



C Programming

File I/O

Adam Sampson (based on material by Greg Michaelson)
School of Mathematical and Computer Sciences
Heriot-Watt University

Files

FILE

- Type for the C standard library's representation of an open file
- In Unix, all files are sequences of bytes
 - i.e. no difference between text and binary files

`FILE *name;`

- Declares name as the address of a FILE value

`#include <stdio.h>` for file I/O types and functions

Files

`FILE * fopen(path, mode)`

- Open file with the name `path` (a string)
- Allocates a new `FILE` object, and returns the address
 - If an error occurs, returns the special pointer value `NULL`
- `mode` is a string saying how to open it:
 - `"r"` – read
 - `"w"` – write – create new empty file – delete old version!
 - `"a"` – append i.e. write at end of existing file
 - (On non-Unix: text by default, `"rb"` etc. for binary)

Files

```
void fclose(FILE *)
```

- Close file, freeing the FILE object
- You must explicitly close a file when you're done using it – if you don't, your changes may not get written to disk
- Some I/O functions handle characters as `ints`
 - 0–255 – byte values in the file
 - Also EOF == special **end of file** constant

Character values

- Each character corresponds to an integer
- ASCII – American Standard Code for Information Interchange – is the Unix standard encoding
- Characters in ranges (A-Z, 0-9...) have sequential values

char	int	hex	char	int	hex	char	int	hex
'0'	48	30	'A'	65	41	'a'	97	61
'1'	49	31	'B'	66	42	'b'	98	62
...				
'9'	57	39	'Z'	90	5A	'z'	122	7A

Character values

- This is often useful for character processing
- `char ch` is:
 - lower case, if: `'a' <= ch && ch <= 'z'`
 - upper case, if: `'A' <= ch && ch <= 'Z'`
 - digit, if: `'0' <= ch && ch <= '9'`
- If `ch` is a digit then `ch - '0'` is its value
 - e.g. `'7' - '0' → 55 - 48 → 7`

Character I/O

```
int getc(FILE *)
```

- Returns next byte from the file (as `int`)
- Returns EOF if at end of file

```
int putc(int, FILE *)
```

- Puts a byte to the file
- Returns EOF on failure

Command-line arguments

```
int main(int argc, char *argv[])
```

- `argc` is the number of command line arguments
 - Includes name of executable: “`ls mydir`” is 2 arguments
- `argv` is an array of arguments
 - Each is a `char *`, a C string (older code: `char **argv`)
 - `argv[0]` = name of executable
 - `argv[1]` = 1st argument
 - `argv[2]` = 2nd argument (etc.)

exit

`exit(int)`

- End program immediately, as if you had returned from the `main` function
- Argument is the **exit code** returned to the operating system, like `main`'s return value
 - 0 means success
 - Non-0 is failure
- `exit` defined in `<stdlib.h>` header

Manual pages

- On Unix-like systems, to view quick documentation for `foo`, run the command: **man foo**
- Manual sections: 1=commands, 2=operating system calls, 3=C library, 5=file formats...
- Useful for C functions: **man fopen**
 - What to `#include`, arguments, return value...
- If a name is in multiple sections, you can specify the section explicitly: **man 3 exit**

copy.c

File copy

Example: file copy

```
$ copy file1 file2
```

- Check correct number of arguments
- Open `file1`
 - If this fails, print an error and exit
- Open `file2`
 - If this fails, print an error and exit
- Copy characters from `file1` to `file2` until EOF
- Close files

File copy

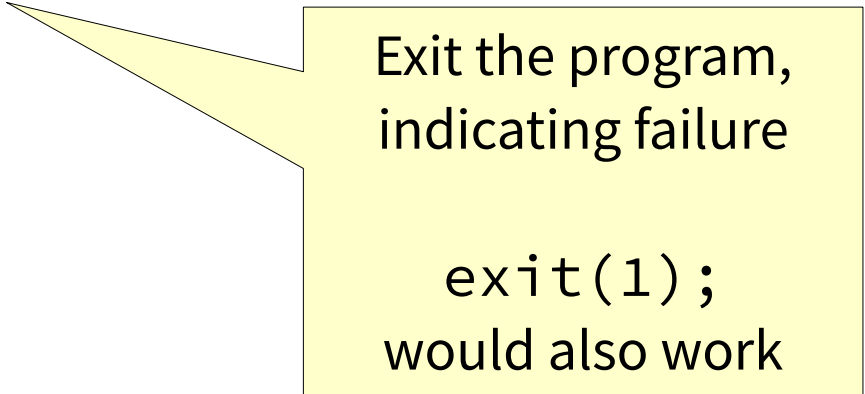
```
#include <stdio.h>

void copy(FILE * fin, FILE * fout)
{
    int ch;
    ch = getc(fin);
    while (ch != EOF) {
        putc(ch, fout);
        ch = getc(fin);
    }
}
```

File copy

```
int main(int argc, char *argv[])
{
    FILE *fin, *fout;

    if (argc != 3) {
        printf("copy: wrong number of arguments\n");
        return 1;
    }
```

