KV Betriebssysteme
Einführung in C

KV Betriebssysteme
Überblick über die LVA

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Ziele und Voraussetzungen

- **Ziele**
  - Einführung in die systemnahe Programmierung mit Hilfe der Programmiersprache C.
  - Vertiefung der Kenntnisse aus der Vorlesung Betriebssysteme
  - Übung der erworbenen Kenntnisse anhand praktischer Beispiele
- **Voraussetzungen**
  - Java-Kenntnisse
  - Programmiererfahrung

Systemvoraussetzungen

- **Hardware**
  - PC (Intel- oder kompatibler Prozessor)
- **Betriebssystem**
  - Windows NT 4, Windows 2000, Windows XP
  - (Windows 95, Windows 98, Windows ME)
  - (Linux)
- **Software**
  - Für C: Microsoft Visual C/C++ 6.0 (5.0)
  - Für Java: Pow! 3.0, Sun JDK

Geplante Termine (I)

Datum	Inhalt	Übung
05.03. / 06.03.	Einführung in C	
12.03. / 13.03.	Einführung in C	1
19.03. / 20.03.	Einführung in C	2
26.03. / 27.03.	Osterferien	
02.04. / 03.04.	Osterferien	
09.04. / 10.04.	Einführung in C	3
16.04. / 17.04.	Einführung in C	4
23.04. / 24.04.	Windowsprogrammierung	5
30.04. / 01.05.	Tag der Arbeit	

Geplante Termine (II)

Datum	Inhalt	Übung
07.05. / 08.05.	Windowsprogrammierung	6
14.05. / 15.05.	Windowsprogrammierung	7
21.05. / 22.05.	Pfingsten	
28.05. / 29.05.	Windowsprogrammierung	8
04.06. / 05.06.	Java VM, Java Multiprocessing	9
11.06. / 12.06.	Java VM, Java Multiprocessing	10
18.06. / 19.06.	Java VM, Java Multiprocessing	
25.06. / 26.06.	Klausur	

Modus

- **Jede Woche wird eine Übung ausgeteilt. Es gibt 10 Übungen. 8 von 10 Übungen müssen abgegeben werden, um zur Klausur antreten zu können.**
- **1/3 der Übungen werden korrigiert. Übungen mit weniger als der Hälfte der möglichen Punktezahl werden als nicht abgegeben gewertet.**
- **Die Endnote setzt sich zusammen aus dem Klausurergebnis, der Anzahl der abgegebenen Übungen, der Mitarbeit und der erreichten Punktezahl bei den korrigierten Übungen.**

KV Betriebssysteme
Einführung in C

Ansprechpartner

- **DI. Roland Eggetsberger**
  - **Tel.: 0732 / 2468 - 8442**
  - **E-Mail: eggetsberger@fim.uni-linz.ac.at**
  - **Sprechstunden: Montag, 9:00 - 12:00, T660**
- **DI. Markus Jöbstl**
  - **E-Mail: joebstl@fim.uni-linz.ac.at**
- **DI. Dr. Peter René Dietmüller**
  - **E-Mail: dietmueller@fim.uni-linz.ac.at**

KV Betriebssysteme Einführung in C

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Geschichte & Literatur

- **Geschichte**
  - **1970 von Dennis Ritchie für eine DEC PDP-11 entw.**
  - **1978 DeFacto Standard durch das erste C-Buch von Dennis Ritchie und Brian Kernighan (Kernighan/Ritchie Standard)**
  - **1982 Definition des heute üblichen Standards des American National Standards Institute (ANSI)**
- **Standardwerke**
  - **Kernighan Brian W., Ritchie Dennis: Programmieren in C, Zweite Auflage: ANSI C, Carl Hanser, 1990**
  - **Stroustrup Bjarne: Die C++ Programmiersprache; Addison-Wesley, 1987**

Eigenschaften von C (I)

- **C ist eine weit verbreitete Programmiersprache.**
  - **C-Compiler existieren für die meisten Zielsysteme.**
- **C kann auf kleinstem Raum beschrieben werden.**
- **C kann schnell erlernt werden.**
- **C ist sehr effizient.**
  - **Die meisten C-Systeme können den erzeugten Code auf Codelänge oder Geschwindigkeit hin optimieren.**
- **C selbst definiert keine Speicherverwaltung.**
- **C selbst hat keine Ein- und Ausgabeoperationen.**

Eigenschaften von C (II)

- **C kennt nur Funktionen, keine Prozeduren.**
- **C kennt keine verschachtelten Funktionen.**
- **C-Funktionen können eine variable Anzahl von Parametern haben.**
- **Parameter werden in C immer als Call-by-value-Parameter übergeben.**

Das erste C-Programm

- **Jedes C-Programm besteht zumindest aus einer Funktion main. Im folgenden Beispiel gibt das Programm den Text "Hello world." aus.**

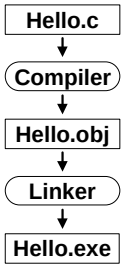
```
#include <stdio.h>
int main (void) {
    printf ("Hello world.\n");
    return 0;
}
```

KV Betriebssysteme
Einführung in C

Struktur eines C-Programmes

- Preprocessor-Anweisungen
  - z.B.: #include, #define
- Typdeklarationen
  - z.B.: typedef, struct
- Funktionsprototypen
- Globale Variablen
- Funktionen
- Hauptprogramm (Funktion main)

C-Programm übersetzen



Grundlegende Syntaxregeln

- Anweisungen sind durch ; zu trennen; sie lassen sich durch { } in Blöcke zusammenfassen (nach } kein Strichpunkt!)
- C ist case-sensitiv
- Funktionen müssen immer mit () aufgerufen werden, auch wenn sie keine Parameter haben!
- Es können nur Variablen und Funktionen verwendet werden, die vorher deklariert wurden.
- Variablen können global (außerhalb der Funktionen) bzw. lokal (an jedem Blockanfang) deklariert werden

Kommentare

- Syntax
  - /* ... */
  - Kommentare können nicht geschachtelt werden!
  - Kommentare mit // gibt es nur in C++! (Ausnahme: MS Visual C++)
- Was soll kommentiert werden?
  - Am Programmanfang (Header): Programmname, Autor, Beschreibung, Verwendung, Dateiformate, Einschränkungen, Fehlerbehandlung
  - Im Programm: Komplizierte Stellen des Programmes

Programmierstil

- Sprechende Bezeichner (mit Kommentar)
  - int a;
  - int a; /* Anzahl der Elemente */
  - int anzahl; /* Anzahl der Elemente */
  - int nAnzahlElemente;
- Einrückung
  - Einheitlich!
- Klarheit (Unterteilung in Einheiten)
  - Funktionen, Module
- Einfachheit

KV Betriebssysteme
Elementare Datentypen

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

KV Betriebssysteme
Einführung in C

Elementare Datentypen (I)

Typ	Länge	von	bis
char	1	-128	127
unsigned char	1	0	255
short	2	-32768	32767
unsigned short	2	0	65535
int		Systemabhängig	
unsigned int		Systemabhängig	
long	4	–2,147,483,648	2,147,483,647
unsigned long	4	0	4,294,967,295

Elementare Datentypen (II)

Typ	Länge	von	bis
float	4	3.4*10^-38	3.4*10^38
double	8	1.7*10^-308	1.7*10^308
long double	10	1.2*10^-4932	1.2*10^4932
void	0		

Literale

- Ganze Zahlen
  - Dezimal: 160
  - Hexadezimal: 0xA0
  - Oktal: 0240
  - Long: 160L
  - Unsigned: 160U
- Reelle Zahlen
  - Dezimal: 34.2
  - Exponential: 3.42E1
- Zeichen
  - 'a'
- Zeichenkette
  - "My name is Bond, James Bond"

Datentyp char

- Der Datentyp „char“ repräsentiert ein einzelnes ASCII-Zeichen.
- Mit dem Datentyp „char“ kann man auch rechnen. Als Wert fungiert dann der ASCII-Wert des Zeichens.
- Beispiel
  - 'B' - 'A' ergibt 1 (66 - 65)
  - 'Z' - 'A' ergibt 25 (90 - 65).
  - '9' - '0' ergibt 9 (57 - 48).

