

Kurseinheit 11: Standardbibliothek

1. Standardbibliothek
2. Wichtige Funktionen aus der Standardbibliothek
3. Compilieren über Konsole

Übersicht KE 11

Lehrveranstaltung Ingenieur-Informatik – 2 SWS/2 Credits: EI1, EI+1, MKA1, MK+1, EI3nat3
Lehrveranstaltung Programmierung 2 (Teil C) – 2 SWS/2 Credits: AI2

Unterrichtsdauer für diese Kurseinheit: 90 Minuten

Korrespondierende Kapitel aus *C-Programmierung – Eine Einführung*: Kapitel 12

Zusatzthemen: Compilieren über Konsole

1. Standardbibliothek

3

Standards C89/C99/C11

In den ANSI-Standards ist hinterlegt, welche Funktionen in der Standardbibliothek vorhanden sind. In jedem neuen Standard kommen neue Funktionen hinzu. Ebenso werden vereinzelt noch die Datentypen der Übergabeparameter angepasst.

Die Standardbibliothek wird **automatisch** eingebunden. Es sind nur die Funktionen, die benutzt werden, über ein `#include` zu deklarieren. In [LUH17] findet sich eine genaue Auflistung.

Im Rahmen dieser Lehrveranstaltung werden Funktionen aus `ctype.h`, `string.h`, `stdio.h`, `math.h`, `stdlib.h` und `time.h` (kurz) behandelt.

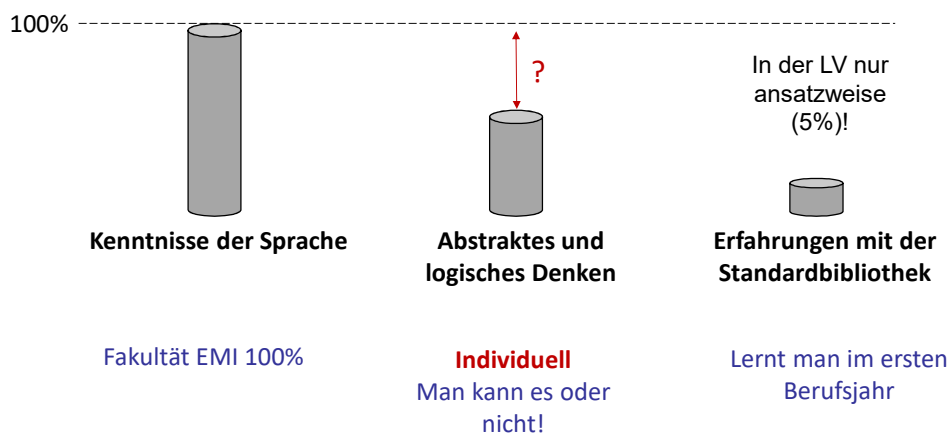
Mircosoft (und andere Compilerherstellen) geht (/gehen) hier noch einen Sonderweg: Es werden der Standardbibliothek noch weitere Funktionen hinzugefügt (z.B. aus `<conio.h>`) die nicht zum Standard gehören. Verwendet ein C-Programm diese, so lässt es sich dann nicht mehr unbedingt immer mit einem anderen C-Compiler übersetzen. C-Compiler erfüllen oft **nicht** alle Features von C99 und C11! Die ausschließliche Verwendung von C89 sichert die Kompatibilität zu andern C-Compilern. ANSI C, C89 und ISO C90 sind äquivalent!

1. Standardbibliothek

4

„Programmierlevel“

Wie gut (oder wie schnell) kann jemand programmieren?
Dazu sind die drei folgenden Säulen notwendig!



1. Standardbibliothek

5

Evolution der Funktionen am Bsp. von strcpy (1)

Brian W. Kernighan, Dennis Ritchie: The C Programming Language, Prentice Hall, 1978

```
void strcpy (char *s, char *t)
{
    while (*s++ = *t++) ;
}
```

Dennis Ritchie entwickelte in den frühen 70er Jahren die Programmiersprache C aufbauend auf der Programmiersprache B. **K&R-C** als quasi-Standard!

Diese Variante hat einen entscheidenden Nachteil. Eine Verkettung ist nicht möglich, da strcpy keinen Zeiger zurückgibt!

