File Input and Output

Chapter 13

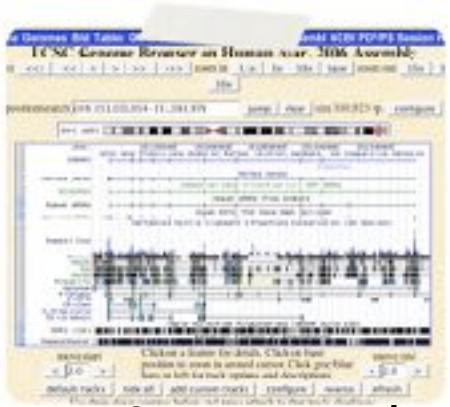
Topics

- Reading from a file (Input)
- Writing to a file (Output)

Files allow programs to easily **store** and **retrieve data** and are used extensively in practically all computer applications!

Spreadsheets

Wordprocessors



Human Genome Project

Databases

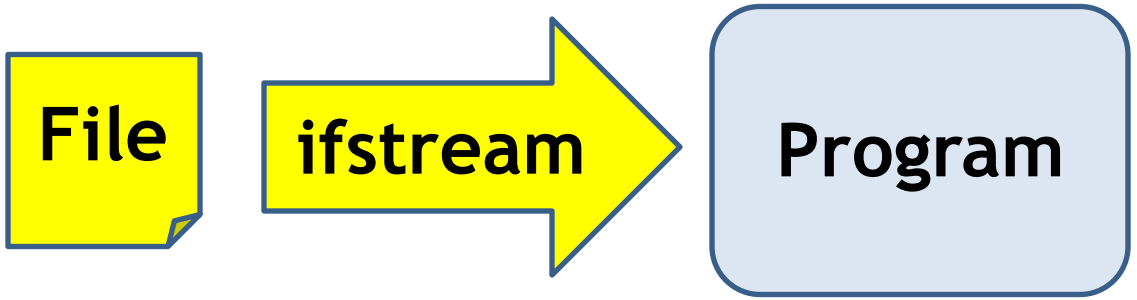
In C++ we use **streams** to interact with data

- For example
  - **`cin`** is an input stream
  - **`cout`** is an output stream
- We can use the operators **`<<`**, **`>>`** on a stream, e.g.

```
cin >> input_variable;
cout << "Hello there";
```

We use a **stream objects** to **input/output** data **from/to** a **file**

INPUT



OUTPUT

INPUT & OUTPUT

6 steps in opening a file for input

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{    ifstream fin;
     fin.open("Z.txt");

     if ( fin.fail() ) {
          cout << "can't open file" << endl;
          return 0;
     }
     string line;
     while( getline(fin,line) )
     {     cout << line << endl;
              // process the string
     }
     fin.close();
}
```

Include <fstream>

Declare ifstream object

Open file

Check if open successful

Read from stream. E.g. get a whole line at a time, and detect End Of File (EOF)!

Close the file

Combining file open with the stream declaration

```
ifstream fin;
fin.open("Z.txt");
```

```
ifstream fin("Z.txt");
```

Some **idioms** used in file Input/ Output…

Testing for successful open

- `if ( !fin )`
- `if ( fin.fail())`

Reading entire file:

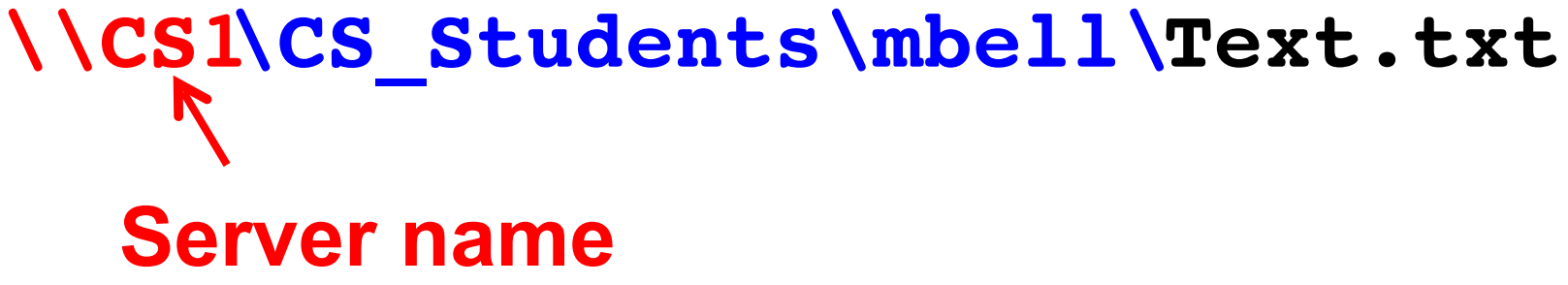
- `while ( getline(fin, line) )`
  - reads an entire line on each loop until **end-of-file** reached
- `while ( !fin.eof() )`
  - Keeps looping till **end-of-file** is reached.
- `while ( fin >> variable1)`
  - reads into variables until the **end-of-file** is reached

Example: Getting a filename from a user and opening the file

