To work with files in C++, we use the header file

```
#include <fstream>
```

Data types:

ifstream – read only
ofstream – write only
fstream – read and write.

```
fstream data;
data.open("info", ios::app);
//info is the name of the file and ios::app is the mode for that file called the file
//access flag.
```

You can use multiple flags at the same time. The way you do this is with a pipe.

```
data.open("info", ios::in|ios::out);
```

The file is located in the same folder as the source file you are debugging or in the same folder as you running the program in. If you want a different folder, you need to include the path in the open command.

To write to the file, you use the << the same as you would with writing to the screen.

To check if a file exists, you can open it

```
data.open("File", ios::in);
```

Then check the fail method

```
if(data.fail())
        //No file
else
        //File, yeah!
```

Checkpoint

12.1 – ios::app, ios::ate, ios::out
12.2 – use pipe to connect them.
12.3 –

```
diskinfo.open("names.dat", ios::out);
```

12.4 –

```
diskinfo.open("customers.txt", ios::app);
```

12.5 –

```
diskinfo.open("payable.txt", ios::out|ios::in);
```

12.6 –

```
fstream datafile("salesfigure.txt", ios::in);
```

to read a line from a file, use the >>

```
string line;
data >> line;
```

anything you can do with cout, you can do with fstreams with regards to formatting.

This applies to special characters such as '\n' or '\t'

We can pass file stream objects to functions.
You should always pass them by reference

```
void getNextLine(fstream &f);
```

Detailed error testing
All file streams have a set of bits maintained internally.
These can be tested using the fstream functions.

Fstream member functions

getline – fetches a full line until you get to the end of line character.
get – gets a single character.
put – writes a single character.

Working with multiple files

You can define multiple stream objects to open more than 1 file at a time.

```
ifstream myInput;
ofstream myOutput;
myInput.open("inputfile", ios::in);
myOutput.open("outputfile", ios::out);
```

Binary files

Using the flat ios::binary opens a file in binary instead of ASCII.
When opened as binary, you use the .read and .write member functions. These functions expect to write the char data type.

.write – 2 parameters
Address – this is the memory address you want to write to the file.
Size – number of bytes you are writing. Use the sizeof function to determine the number of bytes

.read – 2 parameters
Address – this is the memory address you want to save the data to from the file.
Size – number of bytes you are writing. Use the sizeof function to determine the number of bytes

When we write other data types (no char), we need to type cast them.

```
file.read(reinterpret_cast<char *> (&x), sizeof(x));
//this will read the value from the file and store it in x.

file.write(reinterpret_cast<char *> (&x), sizeof(x));
```

When you write structures or classes to a file, any address will be invalid.
To save stuff in structures and classes to a file, store the members one at a time.

We can also access files randomly – this means we can access any byte at any time without regards to what we have previously done.

seekp - seek "put" used to write
seekg – seek "get" used to read.

seekp and seekg move the stream to particular byte. They don't actually read or write the data.

Arguments
offset - this is a long integer telling how far in bytes and which direction to move.
positive is forward and negative is backwards.
Mode flag – tells us what to do with that offset

When you are done, you should test to see if you moved past the beginning or end of file.
Different compilers treat EOF differently so to be safe, call .clear before you seek.

tellp - tell where the cursor is for "putting"
tellg - tell where the cursor is for "getting"