I/O Library

Reading and writing to files

- Use streams from STL
- Syntax similar to `cerr, cout`

```
#include <fstream>
using std::string;
using Mode = std::ios_base::openmode;

// ifstream: stream for input from file
std::ifstream f_in(string& file_name, Mode mode);

// ofstream: stream for output to file
std::ofstream f_out(string& file_name, Mode mode);

// stream for input and output to file
std::fstream f_in_out(string& file_name, Mode mode);
```

There are many modes under which a file can be opened

Mode	Meaning
`ios_base::app`	append output
`ios_base::ate`	seek to EOF when opened
`ios_base::binary`	open file in binary mode
`ios_base::in`	open file for reading
`ios_base::out`	open file for writing
`ios_base::trunc`	overwrite the existing file

Regular columns

Use it when:

- The file contains organized data
- Every line has to have all columns

```
1 1 2.34   One     0.21
2 2 2.004  two     0.23
3 3 -2.34  string  0.22
```

O.K.

```
1 1 2.34   One     0.21
2 2 2.004  two     0.23
3 3 -2.34  string  0.22
```

Fail

```
1 1 2.34   One     0.21
2 2 2.004  two
3 3 -2.34  string  0.22
```

Reading from ifstream

```
#include <fstream>  // For the file streams.
#include <iostream>
#include <string>
using namespace std;  // Saving space.
int main() {
  int i;
  double a, b;
  string s;
  // Create an input file stream.
  ifstream in("test_cols.txt", ios_base::in);
  // Read data, until it is there.
  while (in >> i >> a >> s >> b) {
    cout << i << ", " << a << ", "
         << s << ", " << b << endl;
  }
  return (0);
}
```

```cpp
#include <fstream>  // For the file streams.
#include <iostream>
using namespace std;
int main() {
  string line, file_name;
  ifstream input("test_bel.txt", ios_base::in);
  // Read data line-wise.
  while (getline(input, line)) {
    cout << "Read: " << line << endl;
    // String has a find method.
    string::size_type loc = line.find("filename", 0);
    if (loc != string::npos) {
      file_name = line.substr(line.find("=", 0) + 1,
                              string::npos);
    }
  }
  cout << "Filename found: " << file_name << endl;
  return (0);
}
```

Writing into text files

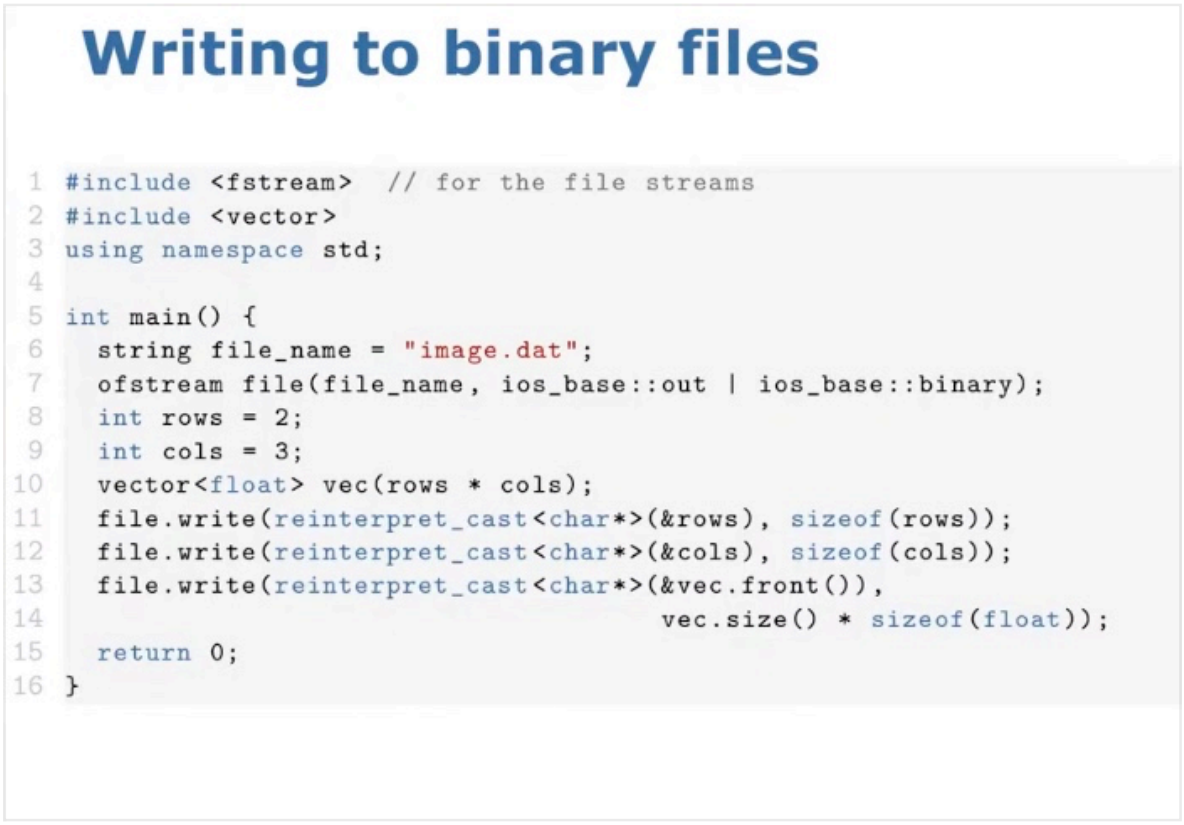
With the same syntax as `cerr` und `cout` streams, with `ofstream` we can write directly into files

```cpp
#include <iomanip>  // For setprecision.
#include <fstream>
using namespace std;
int main() {
  string filename = "out.txt";
  ofstream outfile(filename);
  if (!outfile.is_open()) { return EXIT_FAILURE; }
  double a = 1.123123123;
  outfile << "Just string" << endl;
  outfile << setprecision(20) << a << endl;
  return 0;
}
```

Read/Write binary files (Serialization)

Writing to binary files

- We write a sequence of bytes.
- Writing/reading is fast.
- No precision loss for floating point types.
- Substantially smaller than ascii files
- Syntax:
  file.write(reinterpret_cast<char*>(&a), sizeof(a));



reinterpret_cast will convert from integer format to binary format.

Reading from binary files

- We read a sequence of bytes.
- Binary files are not human readable.
- We must know the structure of contents.
- Syntax:
  file.read(reinterpret_cast<char*>(&a), sizeof(a));

Reading from binary files

```cpp
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;
int main() {
  string file_name = "image.dat";
  int r = 0, c = 0;
  ifstream in(file_name,
              ios_base::in | ios_base::binary);
  if (!in) { return EXIT_FAILURE; }
  in.read(reinterpret_cast<char*>(&r), sizeof(r));
  in.read(reinterpret_cast<char*>(&c), sizeof(c));
  cout << "Dim: " << r << " x " << c << endl;
  vector<float> data(r * c, 0);
  in.read(reinterpret_cast<char*>(&data.front()),
          data.size() * sizeof(data.front()));
  for (float d : data) { cout << d << endl; }
  return 0;
}
```

Pros:

- I/O Binary files is faster than ASCII format.
- Size of files is drastically smaller.
- There are many libraries to facilitate serialisation.

Cons

- Ugly syntax.
- File is not readable.
- You need to know the format before reading.