

Inhaltsverzeichnis

Unix Kommandos	4
Die wichtigsten Befehle mit Erklärung.....	4
Programm Anlegen	4
Zeichen.....	5
ASCII	5
Steuer Zeichen	5
Kommentare	5
Ein/Ausgabe	5
Eingabe:.....	5
Ausgabe:.....	6
Elementary Datentypen	6
Operatoren.....	6
Arten von Operatoren.....	6
Kurzformen	7
Bit-Operatoren.....	7
sizeof-Operator	7
Typenumwandlung	8
Type-Casting.....	8
Kontrollstrukturen	8
Vergleichsoperatoren	8
Der !-Operator (logischer Operator).....	8
Logisches UND (&&) – Logisches ODER ()	8
.....	8
&&	8
Bedingungsoperator ?.....	9
Fallunterscheidung: die switch-Verzweigung	9
Schleifen.....	9
Funktionen	10
Definition von Funktionen	10
Funktionsdeklaration	10
Lokale / Globale Variablen	11
Lokale Variablen.....	11
Globale Variablen Bsp.	11
Speicherklassen / Schlüsselwörter für Variablen.....	11
auto	11
extern	11

Register	11
static.....	11
const.....	12
volatile.....	12
Lebensdauer:.....	12
Speicherklassen / Schlüsselwörter für Funktionen.....	12
extern.....	12
static.....	12
volatile.....	12
Funktionen Werteübergabe (call-by-value).....	12
Ablauf bei der Datenübergabe (call-by-reference):.....	12
Rückgabewert von Funktionen	12
Hauptfunktion main() und return	13
Rückgabewert beim Beenden eines Programms Bsp.	13
Getrenntes Kompilieren von Quelldateien	14
Rekursive Funktionen.....	14
Präprozessor-Direktiven	15
#include.....	15
#define	16
Bsp. Konstanten	16
Bsp. Makros.....	16
Arrays	17
Initialisierung.....	17
Anzahl der Elemente eines Arrays ermitteln (sizeof())	17
Arrays an Funktionen Übergaben	17
Mehrdimensionale Arrays.....	17
Arrays in Tabellenkalkulation einlesen (*.CSV-Dateien)	17
Strings/Zeichenketten (char Array).....	18
Einlesen von Strings	19
Standard Bibliothek <string.h>	19
Strings aneinander hängen strcat()	19
Ein Zeichen im String suchen strchr()	19
Strings vergleichen strcmp()	19
Einen String kopieren strcpy()	20
Einen Teilstring ermitteln strcspn()	20
Länge eines Strings ermitteln strlen()	20
String mit n Zeichen aneinander hängen strncat()	20

n Zeichen von zwei Strings miteinander vergleichen strncmp()	20
String mit n Zeichen kopieren strncpy()	20
Auftreten bestimmter Zeichen suchen strpbrk()	20
Das letzte Auftreten eines bestimmten Zeichens im String suchen strrchr()	21
Erstes Auftreten eines Zeichens, das nicht vorkommt strspn()	21
String nach Auftreten eines Teilstrings durchsuchen strstr()	21
String anhand bestimmter Zeichen zerlegen strtok()	22
Index.....	23

Unix Kommandos

Die wichtigsten Befehle mit Erklärung

Befehle	Erklärung
cat [Dateinamen]	Wird das Text Dokument angezeigt
cd [Directory]	Verzeichnis Wechsel
cp [dat1 ... datx] [Directory]	Kopiert Dateien in Verzeichnis
mkdir [Ordner Name]	Erstellen eines neuen Ordners
more und less	Erweiterte Versionen von cat
mv dat.alt dat.neu	Umbenenne von Dateien
rm [Dateiname/Ordner]	Löschen -i(fragt nach),-f(fragt nicht),-r(rekursiv)
rmdir Directory	Löscht Leeres Verzeichnis
lpr [Drucker] Dateiname	Schickt Datei an Standard Drucker, wenn nicht def.
lpq [Drucker]	Zeigt Drucker Warteschlange
lprm [Job Nr.]	Bricht Druck Job Nr. (lpq) ab
df	Zeigt aktuelle Datenträger Verwendung
Strings [Dateiname]	Sehr genauer Inhalt z.B. von Word-Datei
dir und ls	Zeigen den Inhalt des aktuellen Verzeichnisses
clear	Reinigt die Konsole
su	Macht einem zum Super User
sudo	Führt Befehl als Super User aus
sudo apt-get update	Aktualisiert die Paketliste
sudo apt-get upgrade	Installiert Updates
sudo apt-get install [Packetname]	Installiert das Paket
sudo apt install /PFAD/ZUR/PAKETDATEI.deb	Lokales Paket installieren
pwd	Zeigt aktuellen Arbeitsverzeichniss
cmp [dat1] [dat2]	Vergleicht die Dateien auf gleichen Inhalt
diff [dat1] [dat2]	Gibt den unterschiedlichen Inhalt aus
man [Kommando]	Zeigt mir eine Anleitung zu dem Kommando
Date +%W	Gibt mir die KW aus

Programm Anlegen

1. .c datei erstellen mit echo hi >> 1Prog.c
2. Include hinzufügen
3. Main
4. Kompilieren mit gcc -Wall -pedantic -o 1Bsp 1Prog.c

