

## **Rechtlicher Hinweis**

**Diese Präsentation ist urheberrechtlich geschützt und darf nur im Rahmen von Lehrveranstaltungen der Friedrich-Schiller-Universität Jena verwendet werden. Eine Nutzung durch Verbreitung oder Veröffentlichung dieses Materials - auch in Auszügen - ist strengstens untersagt und wird die Geltendmachung von Unterlassungs- und Schadenersatzansprüchen durch die Friedrich-Schiller-Universität Jena zur Folge haben.**

## **Legal notice**

**These slides are protected by copyright and may only be used as part of courses at the Friedrich Schiller University Jena. Any use through the dissemination or publication of this material - even in extracts - is strictly prohibited and will result in the assertion of injunctive relief and claims for compensation by the Friedrich Schiller University Jena.**

# Informatik I (B.Sc. Physik)

## Zeichen und Zeichenketten

**Dr. Paul Bodesheim**

(Paul.Bodesheim@uni-jena.de)



**FRIEDRICH-SCHILLER-  
UNIVERSITÄT  
JENA**

**Fakultät für Mathematik und Informatik  
Lehrstuhl für Digitale Bildverarbeitung**

**SoSe 2020**

- 1 Zeichen in C++ (Typ char)
- 2 Zeichenketten in C++ (Typ string)

# Inhalt

1 Zeichen in C++ (Typ char)

2 Zeichenketten in C++ (Typ string)

## Der Datentyp char

服务于代表单个字符和代表数字

基础字母字符或数字值

另外：变量的最小存储单位

运算符sizeof将变量和类型的内存大小确定为char的内存大小的倍数

通常：char为1个字节

始终严格区分char的不同用法

- Dient sowohl zur Darstellung von Einzelzeichen als auch zur Repräsentation von Zahlen
- Zeichen des zugrundeliegenden Alphabets oder numerische Werte
- Außerdem: kleinste Speichereinheit von Variablen
- Operator **sizeof** ermittelt Speichergröße von Variablen und Typen als Vielfaches der Speichergröße von **char**
- Typischerweise: 1 Byte für **char**
- Verschiedene Verwendungen von **char** stets strikt trennen

## char für numerische Werte

- Verwendung als Zahl: immer inklusive **signed** oder **unsigned**
- **signed char** `i = -1;` // Wertebereich `-128 ... 127`
- **unsigned char** `j = 128;` // Wertebereich `0 ... 255`
- Beispiel: Farbwerte eines Mega-Pixel-Bildes als **unsigned char** statt **int**
  - ⇒ geringer Speicherbedarf da kleiner Wertebereich
  - ⇒ spart Speicherplatz
- Rechnen wie mit Integer-Datentypen
- Über- oder Unterschreiten des Wertebereichs beachten

示例：百万像素图像的颜色值作为无符号字符而不是int  
    ) 由于值范围较小，因此内存需求较低  
    ) 节省存储空间  
像整数数据类型一样进行计算  
注意超出或低于值范围

## char für Zeichen

- Verwendung als Zeichen: ohne **signed** oder **unsigned**  
(Zeichen haben kein Vorzeichen)
- Zeichenliterale in Hochkommas (keine Anführungszeichen!)
- **char** c = 'A';
- Sonderzeichen (gekennzeichnet durch Backslash, auch Escape-Sequenz genannt:  
Ausbruch aus normaler Schreibweise):

Zeichen	Bedeutung	
\n	Zeilenvorschub	换行
\t	Tabulator	\ t 标签
\'	Hochkomma	\ ' 撇号
\"	Anführungszeichen	\ " 引号
\\	Backslash	\\ 反斜杠

- Es gibt noch weitere solcher Sequenzen

## Zahl oder Zeichen?

char 类型的变量始终代表一个字符和一个数字  
 关系来自字符的基础编码  
 ( 计算器中的所有字符均以数字 ( 二进制 ) 编码。 )  
 标准中未指定编码, 但取决于编译器和操作系统  
 很多时候: 前128个字符的ASCII码  
 您自己的编程应独立于编码!  
 这通常很容易!

