Diploma in

Computer Science

File Management

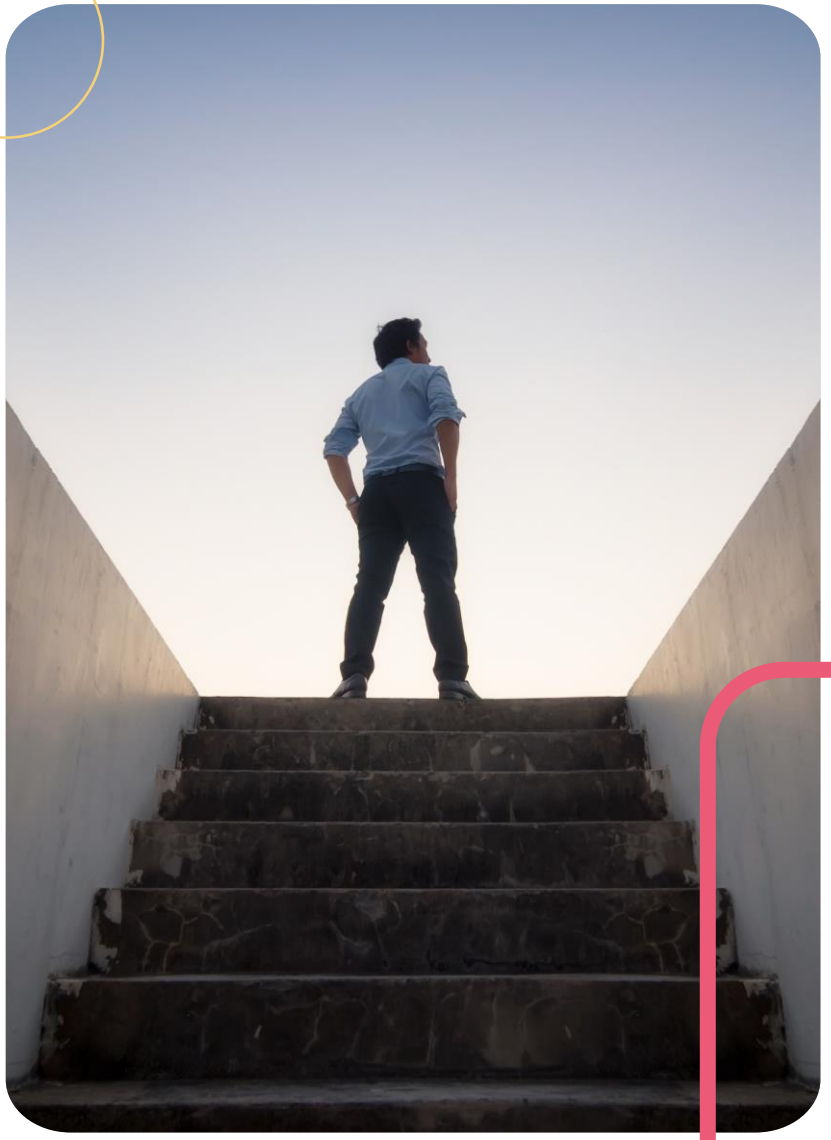
Lesson 3 Challenge Feedback

Which C standard does your compiler conform to?

- Go to HELP.
- Click ‘About’ to see all the info you need to know.

  For example:

