

Exkurs: Wahl-O-Mat: Kennen Sie den Wahl-O-Mat¹ der Bundeszentrale für politische Bildung (bpb)?



LS 1.2 Situation „Wahl-O-Mat“

Herr Müller benötigt für die kommende Stichwahl des Betriebsratsvorsitzenden einen kleinen „Wahl-O-Mat“, welcher die Stimmen der Kandidaten einliest und die Prozentverteilung ausgibt.



Hr. Lars Müller

Geschäftsführung

-----Wahl-O-Mat-----

Bitte geben Sie die Stimmen für Kandidat 1 an: 123
Bitte geben Sie die Stimmen für Kandidat 2 an: 234

Kandidat 1 (in Prozent): 34.4538
Kandidat 2 (in Prozent): 65.5462

Wie viele Variablen benötigen wir für die Umsetzung dieses Programms?

5-9 je nachdem wie man es coded

¹ <https://www.bpb.de/politik/wahlen/wahl-o-mat/>

LS1.2.1 Variablen

Variablen ermöglichen es, Werte oder Informationen im Programm zu halten, um sie später zu verwenden oder zu verarbeiten. Bei der Definition einer Variable wird der Datentyp und Name der Variable festgelegt und der erforderliche Speicherplatz für den entsprechenden Datentyp im Arbeitsspeicher reserviert. Anschließend kann der Variable ein Wert zugewiesen werden (Erstzuweisung = Initialisierung). Welcher Wert zugewiesen werden darf, ist vom Variablentyp abhängig.

LS1.2.2 Übersicht über die wichtigsten Datentypen in C#

C# Datentyp	beinhaltet	Minimum kleinster Wert	Maximum größter Wert	Speicherplatz
Integrale numerische ganze Zahlen				
bool (boolean)	entweder Wert true oder false	false	true	8 Bit, 1 Byte
char	Zeichen (Character)	0	65.535	16 Bit, 2 Byte
short	pos. und neg. Ganzzahl	-32.768	+32.767	16 Bit, 2 Byte
ushort	pos. Ganzzahl	0	65.535	16 Bit, 2 Byte
int	pos. und neg. Ganzzahl	-2.147.483.648	2.147.483.647	32 Bit, 4 Byte
uint	pos. Ganzzahl	0	4.294.967.295	32 Bit, 4 Byte
long	pos. und neg. Ganzzahl	-9.223.372.036.854.775.808	9.223.372.036.854.775.807	64 Bit, 8 Byte
ulong	pos. Ganzzahl	0	18.446.744.073.709.551.615	64 Bit, 8 Byte
Gleitkommatypen				
float	Gleitkommazahl (ca. 6-9 Stellen)	-3,4E+38	+3,4E+38	32 Bit, 4 Byte
double	genauere Gleitkomma- zahl (ca. 15-16 Stellen)	-1,7E+308	+1,7E+308	64 Bit, 8 Byte
decimal	noch genauere Gleitkom- mazahl (ca. 28-29 Stellen)	-7,9E+28	+7,9E+28	128 Bit, 16 Byte

Um Verständlichkeit und Klarheit zu gewährleisten, halten Sie sich bitte im IT-Unterricht an die zweckmäßige Verwendung der verschiedenen Datentypen.

Die Nutzung von Datentypen auf eine Art und Weise, die nicht der allgemein gültigen Konvention entspricht, wie beispielsweise die Verwendung von `char` für die Speicherung ganzer Zahlen, kann zu Verwirrung und potenziellen Fehlern führen, besonders wenn diese Werte dann in Berechnungen verwendet werden.

```
// How a shower should work:  
int temperature;  
  
// How they really work:  
boolean temperature;
```

LS1.2.3 Anlegen von Variablen (Wiederholung, wie bei RobotC)

Variable **anlegen** (in C# spricht man von deklarieren):

```
int i_zahl;  
//Reservierung von Speicherplatz für einen Datentyp int mit dem Namen i_zahl
```

Variable **anlegen und** gleich danach **initialisieren** (d. h. einen Erstwert zuweisen):

```
int i_zahl = 7;  
//Speicherreservierung für eine Variable vom Datentyp int mit dem Namen i_zahl und speichern  
des Wertes 7 in der Variablen.
```

Einer bereits angelegten Variable wird ein Wert **zugewiesen**:

```
i_zahl = 7;  
// speichern des Wertes 7 in der Variablen i_zahl
```

