File IO part 2

Files

- Programs and data are stored on disk in structures called files
- Examples

  a.out – binary file

  lab1.c - text file

  term-paper.doc - binary file

Overview

File Pointer (FILE *)

Standard: stdin, stdout, stderr

Or fopen

Usage:

```
FILE* file=fopen(filename, modes);
      modes: “r”, “w”, or “a” : read, write, or append

   char ch=fgetc(file);
fclose(file);
```

File Pointers

- Each stream in C is manipulated with the file pointer type
- FILE *stream
  - FILE is a struct 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

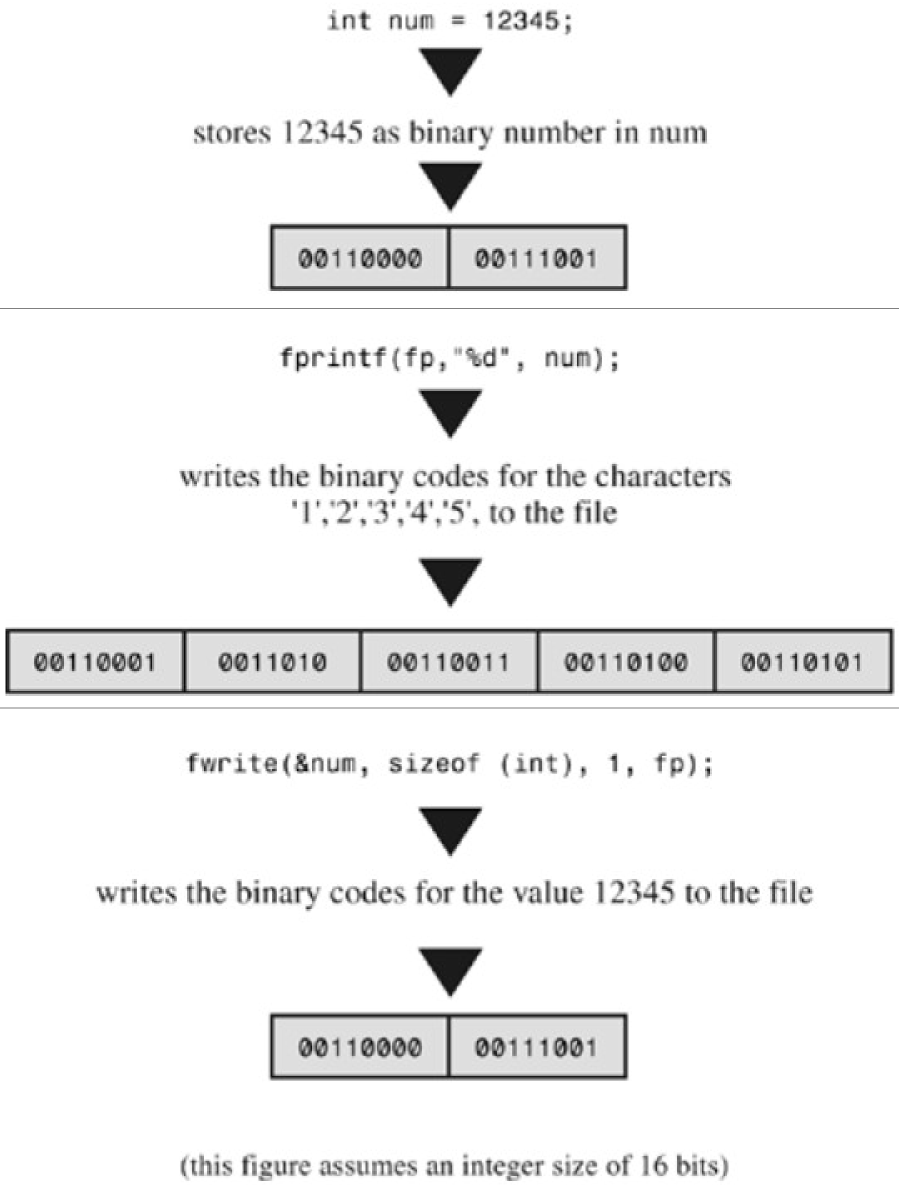
- <stdio.h> contains three standard file pointers that are created for you (each of type FILE *)
  - stdin - file pointer connected to the keyboard
  - stdout - file pointer connected to the output window/terminal
  - stderr - file pointer connected to the error window (generally the output window)/terminal

Text Files and Binary 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
- UTF-8 for unicode texts
- Other files are generally referred to as binary files

Memory

Figure 13.3. Binary and text output.



```
int num = 12345;
```

stores 12345 as binary number in num

00110000	00111001

Text file

```
fprintf(fp,"%d", num);
```

writes the binary codes for the characters
'1','2','3','4','5', to the file

00110001	0011010	00110011	00110100	00110101

Binary file

```
fwrite(&num, sizeof (int), 1, fp);
```

writes the binary codes for the value 12345 to the file

00110000	00111001

(this figure assumes an integer size of 16 bits)

