Text files

Program Persistent Data

Lecture 2

Review

- In C there are buffers required to work with files.
- Streams are declared using **`FILE *fp`**;
- These streams are required for each file that you work on.
- To open and use the stream, error check that the file exists then close when finished:

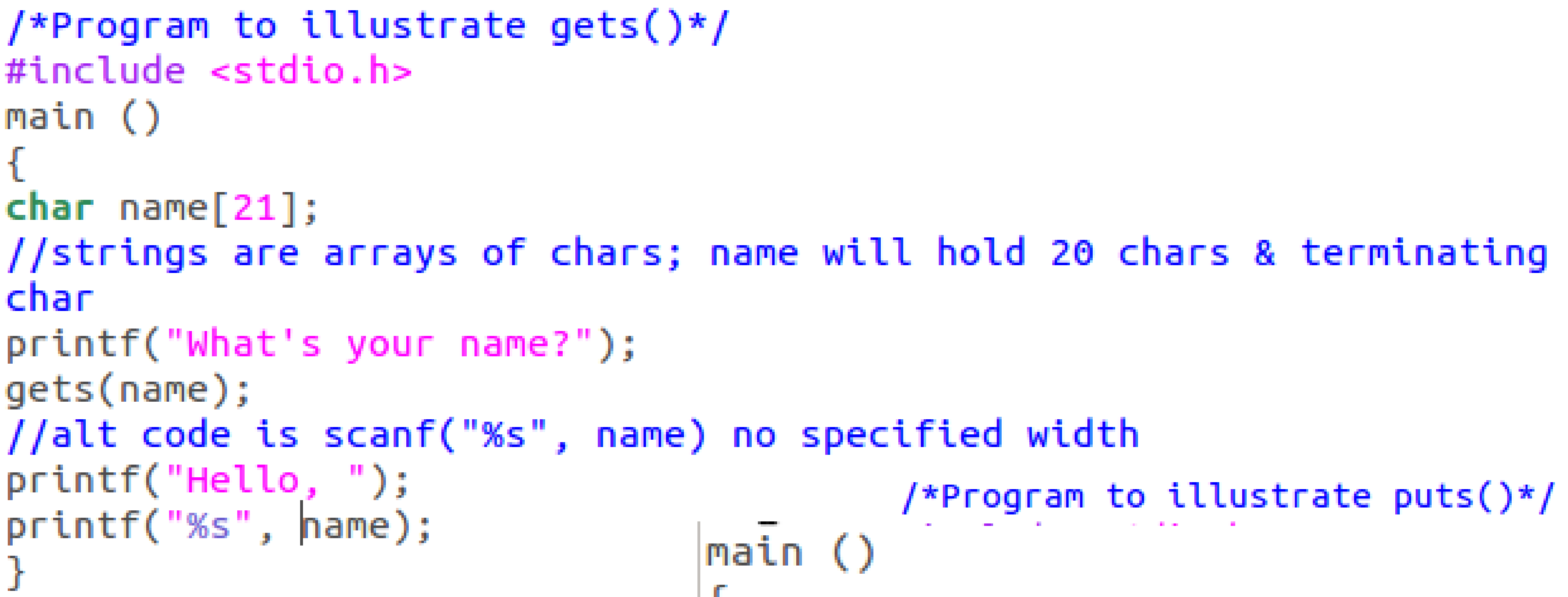
```
fp = fopen("write.txt","w");
if (fp == NULL)
     {printf("Can't open file.\n");
fclose(fp);
```