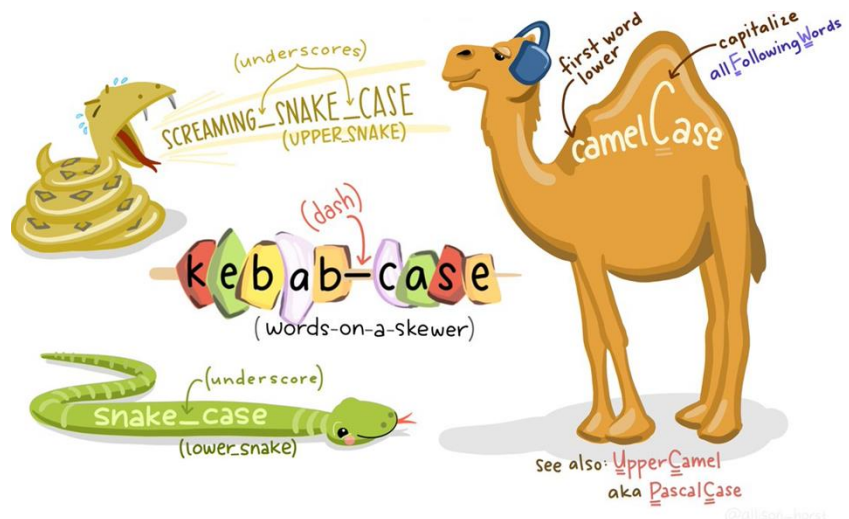
Die Programmierrichtlinie für den AEuP-Unterricht sieht vor, dass der eigentliche Variablenname noch den Datentyp als Buchstabe vorangestellt bekommt. Ferner sollen immer sprechende Namen, die den Variableninhalt gut beschreiben, verwendet werden. Im letzten Beispiel also statt:

```
int a = 6; → besser: int i_Unterrichtsstunden=6;
```

Exkurs: Namenskonventionen

Namenskonventionen sind Teil von Kodierrichtlinien in großen Projekten. Sie sorgen für Einheitlichkeit und Verständlichkeit des Codes. Viele Programmiersprachen und Projekte haben etablierte Richtlinien, die als Basis oder vollständige Anleitung dienen können.

- **Variablen und Objekte:**
meist in lowerCamelCase
- **Funktionen und Methoden:**
oft in lowerCamelCase, mit Verben im Namen
- **Klassen:** in PascalCase, um sie von Instanzen zu unterscheiden
- **Konstanten:** in SNAKE_CASE für bessere Erkennbarkeit
- **Dateinamen:** variieren, oft Snake Case oder Kebab Case, um Kompatibilitätsprobleme zu vermeiden.



Weitere Beispiele zum Anlegen von Variablen mit passender Initialisierung

```
bool b_korrekt = 1;
bool b_falsch = false;
float f_Milchtuete = 0.99f;
double d_Pi = 3.14159265359;
```

Um einer char-Variable einen Buchstaben zuzuweisen darf man **NICHT**

```
char c_zeichen = "a";
```

schreiben, sondern muss

```
char c_zeichen = 'a';
```

schreiben, denn "a" ist ein String (also eine Kette von mehreren Zeichen) und 'a' ein einzelnes Zeichen.

Folgende Standardtypen werden im AWP-Unterricht stets vorausgesetzt		
Bezeichnung	beinhaltet	Größe in Byte bei 32bit OS
int	pos. und neg. Ganzzahl	4
float	Gleitkommazahl	4
double	genauere Gleitkommazahl	8
char	Zeichen (Character)	2
bool	entweder Wert true oder false	1
uint	pos. Ganzzahl	4

Merke: Bei einer **Zuweisung** steht die Variable, der ein Wert zugewiesen werden soll, immer auf der linken Seite vom „**=**“. Die Zuweisung erfolgt also von rechts nach links.

Beispiel: `i_Nummer = 5;` //bedeutet, der Variable i_Nummer wird der Wert 5 zugewiesen

LS1.2.4 Planung Wahl-O-Mat:

Welche Variablen benötigt man, um für zwei Kandidaten die Stimmen zu zählen, alle abgegebenen Stimmen zu ermitteln und daraus die prozentuale Stimmverteilung zu berechnen? Überlegen Sie sich eine sinnvolle Benennung und definieren sowie initialisieren Sie die Variablen.