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Zeichen

ASCII

000	NUL	033	!	066	B	099	c	132	ä	165	ñ	198	ä	231	þ
001	Start Of Header	034	"	067	C	100	d	133	å	166	ª	199	Å	232	þ
002	Start Of Text	035	#	068	D	101	e	134	ä	167	º	200	Ĺ	233	Û
003	End Of Text	036	\$	069	E	102	f	135	ç	168	¿	201	Œ	234	Ü
004	End Of Transmission	037	%	070	F	103	g	136	ê	169	®	202	Š	235	Ý
005	Enquiry	038	&	071	G	104	h	137	ë	170	™	203	Ŧ	236	ÿ
006	Acknowledge	039		072	H	105	i	138	è	171	½	204	Ţ	237	Ÿ
007	Bell	040	(073	I	106	j	139	í	172	¼	205	=	238	ˉ
008	Backspace	041)	074	J	107	k	140	î	173	⅓	206	⌥	239	˘
009	Horizontal Tab	042	*	075	K	108	l	141	ï	174	¼	207	×	240	-
010	Line Feed	043	+	076	L	109	m	142	Ä	175	»	208	ä	241	±
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	∴	209	ð	242	ˉ
012	Form Feed	045	-	078	N	111	o	144	É	177	∞	210	É	243	¾
013	Carriage Return	046	.	079	O	112	p	145	æ	178	⌘	211	Ê	244	ŋ
014	Shift Out	047	/	080	P	113	q	146	Æ	179		212	Ë	245	§
015	Shift In	048	0	081	Q	114	r	147	ö	180	¡	213	Ì	246	÷
016	Delete	049	1	082	R	115	s	148	ø	181	À	214	Í	247	˙
017	-- frei --	050	2	083	S	116	t	149	ò	182	Á	215	Î	248	˚
018	-- frei --	051	3	084	T	117	u	150	ú	183	Â	216	Ï	249	˛
019	-- frei --	052	4	085	U	118	v	151	û	184	Ë	217	Ĵ	250	˜
020	-- frei --	053	5	086	V	119	w	152	ÿ	185	Ĥ	218	Œ	251	ˆ
021	Negative Acknowledge	054	6	087	W	120	x	153	Ö	186	Ĭ	219	█	252	˜
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	Ĵ	220	■	253	˚
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	Ĵ	221	;	254	■
024	Cancel	057	9	090	Z	123	{	156	£	189	ϕ	222	ì	255	
025	End Of Medium	058	:	091	[124		157	∅	190	¥	223	■		
026	Substitute	059	;	092	\	125	}	158	×	191	₹	224	Ó		
027	Escape	060	<	093]	126	~	159	ƒ	192	₹	225	Ḃ		
028	File Separator	061	=	094	^	127	¸	160	á	193	₹	226	Ḍ		
029	Group Separator	062	>	095	_	128	Ç	161	í	194	₹	227	Ḍ		
030	Record Separator	063	?	096	`	129	ü	162	ó	195	₹	228	Ḍ		
031	Unit Separator	064	@	097	a	130	é	163	ú	196	—	229	Ḍ		
032		065	A	098	b	131	â	164	ñ	197	†	230	µ		

Steuer Zeichen

\n für eine neue Zeile

\0 ist die Endemarkierung eines Strings

Kommentare

Zeilen Kommentar: //

Block Kommentar: /* */

Ein/Ausgabe

Eingabe:

scanf wird zum einlesen verwendet zu beachten ist das der Datei Type nach dem “%” angegeben sein muss und je nach Dateitype muss man auch noch vor den Variablen ein & setzen

```
/* FALSCH, da Adressoperator & fehlt */
scanf("%d", zahl);
/* Richtig, eine Zeichenkette benötigt keinen Adressoperator */
scanf("%s", string);
```

%d	decimal
%f	float
%c	Char
%s	string
%x	Hex
%lf	Double

Ausgabe:

printf wird zum Ausgeben verwendet zu beachten ist wie bei der Eingabe der richtige %_ Code. Mit & kann bei der Ausgabe die Speicherstelle ausgegeben werden.

```
double temp = 1234,2264;
printf("%f",temp); // 00001234,226400
printf("%4.2f",temp); //
```

Formatierungszeichen	Es wird ausgegeben (eine) ...
%d, %i	... vorzeichenbehaftete ganze Dezimalzahl.
%o	... vorzeichenlose ganze Oktalzahl.
%u	... vorzeichenlose ganze Dezimalzahl.
%x, %X	... vorzeichenlose ganze Hexzahl (a,b,c,d,e,f) bei x; (A,B,C,D,E,F) bei X
%f	... Gleitpunktzahl in Form von ddd.ddddd
%e, %E	... Gleitpunktzahl in Form von d.ddde+-dd bzw. d.dddE+-dd. Der Exponent enthält mindestens 2 Ziffern.
%g, %G	... float ohne Ausgabe der nachfolgenden Nullen
%c	... Form von einem Zeichen (unsigned char)
%s	... Form einer Zeichenkette
%p	Ausgabe eines Zeigerwertes
%n	Keine Ausgabe. Dieses Argument ist ein Zeiger auf eine Ganzzahl.
%%	... das Zeichen %

Elementary Datentypen

Name	Größe	Wertebereich
char, signed char	1 Byte = 8 Bit	-128...+127 bzw. 0 ... 255
unsigned char	1 Byte = 8 Bit	0...255
short, signed short	2 Byte = 16 Bit	-32768...+32767
unsigned short	2 Byte = 16 Bit	0...65535
int, signed int	4 Byte = 32 Bit	-2147483648...+2147483648
unsigned int	4 Byte = 32 Bit	0...4294967295
long, signed long	4 Byte = 32 Bit	-2147483648...+2147483648
unsigned long	4 Byte = 32 Bit	0...4294967295
float	4 Byte = 32 Bit	3.4*10 ⁻³⁸ ...3.4*10 ³⁸
double	8 Byte = 64 Bit	1.7*10 ⁻³⁰⁸ ...1.7*10 ³⁰⁸
long double	10 Byte = 80 Bit	3.4*10 ⁻⁴⁹³² ...3.4*10 ⁴⁹³²

Mit dem Schlüsselwort unsigned weiß der Compiler, dass es sich beim darauffolgenden Datentype um eine positive Zahl handelt. Denn mit unsigned wird die Vorzeichenbehaftete Stelle als normaler Speicher verwendet. Mit signed passiert genau gar nichts, es wird eine normale Variable erstellt d.h. int a ist dasselbe wie signed int a.

Operatoren

Arten von Operatoren

- Infix – der Operator steht zwischen den Operanden.
- Präfix – der Operator steht vor den Operanden.
- Postfix – der Operator steht hinter den Operanden.

Operator	Bedeutung
+	Addiert zwei Werte
-	Subtrahiert zwei Werte
*	Multipliziert zwei Werte
/	Dividiert zwei Werte
%	Modulo (Rest einer Division)

Kurzformen

Erweiterte Darstellung	Bedeutung
+=	a+=b ist gleichwertig zu a=a+b
-=	a-=b ist gleichwertig zu a=a-b
=	a=b ist gleichwertig zu a=a*b
/=	a/=b ist gleichwertig zu a=a/b
%=	a%=b ist gleichwertig zu a=a%b

Verwendung	Bezeichnung
var++	Postfix-Schreibweise
++var	Präfix-Schreibweise
var--	Postfix-Schreibweise
--var	Präfix-Schreibweise

- Die Postfix-Schreibweise erhöht bzw. erniedrigt den Wert von var, gibt aber noch den alten Wert an den aktuellen Ausdruck weiter.
- Die Präfix-Schreibweise erhöht bzw. erniedrigt den Wert von var und gibt diesen Wert sofort an den aktuellen Ausdruck weiter.

