

Files

- Programs and data are stored on disk in structures called files
- Examples
 - Turbo C++ - binary file
 - Word 4.0 - binary file
 - lab1.c - text file
 - lab1.data - text file
 - term-paper - text file

Outline

Text Files

- Files
 - Text vs Binary
- File Pointer (FILE *)
 - Standard: stdin, stdout, stderr
- Open/Closing
 - fopen
 - modes ("r" "w" "a")
 - return values
 - fclose
 - return values

Outline (cont)

Text Files

- File input
 - fscanf
 - file pointer, format string, address list
 - return values
 - single character
 - return value
 - getchar,getc,fgetc,ungetc
- File output
 - fprintf
 - file pointer, format string, value list
 - return value
 - single character
 - return value
 - putchar,putc,fputc

Text Files

- All files are coded as long sequences of bits (0s and 1s)
- Some files are coded as sequences of ASCII character values (referred to as *text* files)
 - files are organized as bytes, with each byte being an ASCII character
- Other files are generally referred to as binary files

File Terms

- Buffer - a temporary storage area used to transfer data back and forth between memory and auxiliary storage devices
- Stream - files are manipulated in C with streams, a stream is a mechanism that is connected to a file that allows you to access one element at a time

File Pointers

- Each stream in C is manipulated with the file pointer type
- FILE *stream
 - FILE is a type containing multiple parts
 - file for stream, current element in file, etc.
 - FILE * is the address where the FILE type is located in memory
 - FILEs always manipulated as FILE *

Standard File Pointers

- <stdio.h> contains three standard file pointers that are created for you (each of type FILE *)
 - stdin - file pointer connected to the keyboard
 - stdout - file pointer connected to the output window/terminal
 - stderr - file pointer connected to the error window (generally the output window)/terminal

Interactive Processing

```
do {
    printf("Enter 4 nums\n");
    scanf("%d%d%d%d",
          &id,&p1,&p2,&p3);
    if (id != 0)
        process(id,p1,
                p2,p3);
} while (id != 0);
```

Enter 4 nums (0 to quit)
723 85 93 99
Enter 4 nums (0 to quit)
131 78 91 85
Enter 4 nums (0 to quit)
458 82 75 86
Enter 4 nums (0 to quit)
0 0 0 0

Batch Processing

- Text file takes the place of input
- Create a file lab.data:
723 85 93 99
131 78 91 85
458 82 75 86
- If the data is in a file, the program can directly read the file rather than prompting the user for the data

Structure of Files

- String of bits:
010000110110000101110100...
- Interpreted as ASCII numbers:
01000011 01100001 01110100 ...
67 97 116
- Files as ASCII:
67 97 116 115 32 97 110 100 10 68
111 103 115 10 0
- As characters:
Cats and\nDogs\n<EOF>
- In editor:
Cats and
Dogs

Structure of Text Files (cont)

- Two special characters
 - \n - end-of-line character
 - <EOF> - end-of-file marker
- File lab.data:
723 85 93 99
131 78 91 85
458 82 75 86
as a string of characters
723 85 93 99\n131 78 91 85\n458
82 75 86\n<EOF>

Manipulating User Files

- Step 1: open a stream connected to the file
 - fopen command
- Step 2: read data from the file or write data to the file using the stream
 - input/output commands
- Step 3: close the connection to the file
 - fclose command

fopen Command

- Syntax: `fopen("FileName", "mode");`
- File Name is an appropriate name for a file on the computer you are working on, example: "C:\My Files\lab.dat"
- Mode indicates the type of stream:
 - "r" - file is opened for reading characters
 - "w" - file is opened for writing characters (existing file deleted)
 - "a" - file opened for writing characters (appended to the end of the existing file)

fopen Command (cont)

- `fopen` returns a value of type `FILE *` that is a stream connected to the specified file
- if the `fopen` command fails, a special value, `NULL` is returned
- reasons for failure:
 - file doesn't exist (read)
 - can't create file (append)

fclose Command

- Syntax: `fclose(FilePointer)`
- The file pointer must be a stream opened using `fopen` (that remains open)
- `fclose` returns
 - 0 if the the `fclose` command is successful
 - special value `EOF` if the `fclose` command is unsuccessful

Open/Closing File

```
int main() {
    FILE *stream;
    if ((stream = fopen("lab.data", "r"))
        == NULL) {
        printf("Unable to open lab.data\n");
        return(1);
    }
    /* Read data from lab.data using FILE *
       variable stream */
    if (fclose(stream) == EOF) {
        printf("Error closing lab.data\n");
        return(2);
    }
}
```

fprintf Command

- Syntax: `fprintf(file, "Format", ValueList);`
- Works similarly to `printf`, but data sent to file rather than screen
 - `printf("Format", ValueList)` is a shorthand for `fprintf(stdout, "Format", ValueList)`
- `fprintf` returns the number of characters printed or `EOF` (-1) if an error occurs
- File pointer should be write/append stream

fscanf Command

- Syntax: `fscanf(file, "Format", AddrList);`
- Works similarly to `scanf`, but data received from file rather than keyboard
 - `scanf("Format", AddrList)` is a shorthand for `fscanf(stdin, "Format", AddrList)`
- `fscanf` returns the number of successful data conversions or `EOF` if end-of-file reached
- File pointer should be a read stream

fscanf/fprintf Example

```
if ((ins = fopen("part.data","r")) == NULL) {
    printf("Unable to open part.data\n");
    return(-1);
}
if ((outs = fopen("sumpart.data","w")) == NULL) {
    printf("Unable to open sumpart.data\n");
    return(-1);
}
while (fscanf(ins,"%d%d%d%d",&id,&p1,&p2,&p3) == 4)
    fprintf(outs,"%3d %3d\n",id,(p1 + p2 + p3));
fclose(ins);
fclose(outs);
```