<stdlib.h>

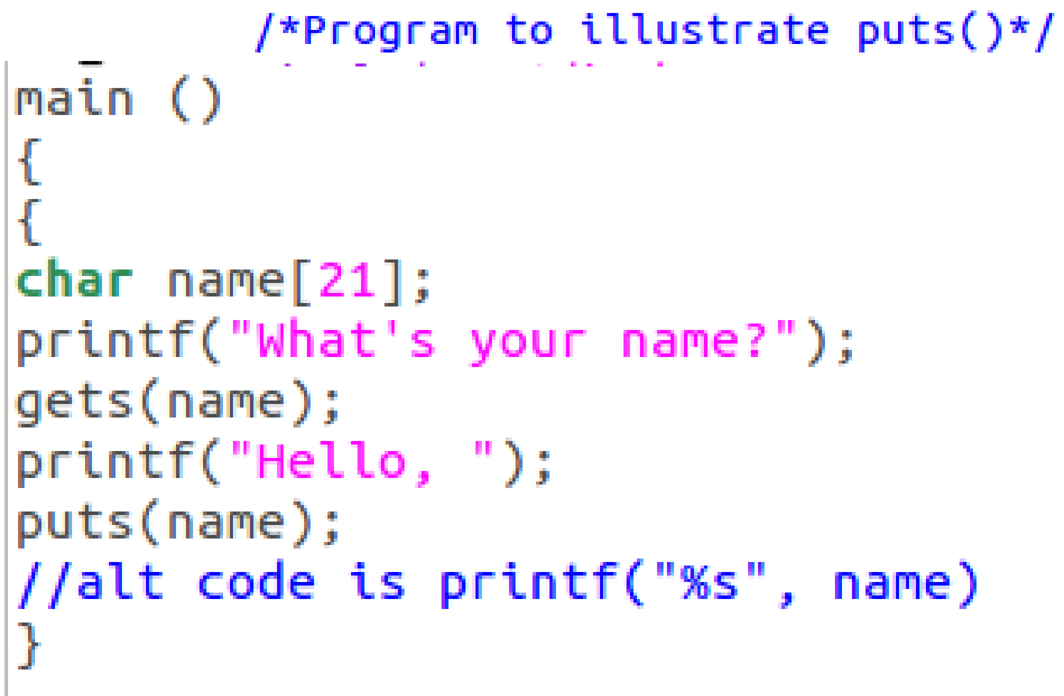
Standard Library

What do we know about this library?
What do we want to do with files??

Recall gets() & puts()



```
/*Program to illustrate gets()*/
#include <stdio.h>
main ()
{
char name[21];
//strings are arrays of chars; name will hold 20 chars & terminating
char
printf("What's your name?");
gets(name);
//alt code is scanf("%s", name) no specified width
printf("Hello, ");
printf("%s", name);
}
```



```
/*Program to illustrate puts()*/
main ()
{
{
char name[21];
printf("What's your name?");
gets(name);
printf("Hello, ");
puts(name);
//alt code is printf("%s", name)
}
```

