C Programmierkurs

4. Stunde: Datei Ein- und Ausgabe

Johannes Hayeß & Mirko Seibt

15. November 2018

Technische Universität Dresden

Datei Ein- und Ausgabe mit Hilfe von Unix

Eine Datei öffnen

Definiert in **fcntl.h**

Schlüsselwort: **open**

Wir können die Schnittstelle zum Betriebssystem benutzen.

Eine Datei öffnen

Definiert in **fcntl.h**

Schlüsselwort: **open**

Wir können die Schnittstelle zum Betriebssystem benutzen.

Jeder Prozess besitzt seine eigenen Datei-Bezeichner.

Einfache Ganzzahlen, die die einzelnen Datei repräsentieren.

Eine Datei öffnen

Definiert in **fcntl.h**
Schlüsselwort: **open**
Wir können die Schnittstelle zum Betriebssystem benutzen.
Jeder Prozess besitzt seine eigenen Datei-Bezeichner.
Einfache Ganzzahlen, die die einzelnen Datei repräsentieren.

```
int [Bezeichner] = open("[Pfad]", [Optionen]);
```

Öffnet die Datei und gibt uns den Datei-Bezeichner zurück.

Optionen:

```
O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, ...
```

Aus einer Datei lesen

Schlüsselwort: **read**

Liest den Inhalt einer Datei:

```
read([Bezeichner], [Variable], [Länge]);
```

Aus einer Datei lesen

Schlüsselwort: **read**

Liest den Inhalt einer Datei:

```
read([Bezeichner], [Variable], [Länge]);
```

Beispiel:

```
int my_file = open("[Pfad]", O_RDONLY);
if (my_file != -1) {
    char my_text[256] = {};
    read(my_file, my_text, 255);
}
```

In eine Datei schreiben

Schlüsselwort: **write**

Schreibt einen String in eine Datei:

```
write([Bezeichner], [String], [Länge]);
```

In eine Datei schreiben

Schlüsselwort: **write**

Schreibt einen String in eine Datei:

```
write([Bezeichner], [String], [Länge]);
```

Beispiel:

```
int my_file = open("[Pfad]", O_WRONLY | O_CREAT, 0644);
char my_text[12] = "Hallo Welt!";
write(my_file, my_text, 12);
```

Einer Datei schliessen

Schlüsselwort: **close**

Schliesst eine Datei wieder:

```
int my_file = open("[Pfad]", O_RDWR);
/* Lesen/Schreiben/... */
close(my_file);
```

Spezielle Datei-Bezeichner

Jeder Unix-Prozess hat Standard-Datei-Bezeichner.
Diese Konstanten sind festgelegt in der **unistd.h**:

Spezielle Datei-Bezeichner

Jeder Unix-Prozess hat Standard-Datei-Bezeichner.
Diese Konstanten sind festgelegt in der **unistd.h**:
STDIN_FILENO für input
STDOUT_FILENO für output
STDERR_FILENO für error

Spezielle Datei-Bezeichner

Jeder Unix-Prozess hat Standard-Datei-Bezeichner.
Diese Konstanten sind festgelegt in der **unistd.h**:
STDIN_FILENO für input
STDOUT_FILENO für output
STDERR_FILENO für error

Diese können auch zum lesen und schreiben verwendet werden.

```
write(STDOUT_FILENO, "Hallo Welt!\n", 14);
```

Der einfachere Weg
stdio.h

Datei öffnen

Spezieller Datentyp:

Bezeichner: **FILE***

Datei öffnen

Spezieller Datentyp:
Bezeichner: **FILE***
Datei öffnen:

```
FILE *[Bezeichner] = fopen("[Pfad]","[Modus]");
```

Modi: **r, w, a, r+, w+, a+**

Datei öffnen

Spezieller Datentyp:
Bezeichner: **FILE***
Datei öffnen:

```
FILE *[Bezeichner] = fopen("[Pfad]","[Modus]");
```

Modi: **r, w, a, r+, w+, a+**

Beispiel:

```
FILE *my_file = fopen("[Pfad]","r+");
```

Daten lesen

Datei lesen:

```
fgets([Variable], [Länge], [Bezeichner]);
```

Daten lesen

Datei lesen:

```
fgets([Variable], [Länge], [Bezeichner]);
```

Beispiel:

```
FILE *my_file = fopen("Pfad", "r");
char my_text[256] = {};
fgets(my_text, 256, my_file);
```

Daten schreiben

Datei schreiben:

```
fputs([String], [Bezeichner]);
```

Daten schreiben

Datei schreiben:

```
fputs([String], [Bezeichner]);
```

Beispiel:

```
fputs("Hello World!\n", stdout);
```

Spezialbezeichner: **stdin, stdout, stderr**

Dateien schliessen

Datei schliessen:

```
fclose([Bezeichner]);
```

Beispiel:

```
fclose(my_file);
```

Kommandozeilen-Argumente

main-Funktion Upgrade

```c
int main (void) { return EXIT_SUCCESS; }
```

main-Funktion Upgrade

```c
int main (void) { return EXIT_SUCCESS; }
```

```c
int main (int argc, char* argv[]) {
    return EXIT_SUCCESS;
}
```

argc gibt die Anzahl an Argumenten in **argv[]** an

argv[] enthält Argumente in Form von Strings

Kommandozeilenargumente

```
$ ./[Programmname] [Argument1] [Argument2] ...
```

Beispiel:

```
$ ./rot13 text.txt
```

Kommandozeilenargumente

```
$ ./[Programmname] [Argument1] [Argument2] ...
```

Beispiel:

```
$ ./rot13 text.txt
```

argv[i]:

```
argv[0] == ./rot13
argv[1] == text.txt
```

Switch

Verschachtelung von Alternativen

```
if ([Bedingung1]) {
    /* 1. if-branch */
} else if ([Bedingung2]) {
    /* 2. if-branch */
} else if ([Bedingung3]) {
    /* 3. if-branch */
} else {
    /* else-branch */
}
```

Verschachtelung von Alternativen

```
if ([Bedingung1]) {
    /* 1. if-branch */
} else if ([Bedingung2]) {
    /* 2. if-branch */
} else if ([Bedingung3]) {
    /* 3. if-branch */
} else {
    /* else-branch */
}
```

ABER: Verschachtelungen können schnell unübersichtlich werden.

Switch

Schlüsselwörter: **switch, case, default, break**

Geeignet für Variablen mit konstanten Ergebnissen.

```
switch ([Variable]) {
    case [Option1] : [Anweisung]; break;
    case [Option1] : [Anweisung]; break;
    ...
    default : [Anweisung];
}
```

Switch

Schlüsselwörter: **switch, case, default, break**

Geeignet für Variablen mit konstanten Ergebnissen.

```
switch ([Variable]) {
    case [Option1] : [Anweisung]; break;
    case [Option1] : [Anweisung]; break;
    ...
    default : [Anweisung];
}
```

Vorsicht: Ohne break werden alle Optionen getestet, unter Anderem default!

Beispiel: Switch

```c
int main(int argc, char* argv[]) {
    switch (argc) {
        case 0 :
            return EXIT_FAILURE;
        case 1 :
            printf("keine Argumente\n"); break;
        case 2 :
            printf("Es gibt 1 Argument\n") break;
        default :
            printf("Es gibt mehr als 1 Argument\n");
    }
    return EXIT_SUCCESS;
}
```