

Dateien:

- zusammengehörende Bytefolge
- permanent (typ.)
- Name Zeichen: 0-9, _, a-z, A-Z, ä, ö, ü, +, -, (\$), (.), evtl. groß/klein
Länge: 8.3, 14, ~128, ~256, ~1024, unendlich
- Pfad/Ort im Dateisystem (je nach OS)
- Attribute ro, Archiv, System
rwxrwxrwx
- Eigentümer, Gruppe, ... ACL (VMS, WIN NT)
- Länge
- ...

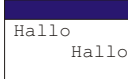
ANSI:

- zusammengehörende Bytefolge
- read, write, löschar
- Name je nach OS
- Länge

Textdatei, Binärdatei

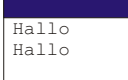
```
printf ("Hallo\n");
```

→ ASCII 10 "Linefeed"



```
printf ("Hallo\r\n");
```

→ ASCII 13 "carriage return"

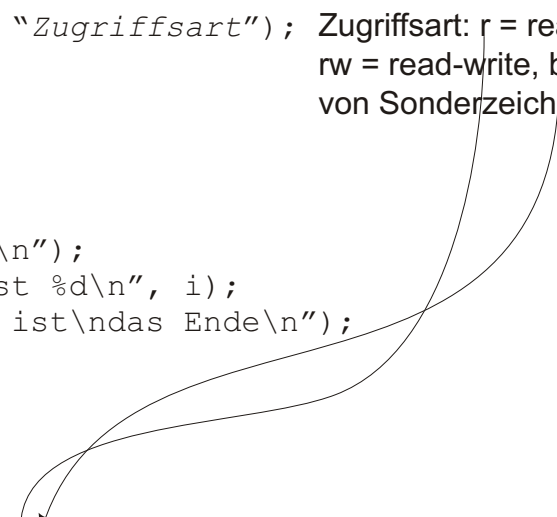


Sequentielle Datei/Direktzugriffsdatei (random access)

```
FILE *f;
fopen("/tmp/a.txt", "Zugriffsart");
"a.txt"
int i=12;
if (!f) {...}
else
{
    fputs(f, "Hallo\n");
    fprintf(f, "i ist %d\n", i);
    fprintf(f, "das ist\ndas Ende\n");
    ...
    fclose(f);
}
```

Zugriffsart: r = read, w = write,
rw = read-write, b = Nichtkonvertierung
von Sonderzeichen (z.B. \n bleibt \n)

```
FILE *f=fopen("a.txt", "rb");
if (!f) {...}
else
{
    char = zeile[1024];
    fgets(f, zeile, 1024);
    ....
}
```



```

fgets(zeile, 1024, f);
fscanf(f, "%1024s", zeile); // liest das nächste Wort
fscanf(f, "%s", zeile); // und keine ganze Zeile(n)
fclose(f);
}

```

```

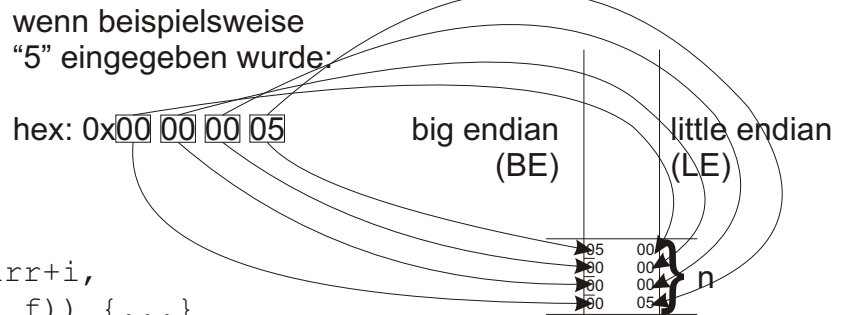
double * arr = NULL;
printf("Bitte Anzahl: \n"); scanf("%d", &n);
arr = (double *) malloc (n * sizeof(double)); if (!arr) {...}
arr[i] = ...;

```

```

FILE *f = fopen("x.dat", "rb"); if (!f) {...}
if (1 != fwrite(&n, sizeof(int), 1, f)) {...};

```



```

for (i=0; i < n; i++)
{
if (1 != fwrite(arr+i,
sizeof(double), 1, f)) {...}
}

```

oder

```

if (fwrite(arr, sizeof(double), n, f) != n) {...}
} close (f);

