Gliederung der Grundvorlesung

1. Grundlagen der Programmierung

~~1.1 Algorithmen und Sprachen~~

~~1.2 Aussagen über Algorithmen~~

1.3 Daten und ihre Strukturierung

1.4 Grundbegriffe der Programmierung

1.5 Die Sprache Ada 95

1.6 Semantik von Programmen

1.7 Komplexität von Algorithmen und Programmen

Gliederung des Kapitels 1.3

1.3 Daten und ihre Strukturierung

1.3.1 Elementare Datentypen

1.3.2 Konstruktoren (für Datenbereiche)

1.3.3 Relationen, Graphen, Referenzen

1.3.4 Spezielle Graphen: Listen, Bäume

1.3.5 Dynamische Datenstrukturen

1.3.6 Keller und Halde

1.3.1 Elementare Datentypen (vgl. Abschnitt 1.1.2)

Die meisten Probleme kann man mit Hilfe von Mengen und auf ihnen definierten Operationen und Relationen beschreiben.

"Elementare" Mengen oder Daten sind: *Wahrheitswerte, Zeichen, natürliche Zahlen, ganze Zahlen, reelle Zahlen.*

Es gibt sie in fast allen Programmiersprachen.

Hinzu kommen selbstdefinierte Mengen (Aufzählungstyp).

In manchen Programmiersprachen gelten auch die Zeichenketten (= Menge der Wörter über dem Alphabet der Zeichen) als elementare Daten mit speziellen Operationen.

Kürzel	Menge	Name	Datentyp
IB	{false, true} oder {0, 1}	Boolesche Werte Wahrheitswerte	Boolean
Â	ASCII-Code EBCDIC-Code	Tastatur-Alphabet Alphabetszeichen	character
IN	{1, 2, 3, 4, ...}	natürliche Zahlen	positive
IN_0	{0, 1, 2, 3, ...}	natürliche Zahlen mit der 0	natural, Cardinal
Z	{ ..., -2, -1, 0, 1, 2, 3, ... }	ganze Zahlen	integer
IR		reelle Zahlen	real, fixed, float

Definition 1.3.1.1: *Datentyp*
Eine Menge (oder mehrere Mengen) zusammen mit den hierauf definierten Operationen nennt man einen (konkreten) Datentyp.
Einen Datentyp, den man nicht auf andere Datentypen zurückführt, nennt man einen elementaren Datentyp.

"Unsere" elementaren Datentypen sind die folgenden 4: Boolean, character, integer und real.

Um sie genau festzulegen, müssen wir zu jeder Menge die zulässigen Operationen hinzufügen. (Mathematisch ist ein Datentyp daher eine Algebra.)

Jede Programmiersprache mag hiervon abweichen. Wir führen die elementaren Datentypen von Ada auf.

1.3.1.1.a: Bei der genauen Beschreibung der Datentypen verwenden wir folgendes Schema; dieses Schema bildet die Signatur des Datentyps.

Der Datentyp xyz:
Zugrunde liegende Menge: ...
Nullstellige Operationen (=besondere Konstanten): ...
Einstellige Operationen: ...
Zweistellige Operationen: ...
höherstellige Operationen (sofern vorhanden): ...
Definition gewisser Operationen: ...
Beziehungen, Gesetzmäßigkeiten: ...
Dieser Datentyp in der Sprache Ada 95: ....

1.3.1.1.b: Darstellung der Operationen:

Hat man zwei Operationen f, g: $M \times M \rightarrow M$ auf einer Menge M, so kann man zusammengesetzte Ausdrücke ("Terme") bilden:

f(g(x, y), x) oder g(g(f(x,y),z), g(z, x))

Diese Darstellung, dass der Operator vor seinen Operanden steht, bezeichnet man als Prefixnotation. Man kann dann die Klammern auch weglassen:

f g x y x oder g g f x y z g z x

und die Auswertung der Ausdrücke bleibt eindeutig.