Sonderzeichen (I) (Escape Sequence)

Konstante	ASCII-Wert	Bedeutung
\0	0	Stringende
\b	8	Backspace
\t	9	Tabulatorsprung
\n	10	Zeilenvorschub
\f	12	Seitenende
\r	13	Zeilenende
\"	34	"-Zeichen
\'	39	' -Zeichen
\\	92	\-Zeichen
\nnn	nnn	Oktal-Zeichen nnn

Sonderzeichen (II)

Zeichenkette	entspricht
"\n"	(Leerzeile)
"Hallo,\nwie geht's.“	Hallo, wie geht's.
"Vorname\tNachname“	Vorname Nachname
"c:\\autoexec.bat"	c:\autoexec.bat
"Hallo\rWelt"	Welt (am Bildschirm)

KV Betriebssysteme
Einführung in C

Datentyp boolean

- C kennt keinen expliziten Datentyp Boolean.
- Stattdessen wird ein Integer-Wert verwendet, der folgendermaßen interpretiert wird:
  - ein Wert von 0 bedeutet FALSE
  - ein Wert ungleich 0 bedeutet TRUE
- Diese Regelung ist manchmal praktisch:
  - Statt „if (i != 0) ...“ kann man schreiben „if (i) ...“

KV Betriebssysteme
Variablen und Funktionen

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Variablendeklaration

- Syntax
  - Typ Variablenname {"," Variablenname}
  - Variablennamen müssen mit einem Buchstaben oder „_“ beginnen. Danach können Buchstaben oder Ziffern kommen. Reservierte Wörter dürfen nicht verwendet werden.
- Beispiel
  - int i, j;
  - char c;

Funktionsdeklaration

- Syntax
  - Typ Funktionsname (Parameter) { Funktionsrumpf }
  - Zur Erinnerung: In C gibt es keine Prozeduren! Dafür gibt es den Datentyp void!
  - Default-Rückgabetyp ist int.
- Beispiel
  - int Max(int a, int b) {
    int c;
    c = (a > b) ? a : b;
    return c;
    }

Funktionsaufruf

- Syntax
  - Funktionsname(Aktualparamter);
  - Parameter werden immer „Call by Value“ übergeben.
  - Der Rückgabewert kann unter den Tisch fallen.
  - Klammern nach dem Funktionsnamen müssen immer angegeben werden, auch wenn keine Parameter übergeben werden!
- Beispiele
  - int c;
    c = Max(1, 2);
  - Max(1, 2);

Datentyp void

- Bedeutung
  - Der Datentyp void wird meist als Rückgabetyp von Funktionen verwendet. Durch Verwendung von void kann man explizit angeben, daß eine Funktion keinen Wert zurückgibt. In diesem Fall darf „return“ nicht verwendet werden.
- Beispiel
  - void DoMax (int a, int b) { /* Funktion o. Rückgabe */
    int c;
    c = Max(a,b);
    }

KV Betriebssysteme
Einführung in C

Funktionsprototyp

- **Bedeutung**
  - Schnittstelle einer Funktion bekanntmachen
- **Syntax**
  - Typ Funktionsname (Parameter);
- **Beispiel**
  - int Max(int x, int y);

    ... (Aufruf der Funktion Max) ...

    int Max(int x, int y) { ... }

Sichtbarkeit von Variablen

```
int i;    /* globale Variable */

void test (void) {
  int i;  /* lokale Variable */
  i = 2;
}

int main (void) {
  i = 1;
  test();
  return 0;
}
```

glob. i	lok. i
0	
	2
1	
1	
1	

Speicherklasse

- **Auto**
  - Standard für lokale Variablen. (Kein Schlüsselwort)
- **Static**
  - Variablen werden nur einmal initialisiert.
  - Variablen können beliebig verändert werden.
  - Bsp.: static int zaehler = 0;
- **Extern**
  - Variablen werden aus anderen Quellen übernommen.
  - Bsp.: extern int zaehler;
- **Register**

Lifetime, Linkage, Visibility

Level	Item	Storage Class	Lifetime	Linkage	Visibility
File scope	Variable	static	Global	Internal	source file
	Variable	extern	Global	External	source file
	Variable	auto	Local	External	source file
	Variable	register	Local	External	source file
	Function	static	Global	Internal	Single source file
	Function	extern	Global	External	source file
	Function	auto	Local	External	source file
Block scope	Variable	static	Global	Internal	Block
	Variable	extern	Global	External	Block
	Variable	auto	Local	Internal	Block
	Variable	register	Local	Internal	Block

KV Betriebssysteme
Ein- / Ausgabe

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Funktion printf

- **Zweck**
  - Formatierte Ausgabe von Werten
- **Modul**
  - #include <stdio.h>
- **Aufruf**
  - printf (Formatstring, ...);
  - Dieser Funktion können eine bel. Anzahl von Parametern übergeben werden. Der erste Parameter muß ein Formatstring sein. Dahinter folgen die auszugebenden Werte.
  - Für jeden auszugebenden Wert sollte eine Formatangabe im Formatstring vorhanden sein. Die übergebenen Werte werden nicht auf Typkorrektheit geprüft!

KV Betriebssysteme
Einführung in C

Formatstring von printf

- **Syntax**
  - %[flags] [width] [.precision] [{h | l}]type
- **Flags**
  - Steuerung der Ausrichtung, Vorzeichen, Leerzeichen, ..
- **Width**
  - Breite der Ausgabe (mit ‚*' variable Breite)
- **Precision**
  - Gesamtanzahl der Ziffern oder Ziffern nach dem Komma
- **Type**
  - Gibt an, wie der Wert interpretiert werden soll.

Beispiel mit printf

- **Beispiel**

```
#include <stdio.h>
int main(void) {
  printf("Name:\t%s\nAlter:\t%d", "Peter", -2);
  return 0;
}
```

- **Ausgabe**

```
Name:   Peter
Alter:  -2
```


Typ in Formatstring (I)

Typ	Datentyp	Ausgabeformat
c	int	einzelnes Zeichen
d, i	int	Dezimalzahl mit Vorzeichen
o	int	Oktalzahl ohne Vorzeichen
u	int	Dezimalzahl ohne Vorzeichen
x	int	Hexadezimalzahl ohne Vorzeichen mit den Buchstaben "abcdef"
X	int	Hexadezimalzahl ohne Vorzeichen mit den Buchstaben "ABCDEF"

Typ in Formatstring (II)

Typ	Datentyp	Ausgabeformat
e, E	double	Dezimalzahl mit Vorzeichen in Exponentialschreibweise
f	double	Dezimalzahl mit Vorzeichen in Gleitkommaschreibweise
g, G	double	e- oder f-Format
p	Pointer	Adresse in der Form xxxx:yyyy
s	String	Zeichenkette. Zeichen werden solange ausgegeben, bis das erste Nullbyte erreicht wird.

Beispiel mit printf (I)

```
#include <stdio.h>
int main (void) {

    char c; float f; int i;

    i = 40;
    printf ("Integer-Ausgabe:\n");
    printf ("i = |%d|\n",i);
    printf ("i = |%5d|\n",i);
    printf ("i = |%-5d|\n",i);
    printf ("i = |%5.3d|\n",i);
    printf ("i = |%1d|\n\n",i);
```

```
Integer-Ausgabe:
i = |40|
i = |   40|
i = |40   |
i = |  040|
i = |40|
```


Beispiel mit printf (II)

```
    c = 'A';
    printf ("Zeichen-Ausgabe:\n");
    printf ("c = |%c|\n",c);
    printf ("c = |%5c|\n",c);
    printf ("c = |%-5c|\n",c);
    printf ("c = |%d|\n",c);
    printf ("c = |%0c|\n\n",c);
```

```
Zeichen-Ausgabe:
c = |A|
c = |    A|
c = |A    |
c = |65|
c = |A|
```

Beispiel mit printf (III)

```
    f = 839.21;
    printf (“Float-Ausgabe:\n");
    printf ("f = |%f|\n",f);
    printf ("f = |%10.3f|\n",f);
    printf ("f = |%10.3e|\n",f);
    printf ("f = |%-10g|\n",f);
    printf ("f = |%2.1f|\n\n",f);
    return 0;
}
```

```
Float-Ausgabe:
f = |839.210022|
f = |   839.210|
f = | 8.392e+02|
f = |839.21    |
f = |839.2|
```