LS1.2.5 Programmierung Wahl-O-Mat:

Programmieren Sie nun (**unter Beachtung der nächsten Kapitel 1.2.6 und 1.2.7**) einen eigenen „Wahl-O-Mat“, welcher die Stimmen für zwei Kandidaten einliest und das prozentuale Stimmverhältnis ausgibt und testen Sie ihn ausführlich.

```
-----Wahl-O-Mat-----
```

```
Bitte geben Sie die Stimmen für Kandidat 1 an: 123  
Bitte geben Sie die Stimmen für Kandidat 2 an: 234
```

```
Kandidat 1 (in Prozent): 34.4538  
Kandidat 2 (in Prozent): 65.5462
```

Laden Sie im Anschluss Ihr fertiges Programm in den Schülerlösungsordner.

LS1.2.6 Wissenswertes zur Programmierung des „Wahl-O-Mat“:

Einlesen mit „`Console.ReadLine()`“

Um Werte, welche im Konsolenfenster über die Tastatur eingegeben werden, in einer Variable zu speichern, verwenden wir die Funktion **`Console.ReadLine()`** (ebenfalls aus der `system`-Bibliothek). Vor der Funktion muss nur die entsprechende Variable genannt werden, um die getätigte Eingabe als Rückgabewert der Funktion in die Variable zu speichern.

Um bspw. einen Wert in die bereits definierte (String) Variable **`s_Text`** zu speichern, schreiben Sie:

```
s_Text = Console.ReadLine();
```

In der Konsole erscheint dann bei Ausführung ein blinkender Unterstrich (Cursor), welcher auf Ihre Eingabe wartet.

Da **`Console.ReadLine()`** immer nur eine Zeichenkette einliest, müssen Zahlenwerte (die bspw. in Integer-Variablen gespeichert werden) erst von String (nach `Int32`) konvertiert werden.

Um bspw. einen Wert in die bereits definierte (int bzw. uint) Variable **`i_Zahl`** bzw. **`ui_Zahl`** zu speichern, schreiben Sie:

```
i_Zahl = Convert.ToInt32( Console.ReadLine() );  
ui_Zahl = Convert.ToUInt32( Console.ReadLine() );
```

Dabei wird der Rückgabewert der `Console.ReadLine`-Funktion (eine Zeichenkette) direkt als Übergabeparameter für die `Convert.ToInt32`-Funktion verwendet. Letztere gibt dann einen Zahlenwert (32-Bit-Integer) zurück, welcher der Variable **`i_Zahl`** zugewiesen wird.

LS1.2.7 EVA-Prinzip

Jeder Informationsverarbeitungsprozess lässt sich in die *drei Stufen* Eingabe, Verarbeitung und Ausgabe einteilen (man spricht vom so genannten *EVA-Prinzip*):

<u>E</u> ingabe	<u>V</u> erarbeitung	<u>A</u> usgabe
-----------------	----------------------	-----------------

Das Prinzip gilt also auch für menschliche Informationsverarbeitungsprozesse (z. B. Schüler-Lehrer):

<u>E</u> ingabe (über das Ohr)	<u>V</u> erarbeitung (im <u>K</u> opf)	<u>A</u> usgabe (Mund)
Der Schüler hört die Frage des Lehrers: "Was ist $250 + 300$?"	Der Schüler denkt darüber nach und errechnet das Ergebnis.	Der Schüler nennt das Ergebnis: "550".

Erst recht gilt dieses Prinzip aber für den Computer und die Programmierung:

<u>E</u> ingabe (per Tastatur)	<u>V</u> erarbeitung (im PC)	<u>A</u> usgabe (am Monitor)
Einlesen der Stimmen: <code>i_Kand1 = Convert.ToInt32(Console.ReadLine());</code> <code>i_Kand2 = Convert.ToInt32(Console.ReadLine());</code>	Berechnung der Summe: <code>i_ges = i_Kand1 + i_Kand2;</code>	Ausgabe im Konsolenfenster: <code>Console.WriteLine(i_ges);</code>

Sie können auch direkt in der Ausgabe rechnen. Bsp.:

```
Console.WriteLine(i_stimmenKand1 + i_stimmenKand2);
```

ABER:

Die Schreibweise entspricht **nicht** dem EVA-Prinzip, da Verarbeitung und Ausgabe nicht getrennt sind.

Bitte beachten Sie stets das EVA-Prinzip.
Es wird bei allen Aufgaben in AWP vorausgesetzt.