Statt dessen kann man die Operatoren auch hinter ihre Operanden schreiben:

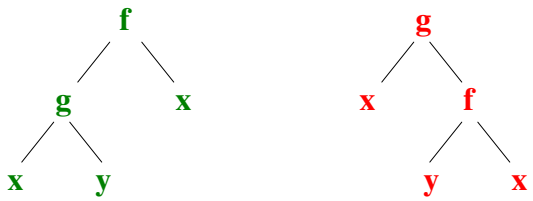
x y g x f oder x y f z g z x g g

Dies nennt man Postfix-Notation (oder polnische Notation). Auch hierbei bleibt die Auswertung der Ausdrücke eindeutig.

Im täglichen Leben verwendet man jedoch in der Regel die Infix-Notation, bei der man das Operationszeichen zwischen die Operanden stellt; obige Beispiele werden dann zu

x g y f x oder x f y g z g z g x

Dies ist nicht mehr eindeutig, da nun z. B. f(g(x, y), x) und g (x, f(y, x)) die gleiche Infix-Notation x g y f x besitzen. Man erkennt dies, wenn man die Baumstruktur betrachtet:



Beide Bäume gehören zu x g y f x

Um bei der Infixnotation Eindeutigkeit zu sichern, verwendet man zum einen Klammern und zum anderen Prioritätsregeln.

Daher müssen bei den Operationen fast immer auch solche Prioritätsregeln für die Operatoren angegeben werden.

Ist nichts angegeben oder haben Operatoren die gleiche Priorität, so wird ein Ausdruck stets von links nach rechts ausgewertet!

Hinweis: Wenden Sie diese Überlegungen auf die Eindeutigkeit von Wörtern oder Grammatiken an (siehe 1.1.5.11):

Stellt man die Ableitungsbäume mit der Prefixnotation dar, so sind zwei Ableitungen genau dann gleich (im Sinne von 1.1.5.10), wenn sie die gleiche Prefixnotation besitzen!

Die Prefixnotation erhält man, indem man den (Ableitungs-) Baum von der Wurzel immer zunächst nach links zu den Blättern hin durchläuft und immer, wenn man einen Knoten erstmals besucht, dessen Inhalt herausschreibt.

Machen Sie sich dies an Beispielen klar. Dadurch kann man die Gleichheit von Ableitungen auch "algorithmisch" relativ leicht nachprüfen.

1.3.1.2: Aufzählungstypen ("enumeration types", vgl. 1.1.2.18)

Zugrunde liegende Menge: Eine endliche Menge M, die man selbst definieren muss. Die Menge M wird eingeführt durch

type <Name des Datentyps> = (<Liste der Elemente>)

Die Reihenfolge in der Liste gibt zugleich die *Anordnung* der Elemente in der Menge $M = \{m_1, m_2, ..., m_n\}$ an.

Es kommt hierbei per Definition nicht zu Namenskonflikten! Das heißt: Wenn ein Element in mehreren Mengen liegt, so muss der Programmierer selbst darauf achten, dass bei jeder Verwendung eines Elementes eindeutig klar ist, zu welchem Datentyp es gehört.

Nullstellige Operationen (= besondere Konstanten) sind alle Elemente m_i der Menge M. Weiterhin seien First das erste und Last das letzte Element der Menge. (Man muss hierbei den Datentyp angeben, d.h., wenn T der Name des Datentyps ist, so schreibt man T'First bzw. T'Last.)

Einstellige Operationen: (Vorgänger, Nachfolger, Position, Wert)

"predecessor" und "successor" pred, succ: $M \rightarrow M$ mit

pred(X) = das Zeichen vor X in der Auflistungs-Reihenfolge,
succ(X) = das Zeichen nach X in der Auflistungs-Reihenfolge.
$pred(m_i) = m_{i-1}$ für $i>1$ und $pred(m_1)$ = undefiniert,
$succ(m_i) = m_{i+1}$ für $i<n$ und $succ(m_n)$ = undefiniert.

Position in der Anordnung der Zeichen: pos: $M \rightarrow \mathbb{N}_0$

pos(X) = n bedeutet: X ist das n-te Zeichen in der Auflistungsreihenfolge (beginnend mit 0).

n-tes Element in der Anordnung: val: $\mathbb{N}_0 \rightarrow M$

val(n) = X bedeutet: X ist das n-te Zeichen in der Auflistungsreihenfolge (beginnend mit 0).