  GCC conforms to the C99 standard



Objectives

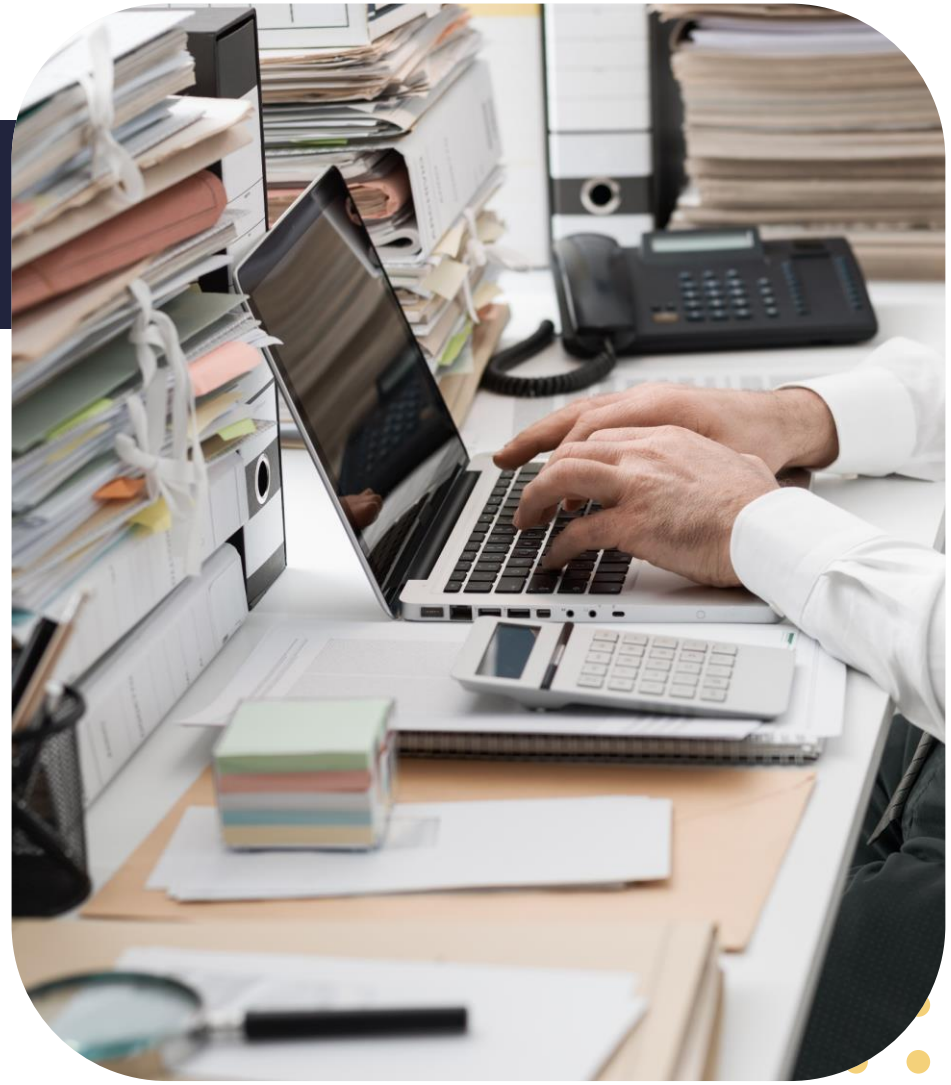
- Explore the various file systems on different platforms
- Identify how programs access files
- Distinguish between file types
- Use C code to access and manipulate files

Why files?

Files and an operating system

An operating system is a collection of files that contain the programs that are run by the computer.

Everything is contained in some form of file.



Storage of data

- A file – a computer's basic means of storing data (binary digits)
- Computer knows what each sequence means