LS1.2.8 Weitere Übungsaufgaben

1. Bitte kreuzen sie an, ob die folgende Aussage wahr oder falsch ist:

Aussage (immer bezogen auf C#)	wahr	falsch
Mit <code>double d_zahl;</code> wird eine double-Variable mit dem Namen <code>d_zahl</code> angelegt in welcher nur ganze Zahlen gespeichert werden können.		✗
<code>char c_zeichen;</code> reserviert 2 Bytes im Arbeitsspeicher.	✓	
Mit <code>bool b_tollesWetter = false;</code> wird eine Bool-Variable angelegt und mit dem Wert <code>false</code> initialisiert.	✓	
<code>char c_zeichen = "B";</code> definiert eine Character-Variable mit dem Namen <code>c_zeichen</code> und dem Wert bzw. Zeichen <code>B</code> .		✗
<code>uint i_zahl = -5;</code> definiert eine Integer-Variable mit dem Namen <code>i_zahl</code> und dem Wert <code>-5</code> .		✗
Variablendefinitionen ohne Initialisierung sind in Ordnung, da neue Variablen standardmäßig immer auf 0 gesetzt werden.		✗

2. Welche Werte können bei einer Modulo 6 Operation herauskommen?

(Lösungshinweis: bspw. einmal testweise alle Ganzzahlen von 18-24 jeweils Modulo 6 rechnen)

0,1,2,3,,4,5

3. Wenn der Befehl `r.Next()` ihnen eine zufällige positive ganze Zahl gibt, wie lautet dann die Codezeile um ein zufälliges Würfelergebnis (eines 6-seitigen Würfels) in die bereits definierte Variable `i_Wurf` zu schreiben?

Random `r = new Random();`

`i_Wurf = r.next()%6+1`

4. Welche (exakten) Werte stehen nach Ausführung der folgenden Befehle (korrektes Programmgerüst vorausgesetzt) in den beiden Ergebnisvariablen?

`double d_Wert = 3.5; int i_Wert = 7;`

`double d_Ergebnis = i_Wert / d_Wert;`

// d_Ergebnis: 2.0 double

`int i_Ergebnis = (int)(i_Wert * d_Wert);`

// i_Ergebnis: 24 int



LS1.2.9 Casting

In C# gibt es zwei Arten von arithmetischen Datentypen:

- Ganzzahlige Typen, z.B. `int, uint`
- Gleitkommatypen, z.B. `float, double`

Automatische Typumwandlung

Bei Berechnungen in Programmen gelten folgende Regeln:

- **Wenn verschiedene Datentypen beteiligt sind, wird das Ergebnis automatisch in den leistungsfähigeren Typ umgewandelt. (=implizites Casting)**

Beispiel:

```
int i_zahl1 = 5;
float f_zahl2 = 7.5f;
double d_zahl3 = 10.3;

wert = i_zahl1 * f_zahl2; // Ergebnis: 37,5 Typ: float
wert = f_zahl2 + d_zahl3; // Ergebnis: 17,8 Typ: double
```

- **Wenn nur gleiche Datentypen beteiligt sind, hat auch das Ergebnis diesen Datentyp.**

Beispiel:

```
int i_zahl1 = 12;
int i_zahl2 = 4;

wert = i_zahl2 / i_zahl1; // Ergebnis: 0 Typ: integer
```

Erzwungene Typumwandlung

Wenn man einen bestimmten Datentyp erzwingen möchte, schreibt man diesen Typ in Klammern vor den Ausdruck, der berechnet wird. (=explizites Casting)

Beispiel:

```
int i_zahl1 = 5;
int i_zahl2 = 6;

f_Ergebnis = ( double ) i_zahl1 / i_zahl2; // Ergebnis: 0,83 Typ: double
```

Diese erzwungene Umwandlungsoperation nennt man auch

LS1.2.10 Weitere Übungsaufgaben zu Casting



Gegeben sind folgende Variablen:

```
int i_a      = 5; float f_c  = 2.5f;
int i_b      = 10; double d_d = 18.999999999;
```

Versuchen Sie zuerst die Aufgaben selbst (falls nötig auch mit Taschenrechner) zu lösen. Falls Sie sich unsicher sind, kopieren Sie den Code in **Visual Studio** und ermitteln so die Ergebnisse. Welcher Wert steht in der jeweiligen Ergebnisvariable.