Zeichenweise Ein-/Ausgabe

- **Modul**
  - #include <stdio.h>
- **Aufruf**
  - Eingabe: int getchar(void);
  - Ausgabe: int putchar(int c);
- **Beispiel**

```
char c;
c = getchar();
while (c != 0x1A) {
    putchar(c); c = getchar();
}
```


KV Betriebssysteme Operatoren

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Operatoren

- Basisoperatoren
- Inkrement- und Dekrementoperator
- Vergleichsoperatoren
- Bitoperatoren
- Logische Operatoren
- Kurzschlußauswertung
- Auswahloperator
- Weitere Operatoren

Basisoperatoren

+	Addition / monadisches positives Vorzeichen
-	Subtraktion / monadisches negatives Vorzeichen
*	Multiplikation
/	Division (auch für Integer!)
%	Divisionsrest
=	Zuweisung (kann auch in Ausdrücken vorkommen)

Inkrement- und Dekrementoperator

- **Inkrement**
  - ++i; i++;
- **Dekrement**
  - --i; i--;
- **Präfix / Postfix:**
  - „++i“ bedeutet Inkrement vor der Bewertung, „i++“ bedeutet Inkrement nach der Bewertung.
  - Vorsicht vor Sideeffects, wie in folgendem Beispiel:
    x = 1; r = (x++ * 2) + (x++ * 3);
    Welchen Wert hat x und r?

KV Betriebssysteme
Einführung in C

Vergleichsoperatoren

Operator	Ergebnis
x < y	1 falls x kleiner als y ist, sonst 0
x <= y	1 falls x kleiner oder gleich y ist, sonst 0
x > y	1 falls x größer als y ist, sonst 0
x >= y	1 falls x größer oder gleich y ist, sonst 0
x == y	1 falls x gleich y ist, sonst 0
x != y	1 falls x ungleich y ist, sonst 0

Bitoperatoren

Operator	Ergebnis
x & y	bitweises UND
x | y	bitweises ODER
x ^ y	bitweises XOR
~x	Negation (Einerkomplement)
x << y	left shift von x um y Stellen (die niederwertigen Bits werden 0)
x >> y	right shift von x um y Stellen (die höherw. Bits werden bei positiven Zahlen 0, bei neg. Zahlen hängt dies vom Compiler ab)

Logische Operatoren

Operator	Ergebnis
x && y	Der Ausdruck ergibt 1, wenn x und y nicht Null sind, sonst ergibt er 0.
x || y	Der Ausdruck ergibt 0, wenn sowohl x als auch y Null sind, sonst ergibt er 1.
!x	Der Ausdruck ergibt 1 wenn x Null ist, sonst ergibt er 0.

Beispiele

- **Bitoperatoren**
  - `2 | 193;` /* 00000010 (2) | 11000001 (193) = 11000011 (195) */
  - `2 & 193;` /* 00000010 (2) & 11000001 (193) = 00000000 (0) */
  - `2 << 3;` /* 00000010 (2) << 3 = 00010000 (16) */
  - `-16 >> 3;` /* 11110000 (-16) >> 3 = 11111110 (-2) */
  - `~ 5;` /* ~ 00000101 (5) = 11111010 (-6) */
- **Logische Operatoren**
  - `2 || 193;` /* ergibt 1, weil beide Operanden ungleich 0 sind */
  - `2 && 193;` /* ergibt 1, weil beide Operanden ungleich 0 sind */
  - `! 5` /* ergibt 0, weil der Operand ungleich 0 ist */

Kurzschlußauswertung

- **Erklärung**
  - **Bei zweistelligen logischen Operatoren wird in Abhängigkeit des ersten Operanden der zweite nicht mehr ausgewertet!**
    - Bei x && y wird y nicht mehr ausgewertet, wenn x Null ist.
    - Bei x || y wird y nicht mehr ausgewertet, wenn x ungleich Null ist.
- **Beispiel**
  - `x = 1; y = (--x) && (--x);`
- **Anwendung**
  - **Prüfung der Indexgrenzen: (i <= 100) && (a[i] == 0)**

Auswahloperator

- **Syntax**
  - `x ? y : z`
- **Bedeutung**
  - **Das Ergebnis des Ausdrucks hängt von x ab. Wenn x ungleich Null ist, ist das Ergebnis des Ausdrucks y, sonst ist es z.**
- **Beispiele**
  - `a = 10; b = 15; c = (a > b) ? a : b;`
  - `c = ‚f‘; c = ((c >= ‚a‘) && (c <= ‚z‘)) ? c - 32 : c;`

Weitere Operatoren

Operator	Name
,	Kommaoperator
()	Typkonvertierung (Cast)
&	Adressoperator
sizeof()	Größenoperator
*	Umleitungsoperator
[]	Feldindexoperator
.	Elementauswahloperator
->	Elementkennzeichnungoperator

KV Betriebssysteme
Ausdrücke

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Typkonvertierung (I)

- **Implizite Typkonvertierung (Reihenfolge!)**

Operand 1	Operand 2	Ergebnis
long double	beliebig	long double
double	beliebig	double
float	beliebig	float
unsinged long	beliebig	unsigned long
long	unsigned int	unsigned long
long	beliebig	long
unsigned int	beliebig	unsigned int
beliebig	beliebig	int

Typkonvertierung (II)

- **Explizite Typkonvertierung (type-cast)**
  - **Der sogenannte cast-Operator erlaubt das explizite Umwandeln von Datentypen.**
- **Beispiele:**
  - **int i; char c;**

```
c = 'A';        /* keine Umwandlung notwendig */
i = c;          /* implizite Umwandlung */
i = 'B';        /* implizite Umwandlung */
c = (char) i;   /* explizite Umwandlung */
                /* (wäre hier nicht notwendig!) */
```


Typkonvertierung (III)

- **Integer-Werte, die auf char umgewandelt werden, verlieren ihre höherwertigen Bits (genauso bei long auf int).**
- **Achtung: nur Konvertierungen von unsigned char in int resultieren in Werten von 0..255. Konvertierungen von char in int resultieren in Werten von -128..127.**

Operatorreihenfolge (I)

- **Der weiter oben in der Tabelle (folgende Seite) stehende Operator hat Vorrang.**
- **Bei zwei in der gleichen Zeile stehenden Operatoren entscheidet das Feld Zusammenfassung (ZF): Bei „L“ hat der linke Operator des Ausdrucks (linksassoziativ), bei „R“ der rechte (rechtsassoziativ) Vorrang .**
- **Ausdrücke mit logischen Operatoren (&& oder ||) werden von links nach rechts ausgewertet bis ihr Wahrheitswert klar ist (Kurzschlußauswertung)**

KV Betriebssysteme
Einführung in C

Operatorreihenfolge (II)

P	ZF	Operator
01	L	() [] -> .
02	R	! ~ ++ -- - (cast) * & sizeof
03	L	* / %
04	L	+ -
05	L	<< >>
06	L	< <= > >=
07	L	== !=
08	L	&

P	ZF	Operator
09	L	^
10	L	|
11	L	&&
12	L	||
13	R	?:
14	R	= += -= ...
15	L	,

Operatorreihenfolge (III)

- **Beispiel**
  - `int x, y, z;` — x y z Ergebnis
  - `x = y = z = 1;` — 1 1 1
  - `++x || ++y && ++z;` — ? ? ? ?
- **Lösungsweg**
  - `(++x) || ((++y) && (++z))`
  - `2 || ((++y) && (++z))` — 2
  - `TRUE || ((++y) && (++z))` — 2 1 1 TRUE(1)
- **Anmerkung:**
  - && und || Auswertung von links nach rechts!