- Sequences kept in files

What is a file?

- Container of binary digits formatted in a special way so that the computer knows that it is a single entity
- Operating system manages files
- Files stored on internal drive or removable drives
- Accessed using a file path

History of files

- Word ‘file’ first used with punched cards in 1940s
- Term initially referred more to storage in hardware
- With removable storage term evolved to refer to binary contents

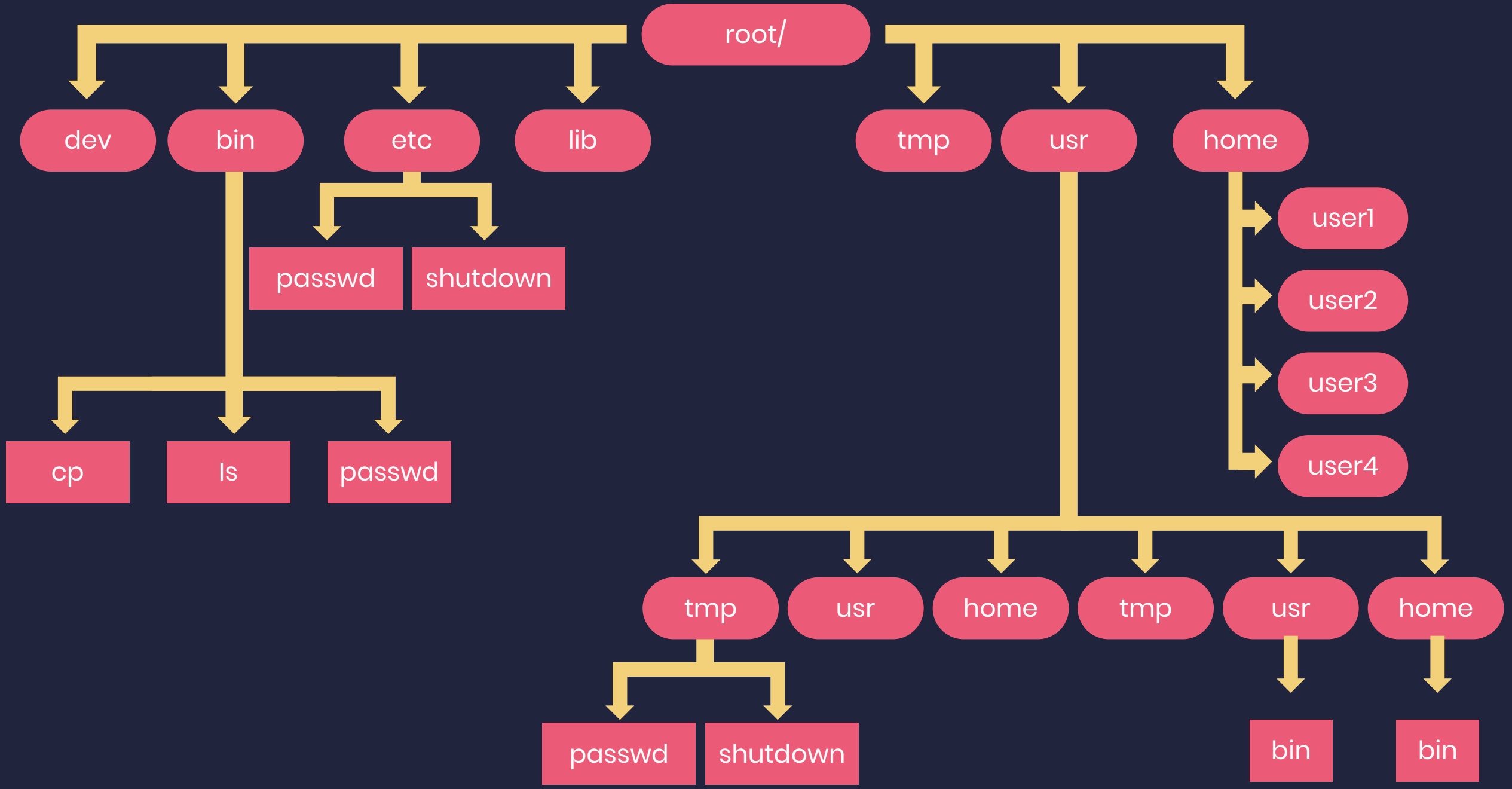
How do files fit into an operating system?