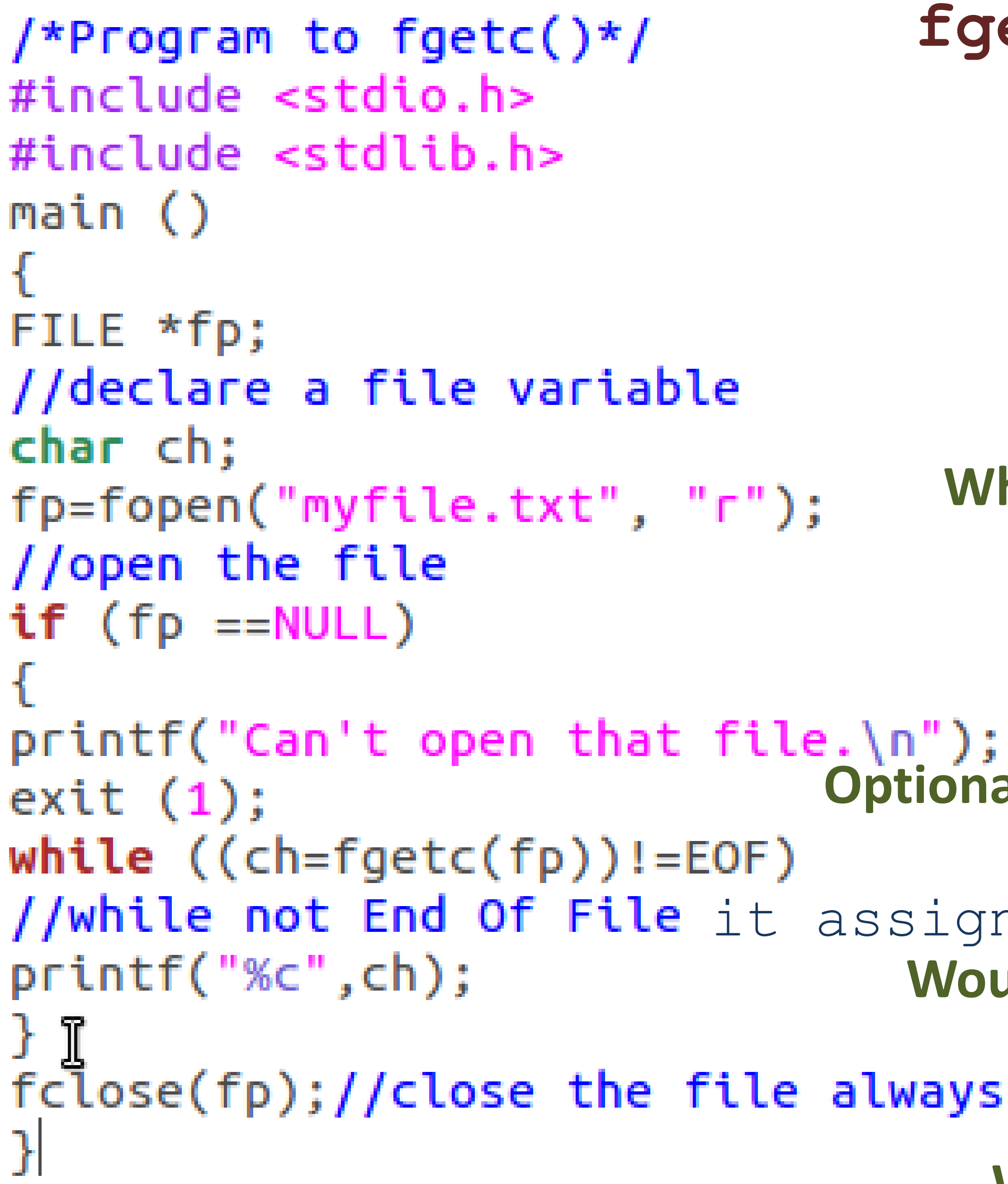
Standard file functions

Instruction	Meaning
`fgetc(f)`	Read a char from file f
`fputc(f)`	Write a char to file f
`fgets(string,size,f)`	Read a string from file f. It reads a string of a specified size
`fputs(string,f)`	Write a string to a file f
`fprintf("Hi %s, you are %i",s,a)`	Write the content to the file f
`fscanf(f,"%s %s %i",a,b,&c)`	Read a formatted line from file f

1. Reading characters `fgetc(file)`

- fgetc() is a *character* oriented function similar to getchar() (from *stdin*).
- Unlike scanf/getchar it reads *all* even whitespace.
- The function returns the *next* character in the file as an **unsigned char** converted to an **int**.
- It will stop reading at the end of file or if a read error occurs.
  - When is it good to read a file by byte
  - When is it good to read a file by byte

fgetc(file)



```
/*Program to fgetc()*/
#include <stdio.h>
#include <stdlib.h>
main ()
{
FILE *fp;
//declare a file variable
char ch;
fp=fopen("myfile.txt", "r");
//open the file
if (fp ==NULL)
{
printf("Can't open that file.\n");
exit (1);
while ((ch=fgetc(fp))!=EOF)
//while not End Of File it assigns into ch
printf("%c",ch);
}
fclose(fp);//close the file always be tidy!
}
```

What's 'r'?

Optional exits or required?

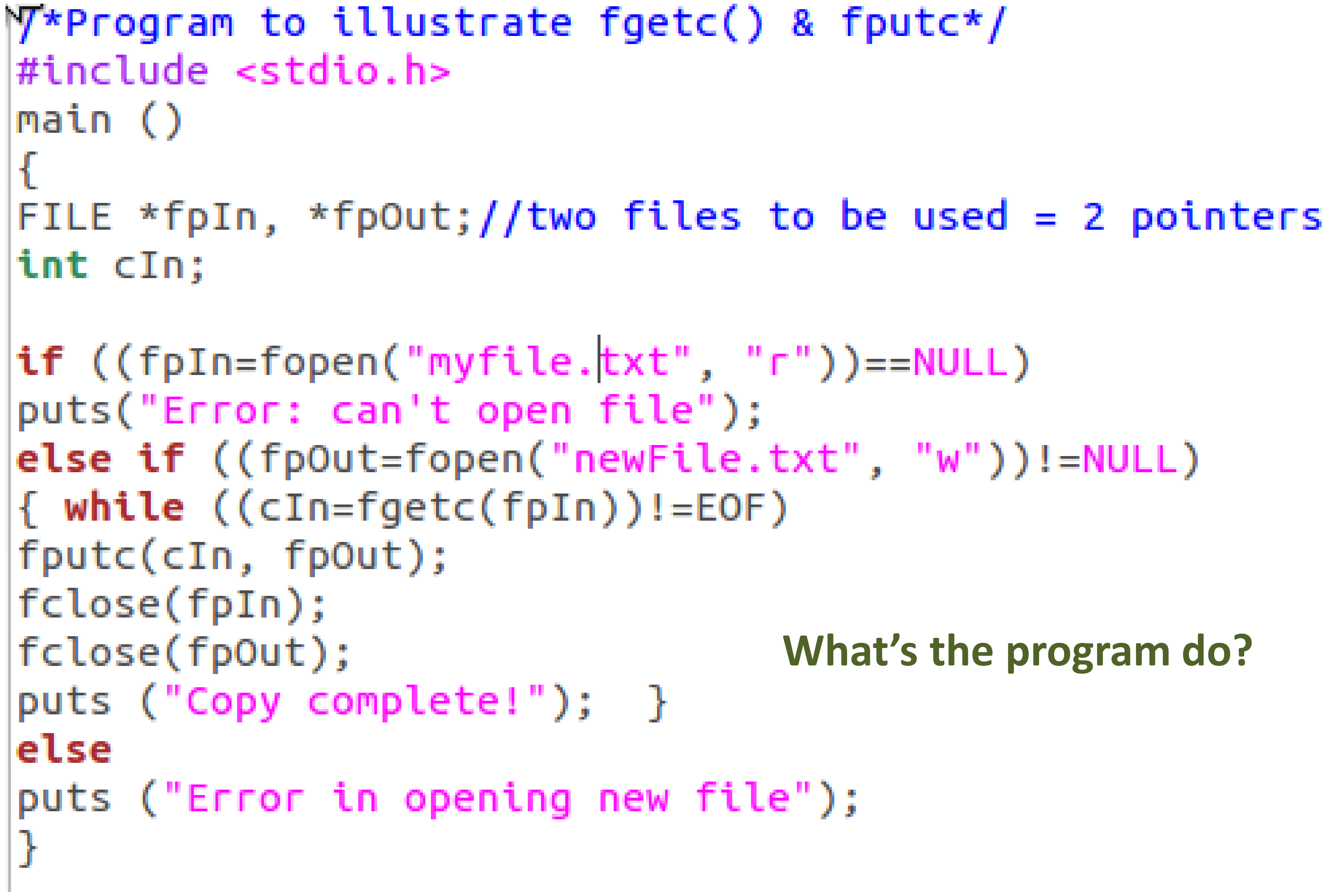
Would puts be appropriate?