Bit-Operatoren

Bit-Operator	Bedeutung
&, &=	Bitweise AND-Verknüpfung
, =	Bitweise OR-Verknüpfung
^, ^=	Bitweise XOR
~	Bitweises Komplement
>>, >>=	Rechtsverschiebung
<<, <<=	Linksverschiebung

Achtung: Geht nur mit ganzzahligen Datentypen

sizeof-Operator

sizeof gibt die Größe aus die ein datatype in Byte belegt.

Beispiel zu sizeof() bei einem 32/64 bit System:

```
/* sizeof_type.c */
#include <stdio.h>
int main(void) {
    printf("char      : %d Byte\n", sizeof(char));
    printf("int       : %d Bytes\n", sizeof(int));
    printf("long      : %d Bytes\n", sizeof(long int));
    printf("float     : %d Bytes\n", sizeof(float));
    printf("double    : %d Bytes\n", sizeof(double));
    printf("66       : %d Bytes\n", sizeof(66));
    printf("Hallo    : %d Bytes\n", sizeof("Hallo"));
    printf("A        : %d Bytes\n", sizeof((char) 'A'));
    printf("34343434 : %d Bytes\n", sizeof(34343434));
    return 0;
}
```

```
char      : 1 Byte
int       : 4 Bytes
long      : 4 Bytes
float     : 4 Bytes
double    : 8 Bytes
66       : 4 Bytes
Hallo    : 6 Bytes
A        : 1 Bytes
34343434 : 4 Bytes
```

Typenumwandlung

Type-Casting

Es wird unterschieden in implizite Datentypenumwandlung und explizite Typenumwandlung.

- Impliziert ist wenn der Compiler die Umwandlung automatisch vornimmt z.B. `float a = 3+4;`
- Explizit ist es wenn ich als Programmierer diese Umwandlung erzwingen z.B. `int a = (int) 3,5;`
Achtung: Ich bin für das Ergebnis verantwortlich d.h. `\\a == 3` ich habe die Kommastelle verloren

```
int main(void) {  
    int x = 5, y = 2;  
    float z;  
    z = x / y;  
    printf("%f\\n", z);          /* = 2.000000 */  
    z = (float) x / (float) y;   /* = 2.500000 */  
    printf("%f\\n", z);  
    return 0;  
}
```

Achtung: Es gilt immer `int / int = int` d.h. ich muss einen der beiden Operatoren Type-Casten siehe oben.

Kontrollstrukturen

Vergleichsoperatoren

Vergleichsoperator	Bedeutung
a < b	Wahr, wenn a kleiner als b
a <= b	Wahr, wenn a kleiner oder gleich b
a > b	Wahr, wenn a größer als b
a >= b	Wahr, wenn a größer oder gleich b
a == b	Wahr, wenn a gleich b
a != b	Wahr, wenn a ungleich b

Der !-Operator (logischer Operator)

Anweisung	==	Anweisung
if(a != 0)	gleich	if(a)
if(a == 0)	gleich	if(!a)
if(a > b)	gleich	if(! (a <= b))
if((a-b) == 0)	gleich	if(! (a-b))

Logisches UND (&&) – Logisches ODER (||)

```
||  
if( (Bedingung1) || (Bedingung2) )  
    /* mindestens eine der Bedingungen ist wahr */  
else{  
    /* keine Bedingung ist wahr */  
}
```

```
&&  
if( (Bedingung1) && (Bedingung2) )  
    /* beide Bedingungen sind wahr */  
else{  
    /* eine oder keine der Bedingung ist wahr*/  
}
```


Bedingungsoperator ?

Syntax: <BEDINGUNG> ? <ANWEISUNG 1> : <ANWEISUNG 2>

Wenn die **BEDINGUNG** wahr ist, wird die **ANWEISUNG1** ausgeführt, sonst **ANWEISUNG2**. Der Programmablaufplan ist identisch mit dem der **if else**-Anweisung. Sie benötigen beispielsweise von zwei Zahlen den höheren Wert.

```
max = (a>b) ? a : b;
```

Fallunterscheidung: die switch-Verzweigung

Syntax:

```
switch(AUSDRUCK) {  
    AUSDRUCK_1 : anweisung_1  
    AUSDRUCK_2 : anweisung_2  
    AUSDRUCK_3 : anweisung_3  
    AUSDRUCK_n : anweisung_n  
}
```

Bsp.

```
int a=2;  
switch(a) {  
    case 1: printf("a ist eins\n"); break;  
    case 2: printf("a ist zwei\n"); break;  
    case 3: printf("a ist drei\n"); break;  
    default: printf("a ist irgendwas\n"); break;  
}
```

Schleifen

Syntax:

- Die while-Schleife:
 - while(Bedingung == wahr) {
/* Abarbeiten von Befehlen bis Bedingung ungleich wahr */
}
- Die do while-Schleife:
 - do {
/* Anweisungen */
} while(BEDINGUNG == wahr);
- Die for-Schleife:
 - for(Initialisierung; Bedingung; Reinitialisierung) {
/* Anweisungen */
}

for-Schleifen Möglichkeiten	
for(sek = 5; sek > 0; sek--)	Zählt rückwärts
for(n = 0; n <= 60; n = n + 10)	Zählt in 10er schritten
for(ch = 'A'; ch <= 'Z'; ch++)	Zählt durchs Alphabet (ASCII)
for(cube = 1; cube * cube * cube <= 216; cube++)	Berechnen der Seitenlänge von einem Würfel bei dem das Volumen gegeben ist
for(zs = 100.0; zs < 150.0; zs = zs * 1.1)	Zählt 10% vom Gesamtwert dazu
for(x = 0; y <= 75; y = (++x*5) + 50)	Zählt in 5er Sprüngen
for(y=2; x<20;	Es müssen nicht alle Variablen deklariert werden
for(;;)	Endlosschleife
for(printf("Bitte eine Zahl eingeben: "); n!=5;)	n muss solange eingegeben werden bis n == 5
for(n1 = 1, n2 = 2; n1 <= 10; n1++)	Initialisieren von mehreren werten

Funktionen

Definition von Funktionen

Syntax:

```
[Spezifizierer] Rückgabetyyp Funktionsname(Parameter) {  
    /* Anweisungsblock mit Anweisungen */  
}
```

Funktionsdeklaration

Funktionen in C beginnen mit einem Kleinbuchstaben ansonsten wie in C# Camel Case.