```

Lesen einer Binärdatei

```

int n; f=fopen("...", "rb"); if (!f) {...}
if (1 != fread(&n, sizeof(int), 1, f)) {...}
if (n != read (arr, sizeof(double), n, f)) {...}
fclose (f);

```

Datei Zeiger abfragen/setzen:

```

int i = ftell(f); //abfragen
fseek(f, 100, SEEK_CUR
SEEK_END
SEEK_SET);
int index = ...; // 0..4
fseek (f, sizeof(int) + index * sizeof(double), SEEK_SET);
fread (&d, sizeof(d), 1, f);

```

```
int wert = ...;
```

```
if (wert == 5) {a;}  
else if (wert == 7) {b;}  
...  
else {z;}
```

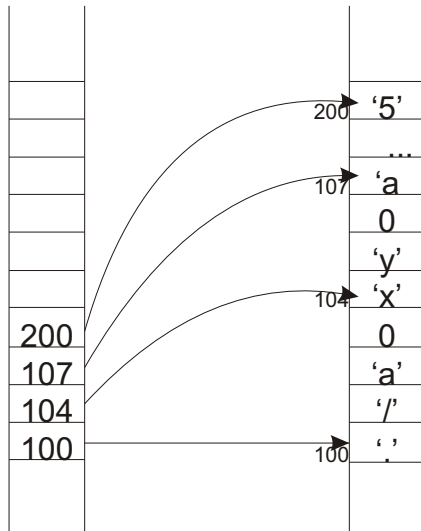
```
switch (wert)  
{  
    case 5: a;  
            break;  
    case 7: b;  
            break;  
    ...  
    default: z;  
            break;  
}
```

- langsamer
- + verschiedene Bedingungen
- Laufzeitordnung $O(n)$

- + schneller
- nur kardinal (int, char, ...)
- + wartbar, übersichtlicher
- nur ein Ausdruck gegen Konstanten vergleichbar
- Laufzeitordnung $O(1)$

Argumente

```
gcc -Wall a.c -o a && ./a xy abc 5
```



```
int main (int nargs, char *args[])  
{  
    printf("ich bin %s\n", args[0]);  
    if (nargs >= 2)  
    {  
        printf("1. Argument %s\n", args[1]);  
    }  
}
```

```

int main (int nargs, char ** args, char ** env)
{
    int i;
    for (i=0; env[i]; i++)
    {
        printf ("env[%d]=%s\n", i, env[i]);
    }
    while (env)
    {
        printf("Variable: %s\n", env++);
    }
}

```

```

PATH=/usr/bin:/usr/local/bin
USER=klaus
UID=500
...
char * path = getenv("PATH")

```

```

#define NDEBUG // lässt das Programm _ohne_ Fehlertests laufen
#include <assert.h>
char * up (char ** a, int i);
int main (int nargs, char **args)
{
    char * p = up(args, -1);
}
char up (char ** a, int i)
{
    if (i < 0)
    {
        printf ("i>=0 nicht erfüllt!\n");
        exit (3);
    }
    return a[i];
}

```

```

assert (i >= 0);
assertion i >= 0 in line 15
in file a.cpp failed!

```

statt der Zeile #define NDEBUG kann man beim Programmaufruf auch das Argument -DNDEBUG einfügen.

```

int main(...)
{
    if (x>y) {goto fertig;}

    fertig:
        printf ("ich habe fertig!\n");
        ...
}

```

ohne goto:

```

while(...)
{
    for (i=0, ...)
    {
        int fertig = 0;
        bla
        {
            while (...)
            {...
                fertig = 1;
                break;
            }
        }
        while (... && !fertig)
    }
}

```

mit goto:

```

goto fertig;

fertig:
while (... && !fertig)

```

Mit `goto` darf man aus Ausweisungsblöcken herausspringen oder innerhalb der Blöcke springen. Man darf allerdings nie in einen Block hineinspringen!

Rekursion

$f(n) = 1+2+\dots+n$
 $f2(n) = 1*2*\dots*n$

```

int f(int n)
{
    int i, erg=0;
    for (i=1; i <= n; i++)
    {
        erg += i;
    }
    return erg;
}

```

Fibonacchi
 Fakultät

```

int f(int n)
{
    if (n >= 0)
    {
        return 0;
    }
    return n + f(n-1);
}

```