```
int main()
{       ifstream fin;
        string filename;
        cout << "Enter a filename" << endl;
        cin >> filename;

        fin.open( filename );
        if ( !fin )
```

Windows-Style **File** Path Name

UNC (Universal Naming Convention) Path



Windows-Style **File** Path Name in C++

C:\Users\jflinn\Desktop\Text.txt

```
string localfile = "c:\\users\jflinn\\desktop\\Text.txt"
```

UNC (Universal Naming Convention) Path

\\CS1\CS_Students\jflinn\Text.txt

```
string netfile = "\\\\CS1\\CS+Students\\jflinn\\Text.txt"
```

6 steps in opening a file for output

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{    ofstream fout;
     fout.open("X.txt");

     if ( fout.fail() ) {
        cout << "can't open file" << endl;
        return 0;
     }
     for ( int i=0; i<10; i++ )
        fout << "line " << i << endl;

     fout.close();
}
```

Include <fstream>

Declare ofstream object

Open file. File will be created if it does not exist. Erased if it does.

check if open successful

Write 10 lines to the output file

close the file

Try it out

- Take the code from the previous slide and run it
  - Try to find X.txt on your computer using windows explorer
  - (it will be somewhere in your project folder)
  - Modify the code so that you take user input, and write that input to the file

sequential and **random access** of data in a file:

- Sequential access:
  - start at beginning of file and go through data in file sequentially
  - to access 100th byte in file, you must go through 99 preceding bytes first
- Random access:
  - read/write data in a file in any order you want!
  - can access the nth byte in a file directly!

`fstream` can be used to declare an **input/output** stream object

- Declare and open an fstream object for **input**.

```
fstream file("myfile.dat",ios::in);
```

- Declare and open an fstream object for **output**.

```
fstream file("myfile.dat",ios::out);
```

- Declare and open an fstream object for **input OR output**.

```
fstream file("myfile.dat",
                    ios::in | ios::out);
```

Bit-wise OR operator

Mode Flags

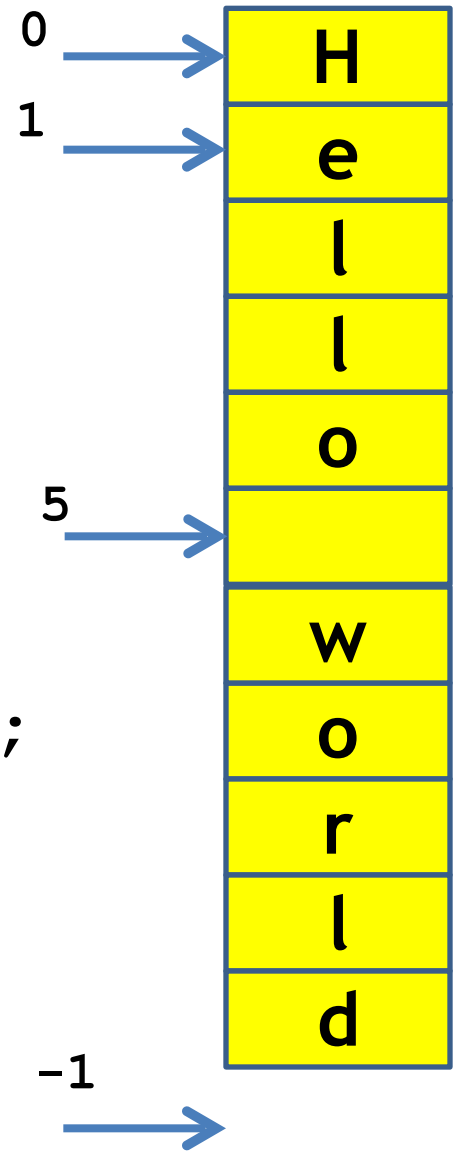
for `fstream, ifstream or ofstream`

`ios::in`	open for input (not valid for **ofstream**)
`ios::out`	open for output (not valid for **ifstream**)
`ios::app`	Open for **output only**, create new file or **append** to existing file
`ios::ate`	Go to end of existing file for **input/output**
`ios::binary`	read/write in binary mode (not text mode)