Operatorreihenfolge (IV)

- **Beispiele**

```
int x, y, z;                    x  y  z  Ergebnis
x = y = z = 1;                  1  1  1
++x && ++y || ++z;              2  2  1   TRUE

int x, y, z;                    x  y  z  Ergebnis
x = y = z = 1;                  1  1  1
++x && ++y && ++z;              2  2  2   TRUE

int x, y, z;                    x  y  z  Ergebnis
x = y = z = -1;                -1 -1 -1
++x && ++y || ++z;              ?  ?  ?      ?
```


Operatorreihenfolge (V)

- **Beispiele**

```
int x, y, z;                    x  y  z  Ergebnis
x = y = z = -1;                -1 -1 -1
++x || ++y && ++z;              ?  ?  ?      ?

int x, y, z;                    x  y  z  Ergebnis
x = y = z = -1;                -1 -1 -1
++x && ++y && ++z;              ?  ?  ?      ?
```


Abkürzende Schreibweisen

- Die Anweisungsfolge „int i; i = 3;“ kann vereinfacht werden zu „int i = 3;“.
- Decken sich Ziel- und Quell-Operand, wie bei „i = i & 1; i = i - 2;“ so kann man vereinfachend schreiben: „i &= 1; i -= 2;“
- Abfragen können mit Zuweisungen verbunden werden:

```
c = getchar ();
while (c != EOF) {
    putchar (c);
    c = getchar ();
}
```

```
while ((c = getchar ()) != EOF)
    putchar (c);
```


KV Betriebssysteme
Ablaufstrukturen

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

KV Betriebssysteme
Einführung in C

Ablaufstrukturen

- **Anweisungen**
  - Ausdrucksanweisung, z.B.: ausdruck;
  - Leere Anweisung, z.B.: ;
- **Block**
  - Zusammenfassung mehrerer Anweisungen
  - Variablendeklaration am Beginn möglich
- **Verzweigungen**
  - if, switch
- **Schleifen**
  - for, while, do

Block

```
int main(void) {
   int a, b;

   a = 1; b = 2;
   printf("Main, a = %d, b = %d\n", a, b);
   {
      /* -- Das ist ein neuer Block -- */
      int a, c;

      a = 10; c = 20;
      printf("Block, a = %d, c = %d\n", a, c);
   }
   printf("Main, a = %d, b = %d\n", a, b);
   return 0;
}
```

Programmausgabe

```
Main, a = 1, b = 2

Block, a = 10, c = 20
Main, a = 1, b = 2
```


If-Statement

- **Syntax**
  - if (ausdruck) ThenAnweisung else ElseAnweisung
- **Bedeutung**
  - Wenn der Ausdruck ungleich 0 ist, wird die „Then-Anweisung“ ausgeführt, sonst die „ElseAnweisung“.
  - Bei mehreren if-Statements gehört else immer zum letzten if. Es ist ratsam, Blöcke zu verwenden.
- **Beispiel**

```
if (i & 1) {
    printf ("ungerade\n"); i = i & 0xFE;
} else
    printf ("gerade\n");
```


Switch-Statement (I)

- **Syntax**

```
switch (Ausdruck) {
    case Konstante: ....; break;
    ....
    default: ....;
}
```

- **Bedeutung**
  - Fallunterscheidung.
  - Ab dem selektierten Zweig werden auch alle nachfolgenden Zweige abgearbeitet, bis ein break-Statement erreicht wird.

Switch-Statement (II)

- **Beispiel**

```
switch (i) {
  case 0: printf("Null\n"); break;
  case 1: printf("Eins\n"); break;
  case 2: printf("Zwei\n"); break;
  case 3: printf("Drei\n"); break;
  case 4: printf("Vier\n"); break;
  case 5: printf("Fünf\n"); break;
  case 6: printf("Sechs\n"); break;
  default: printf("Sonst\n");
}
```


While-Schleife

- **Syntax**
  - while (Bedingung) SchleifenBlock
- **Ausführung**
  - Bedingung prüfen, SchleifenBlock, Bedingung ...
- **Beispiel**

```
i = 0;
while (i <= 255) {
    printf ("%d  %c\n", i, i);
    i++;
}
```

KV Betriebssysteme
Einführung in C

For-Schleife

- **Syntax**
  - for (Initialisierung; Bedingung; Zähler)
    SchleifenBlock
- **Ausführung**
  - Initialisierung, Bedingung prüfen, SchleifenBlock, Zähler, Bedingung prüfen, SchleifenBlock, ...
- **Beispiel**

```
for (i = 0; i <= 255; i++) {
    printf ("%d  %c\n", i, i);
}
```


Do-Schleife

- **Syntax**
  - do SchleifenBlock while (Bedingung);
- **Ausführung**
  - SchleifenBlock, Bedingung prüfen, SchleifenBlock, ...
- **Beispiel**

```
i = 0;
do {
    printf ("%d  %c\n",i,i);
    i++;
} while (i <= 255);
```


Break

- **Bedeutung**
  - Die break-Anweisung kann nur innerhalb von Schleifen oder einer switch-Anweisung verwendet werden und dient zur Beendigung der Schleife bzw. der Anweisung. break terminiert immer nur die innerste Schleife oder switch-Anweisung!
- **Programmierstil**
  - Die break-Anweisung sollte nur für switch-Anweisungen und nicht für das Verlassen einer Schleife verwendet werden!

Continue

- **Bedeutung**
  - Die continue-Anweisung kann nur innerhalb von Schleifen verwendet werden. In while- und do-Schleifen überträgt sie die Steuerung an die Testbedingung, in einer for-Schleife an den Inkrement-Ausdruck. Auch continue wirkt sich immer auf die innerste einschließende Schleife aus.
- **Programmierstil**
  - continue sollte überhaupt nie verwendet werden!

Pythagoräische Tripel

- **Problem:**
  - Gesucht sind ganzzahlige Tripel (a,b,c) für die gilt:

$$a^2 + b^2 = c^2$$

- **Beispiel:**
  - Source\Tripel.c

KV Betriebssysteme
Preprocessor

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

KV Betriebssysteme
Einführung in C

Preprocessor

- In C wird vor dem eigentlichen Compiler ein sogenannter Präprozessor gestartet.
- Der Präprozessor wertet bestimmte Kommandos im Code aus und manipuliert den Quellcode. Der C-Compiler selbst bekommt bereits modifizierten Quellcode.
- Präprozessor-Anweisungen sind durch ein vorangestelltes # zu erkennen.
- Präprozessor-Anweisungen haben am Ende keinen Strichpunkt!

#define, #undef (I)

- Bedeutung
  - Mit #define können in C Konstanten nachgeahmt werden. Durch #define wird der Präprozessor angewiesen alle Vorkommen des definierten Symbols durch den angegebenen Text zu ersetzen.
  - Texte innerhalb von Zeichenketten werden durch den Preprocessor nicht verändert.
- Syntax
  - #define symbol text
  - #undef symbol

#define, #undef (II)

- Beispiel
  - #define MAX 100
  - Definiert das Symbol MAX und weist den Preprozessor an, alle Vorkommen von MAX durch den Wert 100 zu ersetzen. Ausgenommen davon sind Zeichenketten und Kommentare.
- TRUE und FALSE sind nicht definiert. Meist werden sie mit einem #define folgendermaßen deklariert.
  - #define FALSE 0
  - #define TRUE 1
  - #define TRUE (!FALSE)

#define, #undef (III)

Quellcode

```
#define  MAX    100
#define  FALSE  0
#define  TRUE   (!FALSE)

int tiles[MAX];
...
for (i = 1; i <= MAX; i++)
   tiles[i] = TRUE;
printf("MAX=%d", MAX);
```

Nach Preprocessor

```
int tiles[100];
...
for (i = 1; i <= 100; i++)
   tiles[i] = (!0);
printf("MAX=%d", 100);
```


Konstanten

- 2 Möglichkeiten
  - #define
  - Schlüsselwort const
- Schlüsselwort const
  - Variablen können vor Änderung geschützt werden.
  - „const“ wird vor allem für Parameter verwendet.
  - Es handelt sich nicht um Konstanten im klassischen Sinn, weil Speicherplatz angelegt wird.
- Beispiele
  - const int a = 1;
  - #define K 2

#include

- Bedeutung
  - Inkludiert die angegebene Datei.
- Syntax
  - #include <xxx.h>
  - #include "xxx.h"
- Beispiel
  - #include <stdio.h>
  - #include "module.h"
  - Inkludiert die Datei stdio.h aus dem INCLUDE-Verzeichnis des Compilers und die Datei module.h aus dem aktuellen Verzeichnis.