- Operating system - a collection of files that do various things
- Includes executables and the kernel
- Also includes non-executable files
- All arranged in a file system

How do files fit into an operating system?

- File system contains guidelines and permissions
- Restrictions - require you to have certain permissions in order to access the file

Directories allow the computer to trace the root of the drive to the point where it gets to the file.


root/
dev
bin
etc
lib
tmp
usr
home
passwd
shutdown
user1
user2
user3
user4
cp
ls
passwd
tmp
usr
home
tmp
usr
home
passwd
shutdown
bin
bin

Four key elements

- Reusability
- Large storage capacity
- Time-saving
- Portability

Data portability - challenges

- Inability to save data limits the usefulness of the program
- Not practical to enter data every time you want to use a program

Data portability - challenges

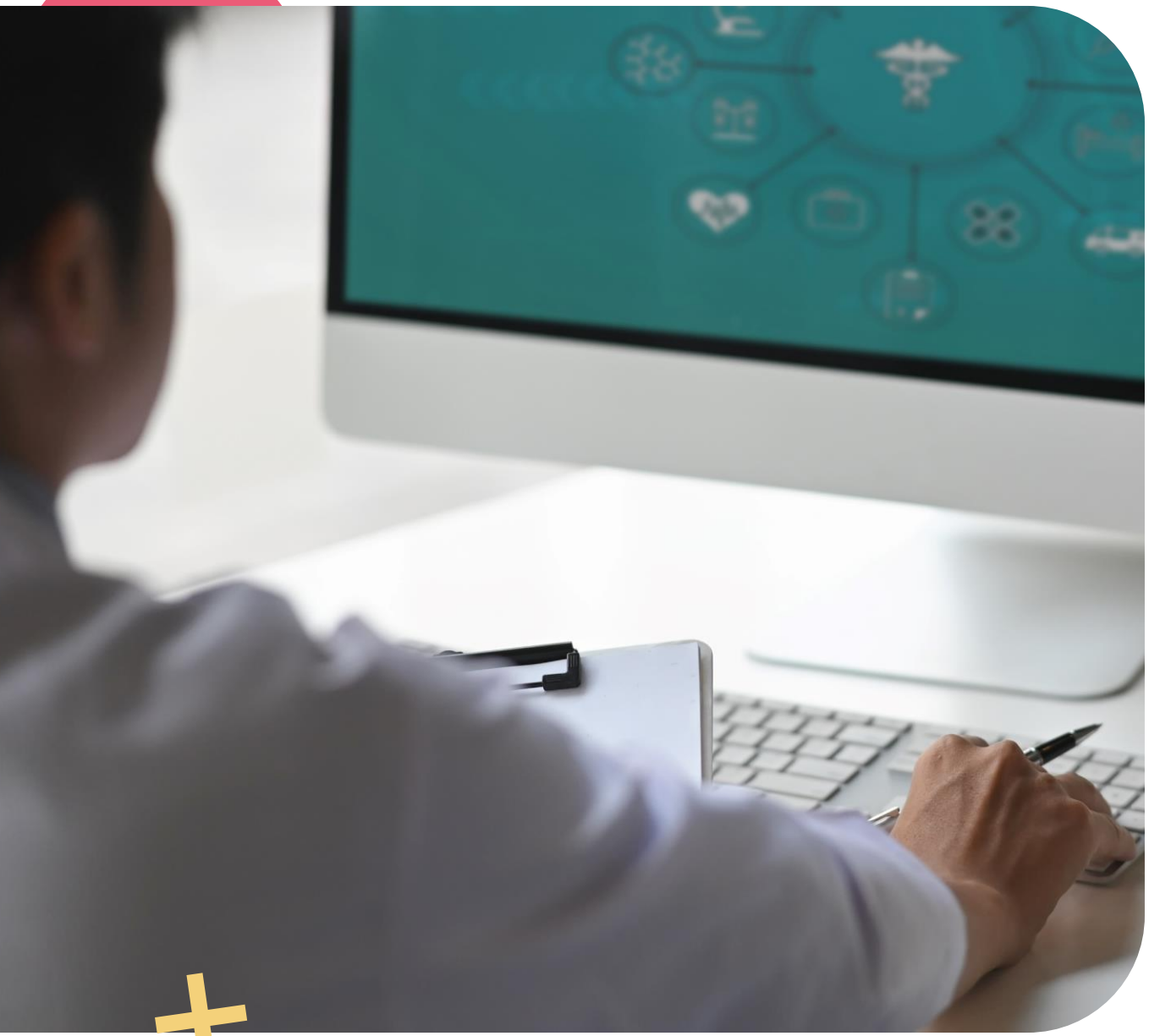
- Manual data entry wastes time by reducing efficiency
- Also introduces errors

C allows us to...

- Read information from files and write information to files
- Store configuration settings so data can be retrieved, processed and stored to same file

Configuration file example: display driver

- Files bundled with the executable can be opened using a regular text editor
- Text file in a folder with info on resolution, colour, refresh rate, etc.



Configuration file example: display driver

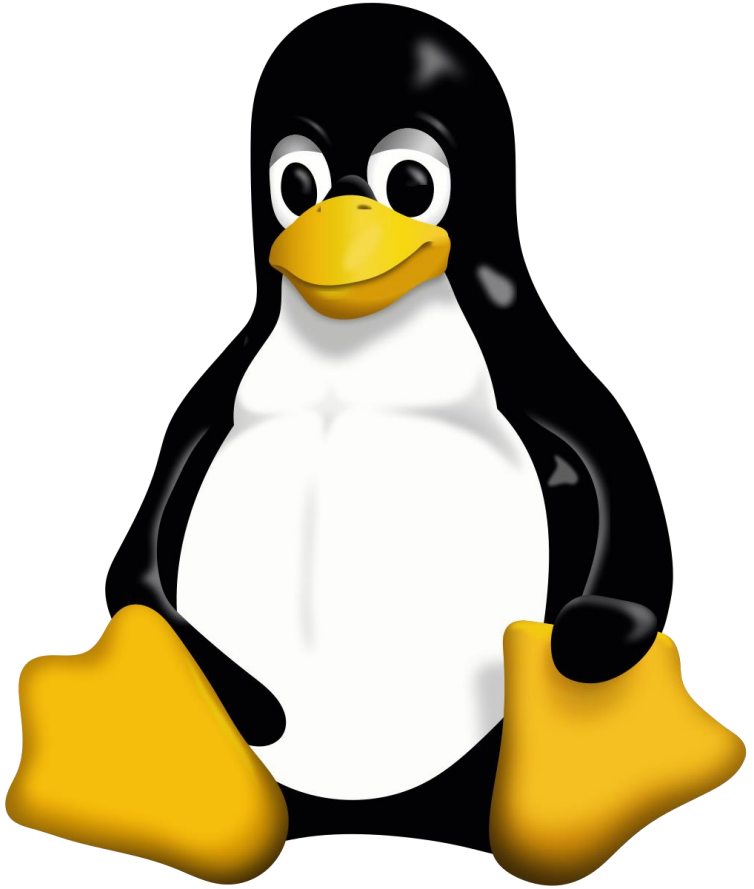
- Known as a configuration file
- If computers couldn’t access files, you would have to set values each time you used it

File system example: Windows

- Each storage device is represented by a drive letter
- Look in C drive for system drive with default configurations (C:\)
- Subsequent directories named after a backslash
- String ends with filename, dot and file extension