Es ist empfehlenswert **Prototypen** von Funktionen anzulegen wie im Bsp. Ein Prototype ist eine Information für den Compiler das es eine Funktion mit dem Rückgabewert z.B. int, Namen z.B. calcSum und den Übergabewerten z.B. (int, int) gibt. Durch die Prototypen könnte man auch die Funktionen erst nach der main() schreiben was in C unüblich ist. Prototypen sind auch wichtig wenn ich eine Funktion verwenden möchte sie ab erst nach der aufrufstell im Code deklariert ist würde der Compiler sie nicht finden.

Beispiel Funktionen

```
/* func3.c */  
#include <stdio.h>  
void func1(void);  
void func2(void);  
void func3(void);  
void func1(void) {  
    printf("Ich bin func1 \n");  
    func3();  
}  
void func2(void) {  
    printf("Ich bin func2 \n");  
}  
void func3(void) {  
    printf("Ich bin func3 \n");  
    func2();  
}  
int main(void) {  
    func1();  
    return 0;  
}
```

Lokale / Globale Variablen

Lokale Variablen

Sind immer nur in einem Anweisungsblock gültig.

Bei gleichnamigen Variablen ist immer die lokalste Variable gültig, also die, die dem Anweisungsblock am nächsten steht.

Globale Variablen Bsp.

```
/* func6.c */
#include <stdio.h>
int i=333; /* Globale Variable i */
void aendern(void) {
    i = 111; /* Ändert die globale Variable */
    printf("In der Funktion aendern: %d\n",i); /* 111 */
}
int main(void) {
    int i = 444;
    printf("%d\n",i); /* 444 */
    aendern();
    printf("%d\n",i); /* 444 */
    return 0;
}
```

In der main() tritt der oben beschriebene Fall ein da es in der main() schon ein i gibt nimmt der dieses den es ist die nächste Variable mit dem richtigen namen.

Speicherklassen / Schlüsselwörter für Variablen

auto

```
int zahl = 5;
auto int zahl1 = 5;
```

Auto ist im Endeffekt ein unnötiges Schlüsselwort es heißt so viel wie das der Compiler die Variable selber anlegt und nach verlassen des Anweisungsblocks auch wieder Automatisch löscht also ist auto ein explizites Zeichen für den Compiler das es sich um eine lokale Variable handelt d.h. beide Programm Zeilen bedeuten das selbe.

extern

Beindet sich die Variable in einer anderen Datei, wird das Schlüsselwort extern davorgesetzt. Diese Speicherklasse wird für Variablen verwendet, die im gesamten Programm verwendet werden können.

Register

Würde den Compiler anweisen das er versucht die Variable so lange wie möglich im CPU Register zu halten da es um ein vielfaches schneller ist. Jedoch ist es letzten Endes dem Compiler überlassen wo er es wirklich speichert.

static

```
/* func7.c */
#include <stdio.h>
void inkrement(void) {
    static int i = 1;
    printf("Wert von i: %d\n",i);
    i++;
}
int main(void) {          //ohne |mit static
    inkrement();          //1    |1
    inkrement();          //1    |2
    inkrement();          //1    |3
    return 0;
}
```

Dies hat damit zu tun das die Variable an einem anderen Bereich ab Speichert zu beachten das static variablen immer gleich mit eine wert initialisiert werden müssen. Sie wird erst beim Schließen des Programmes gelöscht.

const

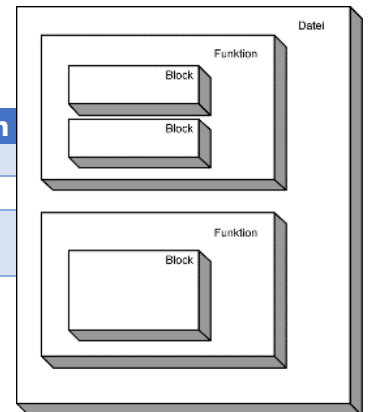
Eine Variable mit dem Schlüsselwort const davor heißt so viel, wie das sie immer den Initialisierungswert hat.

volatile

Die Variable muss vor jedem Zugriff neu aus den Hauptspeicher geladen werden und auch wieder sofort dort abgelegt werden.

Lebensdauer:

Position	Speicherklasse	Lebensdauer	Geltungsbereich
In einer Funktion	keine, auto, register	automatisch	Block
In einer Funktion	extern, static	statisch	Block
Außerhalb Funktion	keine, extern, static	statisch	Datei



Speicherklassen / Schlüsselwörter für Funktionen

extern

Jede Funktion ist wenn nicht anders Deklariert extern d.h. sie kann auch in einer andern Quelldatei aufgerufen werden.

static

Wenn die Funktion mit static Deklariert ist kann sie nur in der Datei verwendet werden in der sie Deklariert ist.

volatile

Verhindert das der Compiler den Code optimiert da die Funktion wie bei den Variablen beschrieben jedes Mal aus dem Hauptspeicher geladen und wieder abgespeichert werden muss.

Funktionen Werteübergabe (call-by-value)

Ablauf bei der Datenübergabe (call-by-reference):

1. Bei der Funktionsdefinition wird die Parameterliste festgelegt (formale Parameterliste).
2. Die Funktion wird von einer anderen Funktion mit dem Argument aufgerufen (muss mit dem Typ des formalen Parameters übereinstimmen).
3. Für die Funktion wird ein dynamischer Speicherbereich (im Stack) angelegt.
4. Jetzt kann die Funktion mit den Parametern arbeiten.

Rückgabewert von Funktionen

Beispiel für Übergabe und Rückgabe

```
/* func10.c */
#include <stdio.h>
float mixed(int x, char y, float z) {
    printf("Stückzahl : %d ",x);
    printf("Klasse   : %c ",y);
    printf("Preis     : %.2f Euro\n",z);
    return x*z;
}
int main(void) {
    float gesPreis = 0;
    gesPreis += mixed(6, 'A', 5.5f);
    gesPreis += mixed(9, 'B', 4.3f);
    printf("Gesamt Preis : %.2f Euro\n",gesPreis);
    return 0;
}
```

Hauptfunktion main() und return

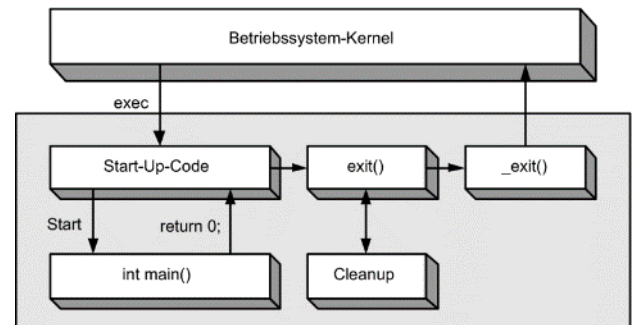
Es gibt verschiedene Varianten der Parameter der main():

```
int main(void) {  
    return 0;  
}
```

ODER

```
int main(int argc, char **argv) {  
  
    return 0;  
}
```

Der Startup-Code wird zu Beginn des Prozesses erzeugt (meist in Assembler) und dient der Beendigung eines Prozesses. Bei Beendigung der main()-Funktion mittels return 0 wird wieder zum Startup-Code zurückgesprungen. Er ruft dann die exit()-Funktion auf. Die exit()-Funktion führt dann noch einige Aufräumarbeiten aus (z.B. Freigabe des Speicherplatzes von benutzten Variablen des Programms). Zuletzt wird der Prozess mit der Funktion _exit() endgültig beendet. Hier eine bildliche.



