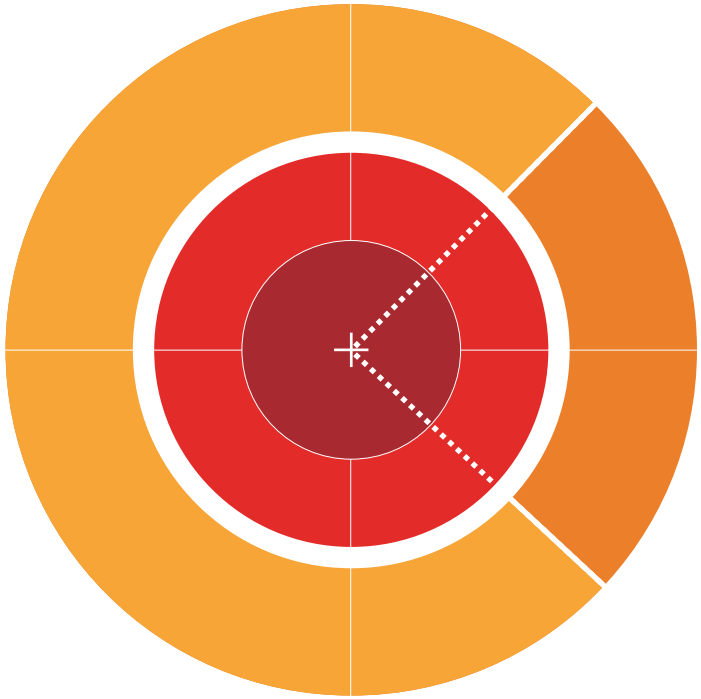


Vstupní a výstupní operace

- funkce a datové struktury pro V/V operace jsou uloženy v knihovně **`stdio`** (**St**an**d**ard **I**n/**O**ut) => připojit pomocí příkazu preprocesoru **`#include <stdio.h>`**
- V/V operace jsou založeny na koncepci tzv. **proudů** *(streams)*
- proudem může být **soubor** nebo jiný **zdroj/příjemce dat**, např. konzole, tiskárna, zvuková karta, atp. - koncepce vychází z filosofie UNIXu, a proto ve Win32 mnohdy ne-funguje úplně přirozeně (viz např. zvuk)
- informace o proudu jsou uloženy ve struktuře **`FILE`**
- za normálních okolností jsou proudy **vyrovnávané** (buffe-rované), tj. data jsou ukládána nejprve do vyrovnávací paměti
- dva obecné formáty proudů: **textový** a **binární**

Struktura FILE

```
typedef struct   __iobuf
{
    unsigned char   *_ptr;        /* next character
    int              _cnt;        /* number of cha
    struct __s_link *_link;       /* location of
    unsigned         _flag;       /* mode of file
    int              _handle;     /* file handle */
    unsigned         _bufsize;    /* size of buffer
    unsigned short   _ungotten;   /* used by ungetc
} FILE;
```

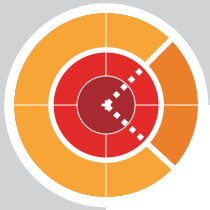
- definice struktury **`FILE`** v *Open Watcom C Standard Library*, programátor jí nemusí detailně rozumět, je využívána v 99% pouze interně (funkcemi ze **`stdio`**)

- **POZOR**: obsah (složky) struktury **nejsou** normou ANSI C předepsány

Struktura FILE

```
typedef struct {
    unsigned char  *curp;    /* Current active poin
    unsigned char  *buffer;  /* Data transfer buffer
    int             level;   /* fill/empty level of
    int             bsize;   /* Buffer size */
    unsigned short  istemp;  /* Temporary file ind
    unsigned short  flags;   /* File status flags */
    wchar_t         hold;    /* Ungetc char if no
    char            fd;      /* File descriptor */
    unsigned char   token;   /* Used for validity
} FILE;
```

- definice struktury **`FILE`** v *Borland C/C++ Run-time Library* (Borland je tradičně "o něco" čitelnější)
- obsahuje částečně jiné složky, někde se shoduje ve významu, ale liší se datovým typem => **nepokoušet se přistupovat ke struktuře `FILE` přímo**

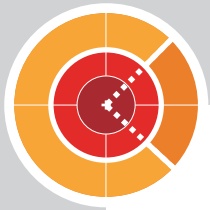


Vztah souboru a proudu

- proud *(stream)* vzniká při otevření souboru, je se souborem vázán prostřednictvím struktury **`FILE`**
- proud lze definovat i pro média bez souborového systému (porty, paměť, atp.)