Zweistellige Operationen (Vergleichsoperatoren):
Alle sechs Vergleichoperationen "=", "≠", "<", "≤", ">" und "≥" bezüglich der Anordnung der Menge sind zugelassen. Das Ergebnis ist vom Typ Boolean (siehe unten).

Hinweis: Tritt ein Element in mehreren Datentypdefinitionen auf

type stuhlteil = (bein, sitzfläche, lehne);
type körperteil = (arm, bein, rücken, sitzfläche, kopf)

so kann man die Elemente eindeutig charakterisieren, indem man den Datentyp abgetrennt durch ' davor schreibt und das Element in Klammern setzt, also:

körperteil'(bein) oder stuhlteil'(bein) oder
körperteil'pos(bein) *(dies ist 1),* stuhlteil'pos(bein) *(dies ist 0).*

Zweistellige Operationen (Fortsetzung):

Für alle Aufzählungstypen T sind die zweistelligen Abbildungen Min und Max definiert: Min, Max: $M \times M \rightarrow M$ mit

$Min(X,Y) = X \Leftrightarrow T'pos(X) \leq T'pos(Y)$ $\quad (\Leftrightarrow T'(X) \leq T'(Y))$

$Max(X,Y) = X \Leftrightarrow T'pos(X) > T'pos(Y)$

Im Falle $X \neq Y$ gilt stets $T'Min(X,Y) \neq T'Max(X,Y)$.

Dreistellige Operationen:

if then else fi: $\mathbb{B} \times M \times M \rightarrow M$

Diese Operation ist wie folgt definiert:

$$\text{if b then m1 else m2 fi} = \begin{cases} \text{m1, falls b den Wert true besitzt} \\ \text{m2, falls b den Wert false besitzt} \end{cases}$$

Prioritäten werden auf diesen Operationen nicht festgelegt.

Beziehungen, Gesetzmäßigkeiten:

Es gilt stets:

$pos(val(k)) = k$ für $0 \leq k \leq n-1$,

$val(pos(X)) = X$ für alle Elemente $X \in M$,

$succ(pred(m_i)) = m_i$ für $1 < i \leq n$,

$pred(succ(m_j)) = m_j$ für $1 \leq j < n$,

$X < Y$ im Datentyp T $\Leftrightarrow$ $pos(X) < pos(Y)$ in $\mathbb{N}_0$.

Der Aufzählungstyp in der Sprache Ada 95:

In Ada werden wie Aufzählungs-Datentypen und ihre ein- und zweistelligen Operationen wie oben angegeben verwendet, allerdings muss man in Ada is statt "=" in der Typdefinition schreiben und ≠ als /=, ≤ als <= und ≥ als >=.

1.3.1.3: Der Datentyp Boolean:

Zugrunde liegende Menge: $\mathbb{B} = \{\text{false}, \text{true}\}$ (= {falsch, wahr})

Nullstellige Operationen (=besondere Konstanten): false und true.

Einstellige Operationen (NICHT):

Negation $\neg$: $\mathbb{B} \to \mathbb{B}$ (statt $\neg$ schreibt man meist not oder man macht einen Querstrich über den jeweiligen Ausdruck).

Zweistellige Operationen (UND, ODER, GLEICH, EXKLUSIVES ODER, FOLGT):

Konjunktion $\wedge$: $\mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (statt $\wedge$ schreibt man meist and)

Disjunktion $\vee$: $\mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (statt $\vee$ schreibt man meist or)

Gleichheit (oder auch Äquivalenz auf $\mathbb{B}$ genannt)

$=$: $\mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (statt = schreibt man auch equiv)

Ungleichheit (oder auch "Exklusives Oder" genannt)

$\neq$: $\mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (statt $\neq$ schreibt man meist exor oder xor)

Implikation $\to$: $\mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (statt $\to$ schreibt man meist impl).

