Programmering i ANSI-C/Filhantering

Från Wikibooks
< Programmering i ANSI-C

Innehåll

Programmering i ANSI-C

Källkoden|Villkorssatser|Preprocessorn|Kompilatorn|Nyckelord|Standardströmmarna

Filhantering|Felhantering|Programexempel|Standardbibliotek|Tabeller

Filhantering

Filhanteringen i ANSI-C görs med hjälp av strömmar. För att hantera strömmar så finns det ett antal funktioner i standardbiblioteket "stdio.h". Vid öppnandet av en fil så skapas en ström som egentligen är ett nummer, den öppna filens ID-nummer. Vidare operationer på filen görs sedan med detta nummer/strömmen som referens. Det är på sådant sätt möjligt att ha flera filer öppna samtidigt där samtliga är knutna till unika strömmar, vilket kan vara nyttigt om man vill flytta data mellan filer till exempel. Det maximala antalet filer som den aktuella processen kan hålla öppna är angivet i makrot "FOPEN_MAX" och det maximala antalet filer datorsystemet kan hålla öppna på en och samma gång kan utläsas ur makrot "SYS_OPEN", (båda definieras i "stdio.h").

Förutom filfunktionerna som är beskrivna på den här sidan så finns det ett antal som gör det möjligt att operera direkt på den öppna filen. Många av dessa funktionen är rester från den tid då datorer främst använde magnetband som lagringsmedia. Alla dessa funktioner kan ersättas med operationer på minnet. Därför är den i princip enda metoden, som brukas idag, att läsa in hela filen till minnet och sedan göra det som skall göras med den och därefter på nytt spara filen till lagringsmediet, (numera oftast en hårddisk).

Se: Standardbibliotek -> stdio.h för en kort beskrivning av dessa funktioner.

Se även: Standardströmmarna för att se hur data matas in och ut (visas), till exempel till och från en filbuffer.

Att öppna en fil

För att knyta en fil till en ström används funktionen "fopen" som är definierad på följande sätt:

```
FILE *fopen( const char *filnamn, const char *mod );
```

"FILE" är en datatyp, (definierad i "stdio.h"), som anger en pekare till den aktuella strömmen. När filen öppnas retuneras denna pekare om allt gått väl. Om något fel däremot föreligger så retuneras NULL-pekaren. "*filnamn" är en pekare till en textsträng som innehåller sökvägen till filen som skall

öppnas. "*mod" är en pekare till en sträng som anger i vilken mod filen skall öppnas. Här följer en lista på dom olika modsträngarna som kan användas:

Tecken i modsträngen:

Tecken	Beskrivning
r	Öppnar fil för läsning (read).
r+	Öppnar fil för läsning och skrivning. Det här är den normala moden om man vill bearbeta data i en redan existerande fil.
w	Skapar fil för skrivning (write). Om filen redan finns kommer den att ersättas med den nya.
w+	Skapar fil för läsning och skrivning. Om filen redan finns kommer den att ersättas med den nya.
a	Öppnar en fil för skriva tillägg på slutet (append). Finns inte filen tidigare så skapas den.
b	Anger att filen är av binärtyp vilket betyder att ett tecken är 8 bitar.
t	Anger att filen är av texttyp vilket betyder att ett tecken är 7 bitar. (ASCII var ursprungligen 0-127)

```
/* Öppna en binärfil för bearbetning */

FILE *min_fil;                          /* skapar filpekare */
char *filnamn = "C:/ANSI-C/data.bin";  /* sökvägen till filen */

min_fil = fopen ( filnamn, "r+b" );    /* öppnar filen */
```

Det finns även en funktion, "freopen", som används för att öppna en ny fil till samma ström som den tidigare filen:

```
FILE *freopen ( const char *filnamn, const char *mod, FILE *ström );
```

Skillnaden jämfört med "fopen" är att här anges även den tidigare filens ström som en av ingångsparametrarna, i övrigt är funktionen densamma som hos "fopen".

Att stänga en öppen fil

När filen är färdigbearbetad så måste stömmen stängas det görs med funktionen "fclose" som är definierad på följande sätt:

```
int fclose ( FILE *ström );
```

"*ström" är den pekare vi fick när filen öppnades. Returvärdet är 0, (noll), om allt gått väl annars retuneras värdet "EOF" som felindikator vilket gäller för de flesta filfunktioner, se felhantering för en närmare beskrivning.

```
fclose ( min_fil );
```

Om flera filer öppnats så kan alla strömmar stängas på en gång med funktionen "fcloseall":

```
int fcloseall ( void );
```

Returvärdet är ett heltal som anger antalet stängda filer. Om något fel uppstått retuneras värdet "EOF" som indikator.

```
int antal_filer;

antal_filer = fcloseall ( );
```

Att läsa från en fil

För att läsa ett valfritt antal poster, (vanligtvis hela filen), från början av en fil så används funktionen "fread" som är definierad på följande sätt:

```
size_t fread ( void *pekare, size_t storlek, size_t antal, FILE *ström );
```

"size_t" är en datatyp, definierad i "stddef.h", som används för att ange storleken på en dataobjekt, till exempel en minnescell eller fält av minnesceller. "pekare" anger adressen till minnet där det lästa avsnittet skall sparas. "storlek" anger storleken på objekten som skall läsas och "antal" hur många objekt som skall läsas. "ström" är pekaren vi fick när vi öppnade filen. Funktionen retunerar sedan antalet lästa objekt, inte bytes, så det totala antalet lästa bytes blir $bytes = \textbf{\textit{sizeof}}\ (\ storlek\ ) \cdot antal$. Om returvärdet är mindre än detta (returvärde < bytes) så har ett fel uppstått. **sizeof** ger storleken på dom datablock som skall läsas, (se nyckelord för en beskrivning).