↪ Müsste für Verkettung char* zurückgeben. Impliziter Cast zu const char*!

```
strcat(acDest, strcpy(acStr1, "Hello"));
```

```
char* strcat(char* pcS1, const char* pcS2);
```

Obige Variante von strcpy verwendet noch Trickprogrammierung:

- Leerer Schleifenrumpf – Semikolon hier korrekt!
- Zuweisungsoperator statt Vergleichsoperator
- Wahrheitswert der Schleife ergibt sich durch Ergebnis der Zuweisung
- Postinkrement und Dereferenzierungsoperator ohne Klammern (implizites Wissen über Priorität der Operatoren ist zum Verständnis notwendig)

Don't do it!

1. Standardbibliothek

6

Evolution der Funktionen am Bsp. von strcpy (2)

```
void strcpy (char *s, char *t)
{
    while (*s++ = *t++) ;
}
```

K&R-C



Verständlichere
Variante mit C-
Coding
Styleguide und
Sanity Checks

```
void strcpy (char *pcDest, char* pcSource)
{
    if ((pcDest != NULL) && (pcSource != NULL))
    {
        *pcDest = *pcSource;
        while (*pcDest != 0x00)
        {
            pcDest++;
            pcSource++;
            *pcDest = *pcSource;
        }
    }
}
```

1. Standardbibliothek

7

Evolution der Funktionen am Bsp. von strcpy (3)

In der Version von **C89** kamen zwei Änderungen hinzu:

- Rückgabewert ist jetzt char* - dadurch Verkettung möglich
- Source-Zeiger wird von char* zu const char*

```
char* strcpy (char *pcDest, const char *pcSource)
```

Mit einem Zeiger, der als const char* deklariert ist, kann der Speicher, auf den dieser zeigt, mit dem Zeiger nicht mehr beschrieben werden. Die Anweisung *pcSource = 0xCC würde einen Compilerfehler generieren! Dies erhöht die Wartbarkeit!

In der Version in **C99** kam noch das neue Schlüsselwort restrict (siehe KE7) dazu.

```
char* strcpy (char* restrict pcDest, const char* restrict pcSource)
```

Intern geschieht für pcSource immer ein impliziter Cast! Es kann ein char* für pcSource übergeben werden.

1. Standardbibliothek

8

Evolution der Funktionen am Bsp. von strcpy (4)

In **C11** kam noch die sichere (Security) Funktion strcpy_s hinzu.

- Eine Verkettung (ist fehleranfällig) ist nicht mehr möglich
- Als Rückgabewert wird ein Fehlercode zurückgegeben.
- Ein dritter Parameter gibt die Anzahl maximal zu kopierender Zeichen an (üblicherweise die Größe des Destination-Buffers) – zu kopierendes 0x00 muss berücksichtigt werden.

```
errno_t strcpy_s(char* restrict pcDest, rsize_t uiDestSize,  
const char *restrict pcSource)
```

Gibt eine 0 zurück, falls erfolgreich kopiert wurde. Wert ungleich 0 gibt im Fehlerfall die Fehlernummer an.

```
typedef int errno_t;
```

In corecrt.h (Core
Common Runtime)

Für den Benutzer dieser Funktion ist nur die Änderung des Rückgabewertes von großer Bedeutung.

```
errno_t iError = 0;  
char acDest[12] = {0};  
char acSource[] = "Hello World";  
  
iError = strcpy_s(acDest, 12, acSource);  
printf("iError: %d\n", iError);
```

corecrt.h wird von verschiedenen Headerdateien inkludiert. Include-Guard!

2. Wichtige Funktionen aus der Standardbibliothek

9

Überblick

Im Moodlekurs findet sich das Dokument KlausurOffiziellerAnhang.pdf. Dort sind die Deklarationen der wichtigsten Funktionen aufgelistet (ctype.h, string.h, stdio.h, math.h, stdlib.h und time.h)

Ggf. wird in der Klausur noch die genaue Beschreibung einer Funktion in Englisch ausgegeben, falls diese Funktion nicht vorher in der Vorlesung oder im Labor verwendet wurde.

Exemplarisch werden hier einige sehr wichtige Funktionen im Detail behandelt.