Definition der Operationen des Datentyps Boolean:

	not
false	true
true	false

and	false	true
false	false	false
true	false	true

or	false	true
false	false	true
true	true	true

equiv	false	true
false	true	false
true	false	true

exor	false	true
false	false	true
true	true	false

impl	false	true
false	true	true
true	false	true

Prioritäten der Operationen:

höchste Priorität: not

Alle übrigen Operationen besitzen eine geringere Priorität.

not a and b bedeutet also (not a) and b.

In der Logik verwendet man für die Operationen meist die Prioritäten:

not vor and vor or vor exor und equiv vor impl,

jedoch wollen wir diese Reihung hier *nicht* benutzen, da wir dann mit der Festlegung in Ada in Konflikt geraten. Statt dessen sollten Sie Boolesche Ausdrücke möglichst vollständig klammern.

Man kann nun Beziehungen und Gesetzmäßigkeiten zwischen diesen Operationen auflisten.

Einige dieser Beziehungen sind (für alle $a, b \in \mathbb{B}$):

not (not a) = a

exor (a, b) = not (equiv (a, b))

impl (a, b) = (not a) or b

not (a and b) = (not a) or (not b)
not (a or b) = (not a) and (not b)
} deMorganschen Regeln

Man benötigt nur die Operationen not und and, um alle anderen Operationen darzustellen (selbst nachweisen).

Der Datentyp Boolean in der Sprache Ada 95:

Der Datentyp Boolean wird in Ada 95 als Aufzählungsdatentyp mit zusätzlichen Operationen aufgefasst, also:

type Boolean is (false, true);

Damit sind zugleich alle Operationen, die unter 1.3.1.2 beschrieben wurden, auch auf Boolean zugelassen. Beispielsweise gilt in Ada false < true, Min(true, false) = false, Boolean'val(1) = true und Boolean'First = false.

Die oben genannten Konstanten und Operationen werden geschrieben als: FALSE, TRUE, NOT, AND, OR, = und XOR. Die Implikation ist nicht vorgesehen, ist aber leicht zu realisieren wegen impl (a, b) = (not a) or b.

Grundsätzlich wird in Ada jeder Ausdruck *vollständig* ausgewertet. Der Boolesche Ausdruck

(A > 3) AND (B/0 = 7)

hat daher den Wert "undefiniert" wegen der Division durch 0.

Oft möchte man aber den Ausdruck a AND b sofort mit dem Ergebnis false verlassen, wenn sich a schon als false erwiesen hat; man spart die Ausrechnung von b.

Für diese "logische Abkürzung" oder "Kaskadenoperation" gibt es in Ada 95 den Operator "AND THEN". Analoges gilt für or und die Operation "OR ELSE".

In Ada stehen daher neben and und or auch die beiden „Kaskadenoperationen“ AND THEN und OR ELSE zur Verfügung, die man aber "sehr bewusst" einsetzen sollte, da man hierdurch Fehler verdecken kann (vgl. später die "exceptions", siehe auch Folie 50 in Kapitel 1.1):

a AND THEN b entspricht:
falls a nicht zutrifft, ist das Ergebnis false, anderenfalls b.

a OR ELSE b entspricht:
falls a zutrifft, ist das Ergebnis true, anderenfalls b.

Im Normalfall, dass a und b einen der Werte false oder true besitzen, stimmen AND und AND THEN bzw. OR und OR ELSE überein. Den Unterschied zu AND und OR erkennt man nur, wenn man die Operationen zusätzlich mit dem Wert "undefiniert" aufschreibt:

AND	false	true	undef
false	false	false	undef
true	false	true	undef
undef	undef	undef	undef

AND THEN	false	true	undef
false	false	false	**false**
true	false	true	undef
undef	undef	undef	undef

OR	false	true	undef
false	false	true	undef
true	true	true	undef
undef	undef	undef	undef

OR ELSE	false	true	undef
false	false	true	false
true	true	true	**true**
undef	undef	undef	undef

Prioritäten in Ada für Boolean:

In Ada werden nur zwei Prioritätsstufen für Boolesche Operationen festgelegt:

Der Operator NOT erhält eine höhere Priorität, die anderen Operatoren AND, OR und XOR erhalten eine niedrigere Priorität. Aber:

In Ada müssen alle Booleschen Ausdrücke, in denen mindestens zwei der Operatoren AND, OR und XOR vorkommen, geklammert werden!

a AND b OR c ist also verboten; man muss (a AND b) OR c oder a AND (b OR c) schreiben.