A file stream is like a ***stream of bytes***

A **file stream pointer** points to the current position in the file stream

```
char ch = 'H';
fin >> ch;


string str = "ello";
fin >> str;

str = " world";
fin >> str;
```



You can **get** the current **file stream pointer** location with …

- **`tellg`** : return current "reading" position in the file in bytes

```
int whereAmI = inFile.tellg();
```

- **`tellp`** : return current "writing" position in output file in bytes

```
whereAmI = outFile.tellp();
```

For a file stream, both tell you the same thing

You can **set** the **file stream pointer** position in an **`fstream`** object for reading and writing

`fstream` member functions:

`seekg(offset)`

`seekp(offset)`

`offset`: long integer specifying number of bytes to move relative to the beginning of the file. g is for reading and p is for writing.

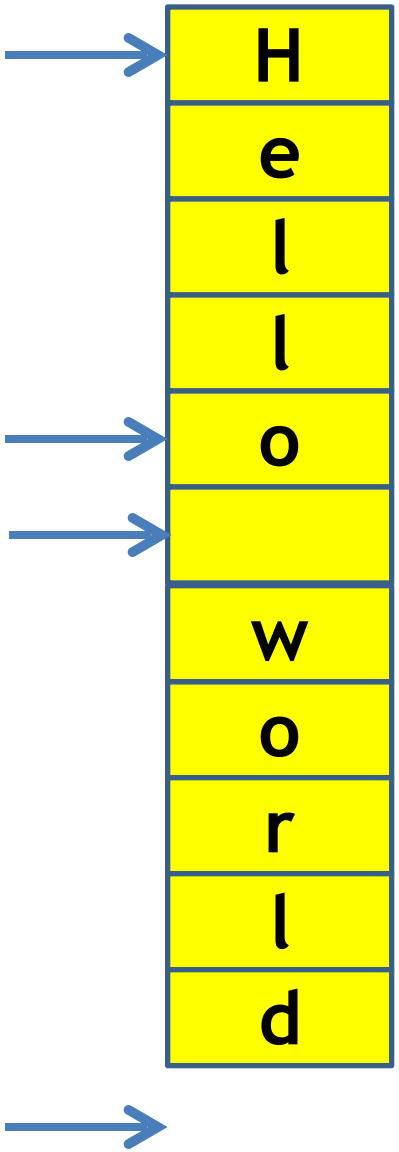
Random Access!

Setting the file pointer for reading

```
fin.seekg(4)
```

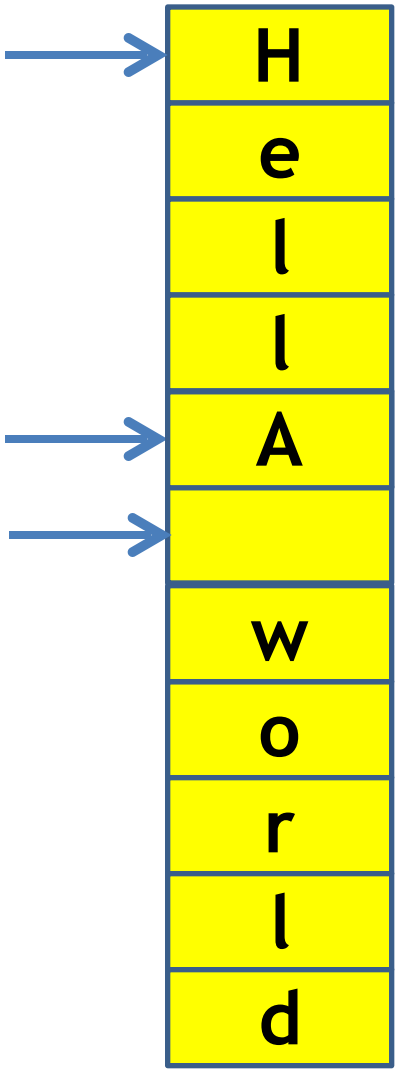
```
char ch;
fin >> ch;
```

```
string str;
fin >> str;
```



Setting the file pointer for writing

```
fin.seekp(4)

fin << 'A';
```



You can set the location from which to apply your offset

`fstream` member functions:

```
seekg( offset, location )
seekp( offset, location )
```