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

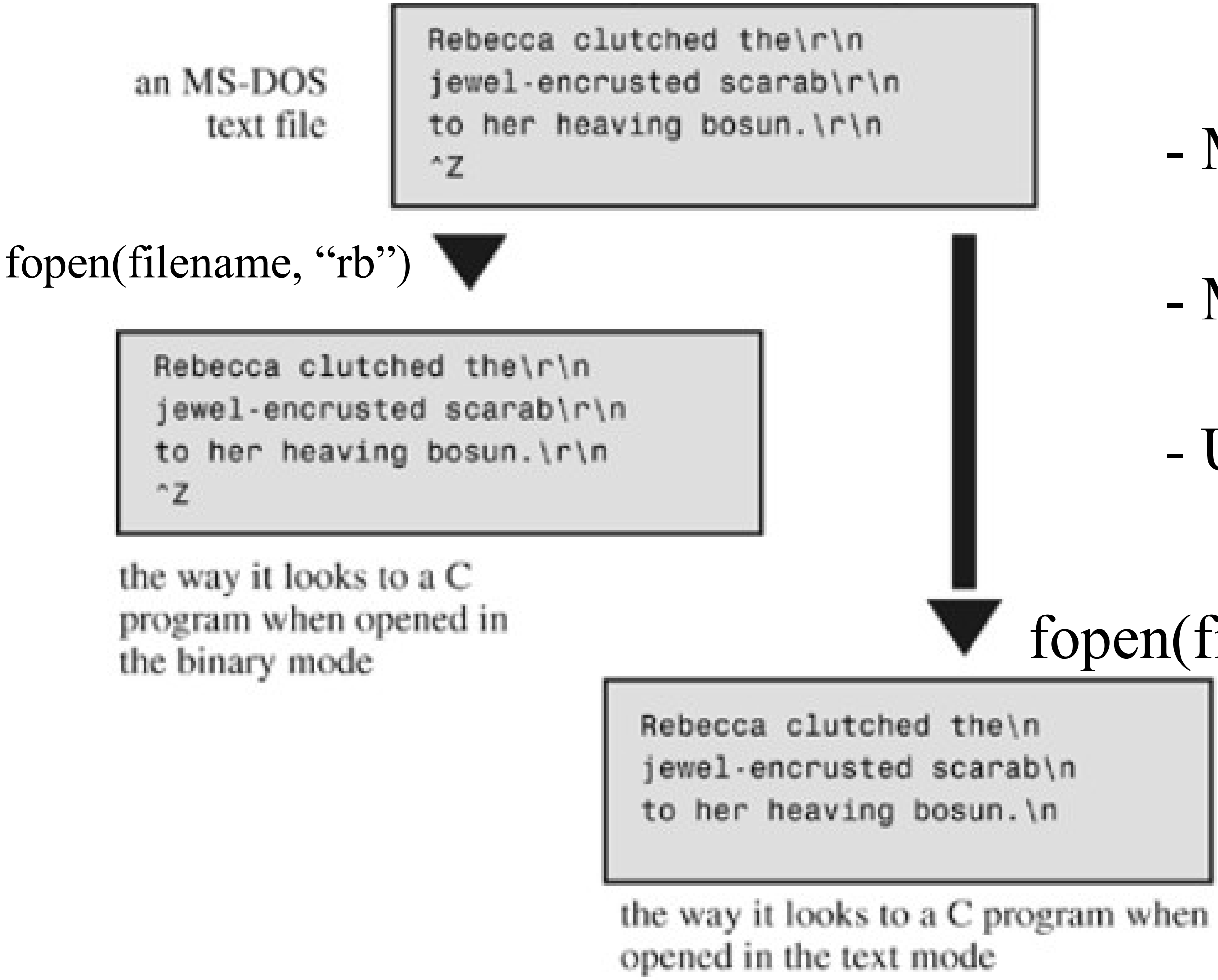
- File lab.data:

```
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>
```

Windows and old mac text files are different!



an MS-DOS text file

```
Rebecca clutched the\r\n
jewel-encrusted scarab\r\n
to her heaving bosun.\r\n
^Z
```

fopen(filename, “rb”)

```
Rebecca clutched the\r\n
jewel-encrusted scarab\r\n
to her heaving bosun.\r\n
^Z
```

the way it looks to a C program when opened in the binary mode

fopen(filename, “r”)

```
Rebecca clutched the\n
jewel-encrusted scarab\n
to her heaving bosun.\n
```

the way it looks to a C program when opened in the text mode

- MS-DOS

  \r\n

- Mac (prior to Mac OS X)

  \r

- Unix/linux

  \n

IO functions

Text Files

File input

- fscanf(file pointer, format string, address list)
- single character
  - getchar, getc, fgetc
  - ungetc