- Variable vom Typ **char** repräsentiert immer Zeichen und Zahl gleichzeitig
- Zusammenhang ergibt sich aus zugrundeliegender Kodierung der Zeichen  
 (Alle Zeichen im Rechner sind numerisch (binär) kodiert.)
- Kodierung im Standard nicht festgelegt, sondern hängt vom Compiler und Betriebssystem ab
- Sehr oft: ASCII-Code für die ersten 128 Zeichen
- Die eigene Programmierung sollte unabhängig von der Kodierung sein!
- Dies ist in der Regel sehr leicht möglich!



## ASCII-Tabelle

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	00000000	NUL	32	0x20	040	01000000	space	64	0x40	100	10000000	@	96	0x60	140	11000000	`
1	0x01	001	00000001	SOH	33	0x21	041	01000001	!	65	0x41	101	10000001	A	97	0x61	141	11000001	a
2	0x02	002	00000010	STX	34	0x22	042	01000010	"	66	0x42	102	10000010	B	98	0x62	142	11000010	b
3	0x03	003	00000011	ETX	35	0x23	043	01000011	#	67	0x43	103	10000011	C	99	0x63	143	11000011	c
4	0x04	004	00000100	EOF	36	0x24	044	01000100	\$	68	0x44	104	10000100	D	100	0x64	144	11000100	d
5	0x05	005	00000101	ENQ	37	0x25	045	01000101	%	69	0x45	105	10000101	E	101	0x65	145	11000101	e
6	0x06	006	00000110	ACK	38	0x26	046	01000110	&	70	0x46	106	10000110	F	102	0x66	146	11000110	f
7	0x07	007	00000111	BEL	39	0x27	047	01000111	'	71	0x47	107	10000111	G	103	0x67	147	11000111	g
8	0x08	010	00010000	BS	40	0x28	050	01010000	(	72	0x48	110	10010000	H	104	0x68	150	11010000	h
9	0x09	011	00010001	TAB	41	0x29	051	01010001	)	73	0x49	111	10010001	I	105	0x69	151	11010001	i
10	0x0A	012	00010010	LF	42	0x2A	052	01010010	*	74	0x4A	112	10010010	J	106	0x6A	152	11010010	j
11	0x0B	013	00010011	VT	43	0x2B	053	01010011	+	75	0x4B	113	10010011	K	107	0x6B	153	11010011	k
12	0x0C	014	00011000	FF	44	0x2C	054	01011000	,	76	0x4C	114	10011000	L	108	0x6C	154	11011000	l
13	0x0D	015	00011001	CR	45	0x2D	055	01011001	-	77	0x4D	115	10011001	M	109	0x6D	155	11011001	m
14	0x0E	016	00011010	SO	46	0x2E	056	01011010	.	78	0x4E	116	10011010	N	110	0x6E	156	11011010	n
15	0x0F	017	00011011	SI	47	0x2F	057	01011011	/	79	0x4F	117	10011011	O	111	0x6F	157	11011011	o
16	0x10	020	00100000	DLE	48	0x30	060	01100000	0	80	0x50	120	10100000	P	112	0x70	160	11100000	p
17	0x11	021	00100001	DC1	49	0x31	061	01100001	1	81	0x51	121	10100001	Q	113	0x71	161	11100001	q
18	0x12	022	00100010	DC2	50	0x32	062	01100010	2	82	0x52	122	10100010	R	114	0x72	162	11100010	r
19	0x13	023	00100011	DC3	51	0x33	063	01100011	3	83	0x53	123	10100011	S	115	0x73	163	11100011	s
20	0x14	024	00101000	DC4	52	0x34	064	01101000	4	84	0x54	124	10101000	T	116	0x74	164	11101000	t
21	0x15	025	00101001	NAK	53	0x35	065	01101001	5	85	0x55	125	10101001	U	117	0x75	165	11101001	u
22	0x16	026	00101010	SYN	54	0x36	066	01101010	6	86	0x56	126	10101010	V	118	0x76	166	11101010	v
23	0x17	027	00101011	ETB	55	0x37	067	01101011	7	87	0x57	127	10101011	W	119	0x77	167	11101011	w
24	0x18	030	00110000	CAN	56	0x38	070	01110000	8	88	0x58	130	10110000	X	120	0x78	170	11110000	x
25	0x19	031	00110001	EM	57	0x39	071	01110001	9	89	0x59	131	10110001	Y	121	0x79	171	11110001	y
26	0x1A	032	00110010	SUB	58	0x3A	072	01110010	:	90	0x5A	132	10110010	Z	122	0x7A	172	11110010	z
27	0x1B	033	00110011	ESC	59	0x3B	073	01110011	;	91	0x5B	133	10110011	[	123	0x7B	173	11110011	{
28	0x1C	034	00111000	FS	60	0x3C	074	01111000	<	92	0x5C	134	10111000	\	124	0x7C	174	11111000	
29	0x1D	035	00111001	GS	61	0x3D	075	01111001	=	93	0x5D	135	10111001	]	125	0x7D	175	11111001	}
30	0x1E	036	00111010	RS	62	0x3E	076	01111010	>	94	0x5E	136	10111010	^	126	0x7E	176	11111010	~
31	0x1F	037	00111011	US	63	0x3F	077	01111011	?	95	0x5F	137	10111011	_	127	0x7F	177	11111011	DEL