2. Wichtige Funktionen aus der Standardbibliothek

10

ctype.h

Relativ einfache Funktionen, die einen ASCII-Code übergeben bekommen. Rückgabewert ungleich 0 falls wahr, ansonsten 0.

```
iRet = isalnum('a');
printf("iRet bei 'a': %d\n", iRet);
iRet = isalnum('9');
printf("iRet bei '9': %d\n", iRet);
iRet = isalnum('+');
printf("iRet bei '+': %d\n", iRet);
```

```
Mi...  —  □  ×
iRet bei 'a': 2
iRet bei '9': 4
iRet bei '+': 0
```

ctype.h

```
int isalnum(int iX);
int isalpha(int iX);
int iscntrl(int iX);
int isdigit(int iX);
int isgraph(int iX);
int islower(int iX);
int isprint(int iX);
int ispunct(int iX);
int isspace(int iX);
int isupper(int iX);
int isxdigit(int iX);
int tolower(int iX);
int toupper(int iX);
```

2. Wichtige Funktionen aus der Standardbibliothek

11

string.h

```
void* memcpy(void* pvS1, const void* pvS2, size_t uiN);
void* memmove(void* pvS1, const void* pvS2, size_t uiN);
char* strcpy(char* pcS1, char* pcS2);
char* strncpy(char* pcS1, char* pcS2, size_t uiN);
char* strcat(char* pcS1, const char* pcS2);
char* strncat(char* pcS1, const char* pcS2, size_t uiN);
int memcmp(const void* pvS1, const void* pvS2, size_t uiN);
int strcmp(const char* pcS1, const char* pcS2);
int strncmp(const char* pcS1, const char* pcS2, size_t uiN);
void* memchr(const void* pvS, int iC, size_t uiN);
char* strchr(const char* pcS, int c);
size_t strcspn(const char* pcS1, char* pcS2);
char* strpbrk(const char* pcS1, const char* pcS2);
char* strchr(const char* pcS, int iX);
size_t strspn(const char* pcS1, const char* pcS2);
const char* strstr(const char* pcS1, const char* pcS2);
void* memset(void* pvM, int iC, size_t uiN);
size_t strlen(const char* pcS);

char* strtok(char* pcStr, const char* pccDelimiters);
char* strlwr(char* pcStr); // converts string to lowercase – no C Standard
char*strupr(char* pcStr); // converts string to uppercase – no C Standard
```

Oft zwei Varianten:
Mit und ohne „n“ (n: Anzahl
der Zeichen)!

Wichtig für Prüfung:
memcpy, strcpy, strncpy,
strcat, strncat, memcmp
strcmp, strncmp,
strchr, strstr, strlen und
strtok.

Im Labor werden die
sicheren Varianten ({}_s)
verwendet.

2. Wichtige Funktionen aus der Standardbibliothek

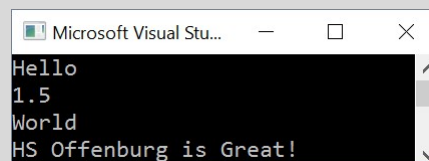
12

string.h – strtok (1)

```
char acStringWithTokens[] = "Hello;1.5,World;HS Offenburg is Great!";
char* pcToken;
```

```
pcToken = strtok(acStringWithTokens, ";,");
```

```
while (pcToken != NULL)
{
    printf("%s\n", pcToken);
    pcToken = strtok(NULL, ";,");
}
```



strtok gilt als unsichere Funktion – Treat Warnings as Errors ausschalten! In der nächsten
Kurseinheit wird strtok_s behandelt!

Eine Zeichenkette (String) kann einzelne Teilzeichenketten (Tokens) enthalten, die durch
Trennzeichen (Delimiter) getrennt sind. Trennzeichen sind hier im Beispiel das Komma und
der Semikolon. Kommas als Trennzeichen sollten in csv-Dateien vermieden werden!

Warum?

```
char* strtok(char* pcStr, const char* pccDelimiters);
```

2. Wichtige Funktionen aus der Standardbibliothek

13

string.h – strtok (2)

```
char* strtok(char* pcStr, const char* pccDelimiters);
```

```
char acStringWithTokens[] = "Hello;1.5,World;HS Offenburg is Great!";
```

H e l l o ; 1 . 5 , W o r l d ; H S O f f e n b u r g i s G r e a t !