File output

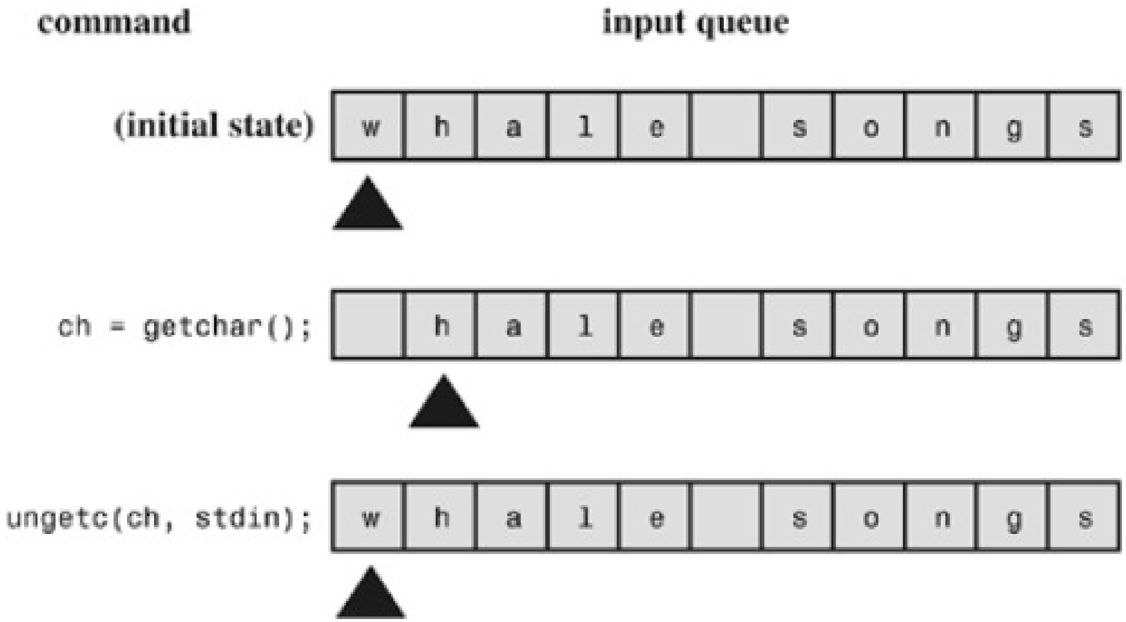
- fprintf(file pointer, format string, value list)
- single character
  - putchar, putc, fputc

The ungetc function

- int ungetc(int c, FILE *fp)

  pushes the charater specified by c back onto the input stream (only one pushback is guaranteed at a time)

Figure 13.2. The `ungetc()` function.



The rewind and gets function

- void rewind(FILE * fp)
  - Sets the file-position pointer to the start of the file
- char * gets (char * str);
  - Reads characters from the standard input (stdin) and stores them into str until a newline character or the end-of-file is reached.
  - On success, the function returns str.
  - If the end-of-file is encountered while attempting to read a character, the eof indicator is set (feof). If this happens before any characters could be read, the pointer returned is a null pointer (and the contents of str remain unchanged).
  - If a read error occurs, the error indicator (ferror) is set and a null pointer is also returned (but the contents pointed by str may have changed).

The fopen function

- 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)

The fopen function (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)

The fprintf function

- Syntax: fprintf(*filep*, “*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

The fscanf function

- Syntax: fscanf(*filep*, “*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

“Add a word”

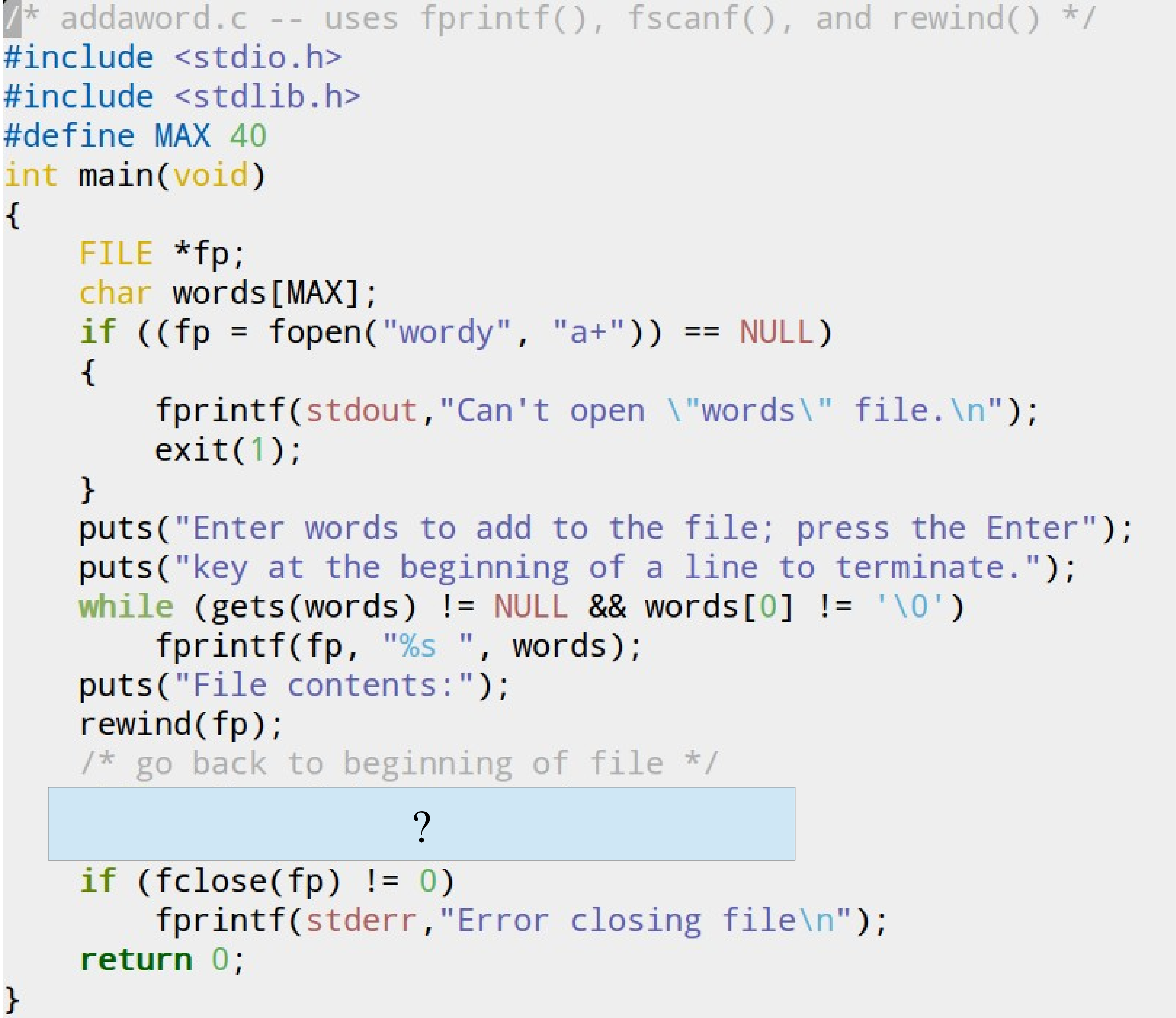
- A sample run

```
$ ./addaword
Enter words to add to the file; press the Enter
key at the beginning of a line to terminate.
The fabulous programmer[enter]
[enter]
File contents:
The
fabulous
programmer
$
```