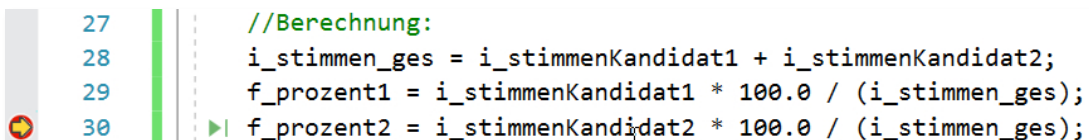
<code>int i_Ergebnis1 = i_b / i_a;</code>	<code>// Wert in i_Ergebnis1: int 2</code>
<code>int i_Ergebnis2 = i_a / i_b;</code>	<code>// Wert in i_Ergebnis2: int 0</code>
<code>float f_Ergebnis1 = i_a / i_b;</code>	<code>// Wert in f_Ergebnis1: int 0</code>
<code>float f_Ergebnis2 = (float)i_a / (float)i_b;</code>	<code>// Wert in f_Ergebnis2: float 0,5</code>
<code>float f_Ergebnis3 = (float)i_a / i_b;</code>	<code>// Wert in f_Ergebnis3: float 0,5</code>
<code>float f_Ergebnis4 = i_a / (float)i_b;</code>	<code>// Wert in f_Ergebnis4: float 0,5</code>
<code>int i_Ergebnis3 = (float)i_a / (float)i_b;</code>	<code>// Wert in i_Ergebnis3: float 0,5</code>
<code>int i_Ergebnis4 = i_b / f_c;</code>	<code>// Wert in i_Ergebnis4: float 4,0</code>
<code>int i_Ergebnis5 = f_c / i_b;</code>	<code>// Wert in i_Ergebnis5: float 0,025</code>
<code>float f_Ergebnis5 = (float)d_d;</code>	<code>// Wert in f_Ergebnis5: float 18,999999999</code>
<code>double d_Ergebnis1 = i_b / (int)d_d;</code>	<code>// Wert in d_Ergebnis1: int 0</code>
<code>double d_Ergebnis2 = d_d / f_c;</code>	<code>// Wert in d_Ergebnis2: double 7,5999999996</code>
<code>bool b_wahr1 = 0;</code>	<code>// Wert in b_wahr1: bool false</code>
<code>bool b_wahr2 = 1;</code>	<code>// Wert in b_wahr2: bool true</code>
<code>double d_Ergebnis3 = (int)((double)f_c * (double)i_a);</code>	<code>// Wert in d_Ergebnis3: int 12</code>

LS1.2.11 Debugging

Der Debugger von Visual Studio kann Ihnen dabei helfen, Probleme in Ihrem Code effizient zu finden. Mit dem Debugger können Sie Ihr Programm ausführen und anhalten, bevor bestimmte Codezeilen ausgeführt werden, damit Sie überprüfen können, was Ihr Programm gerade macht.

Breakpoints:

Ein Breakpoint/Haltepunkt ist ein Marker, der angibt, vor welcher Zeile in Visual Studio die Ausführung anhalten soll, sodass Sie einen Blick auf die Werte von Variablen oder das Verhalten des Arbeitsspeichers werfen können. Wenn Ihr Programm in einen Haltepunkt läuft und pausiert, können Sie Teile Ihres Programms überprüfen, um Probleme in diesem Bereich Ihres Codes zu finden. Es ist die grundlegendste Funktion beim Debuggen. Um einen Haltepunkt festzulegen, klicken Sie links neben der Zeilennummer in den grauen Balken, sodass ein roter Punkt angezeigt wird.



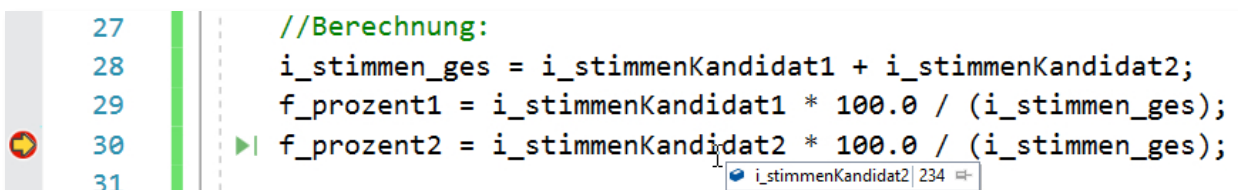
```
27 //Berechnung:
28 i_stimmen_ges = i_stimmenKandidat1 + i_stimmenKandidat2;
29 f_prozent1 = i_stimmenKandidat1 * 100.0 / (i_stimmen_ges);
30 f_prozent2 = i_stimmenKandidat2 * 100.0 / (i_stimmen_ges);
```

Starten des Debuggers

Um das Debugging zu starten, starten Sie Ihren Code mit der grünen Play-Taste in der Toolbar oder drücken Sie F5. Ihr Programm wird (normal) ausgeführt, bis es den ersten gefundenen Haltepunkt erreicht. Der Haltepunkt, welcher gerade aktiv ist, wird dann mit einem gelben Pfeil gekennzeichnet.