## Arbeiten mit char

- `char z = 'A' + 5; // 'F'`
- Wert einer Ziffer (solange Reihenfolge der Ziffern in der Kodierung aufsteigend ist, für ASCII erfüllt):  
 一个数字的值 (只要编码中的数字顺序升序, 就满足 ASCII) :  

```
char c = '7';
int i = c - '0'; // 7
```
- Vordefinierte Funktionen: (`char c = '5';`)
  - Liefern Wahrheitswert (**bool**)
  - Ist c ein Buchstabe?: `isalpha(c)`
  - Bedeutung: `isalphabetic`, nicht `isalphanumeric`!
  - Für ASCII äquivalent: `(c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')`
  - Ist c eine Ziffer?: `isdigit(c)`
  - Für ASCII äquivalent: `(c >= '0' && c <= '9')`

⇒ Vergleichsoperatoren (`>`, `>=`, `<`, `<=`, `==`, `!=`) für **char** arbeiten auf den Zahlen der Kodierung

# Inhalt

- 1 Zeichen in C++ (Typ char)
- 2 Zeichenketten in C++ (Typ string)**

## Datentyp string

- Zeichenketten ( string ) bestehen aus einer Anzahl von Zeichen
- string so ähnlich wie vector<char>, aber nicht gleich
- Verwendung von strings erfordert: **#include** <string>
- Initialisierungen:

```
int main()  
{  
    string s1;  
    string s2 = "Text";  
    string s3("Text");  
    ...  
}
```

- Initialisierung von s1 mit dem Leerstring ""
- Leerstring "" ungleich Leerzeichen " "

## Länge von Zeichenketten

- Zwei Funktionen: `length` und `size`
- Beide liefern die Anzahl der Zeichen / Anzahl der Elemente  
(`size` wegen Analogie zu `vector`)

```
int main()
{
    string s1;
    string s2 = "Text";

    cout << s1.length() << endl; // 0
    cout << s1.size() << endl; // 0
    cout << s2.length() << endl; // 4
    cout << s2.size() << endl; // 4
    ...
}
```

## Vordefinierte Operatoren: Zuweisung und Zugriff

- Zuweisung: =
- Elementzugriff mit eckigen Klammern: [ ] (wie bei vector)  
⇒ Liefert Referenz auf Einzelzeichen vom Typ **char**!

```

int main()
{
    string s1;
    string s2 = "Text";
    s1 = s2; // "Text"

    char c1 = s2[2];
    cout << c1 << endl; // x
    char c2 = "Hallo"[1];
    cout << c2 << endl; // a

    s1[2] = 's';
    cout << s1 << endl; // Test

    bool b1 = (s1[1]=='e'); // true
    bool b2 = (s2[3]=='T'); // false

    ...
}