“Add a word”

- A sample run

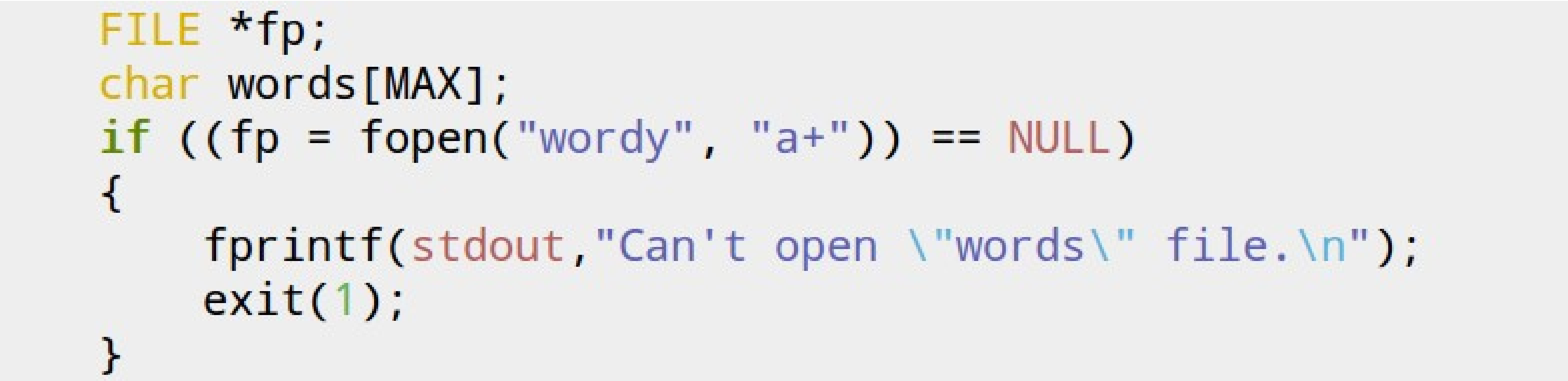
```
$ ./addaword
Enter words to add to the file; press the Enter
key at the beginning of a line to terminate.
The fabulous programmer[enter]
[enter]
File contents:
The
fabulous
programmer
$ ./addaword
Enter words to add to the file; press the Enter
key at the beginning of a line to terminate.
enchanted the[enter]
large[enter]
[enter]
File contents:
The
fabulous
programmer
enchanted
the
large
```



