FEATURE

The Techie's Guide to C Part13

This month we're going to look at the very useful facility of reading the disk directory from a C language program.

STEVE RIMMER

Aside from providing you with a command line to type on and some basic utility programs, DOS has a lot of features which you'll only encounter at the programming level. It provides a myriad of "services" which are available to programs to make them easier to write. These range from simple things like printing a character to the screen right on through disk management and some local area network functions.

As DOS is, in fact, a disk operating system, it's not surprising that a lot of its services have to do with disks. These fall nicely into file management facilities, such as those we've been tapping into over the past few months, and directory management facilities. The latter group can be especially useful and kind of interesting if you're writing programs which work with files.

DOS... and hence C... provide you with tools to allow you to see what files are on your disk. These tools can also be used to check out whether a specific file is there. As DOS treats many different, and apparently unrelated, disk phenomena as files, these same tools will also allow you to locate things like subdirectories and the volume name of a disk.

This month we're going to see how these tools actually work.

First and Next

There are two functions under Turbo C which deal with reading a disk directory, called *findfirst* and *findnext*. Used together, these functions allow you to step through all the files in a directory which match a wild card specification.

Let's see what these functions do before we get into exactly how they do it. We'll search for all the C program files in the current directory, that is, "*.C".

We would begin by passing "*.C" to *findfirst*. Assuming that there was at least one file that matched this specification, it would place the file name in an as yet undiscussed data structure and return zero. We would the pass the aforementioned data structure to *findnext*. If it found a second file which matched the specification, it too would return zero and place the name of the second file in the data structure. We would continue to call *findnext* until it did not return zero, at which time we could assume that all the matching files had been found.

The data structure in question is called an *ffblk*. It looks like this.

```
structffblk{
char ff_reserved[21];
char ff_attrib;
unsigned ff_ftime;
unsigned ff_fdate;
long ff_fsize;
char ff_name[13];
};
```

This is everything which can be known about a file from its directory entry.

The first field, *ff_reserved*, is fairly meaningless for our purposes... it's used by DOS to communicate between *findfirst* and subsequent calls to *findnext*. The *ff_attrib* field is the file attribute of the directory entry which is found, something we'll discuss later on. The *ff_ftime* and *ff_fdate* fields are the file time and date of creation, the same ones you see in a DOS DIR listing. The *ff_fsize* field is the number of bytes in the file. Finally, *ff_name* is a C style string which holds the file's name.

The proper form of *findfirst* is like this.

```
structffblkf
intn;

  n=findfirst(path,&n,0);
```

In this case, *path* is a string which holds a file specification. It can include a path component... you could pass "C:\TC*.C", for example, or more correctly "C:\\TC*.C", as the C precompiler gobbles up individual backslashes. The path can be in upper or lower case.

The third argument is the file attribute. We'll leave this as zero for the time being.

The value in *n* will be zero if *findfirst* found a file that matched the file specification.

You'd use *findnext* like this.

```
n=findnext(&f);
```

This assumes that *f* has been through *findfirst* at least once.

The following bit of code will print a listing of every file in your current directory.

```
dir()
{
struct ffblk f;
int n;

n=findfirst("*.*",&f,0);
if(n==0){
do{
puts(f.ff_name);
n=findnext(&f);
}while(n==0);
}
}
```

This is actually a bit more involved that it needs to be. There's a rather more elegant... if somewhat more obtuse... structure for doing the same thing.

```
dir()
{
struct ffblk f;
int n;

for(n=findfirst("*.*",&f,0);!n;n=findnext(&
f))
puts(f.ff_name);

}
```

If you understand how a *for* statement works under C, you'll have no trouble figuring out what this does.

Practical Directory

Let's digress for a moment and look at some practical considerations in using this facility. We're going to write a simple sorted directory program.

In this program, we'll allocate a buffer big enough to hold a reasonable number for file names... a thousand should do... and then load the *ffblk* structs returned by *findfirst* and *findnext* into the buffer in the order they're found. We'll then use the Turbo C *qsort* function to sort them into alphabetical order. The *qsort* function is a fairly complex bit of work... we'll leave its complete discussion for another time.

Having sorted the names into alphabetical order, they can be displayed on the screen. There are four potentially useful bits of information associated with each file entry, these being the file name itself, its size in bytes, its time of creation and its date of creation.

The time and date of creation of a file are formatted in a particularly obtuse way. Their various component parts are stored as bits in an unsigned integer. You can see how this works if you regard an integer as being sixteen bits.

The time is formatted line this

```
Bits 0 through 4  two second blocks
Bits 5 through 10 minutes
Bits 11 through 15 hours
```

We'll ignore the minutes field in this example. If we wanted to know the minutes portion of the file creation time, we would do this.

```
minutes=(n f.ff_fdate & 0x07e0)>>5;
```

The number being ANDed with the date value is a mask which selects the appropriate bits. If you consider that an integer is sixteen bits, this would be the binary representation of a number having all the bits from five through ten selected.

```
0000011111100000
```

This number in hex would be 0x07e0.

We must shift the result of the AND right by five bits to wind up with a proper integer value.