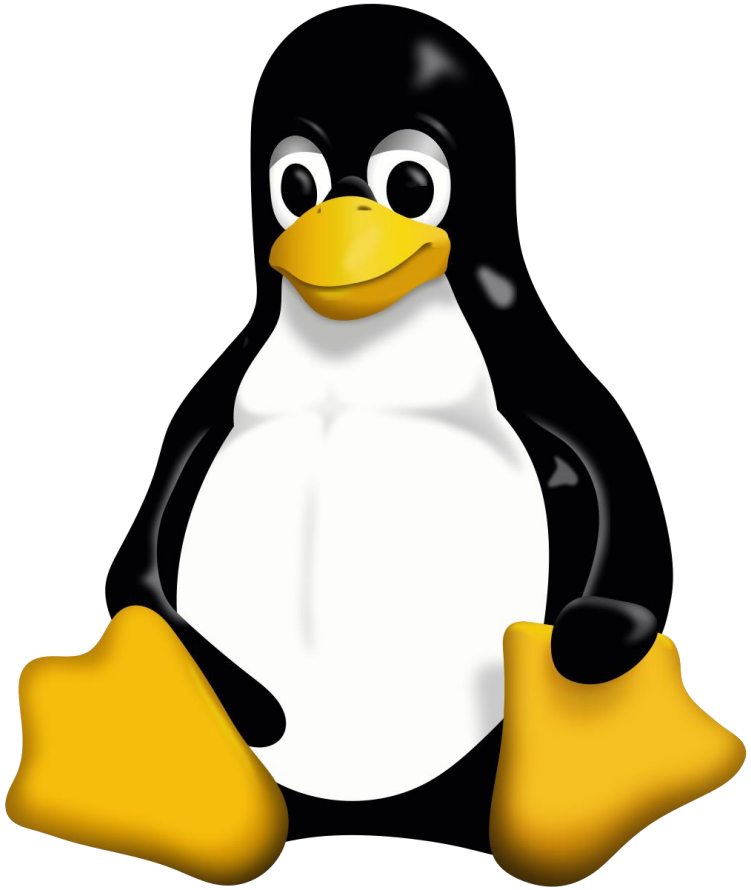
File system example: Linux

- Linux drives don't have a drive letter
- Each storage device is listed under /dev
- Filesystem unifies all physical hard drives and partitions into a single directory structure
- Other directories located under single root directory



File system example: Linux

- Directory structure uses forward slashes
- Start with a forward slash and name of folder
- Starts at the top with the root directory and ends with the filename followed by a dot and the file extension



Did you know?

The word ‘file’ derives from the Latin word *filum* meaning ‘a thread’.

When you save files they are encoded in a specific format reflected by the file extension.

Files supported in C

- Text files
- Binary files

Text files

- Normal files that can be opened by text editor
- Have file extension dot txt

Opening text files

- Contents are in plain text
- Most text files use ANSI and Unicode encoding
- Very easy to open, edit, and delete
- Have widespread compatibility
- Require minimum effort to maintain

The problem with text files...

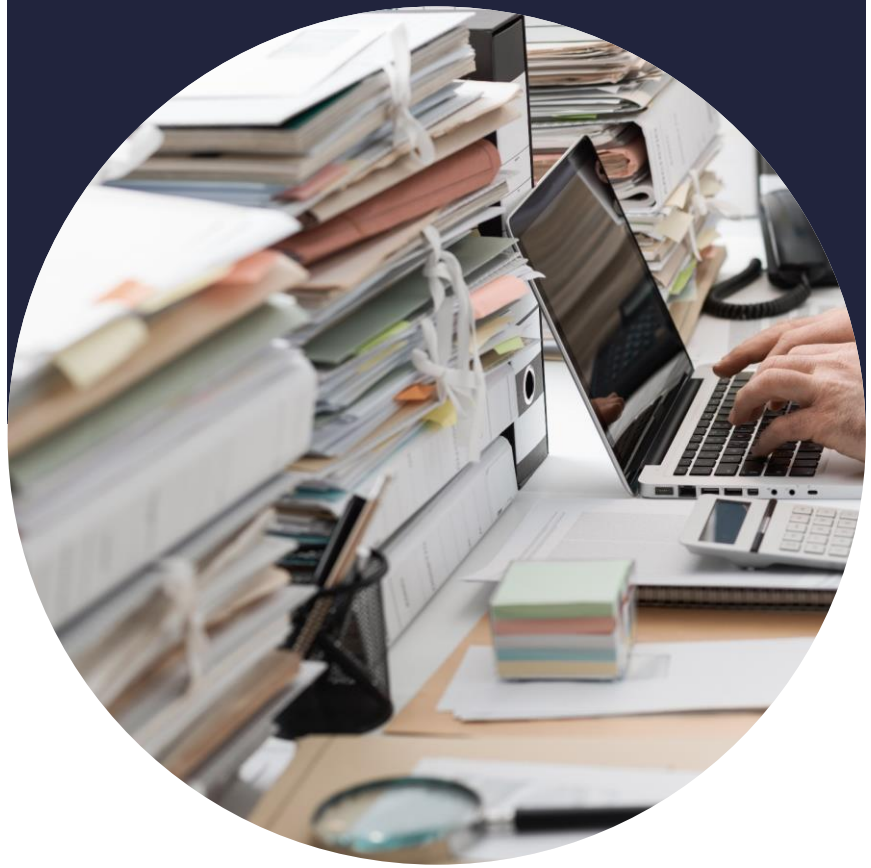
- Limited security - nearly every system can open text files
- Require more space to store