Här ett exempel som läser filen vi öppnade tidigare, "min_fil", till minnet:

```
/* Läs från en öppen fil till minnet */

short int filbuffer [ 256 ];

fread ( filbuffer, sizeof ( short int ), 256, min_fil );
```

Först skapas en buffer dit filen skall läsas. Den måste vara tillräcklig stor[1] för att det lästa avsnittet skall rymmas helt. Sedan läses filen in till buffern med "fread". Eftersom buffern är av typen **short int** så anges "storlek" med hjälp av **sizeof** operatorn till samma typ, "sizeof (short int)". Sedan anges att 256 objekt av denna storlek skall läsas från strömmen "min_fil" till minnet.

[1] Om den är större så gör det inget men ett mindre minnesblock gör att läsningen av filen även kommer att skriva över dom datablock som ligger efter det angivna i minnet, vilket i sin tur brukar medföra att datorn kraschar, dyker, flippar, (valfri synonym), med tiden. Rekommenderas inte =).

Att skriva till en fil

För att skriva ett valfritt antal poster, (vanligtvis hela filen), från början av en fil så används funktionen "fwrite" som är definierad på följande sätt:

```
size_t fwrite ( void *pekare, size_t storlek, size_t antal, FILE *ström );
```

Samtliga parametrar beskrivs under "fread" ovan. Även returvärdet har samma funktion som hos "fread".

```
/* Skriv från minnet till en öppen fil */

short int filbuffer [ 256 ];

kod som på något sätt fyller filbuffern med data.

fwrite ( filbuffer, sizeof ( short int ), 256, min_fil );
```

Namnbyte och radering

För att byta namn på en fil används funktionen "rename" som är definierad på följande sätt:

```
int rename ( const char *tidigare, const char *nytt );
```

"*tidigare" är en pekare till en sträng som innehåller sökvägen till filen. "*nytt" är en pekare till det nya filnamnet. Om hela sökvägen anges så måste den vara densamma i hela utom själva filnamnet. Går allt väl retunerar funktionen 0. Om ett fel uppstått retuneras värdet -1 och den globala variabeln "errno" antar något av följande värden: EEXIST, ENOENT, ENOTSAM. Se Felhantering för en närmare beskrivning.

```
/* Döp om en fil */

char gammalt_namn[] = "C:/ANSI-C/Min_fil.bin";
char nytt_namn[]    = "C:/ANSI-C/Din_fil.bin";

rename ( gammalt_namn, nytt_namn );
```

Det finns även en funktion "remove" som raderar filer från lagringsmediet (normalt hårddisken). "remove" är definierad på följande sätt:

```
int remove ( const char *filnam );
```

"*filnamn" är en pekare till sökvägen för aktuell fil. Går allt väl retunerar funktionen 0. Om ett fel uppstått retuneras värdet -1 och den globala variabeln "errno" antar något av följande värden: EACCES eller ENOENT. Se Felhantering för en närmare beskrivning.

```
/* Radera en fil */

char filens_namn[] = "C:/ANSI-C/Min_fil.bin";

remove ( filens_namn );
```

Felhantering

Den allmänna felhanteringen i ANSI-C beskrivs på sidan felhantering.

Om ett fel uppstått vid filhantering så måste strömmens felindikator nollställas annars kommer felet att kvarstå. Nollställer även "EOF" indikatorn för strömmen.

```
void clearerr ( FILE *ström );
```

"*ström" är pekaren till filen som vi fick när den öppnades. Funktionen ger inget returvärde så det finns inget sätt att ta reda på om anropet lyckats. Även funktionen "rewind" har samma effekt på indikatorerna.

```
/* Nollställ felindikatorn */

clearerr ( min_fil );
```

Det går även att testa om en öppen fil har har läs eller skrivfel. Det görs med funktionen "ferror":

```
int ferror ( FILE *ström );
```

Om filen har ett fel retuneras ett värde som är sklilt från noll, annars noll ("0").

```
/* Testa om filen har läs eller skrivfel */

if ( ferror ( min_fil ) )
{
    puts ( "Filen har läs eller skrivfel!!!\n" );
    return ( 0 );
}

Här fortsätter programmet om filen är felfri.
```

Först testar **if** satsen om det retunerade värdet är SANT (icke noll). Om så är fallet skrivs ett felmedelande ut och funktionen retunerar (avslutas) annars forsätter programmet.