

# **WORKSHOP 4**

## **FILHANTERING**

Lennart Steinvall

(baserad på tidigare DV-  
kursers föreläsning om filer)



UMEÅ UNIVERSITET

# INNEHÅLL

- Genomgång WS 2
- Filhantering och kommandoradsparametrar
- Grupparbete med övningsuppgifter



# FILHANTERING

- Ni har jobbat mycket med inläsning och utskrifter (IO) i era program. Detta har dock skett främst till skärmen.
- En fil kan ses som en följd av minnesceller som behåller sitt innehåll även när datorn är avstängd.
- För att kommunicera med filer för vi in ett mellanled, en *ström*.
  - Programmen kommunicerar med strömmen, öppnar den för att läsa från filen. Vi säger dock oftast att vi öppnar filen.
  - För att komma åt strömmen används filpekare.



# FILHANTERING

- Filpekaren `FILE *` definieras i `stdio.h`, men hur den exakt ser ut är oviktigt.
- Även utskrifter till skärm och inläsning från tangentbord sköts med filpekare.
- Strömmarna benämns `stdout` och `stdin`.
- Dessutom finns en filpekare för utskrifter av felmeddelanden, `stderr`.



# TEXTFILER OCH BINÄRA FILER

- Textfiler är lätta för människor att läsa medan binärfiler oftast kräver program.
  - Källkoden för ett c-program är en textfil medan den körbara koden till programmet är en binärfil.
- Binärfilen lagrar datat exakt så som datorn representerar det medan textfilen är en representation av detta data.
- Lagras talet 267 i en textfil blir det som tre tecken '2', '6' och '7'.
  - Tecknen har en ascii kod tex har '2' ascii koden 50 som lagras som 00110010 binärt.
- Lagras talet 267 i en binärfil lagras det som 00000000 1100100.



# ÖPPNA OCH STÄNGA FILER

- För att kunna läsa eller skriva till en fil måste den först öppnas.

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

```
FILE *fopen(const char* filename, const char *mode);
```

Returnerar: Filpekare vid OK, annars NULL.

- Vi öppnar filer genom att ange bl. a. dess namn.

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

```
int main(void) {
```

```
    FILE* ifp;
```

```
    FILE* ofp;
```

```
    /* open for reading */
```

```
    ifp = fopen("any_file", "r");
```

```
    /* open for writing */
```

```
    outfp = fopen("another_file", "w");
```

```
    ...
```

```
}
```



# ÖPPNA OCH STÄNGA FILER

- Vid öppnandet anger man också vad man vill kunna göra med filen. Detta för att undvika misstag.
- Dessa kan vara
  - "r" Öppnar textfil för läsning.
  - "w" Öppnar textfil för skrivning.
  - "a" Öppnar textfil för att lägga till.
  - "rb" Öppnar binär fil för läsning.
  - "wb" Öppnar binär fil för skrivning.
  - "ab" Öppnar binär fil för att lägga till.
  - "r+" Öppnar fil för läsning och skrivning.
  - "w+" Öppnar fil för skrivning och läsning.
  - ... O.S.V.



# ÖPPNA OCH STÄNGA FILER

- När man är klar med en fil bör den stängas.

```
#include <stdio.h>
int fclose(FILE *stream);
Returnerar: 0 vid OK, annars EOF.
```

```
#include <stdio.h>

int main(void) {
    FILE* fp;
    /* open for reading */
    fp = fopen("any_file", "r");
    /* do something with the file */
    /* close the file */
    fclose(fp);
}
```





# FILER

- Slutet på en fil markeras med ett speciellt slut-på-filen-tecken.
  - C använder sig av ett särskilt värde, EOF för att representera detta.
- Många av input-funktionerna från `stdio.h` returnerar EOF när de läser , vilket vi kan använda oss av för att styra programflödet:

```
while ( scanf ("%d", &myNumber) != EOF ) {  
    printf ("%d\n", myNumber) ;  
}
```




