

KV Betriebssysteme

Einführung in C

KV Betriebssysteme Programmargumente

DI. Dr. Peter René Dietmüller

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

Programmargumente

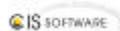


↳ Argumente an ein C-Programm werden als Zeichenketten an die Funktion main übergeben. Die vollständige Deklaration von main lautet:

```
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;

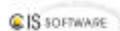
    for (i = 0; i < argc; i++)
        printf("Argument %d: %s\n", i, *argv++);
    printf("%s\n", getenv("PATH"));
    return 0;
}
```

KV Betriebssysteme 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 (l)

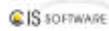


- "r" read only
- "r+" read and write
- "w" write only (create new file)
- "w+" create new file and allow read
- "a" append
- "a+" append + read allowed from end of old file
- "b" Binärmodus (kombinierbar, z.B.: „r+b“)
- „t“ Textmodus (kombinierbar, z.B.: „w+t“),
CR LF -> LF, 0x26 beendet Verarbeitung

KV Betriebssysteme

Einführung in C

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 - fputc**

- `int putchar (int c);`
- `int fputc (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:

↪ **stdin** 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
Lorchestraße 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.

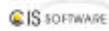
↪ **Beispiel**

- `#define Min(a,b) (a<b ? a : b)`
- `z = Min(x,y); ==> z = (x<y ? x : y);`

KV Betriebssysteme

Einführung in C

Makros (II)



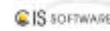
Beispiel:

```
#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 */
```

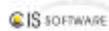
```
x=2;
y=Square(++x);
/* y=9 x=3 */
```

```
#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

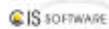
DI, Dr. Peter René Dietmüller

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

KV Betriebssysteme

Einführung in C

Elemente eines Moduls



↪ Schnittstellenbeschreibung

- „Header-File“, hat meist die Endung „.h“

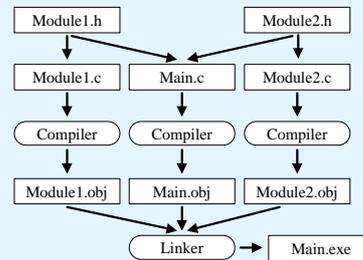
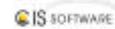
↪ Implementierung

- „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
- ... Dateinhalt
- #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

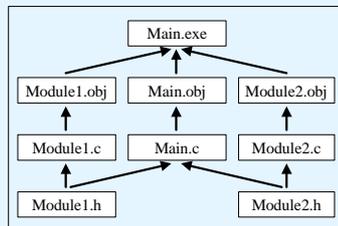
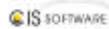
DI, Dr. Peter René Dietmüller

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

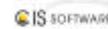
KV Betriebssysteme

Einführung in C

Abhängigkeiten



Beschreibungsblock



↪ Bedeutung

- Ein Beschreibungsblock (Description Block) ist eine Zeile, die Abhängigkeiten definiert:

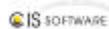
↪ Syntax

- targets... : depends...
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.

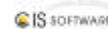
↪ Syntax

- .VonErw.ZuErw:
Befehle

↪ Beispiel

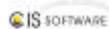
- .c.obj:
cl -c \$*.c

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



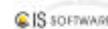
↪ Bedeutung

- 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 $<
```

KV Betriebssysteme

Einführung in 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.

DI. Dr. Peter René Dietmüller KV Betriebssysteme 31

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.

DI. Dr. Peter René Dietmüller KV Betriebssysteme 32

KV Betriebssysteme Bibliotheken

DI. Dr. Peter René Dietmüller

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

DI. Dr. Peter René Dietmüller KV Betriebssysteme 33

Bibliotheken

```

graph TD
    studio_h[studio.h] --> Main_c[Main.c]
    Module1_h[Module1.h] --> Module1_c[Module1.c]
    Main_c --> Compiler1((Compiler))
    Module1_c --> Compiler2((Compiler))
    Compiler1 --> Main_obj[Main.obj]
    Compiler2 --> Module1_obj[Module1.obj]
    Main_obj --> Linker((Linker))
    Module1_obj --> Linker
    Linker --> Main_exe[Main.exe]
    lib[Bibliotheken (z.B. Laufzeitbibliothek)] --> Linker
  
```

DI. Dr. Peter René Dietmüller KV Betriebssysteme 34

Bibliotheken

```

graph TD
    Stdlib_h[Stdlib.h] --> Stdlib_c[Stdlib.c]
    Stdio_h[Stdio.h] --> Stdio_c[Stdio.c]
    String_h[String.h] --> String_c[String.c]
    Stdlib_c --> Compiler1((Compiler))
    Stdio_c --> Compiler2((Compiler))
    String_c --> Compiler3((Compiler))
    Compiler1 --> Stdlib_obj[Stdlib.obj]
    Compiler2 --> Stdio_obj[Stdio.obj]
    Compiler3 --> String_obj[String.obj]
    Stdlib_obj --> Lib[Lib]
    Stdio_obj --> Lib
    String_obj --> Lib
    Lib --> crt_lib[crt.lib]
  
```

DI. Dr. Peter René Dietmüller KV Betriebssysteme 35

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

DI. Dr. Peter René Dietmüller KV Betriebssysteme 36

KV Betriebssysteme

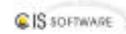
Einführung in C

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

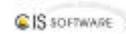
- `char *fgets(char *string, int n, FILE *stream);`
- `char *gets(char *buffer);`

↪ Beispiel

- `char line[80];`

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

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 printf, 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] ...);`

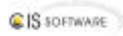
↪ Beispiel

- `char buffer[200]; int iAlter = 20;`
- `sprintf(buffer, "Alter: %d\n", iAlter);`
- `sscanf(buffer, "%d\n", &iAlter);`

KV Betriebssysteme

Einführung in C

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.