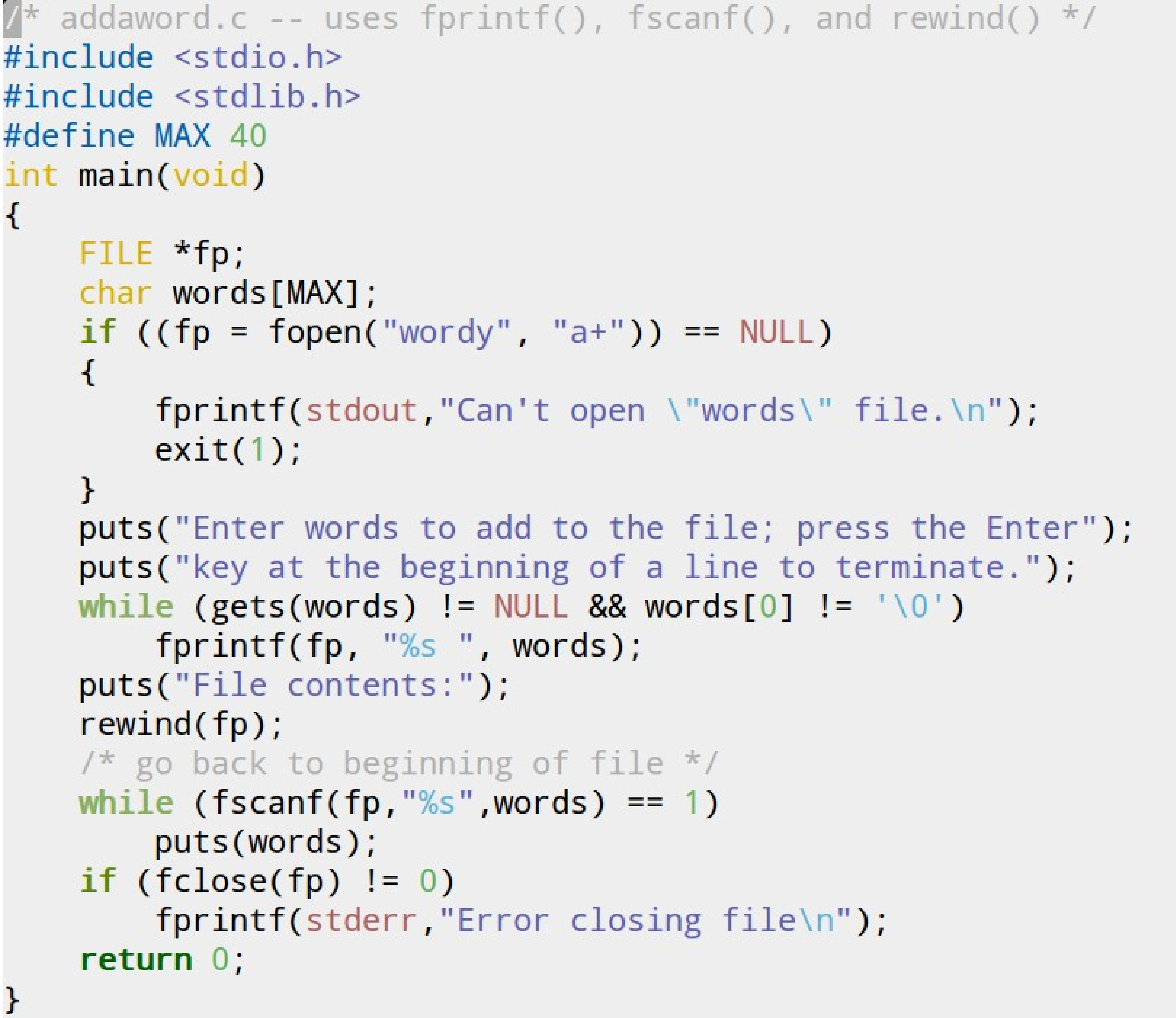
```
/* addaword.c -- uses fprintf(), fscanf(), and rewind() */
#include <stdio.h>
#include <stdlib.h>
#define MAX 40
int main(void)
{
    FILE *fp;
    char words[MAX];
    if ((fp = fopen("wordy", "a+")) == NULL)
    {
        fprintf(stdout,"Can't open \"words\" file.\n");
        exit(1);
    }
    puts("Enter words to add to the file; press the Enter");
    puts("key at the beginning of a line to terminate.");
    while (gets(words) != NULL && words[0] != '\0')
        fprintf(fp, "%s ", words);
    puts("File contents:");
    rewind(fp);
    /* go back to beginning of file */
                          ?
    if (fclose(fp) != 0)
        fprintf(stderr,"Error closing file\n");
    return 0;
}
```

Append mode

- By using the "a+" mode, the program can both read and write in the file
- The first time the program is used, it creates the wordy file
- When you use the program subsequently, it enables you to add (append) words to the previous contents.
- The append mode “a” only enables you to add material to
- the end of the file
- But the "a+" mode does enable you to read the whole file.



```
FILE *fp;
char words[MAX];
if ((fp = fopen("wordy", "a+")) == NULL)
{
    fprintf(stdout,"Can't open \"words\" file.\n");
    exit(1);
}
```



```c
/* addaword.c -- uses fprintf(), fscanf(), and rewind() */
#include <stdio.h>
#include <stdlib.h>
#define MAX 40
int main(void)
{
    FILE *fp;
    char words[MAX];
    if ((fp = fopen("wordy", "a+")) == NULL)
    {
        fprintf(stdout,"Can't open \"words\" file.\n");
        exit(1);
    }
    puts("Enter words to add to the file; press the Enter");
    puts("key at the beginning of a line to terminate.");
    while (gets(words) != NULL && words[0] != '\0')
        fprintf(fp, "%s ", words);
    puts("File contents:");
    rewind(fp);
    /* go back to beginning of file */
    while (fscanf(fp,"%s",words) == 1)
        puts(words);
    if (fclose(fp) != 0)
        fprintf(stderr,"Error closing file\n");
    return 0;
}
```

The fgets function

- char * fgets (char * str, int num, FILE * stream);
- Reads characters from stream and stores them as a C string into str until (num-1) characters have been read or either a newline or the end-of-file is reached, whichever happens first.
- A newline character makes fgets stop reading, but it is considered a valid character by the function and included in the string copied to str.
- On success, the function returns str.
- If the end-of-file is encountered while attempting to read a character, the eof indicator is set (feof). If this happens before any characters could be read, the pointer returned is a null pointer (and the contents of str remain unchanged).
- If a read error occurs, the error indicator (ferror) is set and a null pointer is also returned (but the contents pointed by str may have changed).