What's the output?

2. Writing a character to a text file `fputc(char, stream)`

```
fputc(int ch, FILE *fp)
```

- Write a single char to the specific stream at the position of the pointer.
- The character is written in the integer format in C.
- If the file doesn’t exist the file will be created in which to put the characters!



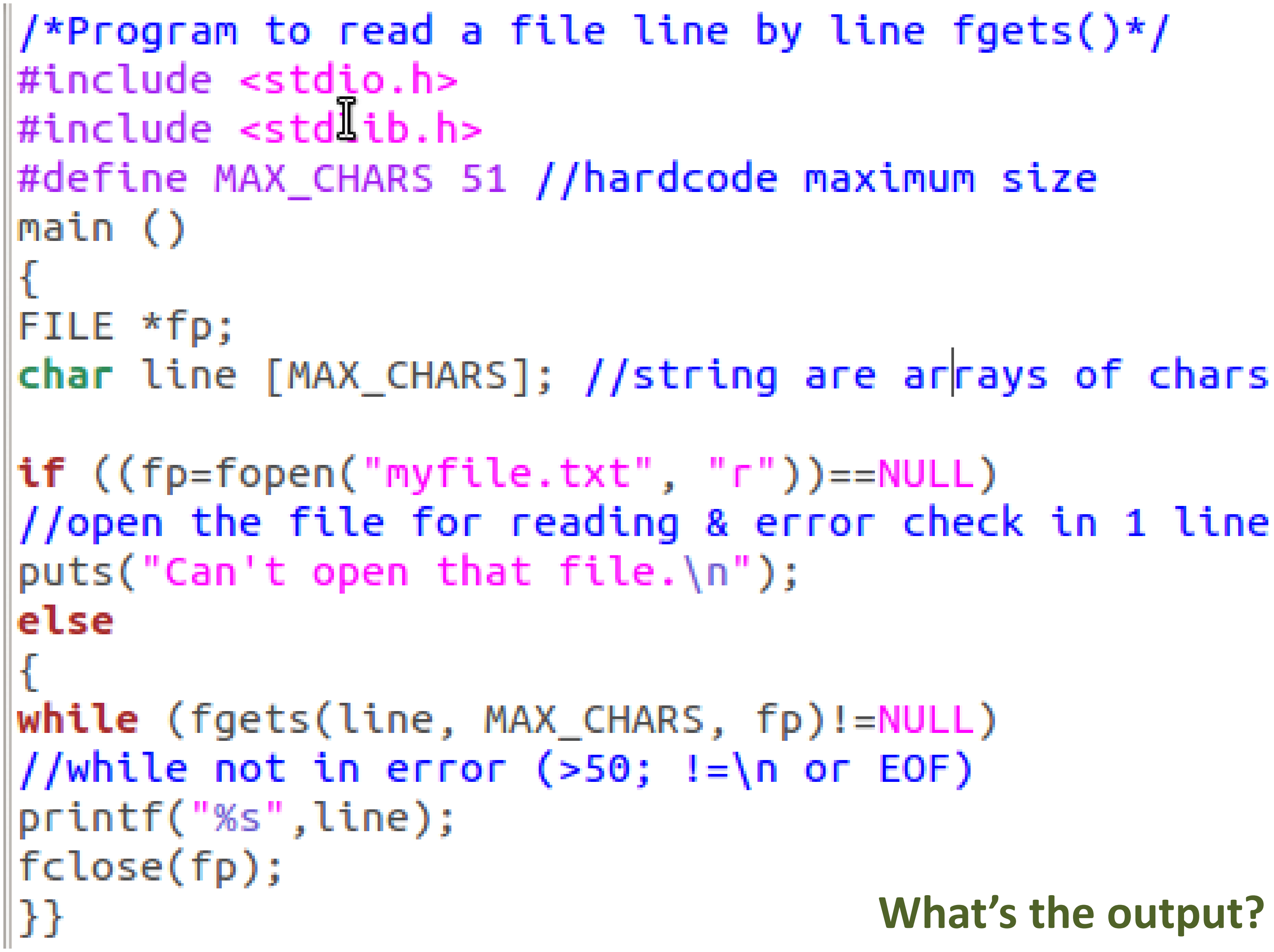
```c
/*Program to illustrate fgetc() & fputc*/
#include <stdio.h>
main ()
{
FILE *fpIn, *fpOut;//two files to be used = 2 pointers
int cIn;

if ((fpIn=fopen("myfile.txt", "r"))==NULL)
puts("Error: can't open file");
else if ((fpOut=fopen("newFile.txt", "w"))!=NULL)
{ while ((cIn=fgetc(fpIn))!=EOF)
fputc(cIn, fpOut);
fclose(fpIn);
fclose(fpOut);
puts ("Copy complete!");  }
else
puts ("Error in opening new file");
}
```

What’s the program do?

