

Anhang A: ASCII-Tabelle

Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen	Dez.	Hex.	Zeichen
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	\$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(72	48	H	104	68	h
9	9	HT	41	29)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Anhang B: Codierung

dezimal	hexadezimal	binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Anhang C: Standardbibliotheken

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);
```

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);
errno_t strcpy_s(char* pcDest, rsize_t uSize, const char* pcSrc); // C11
char* strncpy(char* pcS1, char* pcS2, size_t uiN);
errno_t strncpy_s(char* dest, rsize_t destsz, const char* src, rsize_t count); // C11
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 strcnmp(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* strrchr(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* strtok_s(char* pcStr, const char* pccDelimiters, char** ppcContext); // C11
char* strlwr(char* pcStr); // converts string to lowercase – no C Standard
char*strupr(char* pcStr); // converts string to uppercase – no C Standard
```

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
char* fgets(char* pcStr, int num, FILE* pFile); //CRLF is part of pcStr
int rename(char* pcFilenameOld, char* pcFilenameNew); // return 0 if success
int remove(const char* pccFileName); // return 0 if success

FILE* fopen(const char* pccFilename, const char* pccModus);
errno_t fopen_s(FILE** ppFile, const char* pccFilename, const char* pccModus); // C11
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 scanf_s(const char * pccFormat, ...); // Zusätzlicher Wert bei %c und %s notwendig -
                                         Anzahl Zeichen (C11)
int sscanf(char* pcStr, const char* pccFormat, ... )
```

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
```


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 int strtol(const char* pccValue, char** ppcEndConversion, int iBase);
unsigned long int 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);
void* malloc(size_t uiSize);
void free (void* pvMem);
```

time.h

Datenstrukturen

```
struct tm
{
    int tm_sec;           /* seconds,  range 0 to 59      */
    int tm_min;           /* minutes,  range 0 to 59      */
    int tm_hour;          /* hours,    range 0 to 23      */
    int tm_mday;          /* day of the month, range 1 to 31 */
    int tm_mon;           /* month,    range 0 to 11      */
    int tm_year;          /* The number of years since 1900 */
    int tm_wday;          /* day of the week, range 0 to 6 */
    int tm_yday;          /* day in the year, range 0 to 365 */
    int tm_isdst;         /* daylight saving time          */
};
typedef long int clock_t;
typedef long int time_t;
```

```
char* asctime(const struct tm* pcsTime);
time_t mktime(struct tm* psTime);
struct tm* localtime(const time_t* pcxTime);
clock_t clock(void);
time_t time(time_t* pxTime);
char* ctime(const time_t* pcxTime);

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

Anhang D: Beispielcode zu Threads

Starten eines Threads per Windows Thread API

```
HANDLE CreateThread(  
    LPSECURITY_ATTRIBUTES    lpThreadAttributes,  
    SIZE_T                   dwStackSize,  
    LPTHREAD_START_ROUTINE   lpStartAddress,  
    __drv_aliasesMem LPVOID  lpParameter,  
    DWORD                     dwCreationFlags,  
    LPDWORD                   lpThreadId  
);
```

Wichtige Parameter:

lpStartAddress: Funktionszeiger auf eine Thread-Funktion.

Deklaration einer Thread-Funktion: `DWORD WINAPI ThreadFunction(void *pParam);`

lpParameter: void-Zeiger auf die Parameter

Beispiel: Starten eines Threads ohne Parameterübergabe:

```
int main(void)  
{  
    HANDLE hThread1 = 0;  
  
    // Beispiel ohne Parameterübergabe an Threadfunktion  
    hThread1 = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);  
  
    if (hThread1 != 0)  
    {  
        // Code  
        CloseHandle(hThread1);  
    }  
    return 0;  
}
```

Beispiel: Starten eines Threads mit Parameterübergabe:

```
int main(void)  
{  
    struct Data myData = {4711, "Hello"};  
  
    // "&myData": Impliziter Cast zu void*(Typenloser Zeiger)  
    CreateThread(NULL, 0, ThreadFunction1, &myData, 0, NULL);  
  
    return 0;  
}  
  
DWORD WINAPI ThreadFunction1(void *pParam)  
{  
    DWORD dwRet = 0;  
    // Auf typenlosen Zeiger kann nicht zugegriffen werden -> „Rückcast“  
    struct Data* pData = (struct Data*) pParam;  
  
    // Code  
  
    return 0;  
}
```

Synchronisation von Threads

Warten auf einen Thread:

```
DWORD WaitForSingleObject(HANDLE hHandle,  
                           DWORD dwMilliseconds );
```

Handle: Handle auf Thread, auf den gewartet wird.

dwMilliseconds: Timeout in Millisekunden. INFINITE bedeutet keine Zeitbeschränkung.

Warten auf mehrere Threads:

```
DWORD WaitForMultipleObjects(DWORD nCount,  
                             const HANDLE *lpHandles,  
                             BOOL bWaitAll,  
                             DWORD dwMilliseconds );
```

nCount: Anzahl der Threads

lpHandles: Zeiger auf ein Array von Handles

bWaitAll: TRUE – Wartet bis alle Threads beendet sind, FALSE – Wartet bis ein Threads beendet ist

dwMilliseconds: Timeout in Millisekunden. INFINITE bedeutet keine Zeitbeschränkung.

Anhang E: Beispielcode zu DLL

DLL Implementierung

Alle zu exportierenden Funktionen müssen mit `__declspec(dllexport)` gekennzeichnet sein.

Headerdatei (z.B. DLL.h):

```
#pragma once
__declspec(dllexport) void PrintFromDLL(char* pcText);
```

Sourcdatei:

```
#include "DLL.h"

__declspec(dllexport) void PrintFromDLL(char* pcText)
{
    //
}
```

Aufruf der DLL von einer Anwendung über Implicit Linking

```
#include <windows.h>
int main(void)
{
    HMODULE hModule;
    void (*fpDLLFunc)(char*);
    hModule = LoadLibrary(TEXT("DLL.dll"));
    if (hModule != 0)
    {
        fpDLLFunc = (void(*)(char*))GetProcAddress(hModule, "PrintFromDLL");
        if (fpDLLFunc != NULL)
        {
            fpDLLFunc("My First Call to a DLL");
        }
        FreeLibrary(hModule);
    }
    return 0;
}
```