This is the format of the date.

```
Bits 0 through 4 day of the month
Bits 5 through 8 month
Bits 9 through 15 year past 1980
```

The month value runs from one through twelve. As such, if we store the month names in an array, this value minus one serves as the index into this array.

Here, then, in the program. This version of it is designed for use with a colour monitor. If you have a black and white tube, change the four colour defines at the top of the program to reflect this. For example,

```
#define name_atr LIGHTCYAN+(RED
<<4)
```

would become

```
#define name_atr WHITE
```

If you do have a colour monitor and you want to alter the colours I've chosen, note that there are two colours for each attribute. The leftmost one is the foreground colour. The rightmost one, red in this case, is the background colour. Screen text colours and how they work will be discussed in a future installment of this series.

```
/*
Sorted directory program
Copyright (c) 1989 Alchemy Mindworks

The source code for the current version of
this
program is available on a 5 1/4 inch floppy
disk
for $10.00 from Alchemy Mindworks Inc.,
P.O. Box
313, Markham, Ontario L3P 3J8.

"Perhaps we could tax sex... this would
allow us to tax not only the act but the
tax as well."
Attributed to a
highly placed Mulroney aid
*/
#include "stdio.h"
#include "dir.h"
#include "alloc.h"
#include "conio.h"

#define maxfile 1024
#define dirsize (sizeof(struct ffblk))

/* text attributes for the directory listing */
#define name_atr LIGHTCYAN+(RED
<<4)
#define size_atr CYAN+(RED<<4)
#define time_atr LIGHTBLUE+(RED
<<4)
#define date_atr BLUE+(RED<<4)

char *filetime(),*filedate();
char *buffer;
int count=0;

main(argc,argv)
int argc;
char *argv[];
{
char filespec[129];

if(argc>1)strcpy(filespec,argv[1]);
else strcpy(filespec,"*.*");

if((buffer=malloc(maxfile*dirsize))!=
NULL){
```

Techie's Guide to C Programming, Part 13

```
get_dir(filespec);
if(count){
sort_dir();
show_dir();
   }else puts("File not found");
free(buffer);
textattr(WHITE);
clrscr();
}else puts("Error allocating memory");
}

get_dir(s) /* load the buffer with the direc-
tory */
char *s;
{
struct ffblk f;
int n;

for(n=findfirst(s,&f,0);!n;n=findnext(&f))
{
if(f.ff_attrib!=0x10)
memcpy(buffer+(dirsize*count++),
(char*)&f,dirsize);
if(count>=maxfile) break;
}
}

sort_dir() /* sort the directory by file name
*/
{
int dircheck();

qsort(buffer,count,sizeof(struct ffblk),dir-
check);
}

dircheck(e1,e2)   /* compare two files */
char *e1,*e2;
{
struct ffblk *p1,*p2;

p1=(struct ffblk*)e1;
p2=(struct ffblk*)e2;
return(strcmp(p1->ff_name,p2-
>ff_name));
}

show_dir() /* display the directory */
{
struct ffblk f;
char b[30];
int i,n;

for(i=0;i<count;++i){
if(!(i%50)){
n=0;
gotoxy(1,25);
if(i)getch();
textattr(name_atr);
clrscr();
}
if(n>24)gotoxy(41,n-24);
elsegotoxy(1,1+n);
memcpy((char*)&f,
buffer+(dirsize*i),
dirsize);
textattr(name_atr);
cprintf("%-12.12s",f.ff_name);
textattr(size_atr);
cprintf("%7lu",f.ff_fsize);

textattr(time_atr);
cprintf(" %-6.6s",filetime(f.ff_ftime));

textattr(date_atr);
cprintf(" %s",filedate(f.ff_fdate));
++n;
}
gotoxy(1,25);
getch();
}

char *filedate(l) /* format a file date */
unsigned int l;
{
static char s[20];
static char m[12][4]={
"Jan","Feb","Mar",
"Apr","May","Jun",
"Jul","Aug","Sep",
"Oct","Nov","Dec"};
unsigned int day,month,year;

day=(l&0x001f);
month=(l&0x01e0)>>5;
year=(l&0xfe00)>>9;

sprintf(s,"%s %u,%u",m[month-
1],day,year+1980);
return(s);
}

char *filetime(l) /* format a file time */
unsigned int l;
{
static char s[10];
unsigned int min,hrs;

min=(l&0x07e0)>>5;
hrs=(l&0xf800)>>11;

   if(hrs < 13)
sprintf(s,"%u:%02.2ua",hrs,min);
   else sprintf(s,"%u:%02.2up",hrs-
12,min);
  return(s);
   }
```

If you call this program SDIR.C and type it into Turbo C or get it on a floppy disk, you can compile it to SDIR.EXE or SDIR.COM. If you type SDIR all by itself, you will see all the files in the current directory one page at a time. If you type SDIR *.C, for example, you'll see all the .C files.

This is a very interesting program to experiment with, and there are all sorts of things you can add to it. Next month, we'll talk about some of the other ways to use the directory search facilities, which might give you some ideas for making this program do additional things.