3. Reading strings `fgets(string,size,file)`

- The function reads from the file and places the output into the character array (string) pointed to by s.
- It will stop reading when any of the following conditions are true:
  - It has read n - 1 bytes (one character is reserved for the null-terminator), or
  - It encounters a *newline* character (a line-feed in the compilers tested here), or
  - It reaches the end of file, or
  - A read error occurs.
  - fgets() appends a null-terminator to the data read.

fgets(string,size,file)



```c
/*Program to read a file line by line fgets()*/
#include <stdio.h>
#include <stdlib.h>
#define MAX_CHARS 51 //hardcode maximum size
main ()
{
FILE *fp;
char line [MAX_CHARS]; //string are arrays of chars

if ((fp=fopen("myfile.txt", "r"))==NULL)
//open the file for reading & error check in 1 line
puts("Can't open that file.\n");
else
{
while (fgets(line, MAX_CHARS, fp)!=NULL)
//while not in error (>50; !=\n or EOF)
printf("%s",line);
fclose(fp);
}}
```

What's the output?

`fgets()` - errors

```
while(fgets( line, MAX_CHARS, fp ) != NULL)
```

//Note the termination condition.

We now check if the result of fgets is NULL (with fgetc we checked the !=EOF condition only)

- At any iteration we read MAX_CHARS-1 characters.

//last char is the terminating char reserved place

- If the line is shorter than MAX_CHARS-1 characters, we only read to the end of the line
- If the line is longer than MAX_CHARS characters, we read only part of the line.

//Choose the correct value of MAX_CHARS, it must be bigger than the biggest line in your text!

Standard error checking functions

`ferror()`

file function error that will return a non zero value if there IS an error.

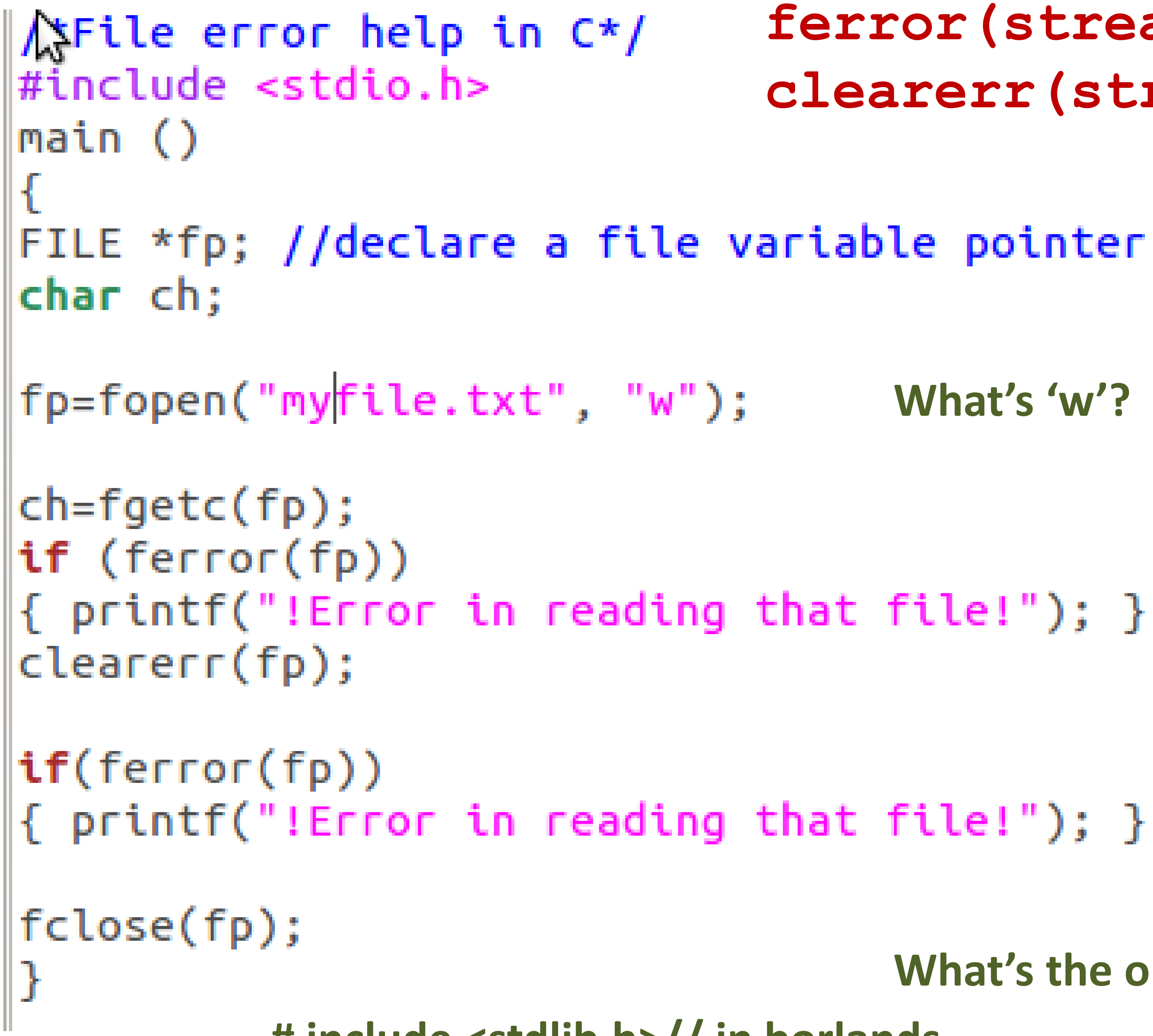
It takes a FILE structure pointer (stream)