“Parrot”

- A sample run

  The silent knight

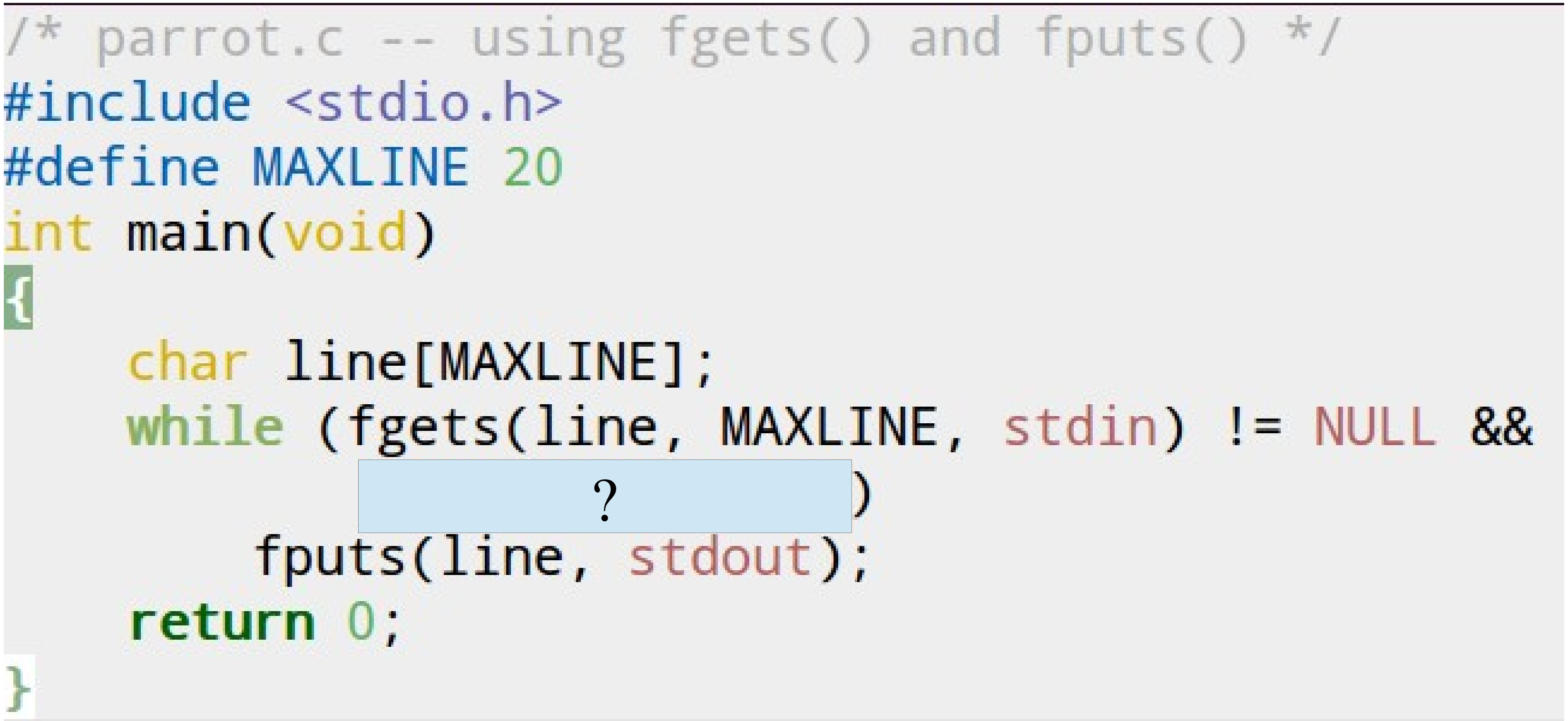
  The silent knight

  strode solemnly down the dank and dark hall.

  strode solemnly down the dank and dark hall.

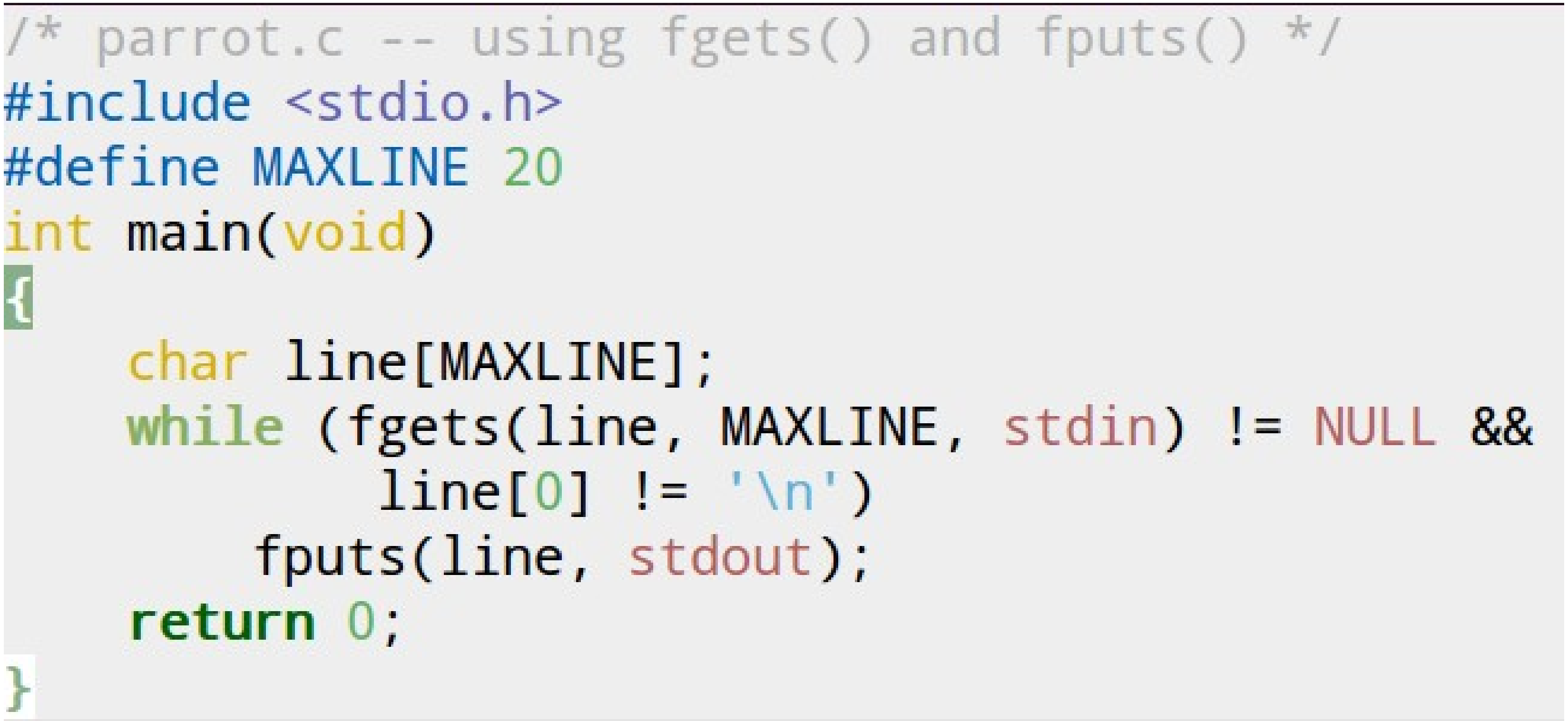
  [enter]