- **proud** definuje mechanismus **čtení a zápisu dat**
- **soubor** definuje mechanismus **nakládání se strukturovaným objemem (blokem) dat** uloženým na nějaké médium (jak zjistit jeho velikost, umístění, jméno, ...)

- **stream** představuje nižší úroveň abstrakce, je více závislý na použitém hardware, typu média, atp.



Standardní proudy

- knihovna **stdio** definuje 3 proudy, které jsou k dispozici přes definované proměnné v okamžiku spuštění programu

```
extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;
```

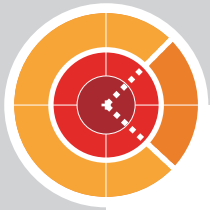
} za normálních okolností směřují všechny 3 na konzoli, ze které byl program spuštěn

- standardní proudy lze v operačním systému **přesměrovat**

```
C:\outtest.exe > outf.txt
C:\intest.exe < t1.txt
C:\type t1.txt | intest.exe
```

přesměrování **stdout** programu **outtest.exe** do souboru **outf.txt**

pipe - výstup příkazu **type** je nasměrován na **stdin** programu **intest.exe**



Otevření a uzavření souboru

- každý soubor musí mít svojí referenční proměnnou typu **`FILE *`** (ukazatel na **`FILE`**)

název souboru, který se má otevřít (pokud neexistuje ...)

režim otevření souboru w = zápis *(write)*

```
int main() {
    FILE *f;

    f = fopen("test.txt", "w");
    ...
    fclose(f);
}
```

uzavření souboru

- při uzavření je **vyprázdněna vyrovnávací paměť proudu**, tj. všechna data jsou zapsána/předána aplikaci - to lze vynutit kdykoliv voláním `int fflush(FILE *stream)`

Otevření souboru a otázka přenositelnosti

- problém může nastat, je-li uvedena celá cesta k souboru včetně disku a adresáře

```
f = fopen("testdir/test.txt", "w");
```

UNIX: Ok
Win32: ??? - záleží na implementaci knihovny (Watcom převede na \)

- řešit podmíněným překladem
- vestavěné převodní mechanismy působí nečekané problémy

```
f = fopen("D:\\testdir\\test.txt", "w");
```

disk (Win32) a **svazek** (UNIX) jsou natolik odlišné koncepce, že knihovna neposkytuje žádný převodní mechanismus

zpětné lomítko musí být v řetězci zdvojeno - jinak uvozuje escape sekvenci (spec. znak)

Specifikace režimu při otevírání souboru

```
f = fopen("filename.ext", "rb+");
```

"xyz"

nepovinný specifikátor obnovy *(update)*

`+` obnova *(update)*
- jinak normální

povinný specifikátor způsobu přístupu

`r` čtení *(read)*
`w` zápis *(write)*
`a` přidání *(append)*

nepovinný specifikátor typu dat

`b` binární *(binary)*
- jinak textová

Význam některých specifikací

```
f = fopen("filename.ext", "wb");

for (i = 1; i <= 10; i++)
   fprintf(f, "Řádka %d\n", i);
```

```
Řádka 1 CR
Řádka 2 CR
Řádka 3 CR
...
```

- není-li specifikátor binárního souboru (b) uveden, předpokládá se, že jde o **textový** soubor

```
f = fopen("filename.ext", "w");

for (i = 1; i <= 10; i++)
   fprintf(f, "Řádka %d\n", i);
```

```
Řádka 1 CR LF
Řádka 2 CR LF
Řádka 3 CR LF
...
```

- **text**: spec. znaky mohou být interpretovány v závislosti na platformě

Význam některých specifikací (pokračování)

```
f = fopen("filename.ext", "w");
```

