  // 012,30
```

> ⚠ **Waarschuwing**
>
> **Komma of punt hangt van de computer af.** Op een Belgisch systeem zie je `12,30`, op een Engels `12.30`. Dat maakt je uitvoer **machine-afhankelijk**. Wil je dat vastzetten, dan leer je later `CultureInfo` kennen.

De char-val: ‘A’ + ‘B’

```
1  char letter1 = 'A';
2  char letter2 = 'B';
3  Console.WriteLine(letter1 + letter2);
```

Op het scherm: `131`, niet “AB”.

Een `char` bewaart een getal: `A` = 65, `B` = 66. De `+`-operator bestaat niet voor `char`, dus C# rekent met `int`: 65 + 66 = **131**.

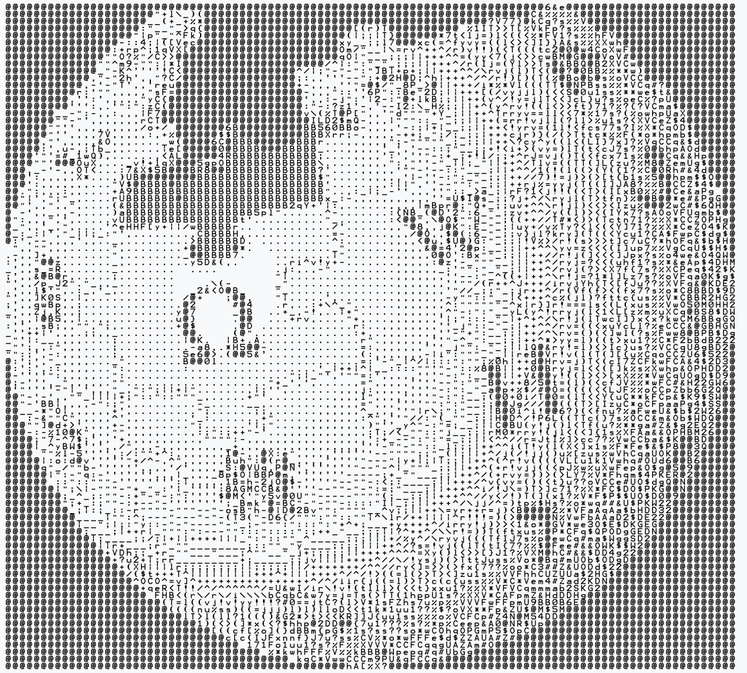
Speciale UNICODE-tekens tonen

Eerst de juiste encoding instellen als **allereerste lijn** in `Main`:

```
1  Console.OutputEncoding = System.Text.Encoding.UTF8;
```

Daarna, bv. het copyright-teken (UNICODE `0x00A9`):

```
1  char copyright = (char)0x00A9;
2  Console.WriteLine(copyright);
3  Console.WriteLine("©");
```



Dit Wikipedia-logo bestaat volledig uit UNICODE-tekens.

UNICODE-kunst

Multiline ASCII-art toon je met een verbatim string `@`:

```
string titel = @"
 _______ _____
|__   __|  __ \
   | |  | |  | |
   | |  | |  | |
   |_|  |_|__/ ";
Console.WriteLine(titel);
```

> **Tip**
>
> `$` en `@` mogen samen: zowel `$@"..."` als `@$"..."` werkt en doet hetzelfde.

De Environment-bibliotheek

`Console` is maar één bibliotheek. **`Environment`** geeft info over de machine waarop je programma draait:

```
1 string pcname = Environment.MachineName;
2 int proccount = Environment.ProcessorCount;
3 string username = Environment.UserName;
4 Console.WriteLine($"Je computer heet {pcname} met {proccount} processoren.");
```

Belangrijk

Op een andere computer krijg je andere resultaten. Test zulke code dus op meerdere systemen.

Programma afsluiten

```
1  Environment.Exit(0);
```

Het getal is de **exitcode**: handig om later in logbestanden te zien waarom een programma stopte.

C# leren lijkt in het begin soms saai, daarom strooi ik er af en toe iets geavanceerds tussen. Niemand kent élke bibliotheek: het is aan jou om te ontdekken welke nuttig zijn voor jouw probleem.

Conclusie

- Een `char` is één teken (een 16-bit getal), een `string` is een reeks chars
- **UNICODE** (UTF-16 in .NET) volgt ASCII op en bevat meer dan een miljoen tekens
- **Escape characters** (`\n`, `\t`, `\"`) en `@` sturen hoe tekst geïnterpreteerd wordt
- **String interpolation** (`$"...{expressie}..."`) is de leesbare, moderne manier om tekst op te bouwen
- Let op de **char-val** (`'A' + 'B'` = 131) en op **machine-afhankelijke** formattering

> **Tip**
>
> Volgende halte: **werken met data**: converteren, casten en rekenen.

Meer weten

Kennisclips

- Char en strings
- Escape characters
- String interpolation
- Unicode tonen
- Environment bibliotheek

Oefeningen H3 · Quizlet flashcards

Zie verder: andere talen

Dezelfde concepten uit dit hoofdstuk, maar dan in andere talen. Handig om later code in Python of JavaScript te leren lezen.

Zie verder: string als char-array

In **C** bestaat er geen echt string-type. Tekst is daar letterlijk een array van `char`, afgesloten met een null-karakter `\0`:

```c
char woord[] = "Tim";   // {'T', 'i', 'm', '\0'}
printf("%s\n", woord);
```

In C# is `string` een volwaardig type met handige methodes. Het idee “een string is een reeks chars” zie je in C dus heel letterlijk terug.

Zie verder: het verbatim-idee elders

Python kent net hetzelfde, daar heet het een *raw string*. Je plaatst een `r` voor de string in plaats van een `@`:

```
1  met_r = r"C:\Temp\Myfile.txt"
```

Backslashes worden gewoon als tekst behandeld, niet als escape character. Handig voor bestandspaden en reguliere expressies.

Zie verder: interpolatie elders

Python (f-string) en **JavaScript** (template literal) doen exact hetzelfde, met een ander tekentje vooraan:

```python
zin = f"Ik ben {naam} en ik ben {leeftijd} jaar."
```

```javascript
const zin = `Ik ben ${naam} en ik ben ${leeftijd} jaar.`;
```

Samenvattende poster

Een handige samenvattende poster (Opgelet: gemaakt m.b.v. ChatGPT Image Gen 2).