■ EOS: End of String (0x00)

Erster Aufruf:

```
pcToken = strtok(acStringWithTokens, ";,");
```

H e l l o 1 . 5 , W o r l d ; H S O f f e n b u r g i s G r e a t !

↑ ↑
pcToken pcRemember

Beim ersten Aufruf wird die zu bearbeitende Zeichenkette übergeben. **Das erste Trennzeichen wird durch ein EOS ersetzt.** Ein statischer Zeiger in strtok (z.B. static char* pcRemember) merkt sich die Position, an welcher die Bearbeitung fortzusetzen ist.

2. Wichtige Funktionen aus der Standardbibliothek

14

string.h – strtok (3)

Zweiter Aufruf:

```
pcToken = strtok(NULL, ";,");
```

H e l l o 1 . 5 W o r l d ; H S O f f e n b u r g i s G r e a t !

↑ ↑
pcToken pcRemember

Wird ein NULL-Zeiger übergeben, so führt die Funktion die Bearbeitung bei der gemerkten Position fort.

Dritter Aufruf:

```
pcToken = strtok(NULL, ";,");
```

H e l l o 1 . 5 W o r l d H S O f f e n b u r g i s G r e a t !

↑ ↑
pcToken pcRemember

2. Wichtige Funktionen aus der Standardbibliothek

15

string.h – strtok (4)

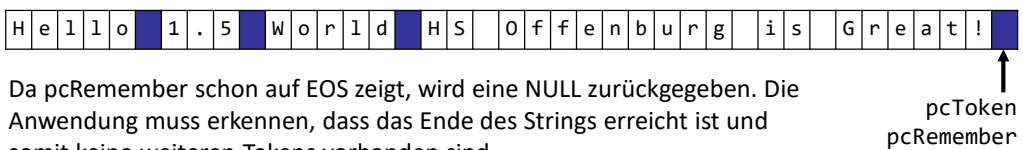
Vierter Aufruf:

```
pcToken = strtok(NULL, ";;");
```



Fünfter Aufruf:

```
pcToken = strtok(NULL, ";;");
```



Da `pcRemember` schon auf EOS zeigt, wird eine NULL zurückgegeben. Die Anwendung muss erkennen, dass das Ende des Strings erreicht ist und somit keine weiteren Tokens vorhanden sind.

Wichtiger abschließender Hinweis: **strtok verändert die Zeichenkette** (Delimiter werden durch EOS ersetzt)!

2. Wichtige Funktionen aus der Standardbibliothek

16

stdio.h

```
int fflush(FILE* pFile);
size_t fread(void* pvData, size_t uiSize, size_t uiNumber, FILE* pFile);
size_t fwrite(void* pvData, size_t uiSize, size_t uiNumber, FILE* pFile);
int fseek(FILE* pFile, long lOffset, int iPos);
long ftell(FILE* pFile);
void rewind(FILE* pFile);
int feof(FILE* pFile); //return not 0 if EOF is reached
int fputc(const char* pccStr, FILE* pFile); //return EOF if error, otherwise non-zero
int rename(char* pcFilenameOld, char* pcFilenameNew); // return 0 if success
int remove(const char* pcFileName); // return 0 if success

FILE* fopen(const char* pccFilename, const char* pccModus);
errno_t fopen_s(FILE** ppFile, const char* pccFilename, const char* pccModus);
int fclose(FILE* pFile); //0 if okay, EOF if Error
int printf(const char* pccFormat, ...);
int sprintf(char* pcStr, const char* pccFormat, ...);
int fprintf(FILE* pFile, const char* pccFormat, ...);
int scanf(const char* pccFormat, ...);
int sscanf(char* pcStr, const char* pccFormat, ...);
```

Input/Output:

- Konsole (Tastatur, Bildschirm)
- File

2. Wichtige Funktionen aus der Standardbibliothek

17

math.h

```
double acos(double dX);
double asin(double dX);
double atan(double dX);
double atan2(double dX, double dY);
double cos(double dX);
double sin(double dX);
double tan(double dX);
double cosh(double dX);
double sinh(double dX);
double tanh(double dX);
double exp(double dX);
double log(double dX);
double log10(double dX);
double pow(double dX, double dY); // x^y
double sqrt(double dX);
double ceil(double dX); // Ganzzahliger Wert durch Aufrunden
double floor(double dX); // Ganzzahliger Wert durch Abrunden
double fmod(double dX, double dY); // Rest der (ganzzahligen) Division der beiden Param.

double fabs(double dX); // C99
float fabsf(float dX); // C99
long double fabsl(long double dX); // C99
```

Vorherrschender Datentyp ist double. Dieser hat eine höhere Genauigkeit als float! Selbst-erklärende Funktionsnamen!

Winkelfunktionen erwarten als Parameter das **Bogenmaß** (Radiant)!
Oft muss eine Anwendung vom geläufigeren Gradmaß (DEG) in Bogenmaß (RAD) umrechnen:

$$\text{Gradmaß}/360 = \text{Bogenmaß}/(2 \cdot \pi)$$

2. Wichtige Funktionen aus der Standardbibliothek

18

math.h – atan und atan2

```
#define _USE_MATH_DEFINES
#include <math.h>
```

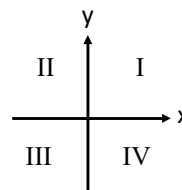
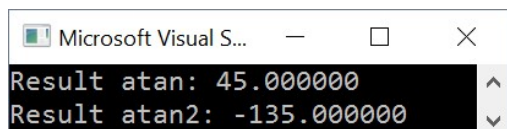
Um die mathematischen Konstanten aus math.h nutzen zu können, muss `_USE_MATH_DEFINES` **vorher** definiert werden. Microsoft spezifisch!!!

```
dRAD = atan(-1. / -1.);
dDEG = (dRAD / (2 * M_PI)) * 360.;
printf("Result atan: %lf\n", dDEG);

dRAD = atan2(-1., -1.);
dDEG = (dRAD / (2 * M_PI)) * 360.;
printf("Result atan2: %lf\n", dDEG);
```

Die Funktion atan liefert nur korrekte Werte aus dem ersten und vierten Quadranten.

Statt einer Fallunterscheidung mit if und Addition von 180 Grad, kann besser atan2 verwendet werden.



2. Wichtige Funktionen aus der Standardbibliothek

19

stdlib.h

```
double atof(const char* pccValue);
int atoi(const char* pccValue);
long atol(const char* pccValue);
double strtod(const char* pccValue, char** ppcEndConversion);
long strtol(const char* pccValue, char** ppcEndConversion, int iBase);
unsigned long strtoul(const char* pccValue, char** ppcEndConversion, int iBase);
int rand(void);
void srand(unsigned int uiStartValue);
int abs(int iValue);
int labs(long int liValue);

char* itoa(int iValue, char* pcStr, int iBase);
```

Konvertierungsfunktionen
und **Zufallszahlen** sind hier
besonders wichtig!

2. Wichtige Funktionen aus der Standardbibliothek

20

stdlib.h - Zufallszahlen

Oft werden für Algorithmen Zufallszahlen benötigt. Intern wird meist ausgehend von der vorherigen Zufallszahl eine neue Zufallszahl bestimmt (rand). In diesem Zusammenhang wird von **Pseudozufallszahlen** gesprochen. Am Anfang wird mittels einer möglichst zufälligen Zahl ein Startwert festgelegt (srand: Set Random). Hier im Beispiel wird die aktuelle Zeit verwendet!

```
srand((unsigned int)time(NULL));

for (iCounter = 0; iCounter < 10; iCounter++)
{
    iRandomValue = rand();
    printf("RandomValue %d: %d\n", iCounter, iRandomValue);
}
```

Die Funktion **rand** gibt eine Zufallszahl zwischen **0 und RAND_MAX** zurück!
Wird eine Zufallszahl zwischen 0 und 10 benötigt, so hilft das folgende Snippet:

```
rand() % 11;
```