Erlaubt ist dagegen a OR b OR c ; dies wird von links nach rechts ausgewertet.

In Booleschen Ausdrücken können auch Vergleiche auftreten.

In Ada erhalten *alle* Vergleichsoperatoren eine höhere Priorität als alle Booleschen Operationen. Statt

(17 < 10) AND (NOT (X=Y))

kann man also auch schreiben:

17 < 10 AND NOT X=Y.

Beachte: Allein schon bzgl. der Prioritätsregeln unterscheidet sich Ada von den meisten anderen Programmiersprachen.

1.3.1.3: Der Datentyp character:

Zugrunde liegende Menge: Wir legen uns hier nicht auf ein Alphabet fest, verlangen jedoch mindestens die Menge der Zeichen, die im ASCII-Code beschrieben sind. Diese Menge ist ein Aufzählungsdatentyp, als dessen Anordnung die Reihenfolge im ASCII-Code verwendet wird, siehe 1.1.2.6.
(Beachte: In Ada wird Latin-1 statt ASCII verwendet, s.u.)

Nullstellige Operationen (die Zeichen sowie First, Last)
Einstellige Operationen (pred, succ, pos, val)
Zweistellige Operationen (Vergleichsoperationen, Min, Max)
Dreistellige Operationen
Beziehungen und Gesetzmäßigkeiten:

Alles wie bei Aufzählungstypen.

Der Datentyp character in der Sprache Ada 95:

Die zugrunde liegende Menge des Datentyps character ist in Ada 95 der Aufzählungsdatentyp der Zeichen (auch "Literale" genannt) des Alphabets "Latin-1", das von 0 bis 127 mit ASCII überein stimmt:
type character is (*NUL*, ..., *US*, ' ', '!', '"', '#', '$', ..., 'ÿ');

Die einzelnen Zeichen werden in Apostroph eingeschlossen, sofern sie graphisch darstellbar sind (das Apostroph selbst wird hierbei durch zwei Apostrophs dargestellt).

Zur Kenntnisnahme geben wir den vollständigen Latin-1 Code, definiert in der ISO-Norm 8859-1, auf den nächsten Folien an. In der Tabelle wird die hexadezimale und die (mit dem Nummerzeichen '#' versehene) dezimale Nummerierung vorangestellt und dann das zugehörige Zeichen genannt.

ISO 8859, Latin-1 Alphabet (festgelegt sind nur die schwarzen Werte; die kursiven blauen bilden eine übliche Ergänzung)