`feof()`

Similar also takes the FILE pointer and will let you know if at the EOF

`clearerr()`

Resets (as does rewind()) the error code on the File pointer, *doesn't fix the errors*.



```
/*File error help in C*/
#include <stdio.h>
main ()
{
FILE *fp; //declare a file variable pointer
char ch;

fp=fopen("myfile.txt", "w");

ch=fgetc(fp);
if (ferror(fp))
{ printf("!Error in reading that file!"); }
clearerr(fp);

if(ferror(fp))
{ printf("!Error in reading that file!"); }

fclose(fp);
}
```

ferror(stream)

clearerr(stream)

What's 'w'?

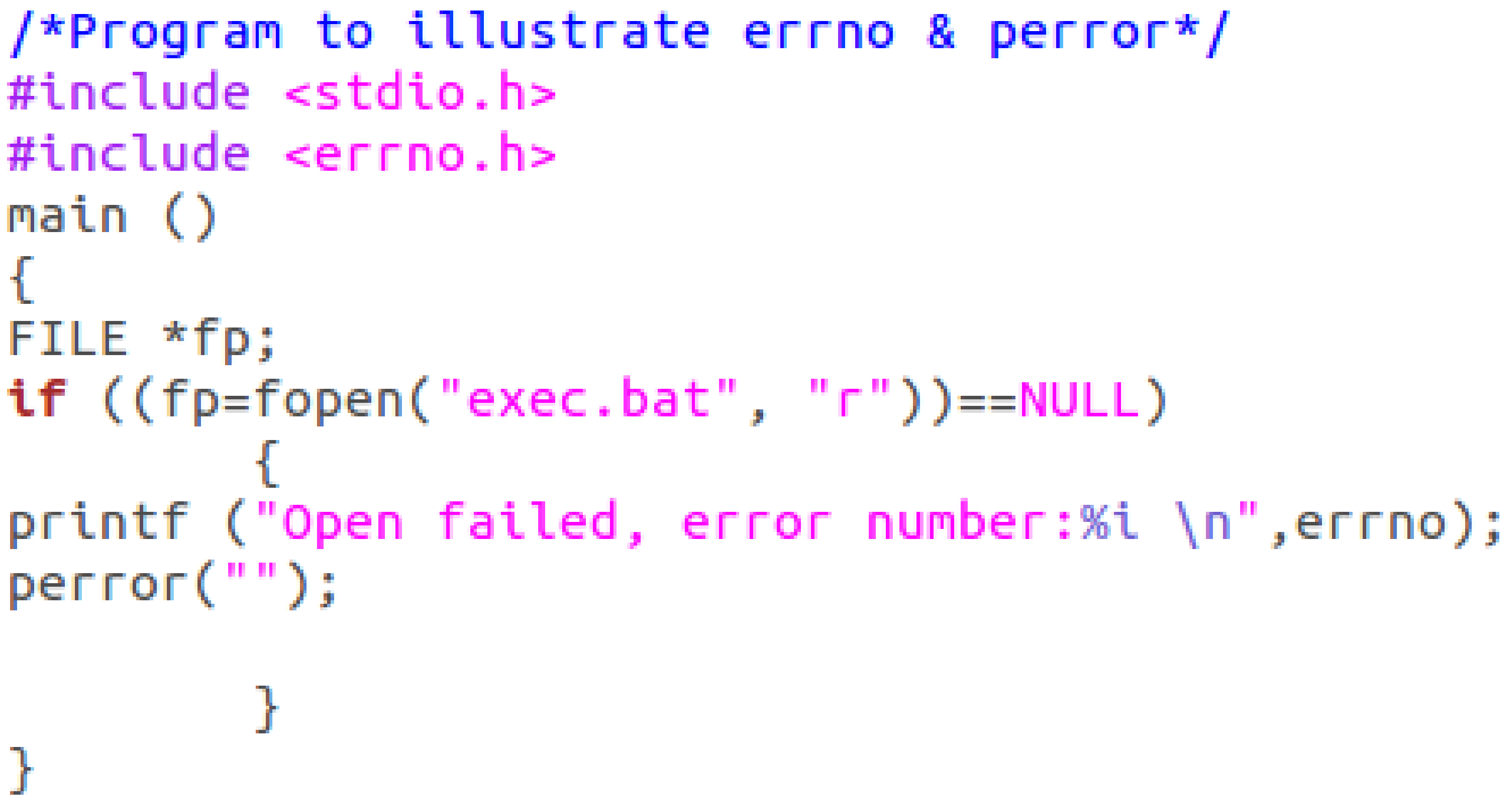
What's the output?