# LÄSA OCH SKRIVA PÅ FILER

- Motsvarigheten till `printf` och `scanf` för filer heter `fprintf` och `fscanf`.
- `int fprintf(FILE *stream, const char* format, ...);`
  - Returnerar: Antalet utskrivna tecken om OK, annars negativt värde.
  - `fprintf` fungerar på samma sätt som `printf` med den skillnaden att resultatet skrivs till en filpekare, `stream`, i stället för till skärmen.
- `int fscanf(FILE* stream, const char* format, ...);`
  - Returnerar: Antalet lyckosamma matchningar. Vid fel EOF.
  - `fscanf` läser text från en filpekare och inte från tangentbordet.
- `char *fgets(char* str, int n, FILE *stream);`
  - Returnerar: En rad från en fil, NULL om något går fel
  - `fgets` läser en rad från en fil tills `n-1` tecken lästs, nyradstecken lästs eller filslutstecken lästs, NULL returneras om filslut nås utan att något tecken lästs in
- Alla funktionerna ovan finns i `stdio.h`



# EXEMPEL: RADNUMRERING

- Antag att vi har ett antal kodfiler och att vi vill skriva ut dem på papper med radnumrering.
  - Det finns visserligen ett antal olika sätt att åstadkomma detta, men vi bestämmer oss för att göra det med hjälp av ett litet C-program.
- Programmet ska heta `rownumbers` och kunna anropas med två filnamn. Tex  
`rownumbers infil.c utfil.txt`  
 **Kommandoradsparametrar!**
- Programmet ska läsa från `infil.c` och skapa en radnumrerad kopia i `utfil.txt`
- Vi antar att ingen av filerna programmet ska hantera har mer än 1000 rader eller någon rad med mer än 300 tecken.



# VAD MÅSTE PROGRAMMET GÖRA?

1. Kontrollera att den fått rätt indata
  - namn på två filer, en att läsa ifrån och en att skriva till
2. Försöka öppna filerna
3. Sätta en räknare till 1
4. För varje rad i infilen
  1. Skriva räknare + raden i utfilen
  2. Öka räknaren med 1
5. Stänga filerna



# KOMMANDORADSPARAMETRAR

- Hittills har vi sett program som tar interaktiv input.
  - Programmet säger till användaren att det förväntar sig indata och läser indata från tangentbordet.
  - I många fall är detta opraktiskt eller rentav omöjligt.
- Vi vill i stället kunna ge programmet den indata det behöver direkt när det startar.
  - Vi ger programmet ***programparametrar (eller kommandoradsparametrar)***, som i praktiken omvandlas till parametrar för funktionen main.
  - Anges direkt efter namnet på den körbara filen  
`rownumbers infil.c utfil.txt`



# KOMMANDORADSPARAMETRAR

- För att ta emot parametern måste programmets main-funktion se ut så här:

```
int main(int argc, char ** argv) {  
    ...  
}
```

- **argc** talar om för oss **hur många** parametrar som skickats till programmet.
- **argv** är en **array av strängar** och innehåller själva parametrarna.



# I PRAKTIKEN...

```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Antal parametrar: %d\n", argc);
    for(int i = 0; i < argc; i++){
        printf("Parameter %d: %s\n", i, argv[i]);
    }
}
```

Om vi kör programmet

```
./a.out one two three
```

så får vi resultatet:

```
Antal parametrar: 4
Parameter 0: ./a.out
Parameter 1: one
Parameter 2: two
Parameter 3: three
```

Programnamnet är också  
en parameter!