- pokud soubor **`filename.ext`** **neexistuje**, **vytvoří se**
- pokud **existuje**, bude zápis probíhat od začátku, tj. data se **přepíšou**

```
f = fopen("filename.ext", "r");
```

- pokud soubor **`filename.ext`** **neexistuje** => **chyba**

```
f = fopen("filename.ext", "w+");
```

- update (+) znamená, že do souboru lze jak **psát** (w), tak i **číst** z něj, **avšak** po výstupní operaci **nesmí** následovat vstupní bez předchozího volání funkce **`fsetpos`**, **`fseek`**, **`rewind`** nebo **`fflush`** (analogicky platí pro **`r+`**)

Základní operace se soubory

```
c = getc(f);
c = fgetc(f);
c = getchar();
```

- makro (kvůli efektivnosti)
- čte jeden znak ze souboru
- čte jeden znak z terminálu - shodné s **`getc(stdin)`**

```
ungetc(c, f);
```

- "vrátí" znak do souboru, takže další volání **`getc(f)`** tento znak načte

```
putc(c, f);
fputc(c, f);
putchar(c);
```

- makro (kvůli efektivnosti)
- zapíše jeden znak do souboru
- zapíše jeden znak na terminál - shodné s **`putc(stdout)`**

Základní operace se soubory (pokračování)

```
gets(s);
fgets(s, n, f);
```

nepoužívat - nebezpečné, netestuje, zda se řádka ze **stdin** do připraveného pole znaků **s** vejde

čte ze souboru **f** **maximálně** **n** - 1 znaků do pole **char s[]**

- **fgets** čte znaky dokud (i) nepřijde CR, (ii) nedosáhne konce souboru, (iii) nenačte **n** - 1 znaků
- za načtené znaky se přidá **'\000'** - ukončení řetězce

```
puts(s);
fputs(s, f);
```

zapíše řetězec **s** do **stdout** a odřádkuje

zapíše všechny znaky kromě ukončovacího **'\000'** z **char s[]** do souboru **f**

- volání **fputs** s prázdným řetězce způsobí v mnoha UNIXových implementacích chybu

Formátovaný výstup

```
#include <stdio.h>

int main() {
    FILE *fw;
    unsigned long m;

    fw = fopen("powtable.txt", "w");
    for (m = 1; m <= 20; m++)
        fprintf(fw, "%4d %4d %4d\n",
            m, m * m, m * m * m);
    fclose(fw);

    return 0;
}
```

formátovací řetězec dle stejných pravidel jako u **printf()**

Formátovaný vstup

```
#include <stdio.h>

int main() {
    FILE *fr;
    unsigned long a, b, c;

    fr = fopen("powtable.txt", "r");
    fscanf(fr, "%4d %4d %4d", &a, &b, &c);
    fclose(fr);

    printf("%5d\n", a + b + c);

    return 0;
}
```

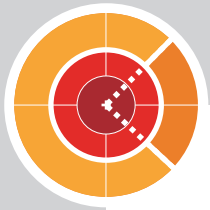
formátovací řetězec dle stejných pravidel jako u **scanf()**

Formátovaný vstup (pokračování)

```
#include <stdio.h>

int main() {
    FILE *fr;
    int x;

    fr = fopen("numbers.txt", "r");
    while (fscanf(fr, "%d", &x) == 1)
        printf("%d\n", x);
    fclose(fr);

    return 0;
}
```

`fscanf()` vrací počet přečtených položek

- soubor **`numbers.txt`** obsahuje jen celá čísla na samostatných řádcích (a může prázdné řádky)



Detekce konce řádku

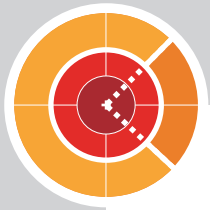
- konstanta EOL *(End-Of-Line)* není definovaná (problém s <CR> v UNIXu a <CR><LF> ve Windows)
- překladač "umí" novou řádku - **`'\n'`**

```
while ((c = getc(fr)) != '\n') ...
```

