Lesen und Schreiben von Binärfiles

F

Bei der Speicherung von Information in "unformatierten" oder Binärfiles werden Daten nicht als ASCII-codierte Buchstaben oder Ziffern, sondern als direktes Abbild der Hauptspeicherinhalte, d.h. in der maschinenspezifischen Darstellung, in das File geschrieben. Das hat neben dem gegenüber ASCII-Files verringerten Platzbedarf noch den Vorteil, daß kein Bit Information durch Konversionen verloren geht, man also ein Programm an einer beliebigen Stelle abbrechen, die Daten sichern und später mit dem Programm forfahren kann, als ob man es nie unterbrochen hätte.

Will man in F ein Binärfile lesen oder schreiben, so muß man es mit der Option `form="unformatted"` im `open`-Statement öffnen

```
open(unit=...,form="unformatted",...)
```

Die eigentlichen Lese- und Schreibbefehle sehen genauso aus wie bei formatierten Files, nur wird die `fmt`-Option weggelassen, also

`read(unit=`*unit*`)` *variable_list*

bzw.

`write(unit=`*unit*`)` *expression_list*

Eine Eigentümlichkeit von Fortran ist, daß in Binärfiles die Daten in Records (Sätzen) organisiert sind, wobei beim Schreiben des Files jeder `write`-Befehl einen neuen Record erzeugt. Umgekehrt wird beim Lesen mit jedem `read` an den Beginn das nächsten Records gesprungen. Stehen also in einem Record mehr Daten als die Variablenliste aufnehmen kann, so werden sie ignoriert. Insbesondere überspringt der Befehl

`read(unit=`*unit*`)`

einen Record, ohne Daten zu lesen.

Wenn nicht anders angegeben, werden Files in Fortran im *sequentiellen* Modus geöffnet, d.h. es wird ein Record nach dem anderen gelesen bzw. geschrieben. (Daneben gibt es aber auch sogenannte Direktzugriffsfiles.) Mit den Befehlen

`backspace(unit=`*unit*`)`

und

`rewind(unit=`*unit*`)`

kann man jedoch an den Beginn des vorhergehenden Records bzw. an den Beginn des Files springen.

C

In C müssen auf POSIX-kompatiblen Systemen beim Öffnen von Binärfiles keine besonderen Optionen angegeben werden. Insbesondere kennt C auch keine Recordstruktur, sondern man kann einen Lese- oder Schreibvorgang an beliebigen Stellen (Byte-Adressen) des Files beginnen lassen.

Der Befehl für das Lesen von einem unformatierten File ist

`fread(`*bufferpointer*`,`*size*`,`*count*`,`*filepointer*`);`

Dabei werden maximal *count* Elemente der Größe *size* (in Bytes) von dem File, auf das *filepointer* zeigt, in den Buffer an der Adresse *bufferpointer* übertragen. Der Rückgabewert der Funktion ist die Anzahl der tatsächlich übertragenen Elemente. Zur Angabe der Größe eines Elements verwendet man günstigerweise den `sizeof`-Operator. Will man z.B. zehn `float`-Werte von einem File mit Filepointer `fp` in einen Vektor `x` einlesen, so gibt man an

```
fread(&x[0],sizeof(float),10,fp);
```

(`&x[0]` ist die Adresse des ersten Elements von `x`.)

Der Befehl für das unformatierte Schreiben auf ein Binärfile ist vollkommen analog

`fwrite(`*bufferpointer*`,`*size*`,`*count*`,`*filepointer*`);`

Ein Nachteil von `fread/fwrite` gegenüber `read/write` in F besteht darin, daß die C-Funktionen keine Liste von Werten oder Variablen verarbeiten können. Will man also mit einem Lese- oder Schreibbefehl mehrere Werte übertragen, so muß man sie in einen Buffer packen.

Ein File kann, ähnlich wie in F, mit

`rewind(`*filepointer*`);`

auf den Anfang positioniert werden. Eine beliebige Stelle eines Files kann man mit

`fseek(`*filepointer*`,`*offset*`,`*from*`);`

erreichen. Das File wird (für den nachfolgenden Lese- oder Schreibvorgang) *offset* Bytes relativ zu *from* positioniert; die speziellen Werte `SEEK_SET`, `SEEK_CUR` und `SEEK_END` für *from* bedeuten Anfang, gegenwärtige Position und Ende des Files. Mit

where`=ftell(`*filepointer*`);`

kann man die `long int`-Variable *where* auf die momentane Position im File setzen.