Untersuchen von Variablen in Ihrem Code

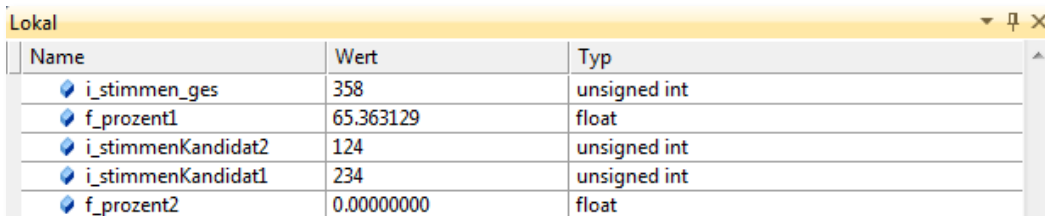
Wenn das Programm angehalten wurde, können Sie die Funktionen des Debuggers verwenden, um die Werte der Variablen in Ihrem Code zu überprüfen. Auf diese Weise können Sie sicherstellen, dass die Werte gespeichert werden, die Sie erwarten. Sie können die Maus verwenden, um den Mauszeiger über eine um den aktuellen Wert einer Variable zu sehen.



```
27 //Berechnung:
28 i_stimmen_ges = i_stimmenKandidat1 + i_stimmenKandidat2;
29 f_prozent1 = i_stimmenKandidat1 * 100.0 / (i_stimmen_ges);
30 f_prozent2 = i_stimmenKandidat2 * 100.0 / (i_stimmen_ges);
31
```

i_stimmenKandidat2 234

Sie können auch die Werkzeugfenster Auto und Lokal zum Überprüfen von Variablen verwenden. Das Auto-Tool-Fenster zeigt den Typ und den aktuellen Wert der Variablen an, die in der aktuellen Zeile oder der vorherigen Zeile verwendet werden. Das Fenster Lokal zeigt Variablen an, die sich derzeit im Bereich befinden. Wenn Sie debuggen, werden diese Fenster standardmäßig in der linken unteren Ecke von Visual Studio angezeigt. Sie können sie aber auch erneut über das Menü **Debuggen** → **Fenster** öffnen.

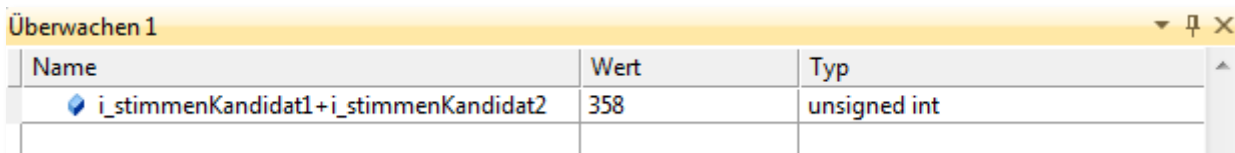


Name	Wert	Typ
i_stimmen_ges	358	unsigned int
f_prozent1	65.363129	float
i_stimmenKandidat2	124	unsigned int
i_stimmenKandidat1	234	unsigned int
f_prozent2	0.00000000	float

Überwachte Ausdrücke

Auch beliebige selbst erstellte Ausdrücke lassen sich überwachen. Diese können während des Debuggings folgendermaßen hinzugefügt werden:

Debuggen → **Fenster** → **Überwachen** → ...



Name	Wert	Typ
i_stimmenKandidat1+i_stimmenKandidat2	358	unsigned int

Schritt für Schritt Analyse

Wenn der Code im Debugger angehalten wird, können Sie Schritt für Schritt durch den Code gehen, um zu sehen, was passiert, wenn die einzelnen Anweisungen ausgeführt werden. Wenn Sie Ihren Code schrittweise bearbeiten, sehen Sie einen gelben Pfeil, der die Anweisung anzeigt, auf der der Debugger angehalten hat. Diese Anweisung wird als nächstes ausgeführt.

Sie können die folgenden Befehle verwenden, um weiter durch Ihren Code zu gehen:



Um das Programm anschließend bis zum nächsten Haltepunkt weiterlaufen zu lassen, klicken Sie auf **Fortsetzen**.