Rückgabewert beim Beenden eines Programms Bsp.

```
/* exit_code.c */  
#include <stdio.h>  
#include <stdlib.h>  
int main(void) {  
    int val, ret;  
    printf("Bitte Eingabe machen : ");  
    ret = scanf("%d", &val);  
    if(ret != 1) {  
        printf("Fehler bei scanf()-Eingabe\n");  
        return EXIT_FAILURE;  
    }  
    if(val < 0) {  
        printf("Fehler - Negative Zahl\n");  
        return EXIT_FAILURE;  
    }  
    return EXIT_SUCCESS;  
}
```

Dies Rückgabewerte sind sinnvoll damit das System oder von diesem Programm abhängige andere Programme wissen ob es erfolgreich ausgeführt wurde. So können sie wenn nicht 0 (UNIX Standard) oder allgemein SUCCESS zurückgegeben wird diesen dienst erneut starten oder eine Info an einen Verwaltungs-Software schicken.

Getrenntes Kompilieren von Quelldateien

Beispiel

```
/*main.c*/
#include <stdio.h>
#include <stdlib.h>
Extern void modul1(void);
Extern void modul2(void);
Int main(void) {
    modul1();
    modul2();
    return EXIT_SUCCESS;
}
/*modul1.c*/
Void modul1(void) {
    printf("Ich bin das Modul 1\n");
}
/*modul2.c*/
Void modul2(void) {
    printf("Ich bin Modul 2\n");
}
```

CMD:

```
gcc -c main.c
```

```
gcc -c modul1.c
```

```
gcc -c modul2.c
```

// Dies hat drei .obj oder .o Dateien erstellt

CMD:

```
gcc main.o modul1.o modul2.o
```

// Dies erzeugt eine .out Datei die sie ausführen können mit -o kann ich eine Dateinamen und type angeben

// z.B. -o myapp.exe

Bei einer Änderung der main() muss jetzt jagdlich dies zwei Schritte ausgeführt werden.

CMD:

```
gcc -c main.c
```

```
gcc main.o modul1.o modul2.o
```

Rekursive Funktionen

Das heißt, dass sich ein Funktion immer wieder selber aufruft bis sie die End Bedingung erreicht/erfühlt.

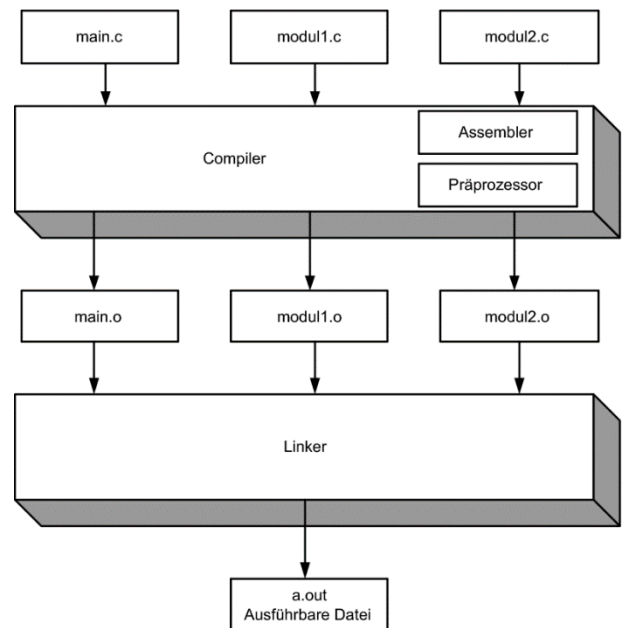
Jeder Neuaufwurf der Funktion wird im Stack gespeichert das ist ein Spezieller Hauptspeicher Bereich und wenn dieser voll ist gibt es eine so genannten Stack Overflow.

Die Fibonacci-Zahlen sollen rekursiv berechnet werden. Fibonacci-Zahlen sind z.B. 1, 2, 3, 5, 8, 13, 21, ...

Errechnet werden können sie mittels ... $1+2=3$, $2+3=5$, $3+5=8$, $5+8=13$. Nach Formel also:

$F(n+2)=F(n+1) + F(n)$. Hierzu der Code:

```
/* fibo.c */
#include <stdio.h>
#include <stdlib.h>
long fibo(long n) {
    if(n)
        return (n <= 2) ? n : fibo(n-2) + fibo(n-1);
}
int main(void) {
    long f;
    long i=0;
    printf("Wie viele Fibonacci-Zahlen wollen Sie ausgeben:");
    scanf("%ld",&f);
    while(i++ < f)
        printf("F(%ld) = %ld\n", i, fibo(i));
    return EXIT_SUCCESS; }
```



Präprozessor-Direktiven

Die Präprozessor-Direktive kümmert sich darum das der Code für den Compiler vorbereitet / aufbereitet wird. Es werden Kommentare entfernt und andere unnötigere Sachen. Anschließend fügt er den Inhalt der include Dateien ein und erstellt mit define Symbolisch Konstanten.

#include

Bindet die Headerdateien ein Info zu den meistverwendeten.

