# The Technic Content of the disk directory from a Clanguage program.

STEVERIMMER

A side from providing you with a command line to type on and some basic utility programs, DOS has a lot of features which you'llonly 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 fileshad 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);

E&TT January 1990

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 timebeing.

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

You'duse 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()

structffblkf; intn;

```
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;
intn;
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

# Bits0through4 two second blocks Bits5through10 minutes Bits11through15 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=(nf.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.

# Bits0 through 4 day of the month Bits5 through 8 month Bits9 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 51/4 inch floppy disk

for \$10.00 from Alchemy Mindworks Inc., P.O. Box

313, Markham, OntarioL3P3J8.

"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; intcount=0;

main(argc,argv)
intargc;
char\*argv[];

char filespec[129];

if(argc>1)strcpy(filespec,argv[1]);
elsestrcpy(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 directory \*/ char\*s; { struct ffblk f; intn;

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
\*/
{

intdircheck();

qsort(buffer,count,sizeof(struct ffblk),dircheck);

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

p1=(structffblk\*)e1; p2=(structffblk\*)e2; return(strcmp(p1->ff\_name,p2->ff\_name)); }

show\_dir() /\* display the directory \*/
{
struct ffblk f;
char b[30];
inti,n;

for(i=0;i<count;++i) {
 if(!(1 % 50)) {
 n=0;
 gotoxy(1,25);
 if(i)getch();
 textattr(name\_atr);
 clrscr();
}</pre>

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) /\* formata filedate\*/ unsigned int l;

staticchars[20]; staticcharm[12][4]={ "Jan","Feb","Mar", "Apr","May","Jun", "Jul","Aug","Sep", "Oct","Nov","Dec" }; unsigned int day, month, year;

day=(1&0x001f); month=(1&0x01e0)>>5; year=(1&0xfe00)>>9;

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

char\*filetime(l) /\*formatafiletime\*/ unsigned intl; {

staticchars[10]; unsigned int min, hrs;

min=(1&0x07e0)>>5; hrs=(1&0xf800)>>11;

if(hrs < 13)
sprintf(s,"%u:%02.2ua",hrs,min);
else sprintf(s,"%u:%02.2up",hrs12,min);
return(s);</pre>

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.



\* Macintosh Plus Computer with Hard Drive

\* Macintosh Computer with external Hard Drive

\* Light Table

\* Draft Table

\* Stat Camera with developer

\* Assorted Software

\* Waxer

Looking to sell as a package. All reasonable offers considered. Perfect for complete Desktop Publishing Company set-up. All you need to get started... or to enhance existing set-up.

Phone anytime 9 a.m. to 5 p.m. David Rusk (416) 445-5600 Ext. 32.

# AMAZING SCIENTIFIC & ELECTRONIC PRODUCTS

GRA1 LC7 RUB4 8TC5 MCP1 EHL EML1 JL3 SD5 LEV1	ANTI GRAVITY GENERATOR 40 WART BURNING CUTTING LASER HI POWER PULSED DRILLING LASER 1 MILLION VOLT TESLA COIL HI VELICITY COIL GUN LASER LIGHT SHOW 3 METHODS ELECTRONIC (HYPNOTISM TECHNIQUES LOWER POWERED COIL GUN LAUNCHER JACOB LADOER 3 MODELS SEE IN THE DARK LEVITATION DEVICE	\$10.00 \$20.00 \$20.00 \$20.00 \$15.00 \$20.00 \$20.00 \$8.00 \$8.00 \$8.00 \$10.00 \$10.00 \$10.00