UMEÅ UNIVERSITET

# REPRIS: VAD MÅSTE PROGRAMMET GÖRA?

1. Kontrollera att den fått rätt indata
  - namn på två filer, en att läsa ifrån och en att skriva till
2. Försöka öppna filerna
3. Sätta en räknare till 1
4. För varje rad i infilen
  1. Skriva räknare + raden i utfilen
  2. Öka räknaren med 1
5. Stänga filerna





# VI BEHÖVER LITE VARIABLER

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

```
#define BUFSIZE 300 /* Max 300 char per line */
```

```
int main(int argc, char ** argv){
```

```
char line[BUFSIZE]; /* Buffer a line at the time from input */
```

```
FILE * infilep; /* Pointer to input file */
```

```
FILE * outfilep; /* Pointer to output file */
```

```
int rowNumber = 1;
```



# 1. KONTROLLERA INDATA

```
/* check that there are two input parameters (in- and outfile)*/  
if(argc <= 2){  
    fprintf(stderr, "Usage: rownumbers <input file name> ");  
    fprintf(stderr, "<output file name>\n");  
    return 0;  
}
```

- Kontrollera alltid att indata är det du förväntat dig först av allt.
  - Rätt antal parametrar och rätt typ av parametrar
- Är datat felaktigt och det ej går “reparera”, avbryt programmet omedelbart.



## 2. ÖPPNA FILERNA – FELHANTERING!

```
/* Try to open the input file */
infilep = fopen(argv[1], "r");

if(infilep == NULL){
    fprintf(stderr, "Couldn't open input file %s\n", argv[1]);
    return 0;
}

/* Try to open the output file */
outfilep = fopen(argv[2], "w");

if(outfilep == NULL){
    fprintf(stderr, "Couldn't open output file %s\n", argv[2]);
    return 0;
}
```

- Finns en massa som kan gå fel:
  - Filen finns inte
  - Vi vill skriva till en fil men saknar rättigheter
  - ...
- Går något fel returnerar fopen NULL.
- Måste kollas, försöker man läsa en fill med en NULL-pekkare kraschar programmet.



## 3-4. LÄSA/SKRIVA TILL FILERNA

```
/* Read a line at a time from the input file.  
 * Write the line, preceded by a row number,  
 * to the output file.  
 */  
while(fgets(line, BUFSIZE, infilep) != NULL) {  
    fprintf(outfilep, "%-3d ", rowNumber);  
    fprintf(outfilep, "%s", line);  
    rowNumber++;  
}
```



## 5. STÄNGA FILERNA

```
fclose(infile);  
fclose(outfile);  
return 0;  
}
```

- `fclose` tar bort strukturen som skapats för att hålla rätt på tillgångsinformationen samt utför ett par andra ”renhållningsuppgifter”.
- Att stänga alla filer programmet öppnat är också synnerligen viktigt!



# BINÄRA FILER

- Om ett program vill lagra data som ska läsas av ett annat C-program kan den göra det i en binär fil utan att bry sig om att konvertera till strängar. Programmet lagrar helt enkelt datorns interna representation av data.
- Ett annat program kan sedan läsa den binära filen, förutsatt att det känner till vad det är som lagrats.
- Lägg till ett b till den andra inparametern i fopen:
  - `binaryp = fopen("myFile.bin", "rb");`



# BINÄRA FILER

- För att skriva till en binär fill används `fwrite`
  - `size_t fwrite (const void *array, size_t size, size_t count, FILE *stream);`
    - `array` är en pekare till en plats i minnet, där `fwrite` ska börja läsa.
    - `size` är storleken på den datatyp som ska skrivas till filen, angedd i bytes.
    - `count` är antalet element vi vill skriva till filen.
    - `stream` är filpekaren.
- `fwrite` kommer att skriva de första `count` blocken av storlek `size` från minnet, med start vid `array`, till filen kopplad till `stream`.



# BINÄRA FILER

- För att läsa från en binär fill används `fread`
  - `size_t fread (const void *array, size_t size, size_t count, FILE *stream);`
    - `array` är en pekare till en plats i minnet, där `fwrite` ska börja läsa.
    - `size` är storleken på den datatyp som ska skrivas till filen, angedd i bytes.
    - `count` är antalet element vi vill skriva till filen.
    - `stream` är filpekaren.
- `fread` kommer att läsa de första `count` blocken av storlek `size` från filen kopplad till `stream` och skriva till minnet, med start vid `array`.