Importieren von Modulen (I)

- Geschieht durch das Inkludieren von Header-Files (Extension .h), in denen die Schnittstelle eines Moduls definiert ist. Das Modulkonzept wird später noch genauer erklärt.
- Um zum Beispiel das Modul STDIO (Standard Input/Output) zu importieren, muß man am Beginn des Programms folgende Zeile aufnehmen.
  - #include <stdio.h>

Importieren von Modulen (II)

```
/* -- 100 Mal Würfeln -- */
#include <stdlib.h>
#include <stdio.h>

int main(void) {
   int i;
   srand(0);                    /* Zufallszahlen init. */
   printf("100 Mal Würfeln:\n");
   for (i = 0; i < 100; i++)
      printf("%2d", rand() % 6 + 1);
   return 0;
}
```


#ifdef, #ifndef, #else, #endif (I)

- Bedeutung
  - Mit diesen Preprocessorbefehlen kann eine bedingte Compilierung durchgeführt werden. Das heißt Teile des Codes können ausgeblendet werden.
- Syntax
  - #ifdef symbol /* bzw. #if defined(symbol) */
  - #ifndef symbol /* bzw. #if !defined(symbol) */
  - #endif
  - if-Abfrage auf Definition bzw. Nicht-Definition; Code nach ifdef und vor endif wird nur übersetzt, falls symbol definiert ist.

#ifdef, #ifndef, #else, #endif (II)

- Beispiel
  - char version[100];

```
#ifdef WIN2000
   strcpy(version, "Windows 2000");
#end
#ifdef WINNT
   strcpy(version, "Windows NT 4.0");
#end
#ifdef WIN98
   strcpy(version, "Windows 98");
#end
```


KV Betriebssysteme
Array

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Array - Modell

KV Betriebssysteme
Einführung in C

Syntax

- **Deklaration**
  - **BaseType Identifier "["Elements"]" {"["Elements"]"}**
- **Beispiele**
  - **int werte[100]; char alphabet[256];**
- **Zugriff**
  - **ArrayName[index]**
- **Beispiele**
  - **c = alphabet[255]; i = werte[1];**
  - **aber: c = alphabet[256]; /* ?? */**

Regeln

- **Die Anzahl der Elemente muß bei der Definition durch einen konstanten Ausdruck bestimmt sein (auch in Verbindung mit dem sizeof-Operator)**
- **Es gibt zur Laufzeit keine Möglichkeit die Anzahl der Elemente eines Arrays zu bestimmen! (siehe aber sizeof-Operator)**
- **Der erste Index eines Arrays ist immer 0! Die zehn Elemente eines Arrays test[10] sind also test[0], test[1], ..., test[9]!**
- **Zuweisung eines Arrays an ein anderes resultiert NICHT in einem Kopieren der Elemente.**

Initialisierung

- **char hex[16];**
- **hex[0] = '0'; hex[1] = '1'; ... hex[15] = 'F';**

- **char hex[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};**

- **int values[3] = {1, 2}; /* Rest wird 0 gesetzt! */**

Parameter (I)

- **Arrays werden als Parameter entgegen den C-Gepflogenheiten Call-by-Reference übergeben (eigentlich: Call-by-Value Übergabe der Array-Adresse!)**
- **Wenn es nicht dem Anlegen von Speicherplatz für einen Array dient, so kann die Größe der ersten Dimension offen gelassen werden (Open Array), z.B. bei Arrays als Funktionsparametern**

Parameter (II)

```
void ArrayAenderung(int werte[]) {
    werte[1] = 5;
}

void ArrayTest(void) {
    int werte[11] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printf("%d", werte[1]);          /* 1 */
    ArrayAenderung(werte);
    printf("%d", werte[1]);          /* 5 */
}
```


Parameter (III)

```
int writeName (char name[]) {
    /* 'name' .. offenes, eindimensionales Array */
}

int getPairs (int p[][2]) {
    /* 'p' .. offenes, zweidimensionales Array */
}
```

KV Betriebssysteme
Einführung in C

Array - Mehrdimensional

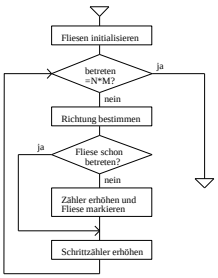
- Mehrdimensionale Arrays werden so gespeichert, daß sich jeweils der letzte Index am schnellsten ändert ("Reihen-Spalten-Ordnung")
- Bei Deklaration und Zugriff muß jede Dimension in eigene Klammern "[" und "]" gestellt werden ([i1][i2], nicht [i1,i2])!
- Beispiel
  - int pairs[4][3] = {{1,2},{3,4}}; /* Rest wird 0 */
  - Speicherabbild

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	..	[3][2]

Array - Beispiel (I)

- Aufgabenstellung
  - Ein Käfer torkelt in einem rechteckigen Raum mit N*M Fliesen. In jedem Zeitschritt tritt er auf irgendeine der Nachbarfliesen (wenn er an die Wand stößt, bleibt er auf seiner Fliese).
  - Gesucht ist die Anzahl der Schritte, die der Käfer braucht, um alle Fliesen mindestens einmal zu betreten.
- Quellcode
  - Source\Kaefer.c

Array - Beispiel (II)



KV Betriebssysteme
Datenstrukturen

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Strukturen (I)

- Bedeutung
  - Zusammenfassung verschiedener Datentypen
  - Jedes Feld wird durch einen Namen referenziert.
- Deklaration
  - struct [tag] { member-list } [declarators];
  - [struct] tag declarators;
  - „tag“ ist der Name eines Datentyps.
  - „declarators“ sind die Namen von Variablen.
- Zugriff
  - Variablenname "." Feldname

Strukturen (II)

- Beispiel

```
struct TADDRESS {
   char  szStreet[30];
   int   iNumber;
   int   iZip;
   char  szCity[20];
};

struct TADDRESS aAddress =
   {"Altenbergerstraße",  69,  4040, "Linz"};

aAddress.iZip = 4020;
```

KV Betriebssysteme
Einführung in C

Strukturen (III)

- **Beispiel zu Arrays und Strukturen**
  - struct TADDRESS {
    char szStreet[30];
    int iNumber;
    int iZip;
    char szCity[20];
    };

    struct TADDRESS address[20];

    address[0].iZip = 1010;

Strukturen (IV)

- **Operationen mit Strukturen**
  - Übergabe an Funktion (Call by value)
  - Rückgabe aus Funktion
  - Zuweisung an eine Struktur (Kopie)
  - Anwendung von Adress- und sizeof-Operator
- **Alignment**
  - Ausrichtung der Daten im Speicher
  - struct { char c; long l; } TEST;
    printf("%ld\n", sizeof(struct TEST));
  - Dieses Beispiel kann 5, 6 oder 8 liefern.

Bitfelder

- **Bedeutung**
  - Aufteilung eines Feldes in einzelne Bits / Bitgruppen
- **Beispiel**
  - struct CELL // Declare CELL bit field
    {
    unsigned character : 8; // 00000000 ????????
    unsigned foreground : 3; // 00000??? 00000000
    unsigned intensity : 1; // 0000?000 00000000
    unsigned background : 3; // 0???0000 00000000
    unsigned blink : 1; // ?0000000 00000000
    } screen[25][80]; // Array of bit fields

Unions

- **Bedeutung**
  - Zusammenfassung verschiedener Datentypen
  - Jedes Feld wird durch einen Namen referenziert.
  - Alle Felder belegen denselben Speicherplatz!
- **Syntax**
  - union [tag] { member-list } [declarators];
  - [union] tag declarators;
- **Beispiel**
  - union TEST { short sZahl; long lZahl; };
    uTest.lZahl = 65537;
    printf("%d\n", uTest.sZahl); /* ?? */

Aufzählungen

- **Bedeutung**
  - Deklaration einer vordefinierten Menge von Werten
  - Jeder Wert bekommt einen Namen.
  - Der erste Wert ist 0.
- **Syntax**
  - enum [tag] {enum-list} [declarator];
  - enum tag declarator;
- **Beispiele**
  - enum tage { mon, die, ... } woche;
  - enum tage { mon = 1, die, ... } woche;
  - enum boolean { FALSE, TRUE };