2. Wichtige Funktionen aus der Standardbibliothek

21

stdlib.h - Zufallszahlen

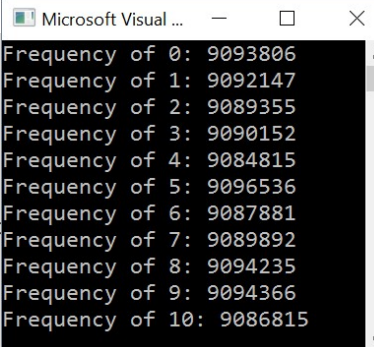
Wie gut ist der Zufallszahlen-Algorithmus aus der stdlib.h?
Falls dieser den Ansprüchen (Gleichverteilung) nicht genügt,
sind andere Zufallszahlen-Algorithmen einzusetzen.

```
int iCounter;
int iRandomValue;
int iArray[11] = {0};

srand((unsigned int)time(NULL));

for (iCounter = 0; iCounter < 100000000; iCounter++)
{
    iRandomValue = rand() % 11;
    iArray[iRandomValue]++;
}

for (iCounter = 0; iCounter < 11; iCounter++)
{
    printf("Frequency of %d: %d\n", iCounter, iArray[iCounter]);
}
```



```
Frequency of 0: 9093806
Frequency of 1: 9092147
Frequency of 2: 9089355
Frequency of 3: 9090152
Frequency of 4: 9084815
Frequency of 5: 9096536
Frequency of 6: 9087881
Frequency of 7: 9089892
Frequency of 8: 9094235
Frequency of 9: 9094366
Frequency of 10: 9086815
```

2. Wichtige Funktionen aus der Standardbibliothek

22

stdlib.h – Konvertierungsfunktionen

Zeichenkette (String) → double (float), int, long int

Einfache Funktionen

```
double atof(const char* pccValue);
int atoi(const char* pccValue);
long int atol(const char* pccValue);
```

Welcher Wert ergibt sich für dVal?

```
dVal = atof("");
dVal = atof("5.5Ups");
dVal = atof("5,5");
```

Fehlerhandling nicht direkt vorhanden!

Komplexere Funktionen mit Fehlerhandling

```
double strtod(const char* pccValue, char** ppcEndConversion);
long int strtol(const char* pccValue, char** ppcEndConversion, int iBase);
unsigned long int strtoul(const char* pccValue, char** ppcEndConversion, int iBase);
```

double (float), int, long int → Zeichenkette

```
char* itoa(int iValue, char* pcStr, int iBase);
```

Es stehen allerdings mit sprintf, bzw. sprintf_s komfortablere Funktionen zur Verfügung (aus stdio.h).

2. Wichtige Funktionen aus der Standardbibliothek

23

stdlib.h – Konvertierungsfunktion strtod (1)

Mit atof kann nicht direkt bestimmt werden, ob alle Zeichen konvertiert wurden. Mit strtod ist dies möglich!

string to double

```
double strtod(const char* pccValue, char** ppcEndConversion);
```

```
double dVal;  
char* pcCh = NULL;  
char acValue[] = "5.5Ups";
```

Erstes Zeichen,
das nicht
konvertiert
wurde.

Erstes Zeichen, das nicht konvertiert wurde

5	.	5	U	p	s	0x00
---	---	---	---	---	---	------

5	.	5	0x00
---	---	---	------

Es wird ein Zeiger benötigt, welcher auf das erste Zeichen zeigt, welches nicht konvertiert wurde. Ist der Wert 0x00, wurden alle Zeichen konvertiert. Hier muss ein Zeiger auf Zeiger übergeben werden. Nur so kann strtod den Zeiger pcCh des Aufrufers ändern.