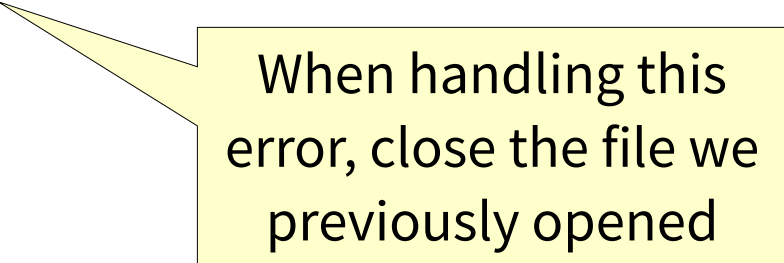


Exit the program,
indicating failure

`exit(1);`
would also work

File copy

```
fin = fopen(argv[1], "r");  
if (fin == NULL) {  
    printf("copy: can't open %s\n", argv[1]);  
    return 1;  
}  
fout = fopen(argv[2], "w");  
if (fout == NULL) {  
    printf("copy: can't open %s\n", argv[2]);  
    fclose(fin);  
    return 1;  
}
```



When handling this error, close the file we previously opened

File copy

```
    copy(fin, fout);  
    fclose(fin);  
    fclose(fout);  
  
    return 0;  
}
```


Keyboard and display I/O

- `stdin`, `stdout`, `stderr`
... are built-in, automatically-opened `FILE *s`
- `stdin` is connected to the keyboard
 - `int getchar() = getc(stdin)`
- `stdout` is connected to the display
 - `int putchar(int) = putc(int, stdout)`
- `stderr` is also the display – for error messages

Formatted I/O

- We've seen `printf` and `scanf` already...
- `fscanf(FILE *, "format", exp1, exp2...)`
 - Like `scanf`, but for any `FILE *` (`scanf` uses `stdin`)
- `fprintf(FILE *, "format", exp1, exp2...)`
 - Like `printf`, but for any `FILE *` (`printf` uses `stdout`)

Formatting numbers

- In `printf`, can precede format character with field width and precision – for example:
- `%3d`
 - Decimal integer
 - At least 3 characters wide
- `%4.2f`
 - Double
 - At least 4 chars wide
 - 2 digits after decimal point

Reading binary numbers

	Input	Value
		0
• Initial value is 0	1 1011	$2 * 0 + \mathbf{1} = 1$
• For each binary digit:	1 011	$2 * 1 + \mathbf{1} = 3$
– Multiply value by 2	0 11	$2 * 3 + \mathbf{0} = 6$
– Add value of binary digit	1 1	$2 * 6 + \mathbf{1} = 13$
	1	$2 * 13 + \mathbf{1} = 27$

Reading binary numbers

- Write a function to read a binary number from a file:
`int getBinary(FILE * fin, int * ch)`
- Parameters for:
 - input file
 - next character variable
- Read each character from file into a non-local variable
 - Pass address of character variable
 - Use indirection (*) when accessing it

Binary number converter

File bdata.txt contains:

0
1
10
11
100
101
...
1111

\$./binary bdata.txt

0
1
2
3
4
5
...
15

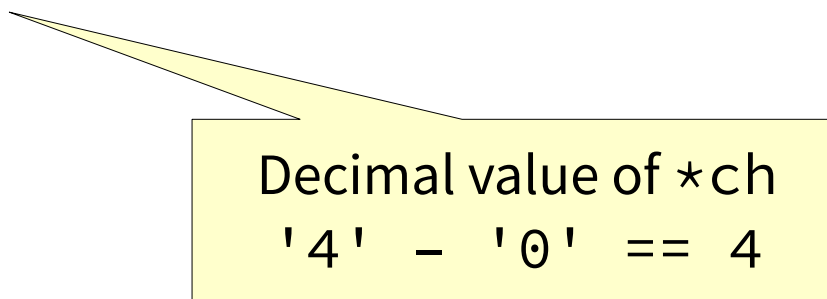
binary.c

Convert binary numbers

Convert binary numbers

```
#include <stdio.h>
```

```
int getBinary(FILE * fin, int *ch)
{
    int val;
    val = 0;
    while (*ch == '0' || *ch == '1') {
        val = 2 * val + (*ch) - '0';
        *ch = getc(fin);
    }
    return val;
}
```



Decimal value of *ch
'4' - '0' == 4

Convert binary numbers

```
int main(int argc, char *argv[])
{
    FILE *fin;
    int ch;
    int val;

    if (argc != 2) {
        printf("getBinary: wrong number of arguments\n");
        return 1;
    }
    if ((fin = fopen(argv[1], "r")) == NULL) {
        printf("getBinary: can't open %s\n", argv[1]);
        return 1;
    }
}
```

Assignment inside expression –
allowed, but can be hard to read!

Convert binary numbers

```
    ch = getc(fin);
    while (1) {
        while (ch != '0' && ch != '1' && ch != EOF)
            ch = getc(fin);
        if (ch == EOF)
            break;
        val = getBinary(fin, &ch);
        printf("%d\n", val);
    }
    fclose(fin);

    return 0;
}
```