FMV1K - PFS1K - NIG7K -	3 MILE FM VOICE TRANSMITTER HAND CONTROLLED PLASMA FIRE SABER HI FLUX NEGATIVE ION GENERATOR	\$34.50 \$49.50 \$34.50
LHC2K -	VISIBLE SIMULATED 3 COLOR LASER	\$49.50
HOD1K -	HOMING/TRACKING BEEPER TRANSMITTER	\$44.50
LGU6K -	2.5 MW HAND-HELD VISIBLE LASER GUN	\$249.50
BTC3K -	250,000 VOLT TABLE TOP TESLA COIL	\$249.50
1062K -	ION RAY GUN, project energy without wires	\$129.95
TRETK -	- 3 MILE AL (TO TELEPHONE TRANSMITTER	\$79.50
LIST10 IPG70 ITM10 TAT30 PSP40 DNE10 LGU20 BLS10	ABSEMBLED IN OUR LABS INFINITY XMTR LSten in via phone lines INVISIBLE PAIN FIELD BLAST WAVE GENERATO 100000 VOLT INTIMIDIATOR UP TO 20' AUTOMATIC TELEPHONE RECORDING DEVICE PHASOR SON'C BLASTPWARE PISTOL ALL NEW 26' VIVIO COLORED NEON STOCK 5 TO TIMW VISIBLE FIED HENE LASER GUN 1000 WATT BLASTER DEFENSE WAND	\$190355 R\$74.50 \$99.50 \$24.50 \$89.50 \$74.50 \$199.50 \$199.50
EASY ORD	ERING PROCEDURE - TOLL FREE 1-800-221-1	705
	GRA1 - LC7 - HUBA - STC5 - MUCP1 - STC5 - HLS1 - EH1 - SD5 - LS5 - LS7 - LS7 - LL7 - EH1 - FMVTIK - PFSTIK - NIG7K - PFSTK - LHC2K - HO7IK - PGSK - LGU6K - STC3K - NIG7K - PGSK - TTGTK - PGSK - STC3 -	GRAT - ANTI GRANTY GENERATOR LT7 - AWUTT BURNING CUTTING LASER RUBA - HI POWER PULSED DRILLING LASER RUTA - HI POWER PULSED DRILLING LASER RUTA - HI POWER PULSED DRILLING LASER RUTA - LASER LIGHT SHOW 3 METHODS EHT - ELECTRONIC MYPNOTISM TECHNIQUES EMT - LOWER POWERD COLL GUN LAUNCHER JL3 - JACOB LADOER 3 MODELS S05 - SEE IN THE DARK LEVT - LEVITATION DEVICE FINTYIK - 3 MILE FM VOICE TRANSMITTER PFSIK - HAND CONTROLLED PLASMA FRE SABER MIGTK - HI RUX NEGATIVE ION GENERATOR PGSK - PLASMA LIGHTNING GLOBE LMC2K - VISIBLE SIMULATED 3 COLOR LASER MIGTK - HI RUX NEGATIVE ION GENERATOR PGSK - 25 MW HAND-HELD VISIBLE LASER GUN BTC3K - 25 MW HAND-HELD VISIBLE LASER GUN BTC3K - TELEKINETIC ENHANCEMPELECTRIC MAN WWINTK - 3 MILE AUTO TELEPHONE TRANSMITTER ISTOM - INFINITY XMTR LISEN IN VAR GENERATOR 1052K - OWN RAY GUN, PDICE IN OUR LAISE TIGTIO - INFINITY XMTR LISEN IN VAR GENERATOR INFORM - 100000 VOLT INFILID BLAST WAR GENERATOR INVISIBLE PINITR ID IN OUR LAISE INTO - INVISIBLE PINITR LISEN IN VAR PINITER ISTOM - INVISIBLE PINITR LISEN IN VAR PINITER ISTOM - INVISIBLE PINITELEPHONE TRANSMITTER INSTIM - 100000 VOLT INFILID BLAST WAR GENERATOR INVISIBLE PINITER IN VAR PINITER ISTOM - INVISIBLE PINITR LISEN IN VAR PINITER ISTOM - INVISIBLE PINITR LISEN IN VAR PINITER ISTOM - INVISIBLE PINITR LISEN IN VAR PINITER ISTOM - INVISIBLE PINITER DEVENDEN INSTICK LISTOM - INVISIBLE PINITR LISE AND HEND BLASER GUN BL310 - 10000 WATT BLASTER DEFENSE WAND EAST ORDERING PROCEDURE - TOLL FREE 1-800-221-11

EASY ORDERING PROCEDURE - TOLL FREE 1-800-221-705 or 24 HRS ON 1-803-673-4730 or FAX 11 T0 1-803-672-5406 VISA, MC, OHECK, MOINUS FUNDS, INCLUDE 1048 SHIPPING, ORDERS \$100.00 & UP ONLY ADD \$10.00 CATALDG \$1.00 OR FREE WITH ORDER.

INFORMATION UNLIMITED P.O. BOX 716, DEPT. ETI AMHERST, NH 03031