00 #0 NULL	*01 #1 START OF HEADING*	*02 #2 START OF TEXT*	*03 #3 END OF TEXT*	*04 #4 END OF TRANSM.*	*05 #5 ENQUIRY*	*06 #6 ACKN.*	*07 #7 BELL*	*08 #8 BACK-SPACE*	09 #9 HORIZ. TAB.	0A #10 LINE FEED	*0B #11 VERTICAL TAB.*	*0C #12 FORM FEED*	0D #13 CARRIAG RETURN	*0E #14 SHIFT OUT*	*0F #15 SHIFT IN*
10 #16 DATA LINK ESC.	*11 #17 DEVICE CONTR.1*	*12 #18 DEVICE CONTR.2*	*13 #19 DEVICE CONTR.3*	*14 #20 DEVICE CONTR.4*	*15 #21 NEGATIV ACKN.*	*16 #22 SYNCHR. IDLE*	*17 #23 END OF T.BLOCK*	*18 #24 CANCEL*	*19 #25 END OF MEDIUM*	*1A #26 SUBSTI-TUTE*	*1B #27 ESCAPE*	*1C #28 FILE SE-PARATOR*	*1D #29 GROUP SEPAR.*	*1E #30 RECORD SEPAR.*	*1F #31 UNIT SEPAR.*
20 #32 SPACE	21 #33 **!**	22 #34 **“**	23 #35 **#**	24 #36 **$**	25 #37 **%**	26 #38 **&**	27 #39 **‘**	28 #40 **(**	29 #41 **)**	2A # 42 *****	2B #43 **+**	2C #44 **,**	2D #45 **-**	2E #46 **.**	2F #47 **/**
30 #48 **0**	31 #49 **1**	32 #50 **2**	33 #51 **3**	34 #52 **4**	35 #53 **5**	36 #54 **6**	37 #55 **7**	38 #56 **8**	39 #57 **9**	3A #58 **:**	3B #59 **;**	3C #60 **<**	3D #61 **=**	3E #62 **>**	3F #63 **?**
40 #64 **@**	41 #65 **A**	42 #66 **B**	43 #67 **C**	44 #68 **D**	45 #69 **E**	46 #70 **F**	47 #71 **G**	48 #72 **H**	49 #73 **I**	4A #74 **J**	4B #75 **K**	4C #76 **L**	4D #77 **M**	4E #78 **N**	4F #79 **O**
50 #80 **P**	51 #81 **Q**	52 #82 **R**	53 #83 **S**	54 #84 **T**	55 #85 **U**	56 #86 **V**	57 #87 **W**	58 #88 **X**	59 #89 **Y**	5A #90 **Z**	5B #91 **[**	5C #92 ****	5D #93 **]**	5E #94 **^**	5F #95 **_**
60 #96 **`**	61 #97 **a**	62 #98 **b**	63 #99 **c**	64 #100 **d**	65 #101 **e**	66 #102 **f**	67 #103 **g**	68 #104 **h**	69 #105 **i**	6A #106 **j**	6B #107 **k**	6C #108 **l**	6D #109 **m**	6E #110 **n**	6F #111 **o**
70 #112 **p**	71 #113 **q**	72 #114 **r**	73 #115 **s**	74 #116 **t**	75 #117 **u**	76 #118 **v**	77 #119 **w**	78 #120 **x**	79 #121 **y**	7A #122 **z**	7B #123 **{**	7C #124 **|**	7D #125 **}**	7E #126 **~**	*7F #127 DELETE*
80 #128 €	*81 #129*	*82 #130 ‚*	*83 #131 ƒ*	*84 #132 „*	*85 #133 …*	*86 #134 †*	*87 #135 ‡*	*88 #136 ˆ*	*89 #137 ‰*	*8A #138 Š*	*8B #139 ‹*	*8C #140 Œ*	*8D #141*	*8E #142 Ž*	*8F #143*
90 #144	*91 #145 ‘*	*92 #146 ’*	*93 #147 “*	*94 #148 ”*	*95 #149 •*	*96 #150 –*	*97 #151 —*	*98 #152 ˜*	*99 #153 ™*	*9A #154 š*	*9B #155 ›*	*9C #156 œ*	*9D #157*	*9E #158 ž*	*9F #159 Ÿ*
A0 #160 NO-BREAK SPACE	A1 #161 **¡**	A2 #162 **¢** CENT	A3 #163 **£** POUND	A4 #164 **¤**	A5 #165 **¥** YEN	A6 #166 **¦**	A7 #167 **§**	A8 #168 **¨**	A9 #169 **©**	AA #170 **ª**	AB #171 **«**	AC #172 **¬**	AD #173 - SOFT HYPHEN	AE #174 **®** REGIS-TERED	AF #175 **¯** MACRON
B0 #176 **°**	B1 #177 **±**	B2 #178 **²**	B3 #179 **³**	B4 #180 **´** ACCENT	B5 #181 **µ** MICRO	B6 #182 **¶**	B7 #183 **·** MID.DOT	B8 #184 **¸** CEDILLA	B9 #185 **¹**	BA #186 **º**	BB #187 **»**	BC #188 **¼**	BD #189 **½**	BE #190 **¾**	BF #191 **¿**
C0 #192 **À**	C1 #193 **Á**	C2 #194 **Â**	C3 #195 **Ã**	C4 #196 **Ä**	C5 #197 **Å**	C6 #198 **Æ**	C7 #199 **Ç**	C8 #200 **È**	C9 #201 **É**	CA #202 **Ê**	CB #203 **Ë**	CC #204 **Ì**	CD #205 **Í**	CE #206 **Î**	CF #207 **Ï**
D0 #208 **Ð**	D1 #209 **Ñ**	D2 #210 **Ò**	D3 #211 **Ó**	D4 #212 **Ô**	D5 #213 **Õ**	D6 #214 **Ö**	D7 #215 **×**	D8 #216 **Ø**	D9 #217 **Ù**	DA #218 **Ú**	DB #219 **Û**	DC #220 **Ü**	DD #221 **Ý**	DE #222 **Þ**	DF #223 **ß**
E0 #224 **à**	E1 #225 **á**	E2 #226 **â**	E3 #227 **ã**	E4 #228 **ä**	E5 #229 **å**	E6 #230 **æ**	E7 #231 **ç**	E8 #232 **è**	E9 #233 **é**	EA #234 **ê**	EB #235 **ë**	EC #236 **ì**	ED #237 **í**	EE #238 **î**	EF #239 **ï**
F0 #240 **ð**	F1 #241 **ñ**	F2 #242 **ò**	F3 #243 **ó**	F4 #244 **ô**	F5 #245 **õ**	F6 #246 **ö**	F7 #247 **÷**	F8 #248 **ø**	F9 #249 **ù**	FA #250 **ú**	FB #251 **û**	FC #252 **ü**	FD #253 **ý**	FE #254 **þ**	FF #255 **ÿ**