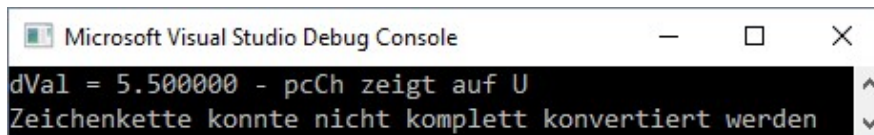
2. Wichtige Funktionen aus der Standardbibliothek

24

stdlib.h – Konvertierungsfunktion strtod (2)

```
dVal = strtod(acValue, &pcCh);  
printf("dVal = %f - pcCh zeigt auf %c\n", dVal, *pcCh);
```

```
if (*pcCh != 0x00)  
{  
    printf("Zeichenkette konnte nicht komplett konvertiert werden");  
}
```



Intern hat strtod einen lokalen char-Zeiger (z.B. pcNext), der auf das nächste Zeichen zeigt. Beim Abbruch der Konvertierung (Ungültiges Zeichen oder 0x00 erreicht) erfolgt dann einfach:

```
double strtod(const char* pccValue, char** ppcEndConversion);
```

„* tilgt ein p“ — *ppcEndConversion = pcNext;

2. Wichtige Funktionen aus der Standardbibliothek

25

time.h

Datenstrukturen

```
struct tm
{
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};
typedef long clock_t;
typedef long time_t
```

```
char* asctime(const struct tm* pctime);
time_t mktime(struct tm* pctime);
struct tm* localtime(const time_t* pctime);
clock_t clock(void);
time_t time(time_t* pctime);
char* ctime(const time_t* pctime);

double difftime(time_t xEndtime, time_t Begintime);
```

Wichtig:

- Bestimmung der aktuellen Zeit
- Laufzeitmessung von Algorithmen

2. Wichtige Funktionen aus der Standardbibliothek

26

time.h – Aktuelle Zeit und Laufzeitmessung

```
struct tm* pMyTime = NULL;
time_t iTime;
clock_t iClockStart;
clock_t iClockEnd;
double dSeconds;
```

Im Jahr 2038 kann es zu einem Problem kommen (**Y2K38**), weil die vorzeichenbehaftete Variable (long int) auf einem 32-Bit-System (typedef long int time_t;) beim Aufruf von time überläuft.

Aktuelle Zeit:

```
time(&iTime);
localtime_s(&sMyTime, &iTime);
asctime_s(acText, sizeof(acText), &sMyTime);
printf("Lokale Zeit/Datum: %s", acText);
```

time gibt die Anzahl der Sekunden seit 1.1.1970, 00:00:00 der Weltzeit UTC zurück. acText **muss** eine Länge von 40 haben.

Laufzeitmessung

```
iClockStart = clock();
// Algorithm
iClockEnd = clock();
dSeconds = ((double)iClockEnd - iClockStart) / CLOCKS_PER_SEC;
printf("Laufzeit: %f\n", dSeconds);
```

Expliziter Cast (double) nicht vergessen!

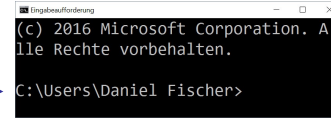
3. Compilieren über Konsole

27

Wichtige Befehle in der Konsole

Öffnen einer Konsole: Windows-Symbol, „cmd“ eingeben, Return-Taste -> Eingabeaufforderung wird gestartet.

Aktueller Pfad



Befehl	Kurzbeschreibung
dir	Gibt alle Verzeichnisse und Unterverzeichnisse aus
cd \	Wechselt ins Rootverzeichnis des Laufwerkes
cd ..	Wechselt zum übergeordneten Verzeichnis
cd HS2	Wechselt ins Unterverzeichnis
cd HS2\Ablage	Wechselt ins Unterverzeichnis
cd c:\Users\Mustermann	Wechselt ins Unterverzeichnis eines Laufwerkes
mkdir NewDir	Legt ein neues Unterverzeichnis mit dem Namen NewDir an
PATH	Gibt die gesetzten Pfade aus

Absoluter Pfad aus Explorer kopieren: Rechte Maustaste und „Adresse als Text kopieren“
Dann in Konsole in cd einfügen




3. Compilieren über Konsole

28

Ausführen eines Programmes in der Konsole

Name des Programmes (.exe kann entfallen)

Übergabeparameter an main (optional)



Damit das ausführbare Programm gefunden werden kann, muss eine der folgenden Bedingungen gelten:

- Das ausführbare Programm muss sich im aktuellen Verzeichnis befinden
- Das ausführbare Programm muss sich im Windowssystemverzeichnis befinden
- Das ausführbare Programm muss sich in einem Verzeichnis befinden, welches in der Umgebungsvariablen PATH der Konsole hinterlegt ist.