`offset`: long integer specifying number of bytes to move

`location`: **(optional)** starting point for the move, specified by

Location Flag	Description
`ios::beg`	Beginning of File
`ios::cur`	Current position in File
`ios::end`	End of File

Example of setting the read and write positions in a file opened for input/output

```
// Set read position 25 bytes
// after beginning of file
fin.seekg(25, ios::beg);

// Set write position 10 bytes
// before current position
fout.seekp(-10, ios::cur);
```

Note: If file stream pointer is at **EOF**, we need to **clear()** the file stream to be able to move it to a new position again.

```
while (file >> word) {}

file.clear();  // clear all flags
file.seekp(0, ios::beg); // back to front

while (file >> word) {} // read again!
```

A program that stores how many times it has been run in an file

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{   int count = 0;
    fstream iofile( "count.dat", ios::in | ios::out  ); // Open a file for input/
output
    if ( ! iofile ) // check to see if the file exists already or not
    {
        iofile.open( "count.dat", ios::out );
        cout << "First time! ";
        count++;
        iofile << count;
        iofile.close();
    }
    else
    {   iofile >> count;                // read in previous count
        count++;
        cout << "This program has run " << count << " times" << endl;
        iofile.clear();                 // clear any flags
        iofile.seekg( 0, ios::beg );    // go to the start of the file
        iofile << count;                // output the new count to file
    }
    iofile.close();  // Close the file
}
```

Hands on Exercise

- Go to the WWW and download an e-book. (https://www.gutenberg.org/)
  - I've downloaded one for you if you don't want to get your own.
- **Open** it for input and output.
- **Read** and count every word in the document.
- **Clear** the file stream flag when you reach the **EOF**
- **Seek** to the end of the file, i.e.
  - **seekp(0, ios::end)**
- **Write** "**\n\nThis file has X words.**"
  - Where **X** is the number of words you counted in the file

WARNING!

PAST THIS POINT AT YOUR OWN INTELLECTUAL RISK

SUPPLEMENTAL: BASICS OF BINARY FILES

Binary file are accessed differently from text files

- **Binary files** store data in the same format that a computer's memory uses.
- **Text files** store data in ASCII characters. Files are opened in text mode (as text files) by default.

Accessing binary files from C++

- Use the **ios::binary** flag to **open** a file in binary mode

```
file.open("myfile.dat",ios::binary);
```

- Reading and writing of binary files requires using **read** and **write** member functions

```
read(char *buffer, int numberBytes)
write(char *buffer, int numberBytes)
```

Exercise: Using **write** to save an object in binary format…

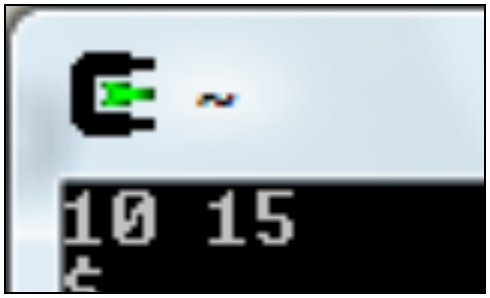
```
#include <fstream>
#include <iostream>
using namespace std;
class point { public: int x; int y; };

int main()
{
    ofstream outFile( "point.dat", ios::binary );
    if (!outFile) {
        return -1;
    }
    point p;
    p.x = 10;
    p.y = 15;
    outFile.write(
              reinterpret_cast<char *>( &p ),
              sizeof( point )
            );
    outFile.close();
}
```

Open a file for output in binary mode

Type cast the pointer to the "point" object to a char pointer as required by the write( ) method

Use sizeof to compute the number of bytes required by the write( ) method.

Exercise: Using **read** to load an object from the file saved by the program shown on the previous slide…




```
#include <fstream>
#include <iostream>
using namespace std;
class point { public: int x; int y; };

int main()
{
    ifstream inFile( "point.dat", i
    point p;
    if (!inFile) {
        return -1;
    }
    inFile.read(
              reinterpret_cas
              sizeof( point
         );
    cout << p.x << " " << p.y << endl;
    inFile.close();
}
```

Open a file for input in binary mode

Type cast the pointer to the "point" object to a char pointer as required by the read( ) method

Output the elements of the point object loaded from the file.

Optional Exercise: Binary Files

- Attempt to open and inspect the file created by your program using notepad. What happens? Why?
- Use Visual Studio to open the file in binary mode - inspect how the struct object was stored in the file.