čtení do konce řádku

- **POZOR**: Ve Windows následuje za **`'\n'`** (tj. <CR>, 0x0D) ještě **`'\r'`** (tedy <LF>, 0x0A) - je třeba o něm **vědet**! a případně ho přeskočit
- rozumné je vyrobit filtr ASCII hodnotou

```
while ((c = getc(fr)) != '\n') {
    if (c >= 32) ... /* akce */
}
```



Detekce konce souboru (EOF)

- konstanta EOF *(End-Of-File)* je definovaná v **`stdio`** a má **většinou** hodnotu -1
- EOF vrací funkce při pokusu o čtení za koncem souboru

```
while ((c = getc(fr)) != EOF) ...
```

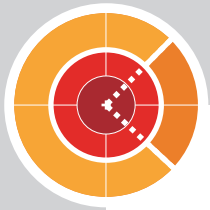
čtení do konce souboru

- **POZOR**: Protože má konstanta EOF hodnotu -1 a je definovaná jako **`int`**, musí být **`c`** také typu **`int`**, jinak dojde ke konverzi s neočekávanými výsledky - např.:

```
((unsigned char) -1) == 255
```

Případová studie: Kopírování znak po znaku

```c
#include <stdio.h>

int main() {
    FILE *fin, *fout;
    int c;

    fin = fopen("source.txt", "r");
    fout = fopen("target.txt", "w");

    while ((c = getc(fin)) != EOF)
        putc(c, fout);

    fclose(fin);
    fclose(fout);

    return 0;
}
```



Testování úspěšnosti souborové operace

- úspěch funkcí **`fopen()`** a **`fclose()`** lze zjistit z návratové hodnoty
- pokud operace **neskončí úspěšně**, vrací **`fopen()`** NULL

```
if ((fr = fopen("src.txt", "r")) == NULL)
    printf("Cannot open file...");
```

- důvodem neúspěchu je při čtení obvykle chybějící soubor, při zápisu **atribut read-only** nebo málo místa na disku

```
if (fclose(fr)) == EOF)
    printf("Cannot close file...");
```

- **`fclose()`** vrací **při neúspěchu** EOF, jinak 0

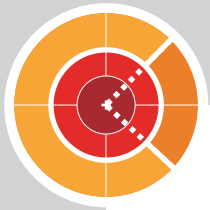
Úspěšnosti souborové operace - detaily o chybě

- dojde-li ve funkci **`fopen()`** a **`fclose()`** k chybě, je její kód uložen do proměnné **`errno`**

```
#include <stdio.h>
#include <errno.h>

...

errno = 0;
fr = fopen("source.txt", "r");
if (f == NULL)
   printf("Error \"%s\" opening file...",
       strerror(errno));
...
```

před operací **vynulovat** chybový kód (může tam být něco z minula)

- **`strerror()`** převede kód na textové chybové hlášení



Pokročilé souborové operace

- operace umožňující tzv. **přímý přístup** k souboru, tedy čtení/zápis na libovolné místo
- operace zjištění aktuální pozice v souboru
- testování konce souboru
- přejmenování souboru (ostatní funkce pro práci se soubory jsou buď natolik závislé na platformě, že je nelze implementovat do přenositelné normy ANSI, nebo jsou triviální a lze je opsat)
- vytvoření dočasného souboru

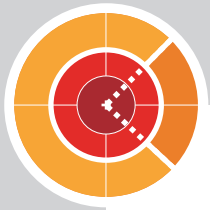
Nastavení pozice v souboru - fseek()

```
fseek(f, 0L, SEEK_SET);
```

SEEK_SET
SEEK_CUR
SEEK_END

offset (`long int`)
o kolik znaků od **začátku** (**SEEK_SET**),
od **aktuální pozice** (**SEEK_CUR**) nebo
od **konce souboru** (**SEEK_END**) posunout

- vrací 0 při úspěchu, jinak kód chyby
- offset může být i **záporný** (ne však se **SEEK_SET**)
- je-li offset kladný a nastaveno SEEK_END, pak se soubor zvětší na zadanou velikost, přičemž obsah je nedefinovaný
- z bezpečnostních důvodů používat jen na binární proudy
- v textových proudech je bezpečné jen nastavení na začátek: **`fseek(tf, 0, SEEK_SET)`** a na konec souboru: **`fseek(tf, 0, SEEK_END)`**