Typdeklaration

- **Bedeutung**
  - Für einen Typ wird ein Synonym vergeben.
- **Syntax**
  - typedef type-declaration synonym;
- **Beispiele:**
  - typedef struct {
    char strasse[30];
    char ort[20];
    } TADRESSE;
    TADRESSE adresse;
  - typedef int (*funcptr)();

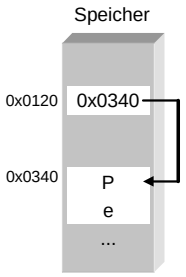
KV Betriebssysteme
Einführung in C

KV Betriebssysteme
Zeiger

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Zeiger - Modell



Zeiger (I)

- **Deklaration eines Zeigers:**
  - **BaseType "*" Identifier**
- **Beispiele:**
  - **int *zeiger1;**
  - **int *zeiger2, wert1; /* 'wert 1' ist kein Zeiger! */**
- **Adresse einer Variablen:**
  - **&-Operator**
- **Beispiele:**
  - **zeiger1 = &wert1;**

Zeiger (II)

- **Derefenzierung eines Zeigers**
  - ***-Operator**
- **Beispiel**
  - **int iValue;**
    int *piValue;

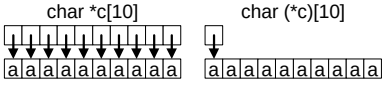
    iValue = 100;
    ***piValue = 200;**
    printf("%d\n", Value); /* ?? */
    printf("%X\n", piValue); /* ?? */
    printf("%d\n", *piValue); /* ?? */

Zeigerarithmetik

- **Addition zu Zeigern entspricht Adress-Addition von n*sizeof(element)**
  - **int *zeiger, wert; zeiger = &wert; zeiger++;**
- **Subtraktion zu Zeigern entspricht Adress-Subtraktion von n*sizeof(element)**
  - **int *zeiger, wert; zeiger = &wert; zeiger--;**
- **Differenzbildung von zwei Zeigern auf Arrays liefert die Anzahl der dazwischenliegenden Elemente**
  - **int *zeiger1, *zeiger2; ...;**
    printf („%d“, zeiger2 - zeiger1);

Zeiger und Arrays

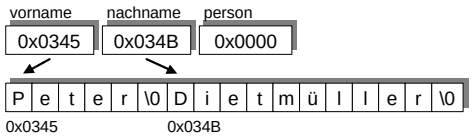
- **Verwendung von arrayName entspricht der Adresse des ersten Arrayelementes.**
  - **int werte[10]; /* werte == &werte[0] */**
- **Unterschied:**
  - **char *c[10]; … array of pointers**
  - **char (*c)[10]; … pointer to array**



KV Betriebssysteme
Einführung in C

Zeiger und Zeichenketten

- In C sind Zeichenketten Character-Arrays, die mit einem Nullbyte abgeschlossen werden.
  - char vorname[] = „Peter“;
  - char *nachname = „Dietmüller“;
  - char *person;



Zeiger - Beispiele (I)

```
/* -- Unterschiedliche Deklarationen -- */
#include <stdio.h>
int main (void) {
   char *a[10],*(b[10]),(*c)[10];
   printf("%d %d %d\n",
      sizeof(a),sizeof(b),sizeof(c));
      /* Ergebnis: 40, 40, 4 */
   return 0;
}
```


Zeiger - Beispiele (II)

- Anhängen eines Zeichens an eine Zeichenkette:

```
while (*c) c++;
*c++='!';
*c=0;
```

- Dasselbe mit einer Funktion aus <string.h>

```
strcat(c, "!");
```


Zeiger - Beispiele (III)

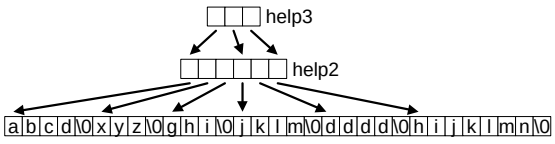
- Zeigerarithmetik:

```
int i, *p, fib[]= {1,1,2,3,5,8};

/* -- Ausgabe des Arrays in drei Varianten -- */
for (i = 0; i <= 5; i++) printf("%d\n", fib[i]);
for (p = &fib[0]; p <= &fib[5]; p++)
      printf("%d\n", *p);
for (p = fib; p <= fib+5; p++) printf("%d\n", *p);
```


Zeigerarithmetik - Beispiel (I)

```
char *string =     "abcdefghijklmnopqrstuvwxyz";
char *help2[] =    { "abcd", "xyz", "ghi",
                     "jklm", "dddd", "hijklmn" };
char **help3[] =  { help2, help2+3, help2+5 };
```



Zeigerarithmetik - Beispiel (II)

```
int main (void) {

   char *help,***help4;

   help = string+10;
   printf("%s\n", help);       /* klmnopqrstuvwxyz */
   printf("%s\n", help-3);     /* hijklmnopqrstuvwxyz */
   printf("%s\n\n", help+5);   /* pqrstuvwxyz */
```

Zeigerarithmetik - Beispiel (III)

```
printf("%s\n", *(help2+3));      /* jklm */
printf("%s\n", help2[5]);        /* hijklmn */
printf("%s\n\n", help2[1]+2);    /* z */

printf("%s\n", **help3);         /* abcd */
printf("%s\n", *help3[1]);       /* jklm */
printf("%s\n\n", **(help3+2));   /* hijklmn */
```

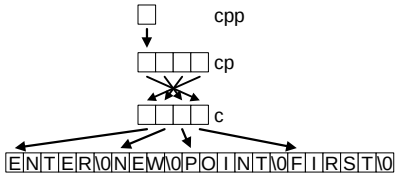

Zeigerarithmetik - Beispiel (4)

```
help4 = help3+1;
printf("%s\n", *++*help4);       /* dddd */
printf("%s\n", *++(*help4++));   /* hijklmn */
printf("%s\n", *--*help4+2);     /* dd */

return 0;
}
```


Zeigersalat (I)

```
char *c[]={"ENTER","NEW","POINT","FIRST"};
char **cp[]= {c+3,c+2,c+1,c};
char ***cpp= cp;
```



Zeigersalat (II)

```
int main (void) {

   printf("%s",**++cpp);          /* POINT */
   printf("%s",*--*++cpp+3);      /* ER */
   printf("%s",*cpp[-2]+3);       /* ST */
   printf("%s\n",cpp[-1][-1]+1);  /* EW */
   return 0;
}
```


Call by Reference (I)

- C unterstützt nur die Call-by-Value Übergabe. Durch Zeiger kann aber Call-by-Reference simuliert werden:

```
void CallByValue (int i) {
    i=1;
}
void CallByReference (int *i) {
    *i=2;
}
```


Call by Reference (II)

```
int main (void) {

  int n = 3;
  CallByValue(n);
  printf("%d\n", n);          /* 3 (nicht 1!!!) */
  CallByReference(&n);
  printf("%d\n", n);          /* 2 */

  return 0;
}
```

KV Betriebssysteme
Dynamische Speicherverwaltung

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Funktionen

- **Funktionen aus <stdlib.h>:**
  - **void *malloc (size_t size);**
    - Legt size Bytes an und gibt einen Zeiger auf das erste Byte zurück.
  - **void *calloc (size_t nitems, size_t size);**
    - Legt nitems*size Bytes an und retourniert Zeiger.
  - **void free (void *block);**
    - Gibt den mit malloc/calloc reservierten Speicher wieder frei.
  - **void *realloc (void *block,size_t size);**
    - Ändert die Größe eines reservierten Speicherbereichs auf size Bytes und retourniert neuen Zeiger (der alte Speicherinhalt bleibt bestehen, der Inhalt von zusätzlichem Speicher ist undefiniert).