Terminating condition



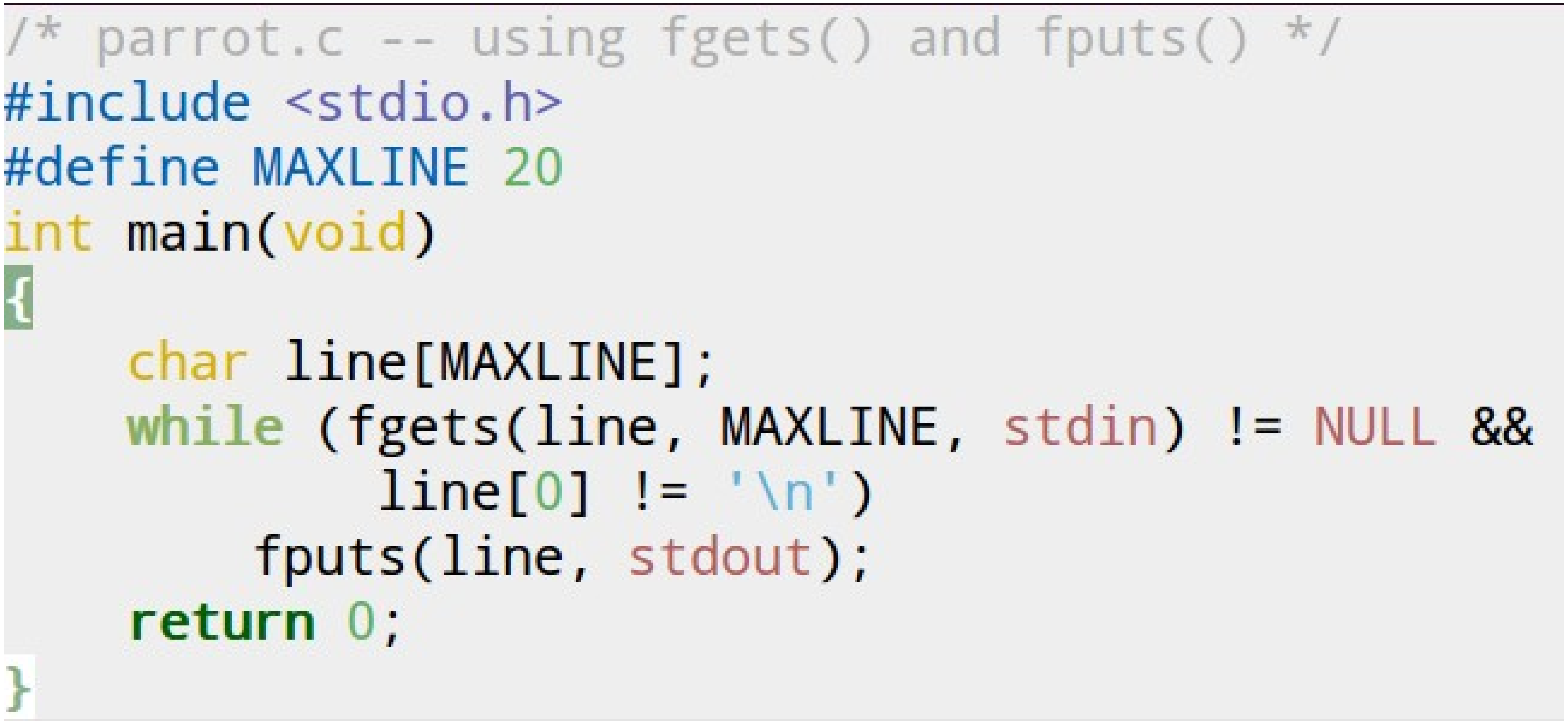
```
/* parrot.c -- using fgets() and fputs() */
#include <stdio.h>
#define MAXLINE 20
int main(void)
{
    char line[MAXLINE];
    while (fgets(line, MAXLINE, stdin) != NULL &&
                   ?        )
        fputs(line, stdout);
    return 0;
}
```