# include <stdlib.h>// in borlands

More information on errors `<errno.h>`

- So you have an issue with the file?!
- How to find out what is going wrong..
- `errno` is the error number variable of C.
- For each error a (!=0) number is returned such as 1: the file is not found.
- Use `perror` to print what the error number is in text.

# include <stdlib.h>// in borlands



```
/*Program to illustrate errno & perror*/
#include <stdio.h>
#include <errno.h>
main ()
{
FILE *fp;
if ((fp=fopen("exec.bat", "r"))==NULL)
        {
printf ("Open failed, error number:%i \n",errno);
perror("");

        }
}
```

- If in error then the system will feed back the error number and text associated.

//Watch the # of brackets and don’t forget to include the errno.h

4 Writing strings to a text file `fputs(string,stream)`

`fputs(string , FILE *fp)`

A string is an array of chars.

This writes a single string when not at the NULL. Similar to the `fgets()` it will terminate at the NULL so allow for extra char.

Doesn't add the formatting of newlines.
Not like `fwrite()` as don't need size or location to write to.

```c
/*Program to illustrate fputs & fgets to take a
line (if 80 chars is a line) at a time*/
#include <stdio.h>
#define MAX_CH 81

main ()
{
FILE *fpIn, *fpOut;
char str[MAX_CH];

if ((fpIn=fopen("myfile.txt", "r"))==NULL)
puts("!Can't open first file!");
else if ((fpOut=fopen("fileNew.txt", "w"))!=NULL)
{
while (fgets(str, MAX_CH, fpIn)!=NULL);
        fputs(str,fpOut);
        puts("Success!");
        fclose(fpIn);
        fclose(fpOut);  }
else
        puts("!Can't open second file!");
}
```

Careful: Copy and paste & be careful to edit
Make sure you use a pointer for each stream & assign consistently
Make sure you give yourself the right access to the files: Read | Write

5. Write to a file

`fprintf(FILE *fp, formatString, variables)`

```
fprintf(fp, "%s %d\n", username,
score);
```

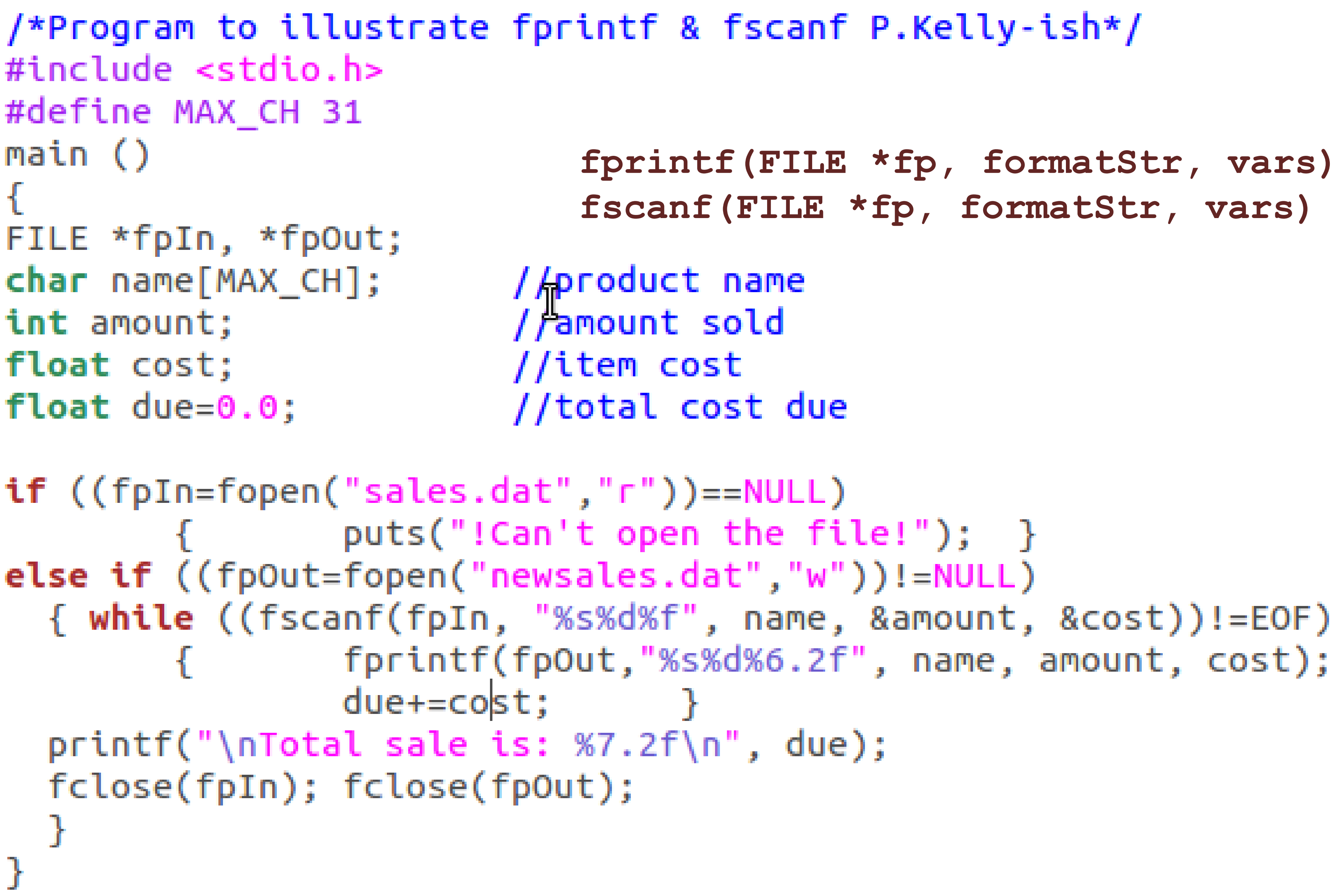
- Similar to printf outputs not to the console but to the file.
- The file is specified by the stream (fp).
- The formatting required is next argument (`"%s %d\n"`).
- The final arguments are the variable names to be written (there are two; a string `username` & an integer `score`) .