Headerdatei	Bedeutung
assert.h	Fehlersuche und Debugging
ctype.h	Zeichentest und Konvertierung
errno.h	Fehlercodes
float.h	Limits/Eigenschaften für Gleitpunkttypen
limits.h	Implementierungskonstanten
locale.h	Länderspezifische Eigenschaften
math.h	Mathematische Funktionen
setjmp.h	Unbedingte Sprünge
signal.h	Signale
stdarg.h	Variable Parameterübergabe
stddef.h	Standard-Datentyp
stdio.h	Standard-I/O
stdlib.h	Nützliche Funktionen
string.h	Zeichenkettenoperationen
time.h	Datum und Uhrzeit
complex.h	Komplexe Arithmetik (Trigonometrics, etc.)
Fenv.h	Kontrolle der Gleitpunktzahlen-Umgebung
inttypes.h	Für genauere Integertypen
iso646.h	Alternative Schreibweisen für logische Operatoren; Zur Verwendung von Zeichensätze im ISO646-Format (seit C95 vorhanden)
stdbool.h	Boolsche Datentypen
stdint.h	Ganzzahlige Typen mit vorgegebener Breite
tgmath.h	Typengenerische Mathematik-Funktionen
wchar.h	Umwandlung von Strings zu Zahlwerten für den erweiterten Zeichensatz; String- und Speicherbearbeitung für den erweiterten Zeichensatz; Ein- und Ausgabe für den erweiterten Zeichensatz (seit C95 vorhanden)
wctype.h	Zeichenuntersuchung für den erweiterten Zeichensatz (seit C95 vorhanden)

Syntax:

```
#include <stdio.h> /* Headerdatei für Standardfunktionen */
#include "/home/myownheaders/meinheader.h"
```

Achtung maximal ein include pro Zeile.

#define

Dies ermöglicht es mir symbolische Konstanten zu verwenden. Dahinter steckt eigentlich nur eine Textersetzung.

Syntax:

```
#define Bezeichner      Ersatzbezeichner
#define Bezeichner(Bezeichner_Liste)  Ersatzbezeichner
#undef Bezeichner //Beendet diese Konstante oder Marko ansonsten bis Ende der Datei gültig
```

Bsp. Konstanten

```
#include <stdio.h>
#define GANZZAHL      int
#define SCHREIB      printf(
#define END          );
#define EINGABE      scanf(
#define ENDESTART    return 0;
#define NEUEZEILE    printf("\n");
#define START        int main()
#define BLOCKANFANG  {
#define BLOCKENDE    }
```

```
START
BLOCKANFANG
    GANZZAHL zahl;
    SCHREIB "Hallo Welt" END
    NEUEZEILE
    SCHREIB "Zahleingabe: " END
    EINGABE "%d", &zahl END
    SCHREIB "Die Zahl war %d", zahl END
ENDESTART
BLOCKENDE
```

Bsp. Makros

```
/* define2.c */
#include <stdio.h>
#include <stdlib.h>
#define MAX(x,y) ( (x)<=(y) ?(y) :(x) )
#define KLEINER_100(x) ((x) < 100)
Void klHundert(int zahl) {
    if(KLEINER_100(zahl))
        printf("Ja! Die Zahl ist kleiner als 100!\n");
    else
        printf("Die Zahl ist größer als 100!\n");
}
Int main(void) {
    int b = 99;
    klHundert(b);
    return EXIT_SUCCESS;
}
```


Arrays

Initialisierung

Datentyp Arrayname[Anzahl_der_Elemente];

Anzahl der Elemente eines Arrays ermitteln (sizeof())

sizeof() gibt ja die Anzahl der reservierten Bytes zurück das heißt um die Anzahl der Elemente eines Arrays zu ermitteln muss man die Größe des Arrays durch die Größe des Datentyps dividieren sizeof(array) / sizeof(Datentype)

```
/* array8.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int zahlen[] = {3,6,3,5,6,3,8,9,4,2,7,8,9,1,2,4,5};
    printf("Speicher Größe aller Elemente: %d\n", sizeof(zahlen)); //68
    printf("Anzahl der Elemente: %d\n", sizeof(zahlen) / sizeof(int)); //17
    return EXIT_SUCCESS;
}
```

Arrays an Funktionen Übergeben

```
#define MAX 4
int funktion12(int matrix[MAX][MAX]){return 0;}
int funktion3(int matrix1[MAX][MAX], char matrix2[MAX][MAX]){return 0;}
int funktion4(int matrix1[MAX][MAX], char matrix2[MAX][MAX], int posZ, int posS){return 0;}
int main(void) {
    int matrix[MAX][MAX];
    int temp = 0;
    temp = funktion12(matrix);
    temp = funktion3(matrix,matrix);
    temp = funktion4(matrix1,matrix2,temp,temp);
}
```

zeigt die Übergabe an Funktionen, zurückgabe ist nicht erforderlich da Array eine Referencetype ist.

Mehrdimensionale Arrays

Syntax:

```
int matrix[4][5];    /* Zweidimensional - 4 Zeilen x 5 Spalten */
```

```
/* 4 Zeilen 5 Spalten */
int Matrix[4][5] = { {10,20,30,40,50},
                     {15,25,35,45,55},
                     {20,30,40,50,60},
                     {25,35,45,55,65}};
```

```
int dreid[][][] = {1.Feldindex{2.Feldindex{3.Feldindex}}};
```

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
10	20	30	40	50
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
15	25	35	45	55
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
20	30	40	50	60
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
25	35	45	55	65

Arrays in Tabellenkalkulation einlesen (*.CSV-Dateien)

```
/* md_array4.c */
#include <stdio.h>
#include <stdlib.h>
#define WOCHEN 4
#define TAGE 7
float stand[WOCHEN][TAGE] = {
    { 12.3f,13.8f,14.1f,12.2f,15.4f,16.5f,14.3f },
    { 15.4f,13.6f,13.6f,14.6f,15.6f,16.3f,19.5f },
    { 20.5f,20.4f,21.5f,23.4f,21.4f,23.5f,25.7f },
    { 25.5f,26.6f,24.3f,26.5f,26.9f,23.6f,25.4f }
};

int main(void) {
    int i, j;
    printf("Tag;Montag;Dienstag;Mittwoch;Donnerstag; "
           "Freitag;Samstag;Sonntag");
    for(i=0; i < WOCHEN; i++) {
        printf("\nWoche%d;",i);
        for(j=0;j < TAGE; j++) {
            printf("%.2f;",stand[i][j]);
        }
    }
    return EXIT_SUCCESS;
}
```

Mit Programmname > noveber.csv aus der Konsole heraus starten. Dies erstellt ein csv Datei und leidet in dies alle printf() Befehle um.