Binary files

- Cannot be read by text editors
- Similar in structure to arrays of structures
- Can be used to create very large collections of data
- Easy to jump instantly to any structure within them

Opening binary files

- Faster read and write times than text files
- Supports the file-of-structures concept

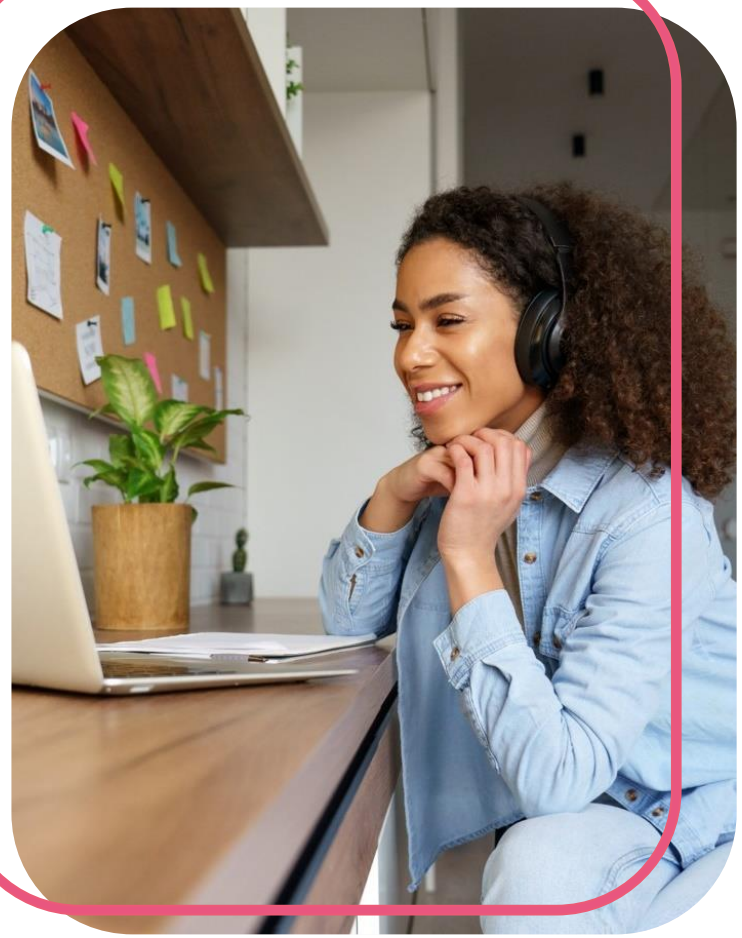


Opening binary files

- Program uses file pointers
- Points to record 0
- ‘Seek’ moves the pointer to the required record
- Can point to any byte location so you need to keep track of things

What can you do with files in C?

- Create a file
- Open a file
- Read a file
- Write to a file
- Close a file



Header files and functions in C

- Functions for file input and output contained in the standard library header file
- C treats files as data structures