Beispiele

- **Dynamische Liste**
  - **Eine Liste von Elementen, die zur Laufzeit wachsen und schrumpfen kann.**
  - **Source\Liste.h, Source\Liste.c, Source\ListTest.c**
- **Dynamisches Array**
  - **Die Größe eines Arrays muß in C zum Zeitpunkt der Deklaration feststehen. Dieses Beispiel zeigt ein Array, das zur Laufzeit wachsen kann.**
  - **Source\Array.h, Source\Array.c**

KV Betriebssysteme
Programmargumente

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Programmargumente

- **Argumente an ein C-Programm werden als Zeichenketten an die Funktion main übergeben. Die vollständige Deklaration von main lautet: int main (int argc, char **argv);**
  - **argc** Anzahl der Argument-Strings [0..argc-1], mindestens 1
  - **argv** Argument-Strings, Null-terminiert (der erste Eintrag ist der Programmname)
  - **Der Rückgabewert kann in einer Batch-Datei über ERRORLEVEL abgefragt werden.**

Beispiel

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv) {

    int i;

    for (i = 0; i < argc; i++)
        printf("Argument %d: %s\n", i, *argv++);
    printf("%s\n", getenv("PATH"));
    return 0;
}
```

KV Betriebssysteme
Einführung in C

KV Betriebssysteme
Dateien

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Dateioperationen

- **Modul**
  - **#include <stdio.h>**
- **Funktionen**
  - **Öffnen: FILE* fopen(char name[], char mode[]);**
  - **Lesen: int fread(buffer[], int size, int n, FILE*);**
  - **Schreiben: int fwrite(buffer[], int size, int n, FILE*);**
  - **Schließen: int fclose(FILE*);**
  - **Position.: int fseek(FILE*, long offset, int whence);**

Dateimodus (I)

"r" read only
"r+" read and write
"w" write only (create new file)
"w+" create new file and allow read
"a" append
"a+" append + read allowed from end of old file
"b" Binärmodus (kombinierbar, z.B.: „r+b“)
„t“ Textmodus (kombinierbar, z.B.: „w+t“), CR LF -> LF, 0x26 beendet Verarbeitung

Dateimodus (II)

	r	w	a	r+	w+	a+
Datei muß existieren	x	-	-	x	-	-
Datei wird auf Null gekürzt	-	x	-	-	x	-
Datei kann gelesen werden	x	-	-	x	x	x
Datei kann beschrieben werden	-	x	x	x	x	x
Schreiben nur ab Ende der Datei	-	-	x	-	-	x

Standarddateien (I)

- **Drei Datei-Handles stehen zur Verfügung, ohne daß dafür eine Datei geöffnet oder geschlossen werden muß (definiert in stdio.h):**
  - **stdin (read-only)**
    - Standard-Eingabe, normalerweise Eingaben über Tastatur
  - **stdout (write-only)**
    - Standard-Ausgabe (Bildschirm, Fenster)
  - **stderr (write-only)**
    - Ausgabe von Fehlermeldungen, benützt meist dasselbe Ausgabemedium wie stdout

Standarddateien (II)

- **Die meisten Ein- und Ausgabefunktionen gibt es auch für Dateien, z.B.**
- **printf - fprintf**
  - **int printf (char *format, …);**
  - **int fprintf (FILE *stream, char *format, …);**
- **putc - fputc**
  - **int putc (int c);**
  - **int fputc (int c, FILE *stream);**
- **aber putchar - fputchar**
  - **int putchar (int c);**
  - **int fputchar (int c); /* kein Unterschied! */**

Dateiumleitung

- stdin ist unter UNIX und DOS an die Tastatur gebunden, stdout und stderr an das Konsolenfenster. Beim Start eines Programmes können diese auch auf Dateien umgeleitet werden:
- stdio von Datei lesen
  - program < infile.txt
- stdout auf Datei schreiben
  - program > outfile.txt
- stderr auf Datei schreiben (nicht unter DOS)
  - program 2> errfile.txt

KV Betriebssysteme
Makros

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Makros (I)

- Syntax
  - #define Symbol [(Symbol1, ... , SymbolN)] Text
  - Definiert ein Symbol mit N Parametern. Die Parameter können innerhalb des Textes werden werden. Der Präprozessor ersetzt das Symbol im Quellcode durch den Text, wobei die Formalparameter durch die aktuellen Parameter ersetzt werden.
- Beispiel
  - #define Min(a,b) (a<b ? a : b)
  - z = Min(x,y); ==> z = (x<y ? x : y);

Makros (II)

- Beispiel:

```
#define Min(a,b) (a<b ? a : b)

int x, y, z;

Min(x,y) = z;          /* expandiert: (x<y ? x : y) = z; Geht nicht! */
z = Min(x+1,y-2);      /* z = (x+1<y-2 ? x+1 : y-2); */
z = Min(x&3,y&3); /* z = (x&3<y&3 ? x&3 : y&3); Fehler! */
```


Makros (III)

```
#define Min(a,b) (a<b ? a : b)

int x, y, z;

x = 103; y = 202;

z = Min(x&3,y&3);
```

Operatorreihenfolge (Auszug)	
< <= > >=	von links
&	von links
?:	von rechts
= += -= ...	von rechts

```
/* z = (x&3<y&3 ? x&3 : y&3) */
/* z = (x&1&3 ? 3 : 2) */
/* z = (1 ? 3 : 2) */
/* z = 3 */
```


Makros (IV)

- Daher sollen in einem Makro alle Parameter geklammert werden.
  - #define Min(x,y) ((x) < (y) ? (x) : (y))
  - z = Min(x&3, y&3);
  - /* z = ((x&3) < (y&3) ? (x&3) : (y&3)); */
- Im Zusammenhang mit Inkrement und Dekrement-Operatoren können auch unerwünschte Seiteneffekte auftreten.

Makros (V)

```
static int Square (int x) {
   return x*x;
}

...
x=2;
y=Square(x++);
/* y=4 x=3 */

x=2;
y=Square(++x);
/* y=9 x=3 */
```

```
#define SQUARE(x) ((x)*(x))