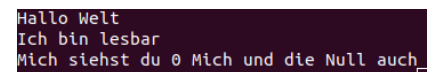
```
Tag;Montag;Dienstag;Mittwoch;Donnerstag;Freitag;Samstag;Sonntag
Woche0;12.30;13.80;14.10;12.20;15.40;16.50;14.30;
Woche1;15.40;13.60;13.60;14.60;15.60;16.30;19.50;
Woche2;20.50;20.40;21.50;23.40;21.40;23.50;25.70;
Woche3;25.50;26.60;24.30;26.50;26.90;23.60;25.40;
```

Strings/Zeichenketten (char Array)

In C gibt es keinen String das heißt man muss ein char[] verwenden jedoch kann man sich einige nützlich Funktionen aus der string.h Bibliothek importieren.

```
const char hallo[] = { 'H', 'a', 'l', 'l', 'o', ' ',
                        'W', 'e', 'l', 't', '\n', '\0' };
```

```
/* string1.c */
#include <stdio.h>
#include <stdlib.h>
char hello1[] = { "Hallo Welt\n" };
char output[] = { "Ich bin lesbar \0 Ich nicht mehr" };
char deznu[] = { "Mich siehst du 0 Mich und die Null auch" };
int main(void) {
    printf("%s",hello1);
    printf("%s\n",output);
    printf("%s\n",deznu);
    return EXIT_SUCCESS;
}
```



```
Hallo Welt
Ich bin lesbar
Mich siehst du 0 Mich und die Null auch
```

Standardmäßig ist bei C /0 das Zeichen dafür das ein Char Array aus ist das heißt man sollte aufpassen und das Array immer um eins größer machen damit sich nicht ein Zeichen an der gleichen Stelle wie das /0 sein sollte.

Buchstaben können auch als Zahl eingegeben werden da dahinter immer der ASCII Code steht d.h. man kann auch mit Chars rechnen.

Einlesen von Strings

```
/* string7.c */
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    char str[100];
    printf("Geben sie ein paar Wörter ein : ");
    fgets(str, 100, stdin);
    printf("Ihre Eingabe: %s\n", str);
    return EXIT_SUCCESS;
}
```

Es wird der String mit der Funktion `fgets()` eingelesen wobei wie im Beispiel die Syntax zu beachten ist.

```
char *fgets(char *string, int anzahl_zeichen, FILE *stream);
```

Standard Bibliothek <string.h>

Strings aneinander hängen `strcat()`

Fügt zwei `char[]` zu einem zusammen es wird empfohlen `strncat()` zu verwenden.

Syntax:

```
char *strcat(char *s1, const char *s2);
```

```
/* stringcat.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char ziel[30] = "Hallo ";
    char name[20];
    printf("Wie heissen Sie: ");
    fgets(name, 20, stdin);
    strcat(ziel, name);
    printf("%s", ziel);
    return EXIT_SUCCESS;
}
```

Ein Zeichen im String suchen `strchr()`

Sucht nach einem bestimmten Zeichen im `char[]`.

Syntax:

```
char *strchr(const char *s, int ch);
```

```
/* strchr.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char str[] = "Ein String mit Worten";
    printf("%s\n", strchr(str, (int)'W')); //Worten
    return EXIT_SUCCESS;
}
```

Strings vergleichen `strcmp()`

`Strcmp()` führt einen lexicographische Vergleich der beiden Strings durch.

Sind beide Strings identisch, gibt diese Funktion 0 zurück. Ist der String `s1` kleiner als `s2`, ist der Rückgabewert kleiner als 0 und ist `s1` größer als `s2`, dann ist der Rückgabewert größer als 0.

Syntax:

```
int strcmp(const char *s1, const char *s2);
```

Einen String kopieren **strcpy()**

Kopiert den Inhalt von String2 in String1.

Dass hierbei der String-Vektor s1 groß genug sein muss, versteht sich von selbst. Bitte beachten Sie dabei, dass das Ende-Zeichen '\0' auch Platz in s1 benötigt.

Syntax:

```
char *strcpy(char *s1, const char *s2);
```

Einen Teilstring ermitteln **strcspn()**

Gibt die Länge bis zu einem gewissen auftretenden Zeichen zurück.

Sobald ein Zeichen, welches in s2 angegeben wurde, im String s1 vorkommt, liefert diese Funktion die Position dazu zurück.

Syntax:

```
int strcspn(const char *s1, const char *s2);
```

Länge eines Strings ermitteln **strlen()**

Gibt die länge des Strings ohne /0 zurück.

Syntax:

```
size_t strlen(const char *s1);
```

String mit n Zeichen aneinander hängen **strncat()**

Das gleiche wie strcat() nur werden hier n Zeichen angehängt was die Funktion deutlich sicherere macht.

size_t ist ein primitiver Datentyp, der meistens als unsigned int oder unsigned long deklariert ist.

Syntax:

```
char *strncat(char *s1, const char *s2, size_t n);
```

n Zeichen von zwei Strings miteinander vergleichen **strncmp()**

Das ist das gleiche wie strcmp() nur werden hier die ersten n Zeichen der beiden Strings verglichen size_t ist wieder ein unsigned int oder unsigned long.

Syntax:

```
int strncmp(const char *s1, const char *s2, size_t n);
```

String mit n Zeichen kopieren **strncpy()**

Das ist das gleiche wie strcpy() nur werden hier die ersten n Zeichen kopiert zu beachten ist das er das /0 nicht mit kopiert. size_t ist wieder ein unsigned int oder unsigned long.

Syntax:

```
char *strncpy(char *s1, const char *s2, size_t n);
```

Auftreten bestimmter Zeichen suchen **strpbrk()**

Diese Funktion arbeitet ähnlich wie strcspn(), nur dass hierbei nicht die Länge eines Teilstrings ermittelt wird, sondern das erste Auftreten eines Zeichens in einem String, welches im Suchstring enthalten ist.

Syntax:

```
char *strpbrk(const char *s1, const char *s2);
```

Das letzte Auftreten eines bestimmten Zeichens im String suchen **strrchr()**

Diese Funktion ist der Funktion **strchr()** ähnlich, nur dass hierbei das erste Auftreten des Zeichens von hinten, genauer das letzte, ermittelt wird.

Syntax: `char *strrchr(const char *s, int ch);`

Die Funktion **fgets()** hängt beim Einlesen eines Strings immer das Newline-Zeichen am Ende an. Manchmal ist das nicht erwünscht. Wir suchen mit **strrchr()** danach und überschreiben diese Position mit dem `'\0'`-Zeichen:

```
/* strrchr.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char string[20];
    char *ptr;
    printf("Eingabe machen: ");
    fgets(string, 20, stdin);
    /* Zeiger auf die Adresse des Zeichens \n */
    ptr = strrchr(string, '\n');
    /* Zeichen mit \0 überschreiben */
    *ptr = '\0';
    printf("%s", string);
    return EXIT_SUCCESS;
}
```

Erstes Auftreten eines Zeichens, das nicht vorkommt **strspn()**

Die Funktion **strspn()** gibt die Position des ersten Auftretens eines Zeichens an, das nicht vorkommt. Die **Syntax** lautet:

`int strspn(const char *s1, const char *s2);`

Das folgende Beispiel liefert Ihnen die Position des Zeichens zurück, welches keine Ziffer ist

```
/* strspn.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char string[] = "75301234-2123";
    int pos = strspn(string, "0123456789");
    printf("Position, welche keine Ziffer ist:");
    printf(" %d\n", pos); /* 8 */
    return EXIT_SUCCESS;
}
```

String nach Auftreten eines Teilstrings durchsuchen **strstr()**

Mit dieser Funktion kann man einen String auf das Auftreten eines Teilstrings durchsuchen.