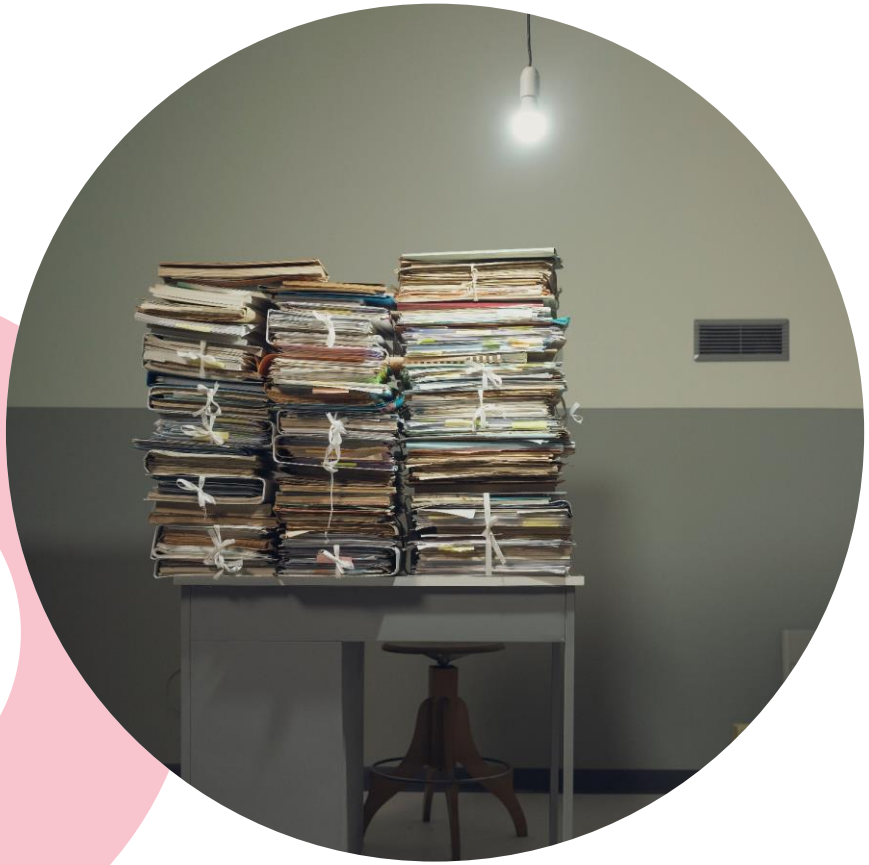
```
FILE *fptr.
```

Syntax

File operations in C

Function	Purpose
fopen ()	Creating a new file
fopen ()	Opening an existing file in your system
fclose ()	Closing a file
getc()	Reading characters from a line
putc()	Writing characters in a file
fscanf()	Reading a set of data from a file
fprintf()	Writing a set of data in a file
getw()	Reading an integral value from a file
putw()	Writing an integral value in a file
fseek()	Setting a desired position in the file
ftell()	Getting the current position in the file
rewind()	Setting the position at the beginning point

A closer look at C's built-in functions...



R	Opens a text file in reading mode
W	Opens or create a text file in writing mode
W	Opens a text file in append mode
R+	Opens a text file in both reading and writing mode
W+	Opens a text file in both reading and writing mode
A+	Opens a text file in both reading and writing mode
Rb	Opens a binary file in reading mode
Wb	Opens or create a binary file in writing mode
Ab	Opens a binary file in append mode
Rb+	Opens a binary file in both reading and writing mode
Wb+	Opens a binary file in both reading and writing mode
Ab+	Opens a binary file in both reading and writing mode

A closer look at C's built-in functions...

```
File = fopen("file_name",
"mode")
```

Syntax

- 'filename' refers to the name of the file that you want to work with
- 'mode' refers to the manner in which the file will be opened

A closer look at C's built-in functions

```
#include <stdio.h>
Int main(){
 FILE * file;
 If (file = fopen("hello.txt",
"r")){
 Printf("File opened in read
mode");
 }
 Else
 Printf("The file wasn't found, and
we cannot create a new file using r
mode");
 Fclose(file);
 Return 0;
}
```

Example

A closer look at C's built-in functions

```
fputc(char, file_pointer)

fputs(str, file_pointer):

fprintf(file_pointer, str,
variable_lists)
```

Stdio library

Writing to files

```
#include <stdio.h>
Int main() {
 Int i;
 FILE * fptr;
 Char fn[80];
 Char str[] = "Hello world\n";
 Fptr = fopen("our_first_file.txt", "w"); //
"w" defines "writing mode"
 For (i = 0; str[i] != '\n'; i++) {
 /* write to file using fputc() function */
 Fputc(str[i], fptr);
 }
 Fclose(fptr);
 Return 0;
 }
```

Example

A shorter way of writing to files

We can use the fputs function to get rid of the for loop.

```
#include <stdio.h>
Int main() {
 FILE * fp;
 Fp = fopen("testfile.txt", "w+");
 Fputs("This is an easier way to save data ,",
fp);
 Fputs("We just got rid of the for loop!\n",
fp);
 Fputs("way easier!\n", fp);
 Fclose(fp);
 Return (0);

 }
```

Example

Reading data from a file

Three different functions:

- Fgetc(file_pointer)
- Fgets(buffer, n, file_pointer
- Fscanf(file_pointer, conversion_specifiers, variable_addresses)

Reading data from files

Use fgets to read a string from a given file.

```
fgets(sptr,max,fptr);
```

Syntax

Closing files

- A file must be closed when we have finished with it
- To save memory and to make sure that no data is accidentally written to the file

```
Fclose(file_pointer)
```