Do you notice anything odd?



```c
/* parrot.c -- using fgets() and fputs() */
#include <stdio.h>
#define MAXLINE 20
int main(void)
{
    char line[MAXLINE];
    while (fgets(line, MAXLINE, stdin) != NULL &&
            line[0] != '\n')
        fputs(line, stdout);
    return 0;
}
```

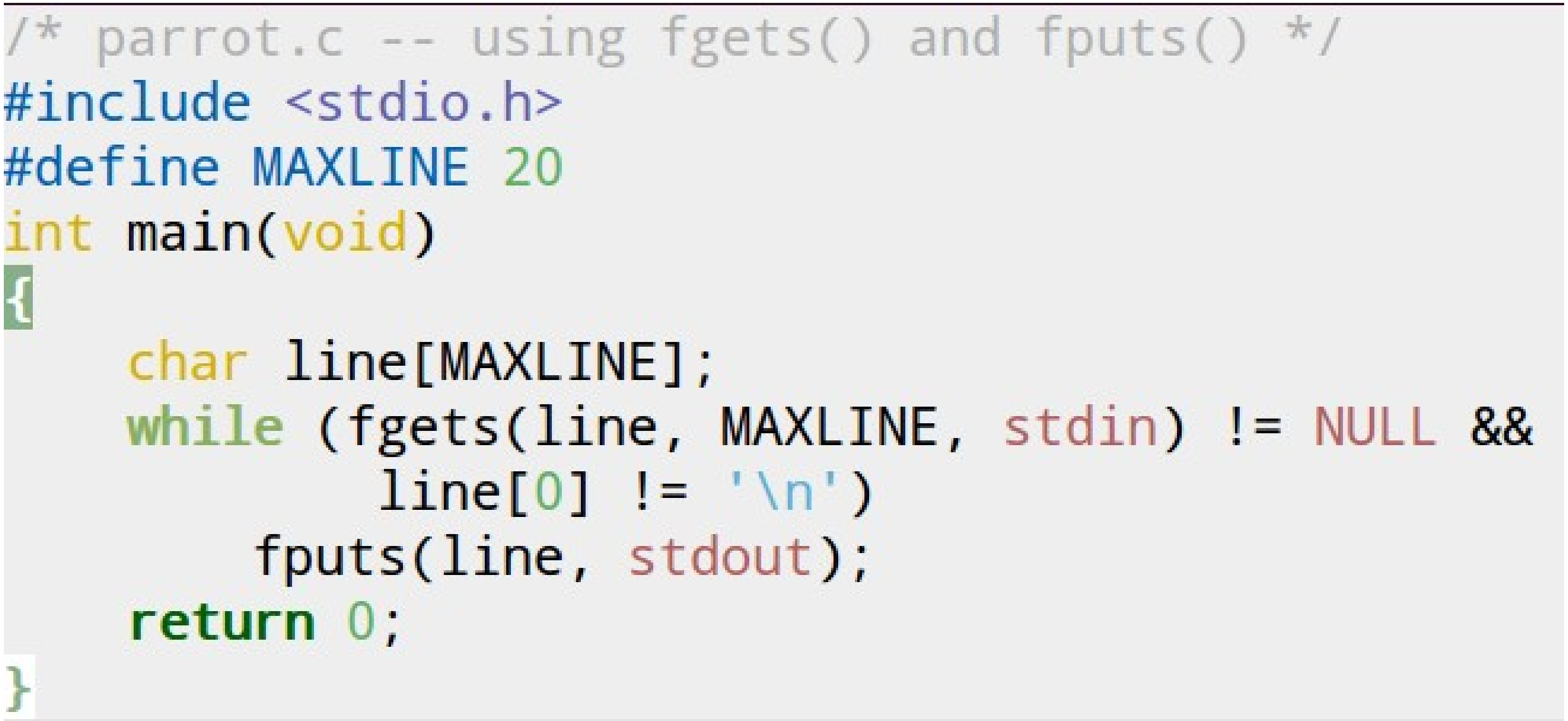
Do you notice anything odd?



```c
/* parrot.c -- using fgets() and fputs() */
#include <stdio.h>
#define MAXLINE 20
int main(void)
{
    char line[MAXLINE];
    while (fgets(line, MAXLINE, stdin) != NULL &&
            line[0] != '\n')
        fputs(line, stdout);
    return 0;
}
```

The program works fine. This should seem surprising because the second line entered contains 44 characters, and the line array holds only 20, including the newline character!

Do you notice anything odd?



```
/* parrot.c -- using fgets() and fputs() */
#include <stdio.h>
#define MAXLINE 20
int main(void)
{
    char line[MAXLINE];
    while (fgets(line, MAXLINE, stdin) != NULL &&
            line[0] != '\n')
        fputs(line, stdout);
    return 0;
}
```

fputs(“strode solemnly dow” , stdout);
fputs(“n the dank and dark hall\n”, stdout);