Um die nächste Anweisung auszuführen, klicken Sie auf **Einzelschritt**.

Aufgabe

Testen Sie nun **all diese Funktionen**, **vor** und **während** der Programmausführung, **ausführlich** an Ihrem eigenen Programm des „Wahl-O-Mat“.

Setzen Sie einen Breakpoint bei Ihrer ersten Befehlszeile und gehen per **Einzelschritt** durch Ihr Programm. Beobachten Sie dabei den Zustand Ihrer Variablen im Überwachungsfenster **Lokal**.

LS 1.3 Situationsbeschreibung „Wahl-O-Mat“ – formatierte Ausgabe

Herr Müller teilt Ihnen erfreulicher Weise mit, dass ein Kunde Ihrer Firma Gefallen an ihrem „Wahl-O-Mat“ gefunden hat. Er möchte jedoch eine schönere und formatierte Ausgabe. Herr Müller zeigt Ihnen dazu einen Bildschirmausdruck.



Hr. Lars Müller

Geschäftsführung

```

-----Wahl-O-Mat-----

Bitte geben Sie die Stimmen für Kandidat 1 an: 54
Bitte geben Sie die Stimmen für Kandidat 2 an: 76

=====
Das amtliche Endergebnis der Stichwahl lautet
für Kandidat 1 (in Prozent): 41.53
für Kandidat 2 (in Prozent): 58.46
=====

```

Wie können Sie die Zeichen für den Rahmen ausgeben?

.....

Die (erweiterte) ASCII-Tabelle:

ASCII-Tabelle und Unicode-Tabelle:

ASCII control characters			ASCII printable characters					
00	NULL	(Null character)	32	space	64	@	96	`
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	\$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(72	H	104	h
09	HT	(Horizontal Tab)	41)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[123	{
28	FS	(File separator)	60	<	92	\	124	
29	GS	(Group separator)	61	=	93]	125	}
30	RS	(Record separator)	62	>	94	^	126	~
31	US	(Unit separator)	63	?	95	_		
127	DEL	(Delete)						

Die ersten 128 Zeichen der Unicode-Tabelle entsprechen der ASCII-Tabelle.

Alle Unicode Zeichen:

<https://www.ssec.wisc.edu/~tomw/java/unicode.html>

Unicode Chart

Range	Decimal	Name
0x0000-0x007F	0-127	Basic Latin
0x0080-0x00FF	128-255	Latin-1 Supplement
0x0100-0x017F	256-383	Latin Extended-A
0x0180-0x024F	384-591	Latin Extended-B
0x0250-0x02AF	592-687	IPA Extensions
0x02B0-0x02FF	688-767	Spacing Modifier Letters
0x0300-0x036F	768-879	Combining Diacritical Marks
0x0370-0x03FF	880-1023	Greek
0x0400-0x04FF	1024-1279	Cyrillic
0x0530-0x058F	1328-1423	Armenian
0x0590-0x05FF	1424-1535	Hebrew
0x0600-0x06FF	1536-1791	Arabic
0x0700-0x074F	1792-1871	Syriac
0x0780-0x07BF	1920-1983	Thaana
0x0900-0x097F	2304-2431	Devanagari
0x0980-0x09FF	2432-2559	Bengali
0x1F00-0x1FFF	8192-16383	Box Drawing

9552 =
9553 |
9554 |
9555 |
9556 |
9557 |
9558 |
9559 |
9560 |
9561 |
9562 |
9563 |
9564 |
9565 |

Beispiele für Konstanten (Literale)

Boolesche Konstante

true = 1 false = 0

true ist also eine Konstante für den Wert 1 und false ist eine Konstante für den Wert 0.

Zahlenkonstanten

- Beispiele für ganzzahlige Konstante

Dezimal	Hexadezimal	Datentyp
16 ($\triangleq 16_{10}$)	0x10 ($\triangleq 10_{16}$)	int

16 ist also eine Konstante für den Wert 16 und 0x10 ist ebenfalls eine Konstante für den Wert 16. Beide werden als Integer interpretiert.