x=2;
y=SQUARE(x++);
/* exp.: y=((x++)*(x++)); */
/* y=4 x=4 (!!!) */

x=2;
y=SQUARE(++x);
/* exp.: y=((++x)*(++x)); */
/* y=16 x=4 (!!!) */
```


Makros (VI)

- **Vorsicht!**
  - Man kann nicht immer davon ausgehen, daß Funktionen in Bibliotheken wirklich als Funktionen und nicht als Makros implementiert sind!
- **Beispiel (aus Visual-C)**
  - #define tolower(_c) ((isupper(_c)) ? _tolower(_c) : (_c))
- **Problem**
  - lower = tolower(c++);
  - Ergibt einen unerwünschten Nebeneffekt, weil der Ausdruck „c++“ zweimal ausgewertet wird!

KV Betriebssysteme
Module

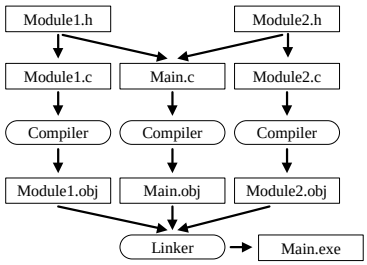
DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Elemente eines Moduls

- **Schnittstellenbeschreibung**
  - „Header-File“, hat meist die Endung „.h“
- **Implementierung**
  - „C-File“, hat meist die Endung „.c“
- **Implementierung besteht aus**
  - 1. Inkludieren der benötigten Module
  - 2. Definition der Datentypen
  - 3. Globale Variablen
  - 4. Forward Deklarationen / Funktionsprototypen
  - 5. Funktionen (keine Verschachtelung möglich!)

Getrennte Übersetzung



Was gehört in eine Header-Datei?

- **Definitionen von Datentypen und #define-Konstanten, die auch in anderen Modulen bekannt sein sollen.**
- **Prototypen von Funktionen, die von anderen Modulen aus aufgerufen werden sollen.**
- **Makros, die auch in anderen Modulen Verwendung finden sollen.**
- **Deklaration von globalen Variablen als „extern“, die in anderen Modulen verwendet werden dürfen.**

Was gehört in eine Header-Datei?

- **Problem**
  - Header-Dateien können mehrfach in einen Quellcode inkludiert werden. Daher muß verhindert werden, daß der Compiler die Deklarationen mehr als einmal zu Gesicht bekommt. Davor wird der gesamte Inhalt der Header-Datei mittels bedingter Compilation geschützt.
- **Beispiel**
  - #ifndef _modulname
    #define _modulname
    … Dateiinhalt
    #endif

Was gehört NICHT in eine Header-Datei?

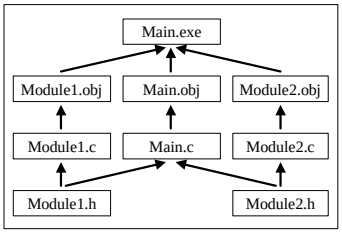
- **Anweisungen, die Code erzeugen (also kein Programmcode)**
- **Anweisungen, die Speicher reservieren (also keine Variablendeklarationen)**

KV Betriebssysteme Makefile

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Abhängigkeiten



Beschreibungsblock

- **Bedeutung**
  - Ein Beschreibungsblock (Description Block) ist eine Zeile, die Abhängigkeiten definiert:
- **Syntax**
  - targets... : dependents...
    commands...
- **Beispiel**
  - main.obj: main.c module1.h module2.h
    cl /c main.c
  - main.exe: main.obj module1.obj module2.obj
    link /out:main.exe main.obj module1.obj module2.obj

Schlussregeln (Inference Rules)

- **Bedeutung**
  - Regeln, die mit Dateinamenserweiterungen arbeiten. (Motto: „.c-Dateien werden folgendermaßen in .obj-Dateien übersetzt ...“)
  - „.SUFFIXES“ gibt die verwendeten Erweiterungen an.
- **Syntax**
  - .VonErw.ZuErw:
    Befehle
- **Beispiel**
  - .c.obj:
    cl -c $*.c

Dateinamen - Makros

$@	Vollständiger Name des aktuellen Ziels (Pfad, Basisname, Erweiterung) wie aktuell angegeben.
$$@	Vollständiger Name des aktuellen Ziels (Pfad, Basisname, Erweiterung) wie aktuell angegeben. Nur gültig als abhängige Datei in einer Abhängigkeit.
$*	Pfad und Basisname des aktuellen Ziels ohne Dateinamenerweiterung.
$**	Alle abhängigen Dateien des aktuellen Ziels.
$?	Alle abhängigen Dateien mit einer späteren Zeitmarkierung als das aktuelle Ziel.
$<	Abhängige Dateien mit einer späteren Zeitmarkierung als das aktuelle Ziel. Nur gültig in Befehlen in Schlussregeln.

Befehlsblock

- **Bedeutung**
  - Nach einem Beschreibungsblock oder einer Schlussregel kann ein Befehlsblock stehen, der aus mehreren Befehlen bestehen kann.
- **Verschiedene Varianten**
  - **Befehl** „Normaler“ Befehl
  - **@Befehl** Verhindert die Anzeige des Befehls.
  - **–[Zahl]Befehl** Deaktiviert die Fehlerprüfung.
  - **!Befehl** Führt den Befehl für jede abhängige Datei aus.

Makefile

```
ALL: "Beispiele.exe"

"Beispiele.exe": "Beispiele.obj"
   link.exe kernel32.lib user32.lib \
   /subsystem:console /incremental:no Beispiele.obj

.c.obj:
   cl /MTd /W3 /D "WIN32" /D "_DEBUG" \
   /D "_CONSOLE" /c $<
```


NMAKE

- **Syntax**
  - NMAKE [Option..][Makros..][Ziele..][@Befehlsdatei..]
- **Optionen (Auszug):**
  - **-f Datei** Name des Makefiles (sonst Datei „Makefile“ im aktuellen Verzeichnis)
  - **-a** Erstelle alle Ziele, auch wenn sie nicht veraltet sind.

Beispiel

E:\..\Source>nmake -f Beispiele.mak -a

```
Microsoft (R) Program Maintenance-Dienstprogramm: Version 1.62.7022
Copyright (C) Microsoft Corp 1988-1997. Alle Rechte vorbehalten.

        cl /MTd /W3 /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /c "Beispiele.c"
Optimierender Microsoft (R) 32-Bit C/C++-Compiler, Version 11.00.7022, fuer x86
Copyright (C) Microsoft Corp 1984-1997. Alle Rechte vorbehalten.

Beispiele.c
Beispiele.c(30) : warning C4101: 'z' : Unreferenzierte lokale Variable
        link.exe kernel32.lib user32.lib /subsystem:console /incremental:no Beispiele.obj
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.
```

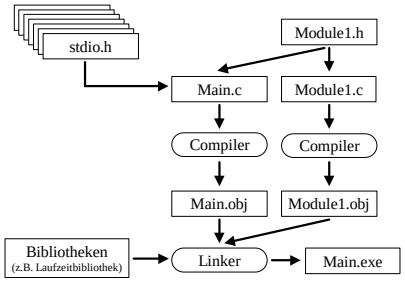

KV Betriebssysteme
Bibliotheken

DI. Dr. Peter René Dietmüller

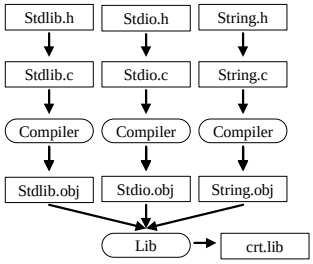
CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

KV Betriebssysteme
Einführung in C

Bibliotheken



Bibliotheken



Bibliotheksprogramm LIB

- **Bedeutung**
  - **„LIB“ erzeugt und verwaltet Bibliotheken.**
- **Syntax**
  - **LIB [Optionen...] Dateien...**
- **Optionen**
  - **/LIST** **Anzeige über die Ausgabebibliothek**
  - **/OUT:Dateiname** **Setzt den Standarddateinamen**
  - **/REMOVE:Objekt** **Entfernt das angegebene Objekt**

KV Betriebssysteme
C-Laufzeitbibliothek

DI. Dr. Peter René Dietmüller

CIS-Software - Peter René Dietmüller
Lorcherstraße 2a, 4470 Enns

Wichtige Module

- **stdio.h** **Standard Input-Output**
- **stdlib.h** **Standard Library**
- **math.h** **Mathematische Funktionen**
- **string.h** **String-Funktionen**
- **time.h** **Datums- und Zeit-Funktionen**

Zeilenweise Eingabe

- **Modul**
  - **#include <stdio.h>**
- **Aufruf**
  - **char *fgets(char *string, int n, FILE *stream);**
  - **char *gets(char *buffer);**
- **Beispiel**
  - **char line[80];**

```
fgets(line, sizeof(line), stdin);
gets(line);
```

KV Betriebssysteme
Einführung in C

Zeilenweise Ausgabe

- **Modul**
  - **#include <stdio.h>**
- **Aufruf**
  - **int fputs(const char *string, FILE *stream);**
  - **int puts(const char *string);**
- **Beispiel**
  - **char line[80] = „Hallo Welt\n“;**

```
fputs(line, stdin);
puts(line);
```


Funktion scanf

- **Zweck**
  - **Liest formatierte Daten von stdin.**
- **Modul**
  - **#include <stdio.h>**
- **Aufruf**
  - **int scanf(const char *format [, argument]...);**
- **Beispiel**
  - **int iAlter;**
    int iGehalt;
    scanf("%d\t%d\n", &iAlter, &iGehalt);

Funktion sprintf, sscanf

- **Zweck**
  - **Formatierte Ein-/Ausgabe in eine Zeichenkette**
- **Aufruf**
  - **int sscanf(const char *buffer,**
    const char *format [, argument] ...);
  - **int sprintf(char *buffer,**
    const char *format [, argument] ...);
- **Beispiel**
  - **char buffer[200]; int iAlter = 20;**
    sprintf(buffer, "Alter: %d\n", iAlter);
    sscanf(buffer, "%d\n", &iAlter);

Stringfunktionen

- **Funktionen aus dem Modul <string.h>**
  - **char *strcat(char *strDest, const char *strSource);**
    - Hängt eine Zeichenkette an eine andere an.
  - **int strcmp(const char *string1, const char *string2);**
    - Vergleicht zwei Zeichenketten miteinander und gibt -1 zurück, wenn string1 < string2, 0 wenn string1 = string2 und +1 wenn string1 > string2.
  - **char *strcpy(char *strDest, const char *strSource);**
    - Kopiert eine Zeichenkette.
  - **size_t strlen(const char *string);**
    - Ermittelt die Länge einer Zeichenkette.