6. Reading from files `fscanf(FILE *fp, formatString, variables)`

```
fscanf(fp, "%s %d\n", username,
score);
```

- Similar to scanf gets *inputs* but from the file.
- The file is specified by the stream (fp).
- The formatting is required (`"%s %d\n"`)
- The final arguments are the variable names to be written (there are two; a string `username` & an integer `score`) .

```
fprintf(FILE *fp, formatStr, vars)
fscanf(FILE *fp, formatStr, vars)
```



```
/*Program to illustrate fprintf & fscanf P.Kelly-ish*/
#include <stdio.h>
#define MAX_CH 31
main ()
{
FILE *fpIn, *fpOut;
char name[MAX_CH];       //product name
int amount;              //amount sold
float cost;              //item cost
float due=0.0;           //total cost due

if ((fpIn=fopen("sales.dat","r"))==NULL)
        {       puts("!Can't open the file!");  }
else if ((fpOut=fopen("newsales.dat","w"))!=NULL)
  { while ((fscanf(fpIn, "%s%d%f", name, &amount, &cost))!=EOF)
        {       fprintf(fpOut,"%s%d%6.2f", name, amount, cost);
                due+=cost;      }
  printf("\nTotal sale is: %7.2f\n", due);
  fclose(fpIn); fclose(fpOut);
  }
}
```

fscanf() extra explanation

`fscanf(FILE *fp, format_string, variables)`

- format_string is a string containing C *format specifiers*, such as: %s for string, %i or %d for integers, %c for char, %f for float.
- Variable list is a list of variables separated by comma. The data are read from the file and saved in these variables in order!
- `fscanf` wants pointers to the variables, only use the symbol **&** in front of normal variables (`int, char`..) , but remember

fscanf() – extra clarifications

`fscanf()` example

```
   int a,b;
   char s1[10];
   char s2[10];
   fscanf(fp,”%i %i %s %s”,&a,&b,s1,s2)
```

//The & operator returns the address of a variable:

```
   int a;  //this is an int variable
   int *p;  //this is a pointer to an integer variable
   a=2 // an assignment
   &a is the address of variable a (for instance XFF00)
   p=&a  //assign to pointer p the address of a
   p contains XFF00 (the address of the variable
   pointed)
   *p contains 2 (the value of a, the variable pointed
   by p)
```

Review – *text* file

Instruction	**Meaning**
`fgetc(f)`	Read a char from file f
`fputc(f)`	Write a char to file f
`fgets(string,size,f)`	Read a string from file f. It reads a string of a specified size
`fputs(string,f)`	Write a string to a file f
`fprintf(“Hi %s, you are %i”,s,a)`	Write the content to the file f
`fscanf(f,”%s %s %i”,a,b,&c)`	Read a formatted line from file f

lab1

- Working within the teams this week.
- Working with the text files & stdlib.h functions:
- Four tasks:

1. Count the number of digits
2. Copy a file line by line
3. Separate a text file
4. Filter & process data with fscanf – files given in folder to work on