255	0xFF	int
32767	0x7FFF	int
32768U	0x8000U	uint
10L	0xAL	long
27UL	0x1BUL	ulong

- Gleitkommakonstante

16.32	ist eine Konstante für den Wert	16,32
12E-2	ist eine Konstante für den Wert	12 * 10 ⁻² also für 0,12
.45E4	ist eine Konstante für	0.45 * 10 ⁴ also für 4500

Zeichenkonstante

Hinter dem Zeichen 'A' steht laut ASCII-Tabelle der Zahlenwert 65

z. B.: 'a'	Kleinbuchstabe a	→ siehe ASCII-Tabelle
' '	Leerzeichen	→ siehe ASCII-Tabelle
' , '	Hochkomma	→ siehe ASCII-Tabelle

Escape-Sequenzen

Einzelzeichen	Bedeutung	ASCII-Zeichen	ASCII-Code
\a	alert	BEL	07
\b	backspace	BS	08
\t	horizontal tab	HT	09
\n	line feed	LF	0A
\v	vertical tab	VT	0B
\f	form feed	FF	0C
\r	carriage return	CR	0D
\"	"	"	22
\'	'	'	27
\?	?	?	3F
\\	\	\	5C
\0	Stringende-Zeichen	NUL	00

MERKE: Konstanten repräsentieren einen Zahlenwert. (d.h. hinter jeder Konstanten steht ein Zahlenwert)

Was sind ASCII- und Unicode-Tabellen und wofür werden Sie verwendet?

.....

.....

.....

LS 1.4 Weitere Übungsaufgaben nach dem EVA-Prinzip

Schreiben Sie jeweils ein Programm in C# für folgende Aufgaben:

1. Quadrat-Kubikmeter-Rechner mit folgenden Funktionen:

- Berechnung des Flächeninhaltes und/ oder Umfangs eines Rechtecks, Dreiecks bzw. Kreises
- Berechnung des Volumens/ Oberflächeninhaltes eines Quaders, Zylinders bzw. einer Kugel

Lösen Sie die Aufgaben nach Ihrem individuellen Leistungsstand ;)

2. Vorgänger-Nachfolger-Ausgabe (Beispiel für Wertebereichsüberschreitung):

- Anlegen aller notwendigen Variablen mit Hilfe des Datentyps `unsigned short` (=ushort)
- Berechnung und Ausgabe von Vorgänger und Nachfolger mit den Testwerten 0 und 65535

3. Einfacher Taschenrechner:

Verrechnen Sie zwei Zahlen zu den Ergebnissen von Summe, Differenz, Produkt und Quotienten

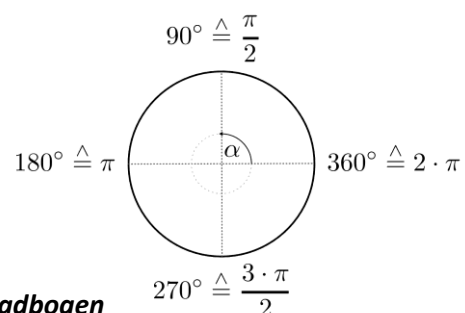
-
4. Schreiben Sie ein Programm, das einen Kleinbuchstaben einliest und anschließend den entsprechenden Großbuchstaben ausgibt.
5. Das Normalgewicht n für Männer wird oft aus der Körpergröße k ermittelt: $n = k - 100$
Das Idealgewicht beträgt dann 95% des Normalgewichts. Lesen Sie die Körpergröße in cm ein und geben Sie das Normalgewicht und das Idealgewicht aus.
6. Erstellen Sie ein Programm, das für einen verzinsten Geldbetrag den Endbetrag errechnet. Diesem Programm muss das Anfangskapital, der Zinssatz in Prozent und die Laufzeit in Jahren mitgeteilt werden.

$$\text{Endkapital} = \text{Anfangskapital} * (1 + \text{Zinssatz} / 100)^{\text{Laufzeit}}$$

7. Schreiben Sie ein Programm, das die Anzahl der Felder eines Spielbretts (Schachbrett) einliest. Anschließend soll das Gewicht eines Reiskorns (25 mg) in kg und das Gesamtgewicht aller Reiskörner auf dem letzten Feld des Spielfeldes ausgegeben werden, wenn man bei jedem Feld die Anzahl der Reiskörner verdoppelt.
8. Ein Winkel soll in Grad eingelesen und im Bogenmaß ausgegeben werden.

Grad	Bogenmaß
90°	1.57
45°	0.79
-277°	-4.83

Bsp.:



Umrechnung von Grad in Bogenmaß: **b3-f.de/gradbogen**

9. Ändern Sie die Umrechnung in Aufgabe 8 so ab, dass Bogenmaß in Grad umgerechnet wird.