Syntax

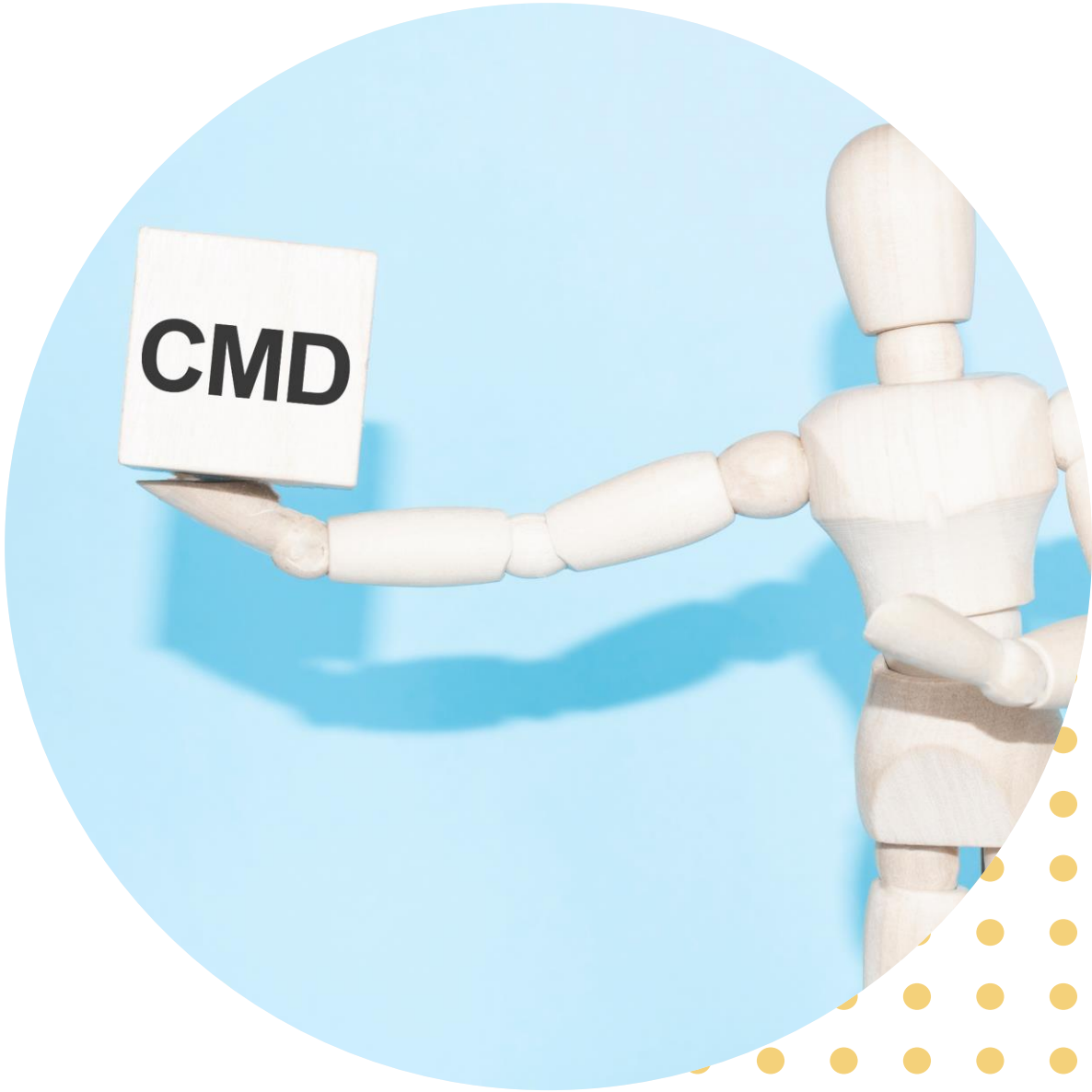




Command line arguments

Command line arguments

- Arguments specified after the name of the program in the system's command line
- Passed on to your program during program execution



Components of a command line argument

Two components:

- Argc: argument count
- Agrv: argument vector

Components of a command line argument

```
Int main( int argc, char *argv[] )
{
// BODY OF THE MAIN FUNCTION
}
Or it can also be written as

Int main( int argc, char **argv[] )
{
// BODY OF THE MAIN FUNCTION
}
```

Syntax

Components of a command line argument

```
int main( int number_of_args, char*
list_of_args[] )
{
 ...
}
 int main( int argc, char** argv )
{
 ...
}
```

Example

Processing command line Args

- Create a function in C to read through all the command line args and return the state
- Can be done in the first few lines of code of the main function