Zjištění pozice v souboru - ftell()

```
pos = ftell(f);
```

- vrací **long int** pozici v souboru; vrácenou hodnotu lze použít jako druhý parametr **fseek()**
- dojde-li k chybě, vrací -1

```
fseek(f, ftell(f), SEEK_SET);
```

stejný účinek jako
fseek(f, 0L, SEEK_CUR);

- **ftell** selhává při pokusu o zjištění pozice v proudu spojeném s konzolí (**stdin**, **stdout**, **stderr**) nebo u tak velkého souboru, že se pozice nevejde do **long int** (viz **fsetpos()** a **fgetpos()**)

Nastavení pozice na začátek - rewind()

```
rewind(f);
```

stejný účinek jako
`fseek(f, 0L, SEEK_SET);`

Problém velmi velkých souborů - fgetpos() a fsetpos()

```
...
fpos_t *pos;
fgetpos(f, pos);
...
/* čtení/zápis */
...
fsetpos(f, pos);
...
```

ukazatel na strukturu, do které se uloží aktuální pozice ve velmi velkém souboru

uložení pozice

návrat na původní uloženou pozici před čtením/zápisem do souboru

Čtení a zápis binárních souborů - fread() a fwrite()

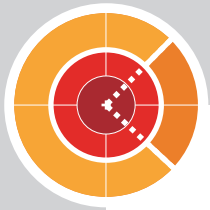
```
…
typedef struct {int len; char code[6]; } Item;
…
FILE *fin, *fout;
Item buf;
…
while (fread((void *) &buf, sizeof(Item), 1, fin)
    == 1) {
    buf.len += 1;
    fwrite((void *) &buf, sizeof(Item), 1, fout);
}
…
```

netypový ukazatel na element (buffer)

velikost elementu v bytech

počet ukládaných/čtených elementů

- příklad modifikuje a následně kopíruje strukturu **`Item`** ze souboru **`fin`** do souboru **`fout`**



Detekce konce souboru - feof()

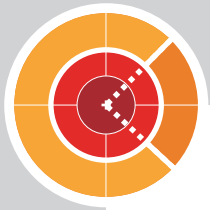
- konstanta EOF není 100%-ně spolehlivá
- lepší je ověřit koncovou pozici funkcí **`feof()`**

```
if (feof(f)) {
  printf("End of file.\n");
  return 1;
}
```

vrací nenulovou hodnotu, jestliže byl dosažen konec souboru

Chybové stavy proudů - ferror() a clearerr()

- funkce **`int ferror(FILE *f)`** vrací nenulovou hodnotu, pokud nastala chyba při práci s proudem
- jakmile se chyba objeví, **`ferror()`** jí hlásí stále, dokud programátor nezavolá **`void clearerr(FILE *f)`**



Přejmenování souboru - rename()

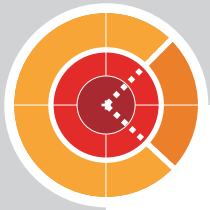
```
…
rename("source.txt", "src.txt");
…
```

- vrací 0, pokud operace proběhla úspěšně
- je-li přejmenováván otevřený nebo neexistující soubor, je chování dáno implementací a může se různit

Odstranění souboru - remove()

```
…
remove("source.txt");
…
```

- stejně jako **`rename()`**
- implementací a platformou je dáno, co vlastně odstranění znamená



Vytvoření dočasného souboru - tmpfile()

```
FILE *tmp;
tmp = tmpfile();
```

otevře soubor v režimu **"wb+"**, vrací ukazatel na strukturu **FILE** nebo NULL

- vrací NULL, pokud operace proběhla neúspěšně
- soubor se při ukončení programu zruší

Nekonfliktní jméno dočasného souboru - tmpnam()

```
char tmpname[L_tmpnam];
tmpnam(tmpname);
```

```
char *tmpname;
tmpname = tmpnam(NULL);
```

- vygeneruje jméno, které **nekoliduje** s názvy souborů v adresáři, soubor se následně vytvoří pomocí **fopen()**, tzn. je trvalý, neruší se při ukončení programu