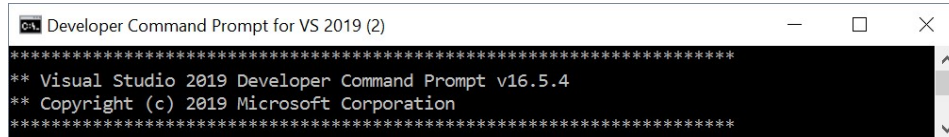
Trifft keine Bedingung zu, so kann das Programm durch Hinzufügen eines absoluten oder relativen Pfades vor dem Programmnamen gestartet werden. Dies ist gerade bei langen Pfadnamen relativ aufwändig und fehleranfällig.

3. Compilieren über Konsole

29

Beispiel (1)

1. Developer Command Prompt von Visual Studio öffnen (Windows Symbol, Folder Visual Studio 2019, Developer Command Prompt für VS 2019. Dies ist ein Konsolenfenster, in welchem die Pfade zu den MSVS Entwicklungstools (Compiler) schon gesetzt sind.



2. Navigieren Sie mit dem Befehl `cd` in ein Verzeichnis und legen Sie dort ein neues Unterverzeichnis an. Wechseln Sie in dieses Unterverzeichnis mit dem Befehl `cd`.
3. C-Datei `Main.c` und `Output.c` mit z.B. notepad dort abspeichern. In `Output.c` soll sich nur eine Funktion befinden, die „Hello EIM“ auf der Konsole ausgibt. Die `main`-Funktion in `Main.c` ruft diese dann auf.
4. Jetzt kann der C-Compiler aufgerufen werden: `cl /c Main.c` und `cl /c Output.c` – es wurden jetzt Objektdateien erzeugt ohne zu Linken (Option `/c`)
5. Um mit Warning Level 4 zu compilieren, ist z.B. `cl /c /W4 Main.c` aufzurufen.
6. Um zu Linken ist `cl Main.obj Output.obj /link /out:FirstCon.exe` einzugeben.
7. Die erzeugte Exe-Datei `FirstCon.exe` kann nun in der Konsole aufgerufen werden.

3. Compilieren über Konsole

30

Beispiel (2)

Letztendlich ist `cl` wiederum nur ein Konsolenprogramm, welches Parameter an `main` übergeben bekommt. Bisher hatte dies nur die IDE im Hintergrund aufgerufen. Bei größeren Projekten und Teams werden Programme meist zentral auf einem Server ohne IDE „gebaut“.

Ein erster Zwischenschritt wäre das Anlegen einer Batchdatei (z.B. „BuildPrg.bat“), welche die einzelnen Schritte nacheinander ausführt:

```
cl /c /W4 Main.c
cl /c /W4 Output.c
cl Main.obj Output.obj /link /out:FirstCon.exe
```

In Notepad einfach abspeichern (Dateityp: Alle Dateien, Name „Buildprg.bat“). Batchdatei mit „Buildprg“ starten.

In der Praxis ist das Bauen von größeren Anwendungen deutlich komplizierter. Sourcecodedateien müssen erst aus einer Versionsverwaltung „ausgecheckt“ werden, automatisierte Tests werden gestartet und beim Übersetzen laufen statische Codeanalysertools mit. Hier haben sich sogenannte Buildskripte bewährt, die von einem Tool (Make oder Build im Namen) ausgeführt werden.

Behandelte Schlüsselwörter in KE 11

Schlüsselwörter C89:

auto ✓	do ✓	goto ✓	signed ✓	unsigned ✓
break ✓	double ✓	if ✓	sizeof ✓	void ✓✓
case ✓	else ✓	int ✓	static ✓✓	volatile ✓
char ✓	enum ✓	long ✓	struct ✓	while ✓
const ✓✓	extern ✓✓	register ✓	switch ✓	
continue ✓	float ✓	return ✓	typedef ✓	
default ✓	for ✓	short ✓	union ✓	

Schlüsselwörter ab C99:

_Bool ✓	_Complex ✓	_Imaginary ✓	inline ✓	restrict ✓✓
---------	------------	--------------	----------	-------------

Schlüsselwörter ab C11:

_Alignas	_Alignof ✓	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			