Field Specification Revisited

%[Flg][W][Pr][Sz]Code	Floating-Point Codes:
Whole Number Codes:	f - standard float
d - decimal int	e - scientific notation (e)
o - octal int	E - scientific notation (E)
x - hexadecimal int (a-f)	g - f or e (shorter)
X - hex int (A-F)	G - f or E (shorter)
u - unsigned decimal int	Character Code: c
i - 0x, 0X hex, 0 oct	Percent sign: %%
Code count: n (print)	
#chars printed, extracted as value (similar to scan)	

Field Specification

Size possibilities:	Precision: . number
Whole number	print: float - digits after .
h - shortint	
l - longint	Flag:
Floating point	scan: * read, don't extract
l - double	print:
L - long double	* - use arg as width
Width: <i>number</i>	- left justify
print: #chars to use	+ add plus in front of nums
scan: max chars read	space - space if no sign
	0 - pad with zeros

Scanning Multiple Arguments

- fscanf will ignore white space to fill args
- Example:

```
while (fscanf(ins,"%d%d",&n1,&n2) == 2)
    printf("%d %d\n",n1,n2);
```

applied to:
25
30 31
32 40
produces as output:
25 30
31 32

Reading Characters

C provides functions for reading single chars:

- int getchar() - returns next char from keyboard
- intgetc(FILE *fp) - returns next char from fp
- int fgetc(FILE *fp) - returns next char from fp
- int ungetc(int oneChar, FILE *fp) - returns char oneChar to stream fp (but only one)

Note, all routines return an int, not a char, this is to allow for returning the value EOF (end-of-file), which is not a char

EOF is also returned if there is an error

Showing a File

```
FILE *ins;
int c;

if ((ins = fopen("file1","r")) == NULL) {
    printf("Unable to open file1\n");
    exit(0);
}

while ((c = fgetc(ins)) != EOF)
    putchar(c);

fclose(ins);
```

Writing Characters

C also provides functions for writing one character:

`int putchar(int c)` - prints char `c` to output window
`int putc(int c, FILE *fp)` - print char `c` to stream `fp`
`int fputc(int c, FILE *fp)` - print `c` to stream `fp`
Routines accept int args (chars are coerced)
Routines return EOF if there is a problem

Creating a File

```
FILE *outs;
int c;

if ((outs = fopen("file2", "w")) == NULL) {
    printf("Unable to open file2\n");
    exit(0);
}

while ((c = getchar()) != EOF)
    fputc(c, outs);

fclose(outs);
```

Copying a File

```
FILE *ins;
FILE *outs;
int c;

if ((ins = fopen("file1", "r")) == NULL) {
    printf("Unable to open file1\n");
    exit(0);
}

if ((outs = fopen("file2", "w")) == NULL) {
    printf("Unable to open file2\n");
    exit(0);
}

while ((c = fgetc(ins)) != EOF)
    fputc(c, outs);

fclose(ins);
fclose(outs);
```

Count # Lines, Chars

```
FILE *instream;
int c;
int linenum = 1;
int charcount = 0;

if ((instream = fopen("file3", "r")) == NULL) {
    printf("Unable to open file3\n");
    exit(-1);
}

while ((c = fgetc(instream)) != EOF) {
    if (c == '\n') {
        printf("%3d: %d\n", linenum, charcount);
        linenum++;
        charcount = 0;
    }
    else
        charcount++;
}

fclose(instream);
```

Reading to End of Line

- From keyboard:
`while (getchar() != '\n');`
- From file (with file pointer `fp`):
`while (fgetc(fp) != '\n');`
- Can be used to discard:
 - unneeded remainder of line
 - problematic input

Dealing with Problem Input (scan)

```
done = 0;
while (!done) {
    printf("Please enter number: ");
    if (scanf("%d", &num) == 1)
        done = 1;
    else
        while (getchar() != '\n');
```

Doing Your Own Formatted Input

```
int main() {
    int value;
    do {
        printf("Enter number (-1 to quit):");
        if (readInt(stdin,&value))
            printf(" You entered %d\n",value);
        else
            printf(" Unable to read value\n");
        while (fgetc(stdin) != '\n');
    } while (value != -1);
    return 0;
}
```

Removing White Space

```
void discardWhiteSpace(FILE *fp) {
    int ch;

    ch = fgetc(fp);
    while ((ch == ' ') || (ch == '\t')
           || (ch == '\n'))
        ch = fgetc(fp);
    ungetc(ch,fp);
}
```

Reading Integer Magnitude

```
int readIntBody(FILE *fp, int *res) {
    int result;
    int ch;
    ch = fgetc(fp);
    if ((ch >= '0') && (ch <= '9')) {
        result = ch - '0';
        do {
            ch = fgetc(fp);
            if ((ch >= '0') && (ch <= '9'))
                result = result * 10 + ch - '0';
        } while ((ch >= '0') && (ch <= '9'));
        ungetc(ch,fp);
        *res = result;
        return 1;
    }
    else {
        *res = 0;
        return 0;
    }
}
```

Safely Reading an Integer

```
int readInt(FILE *fp, int *res) {
    int ch;
    int retval;
    discardWhiteSpace(fp);
    ch = fgetc(fp);
    if (ch == '-') {
        retval = readIntBody(fp,res);
        *res *= -1;
        return retval;
    }
    else if ((ch >= '0') && (ch <= '9')) {
        ungetc(ch,fp);
        retval = readIntBody(fp,res);
        return retval;
    }
    else
        return 0;
}
```

Reading a Float

- Need to add mechanisms for:
 - decimal point
 - digits after decimal point
 - exponent e
 - exponent minus sign
 - exponent magnitude
- Not an easy undertaking!