Syntax:

`char *strstr(const char *s1, const char *s2);`

Damit wird der String **s1** nach einem String mit der Teilfolge **s2** ohne `'\0'` durchsucht. Ein Beispiel:

```
/* strstr.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char string[] = "Das ist ein Teststring";
    char suchstring[] = "ein";
    if ( strstr(string, suchstring) != NULL)
        printf("Suchstring \"%s\" gefunden\n", suchstring);
    return EXIT_SUCCESS;
}
```

String anhand bestimmter Zeichen zerlegen `strtok()`

Mit der Funktion `strtok()` können Sie einen String in einzelne Teilstrings anhand von Tokens zerlegen.

Syntax:

```
char *strtok(char *s1, const char *s2);
```

Damit wird der String `s1` durch das Token getrennt, welches sich im `s2` befindet. Ein Token ist ein String, der keine Zeichen aus `s2` enthält. Ein Beispiel:

```
/* strtok.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void) {
    char string[] = "Ein Teststring mit mehreren Worten\n"
                   "und mehreren Zeilen.\t Ende\n";

    int i=1;
    char *ptr;
    ptr = strtok(string, "\n\t ");
    while(ptr != NULL) {
        printf("%d. Wort: %s\n", i++, ptr);
        ptr = strtok(NULL, "\n\t ");
    }
    return EXIT_SUCCESS;
}
```

Mit der Zeile

```
ptr = strtok(string, "\n\t ");
```

würde nur das erste Wort anhand eines der Whitespace-Zeichen Newline, Tabulator oder Space getrennt werden. Der String wird jetzt von der Funktion `strtok()` zwischengespeichert. Wollen Sie jetzt den String mit weiteren Aufrufen zerlegen, müssen Sie `NULL` verwenden.

```
ptr = strtok(NULL, "\n\t ");
```

Dabei gibt jeder Aufruf das Token zurück. Das jeweilige Trennzeichen wird dabei mit `'\0'` überschrieben. In diesem Beispiel ist die Schleife am Ende, wenn `strtok()` den `NULL`-Zeiger zurückliefert.

Index

#define	16	Kompilieren von Quelldateien	14
#include	15	Konstanten	15, 16
% 5		Kontrollstrukturen	8
(sizeof())	17	Kurzformen	7
Arrays	17	Länge eines Strings ermitteln	20
Arrays an Funktionen Übergaben	17	Lebensdauer	12
Arrays in Tabellenkalkulation einlesen	18	letzte Auftreten eines bestimmten Zeichens im String suchen	21
ASCII	5	lexigraphische Vergleich	19
Auftreten bestimmter Zeichen suchen	20	logischer Operator	8
auto	11	Logisches ODER ()	8
Bedingungsoperator	9	Logisches UND (&&)	8
Bit-Operatoren	7	Lokale Variablen	11
Buchstaben	18	main	13
call-by-reference	12	Makros	16
call-by-value	12	Mehrdimensionale Arrays	17
Char Array	18	n Zeichen von zwei Strings miteinander vergleichen	20
CMD	14	Newline-Zeichen	21
Code für den Compiler	15	NULL-Zeiger	22
const	12	Operator	7
csv Datei	18	Operatoren	6
Datenübergabe	12	Postfix	6, 7
define	15	Präfix	6, 7
do while-Schleife	9	Präprozessor-Direktiven	15
Elementary Datentypen	6	printf	6
End Bedingung	14	Programm Anlegen	4
erste Auftreten eines Zeichens in einem String	20	Prototypen	10
Erstes Auftreten eines Zeichens, das nicht vorkommt	21	Referencetype	17
exit	13	Register	11
EXIT_FAILURE	13	Rekursive Funktionen	14
EXIT_SUCCESS	13	return	13
explizite Typenumwandlung	8	Rückgabewert beim Beenden eines Programms	13
extern	11, 12	Rückgabewert von Funktionen	12
Fallunterscheidung: die switch-Verzweigung	9	scanf	5
fgets()	21	Schleifen	9
Fibonacci-Zahlen	14	Schlüsselwörter für Funktionen	12
Formatierungszeichen	6	Schlüsselwörter für Variablen	11
for-Schleife	9	signed	6
for-Schleifen Möglichkeiten	9	sizeof	7
Funktion fgets()	19	sizeof()	17
Funktionen	10	Speicherklassen	11, 12
Funktionen Werteübergabe	12	Stack	14
Funktionsdeklaration	10	Stack Overflow	14
Globale Variablen	11	<i>Standard Bibliothek <string.h></i>	19
Hauptfunktion main() und return	13	Startup-Code	13
Headerdateien	15	static	11, 12
implizite Datentypenumwandlung	8	stdlib.h	13
include Dateien	15	Steuer Zeichen	5
Infix	6	strchr()	19
Initialisierung	17	Strcmp()	19
Kommentare	5	strcpy()	20

strcspn()	20	strtok()	22
String anhand bestimmter Zeichen zerlegen	22	SUCCESS	13
String kopieren	20	Teilstring ermitteln	20
String mit n Zeichen aneinander hängen	20	Token	22
String mit n Zeichen kopieren	20	Tokens	22
String nach Auftreten eines Teilstrings durchsuchen	21	Type-Casting	8
string.h	18	Typenumwandlung	8
Strings	18	Übergabe an Funktionen	17
Strings aneinander hängen	19	Unix Kommandos	4
Strings vergleichen strcmp()	19	UNIX Standard	13
strlen()	20	unsigned	6
strncat()	19, 20	Vergleichsoperatoren	8
strncmp()	20	volatile	12
strncpy()	20	while-Schleife	9
strpbrk()	20	Whitespace-Zeichen Newline, Tabulator oder Space..	22
strrchr()	21	wichtigsten Befehle	4
strspn()	21	Zeichen im String suchen	19
strstr()	21	Zeichenketten (char Array)	18