Latin-1 enthält ASCII als die ersten 128 Zeichen. Die Zeichen mit den Nummern 128 bis 159 wurden in Latin-1 frei gelassen; sie werden von verschiedenen Softwareherstellern unterschiedlich verwendet. Oben sind die Zeichen von Microsoft Windows eingetragen (die Nummern 128, 129, 141-144, 157, 158 werden dabei nicht festgelegt; 128 wird meist für das Eurozeichen, 142 und 158 für das modifizierte Z benutzt), in Ada kann man auf diese Zeichen mit symbolischen Bezeichnungen zugreifen, siehe nächste Folie.

Die Werte mit den Nummern 0 bis 31 und 127 bis 159 können zwar verwendet, aber oft nicht angezeigt werden. Sie werden durch zwei- oder dreibuchstabige Kürzel bezeichnet und zwar:

Nr.	Kürzel	Nr.	Kürzel			Nr.	Kürzel
0	NUL	16	DLE	127	DEL	144	DCS
1	SOH	17	DC1	128	128	145	PU1
2	STX	18	DC2	129	129	146	PU2
3	ETX	19	DC3	130	BPH	147	STS
4	EOT	20	DC4	131	NBH	148	CCH
5	ENQ	21	NAK	132	132	149	MW
6	ACK	22	SYN	133	NEL	150	SPA
7	BEL	23	ETB	134	SSA	151	EPA
8	BS	24	CAN	135	ESA	152	SOS
9	HT	25	EM	136	HTS	153	153
10	LF	26	SUB	137	HTJ	154	SCI
11	VT	27	ESC	138	VTS	155	CSI
12	FF	28	FS	139	PLD	156	ST
13	CR	29	GS	140	PLU	157	OSC
14	SO	30	RS	141	RI	158	PM
15	SI	31	US	142	SS2	159	APC
				143	SS3		

First, Last, die Funktionen pred, succ, pos, val, Min, Max und die Vergleichsoperationen sind hiermit definiert. Zum Beispiel gilt Character'First = NUL, val(51) = '3', pos(pred(pred(DEL))) = 125, 'C' < '©', val(145) = PU1 und succ(Character'Last) ist undefiniert.

Hinweis: Es gibt verschiedene Zeichensätze, vgl. auch 1.1.2.7. Daher gint es auch in Ada mehrere vordefinierte Character-Datentypen. Diese werden angesprochen durch

Ada.Characters.<Name des Zeichensatzes>

Beispiel: Der oben beschriebene übliche Zeichensatz ist

Ada.Characters.Latin_1

Man kann über

Ada.Characters.ASCII
Ada.Characters.Wide_Character

andere Zeichensätze nutzen. Details siehe spezielle Literatur.