```

作业：=  
 方括号中的元素访问：[]（与矢量一样）  
 ）返回对char类型的单个字符的引用！

## Vordefinierte Operatoren: Verketten

- Verketten durch: +, +=
- Erweiterte Zuweisung (+=) effizienter!  
⇒ Direktes Anhängen statt Erzeugen einer Kopie auf der rechten Seite der Zuweisung

```
int main()
{
    string s1 = "Hallo ";
    string s2 = "Hello ";
    string s3 = "Jena";

    s1 = s1+s3; // "Hallo Jena"
    s2 += s3; // "Hello Jena"

    s1 = s1 + '!'; // "Hallo Jena!"
    s2 += '!'; // "Hello Jena!"
    ...
}
```

## Vordefinierte Operatoren: Vergleiche

- Vergleichsoperatoren: `>`, `>=`, `<`, `<=`, `!=`, `==`
- Test auf Gleichheit (`==`) oder Ungleichheit (`!=`): alle Zeichen stimmen überein (Groß- und Kleinschreibung sowie Reihenfolge und Zeichenanzahl beachten)

```
int main()
{
    string s1;
    string s2 = "Hallo";
    string s3 = "Hallo ";
    string s4 = "HaLLo";

    bool b1 = (s1==""); // true
    bool b2 = (s1==" "); // false
    bool b3 = (s2==s3); // false
    bool b4 = (s2==s4); // false
    ...
}
```



## Vordefinierte Operatoren: Vergleiche (2)

对于其他运算符，适用以下条件：字符串上有一个顺序（根据各个字符的编码），例如：“A”，“B”，“A”，“A”，“Z”，“A”，...  
通过成对比较从左到右逐字符检查此顺序

- Für die anderen Operatoren gilt: es gibt eine Ordnung auf Zeichenketten (gemäß der Kodierung der Einzelzeichen), z.B.: 'A' < 'B', 'A' < 'a', 'Z' < 'a', ...
- Diese Ordnung wird zeichenweise von links nach rechts durch paarweises Vergleichen überprüft
- "BUCH" > "AUTO", da 'B' > 'A'
- "BUCH" < "bUCH", da 'B' < 'b'
- "BUCH" > "BAUCH", da 'U' > 'A'
- "BUCH" < "BUCHSTABE", hier wegen Zeichenanzahl

## Beispiel: Anzahl Leerzeichen bestimmen

- Typische for-Schleife zum Durchlaufen einer Zeichenkette

```
int countSpaces(const string & s)
{
    int ct = 0;
    for (int i = 0; i < s.length(); i++)
        if (s[i] == ' ')
            ct++;
    return ct;
}
```

- Test, ob (mindestens ein) Leerzeichen vorkommt:

```
bool hasSpace(const string & s)
{
    return countSpaces(s) > 0;
}
```

## Beispiel: effizientere Umsetzung

- Variante 1:

```
bool hasSpace(const string & s)
{
    bool found = false;
    for (int i = 0; i < s.length() && !found; i++)
        if (s[i] == ' ')
            found = true;
    return found;
}
```

- Variante 2 (kürzer):

```
bool hasSpace(const string & s)
{
    int i;
    for (i = 0; i < s.length() && s[i] != ' '; i++) /*nichts*/;
    return i < s.length();
}
```

## Beispiel: Leerzeichen ersetzen

- Leerzeichen durch Unterstriche ersetzen
- Direkte Manipulation der Zeichenkette (Ersetzen) möglich

```
void replaceSpaces(string & s)
{
    for (int i = 0; i < s.length(); i++)
        if (s[i] == ' ')
            s[i] = '_';
}
```

- Direkte Manipulation durch Einfügen oder Löschen von Zeichen auch möglich aber etwas komplizierter
- Dafür gibt es vordefinierte Funktionen!

## Beispiel: Leerzeichen entfernen

- Neue Zeichenkette aufbauen, die Schritt für Schritt aus Originalzeichenkette zusammengesetzt wird

```
string removeSpaces(const string & s)
{
    string result; // result=""
    for (int i = 0; i < s.length(); i++)
        if (s[i] != ' ')
            result += s[i];
    return result;
}
```

- Alternativ:

```
void removeSpaces(string & s)
{
    string result; // result=""
    for (int i = 0; i < s.length(); i++)
        if (s[i] != ' ')
            result += s[i];
    s = result;
}
```

Gibt es Fragen?

(Es gibt keine dummen Fragen!)

