CSE 1325: Object-Oriented Programming

Lecture 23

# File and String Streams With Iterators

Mr. George F. Rice
george.rice@uta.edu

Office Hours:
Prof Rice 12:30 Tuesday and Thursday in ERB 336
For TAs see this web page

Where are we going?
And why am I in this handbasket?

# Today's Topics

- Streaming in C++
  - File Streams
  - String Streams
  - Error Handling
- Iterators

# Overview
# C++ vs Java Streams

- In C++, instance **`std::ifstream{filename}`** for file input, **`std::ofstream{filename}`** for file output
  - These are similar to **`BufferedReader(FileReader)`** and **`BufferedWriter(FileWriter)`** in Java
- C++ can also stream from and to *strings* using “String Streams”, **`std::istringstream iss{s}`** and **`std::ostringstream oss;`**
  - **`oss.str()`** will return all text streamed to **`oss`** thus far
  - **`oss.str("")`** to *clear* all text streamed to **`oss`** and start over
- In C++, use **`<<`** and **`>>`** for file, string, and all other streams
  - And of course **`std::getline(iss, s);`** or **`std::getline(iss, s, c);`**
    - The next token goes in **`s`** (delimited by **`c`** if provided, **`‘\n’`** otherwise)
  - To parse a complex **`s`**, try **`std::istringstream iss{s}; std::getline …`**
- C++ requires polling stream error states (**`good, bad, fail, eof`**) rather than **`IOException`** as in Java
  - Although you can request an exception if the state changes

Contrasting C++ with Java

## Opening a Text File for Reading

Read.java

**Java**

(We would normally put this line in a try-with-resources. We omit that here to highlight symmetry with C++.)

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Read {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader((args[0])));
        String s;
        while((s=br.readLine())!=null) System.out.println(s);
    }
}
```

read.cpp

**C++**

**`std::ifstream`**, equivalent to Java's **`BufferedReader`**, works with both **`std::getline`** and the **`>>`** operator.

```cpp
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    std::ifstream ist{std::string{argv[1]}};
    if (!ist) throw std::runtime_error{"can’t open input file"};

    std::string s;
    while (std::getline(ist, s)) std::cout << s << std::endl;
}
```

Know this idiom!

http://www.cplusplus.com/reference/fstream/ifstream/

# Contrasting C++ with Java

## Symmetry!

Read.java

**Java**

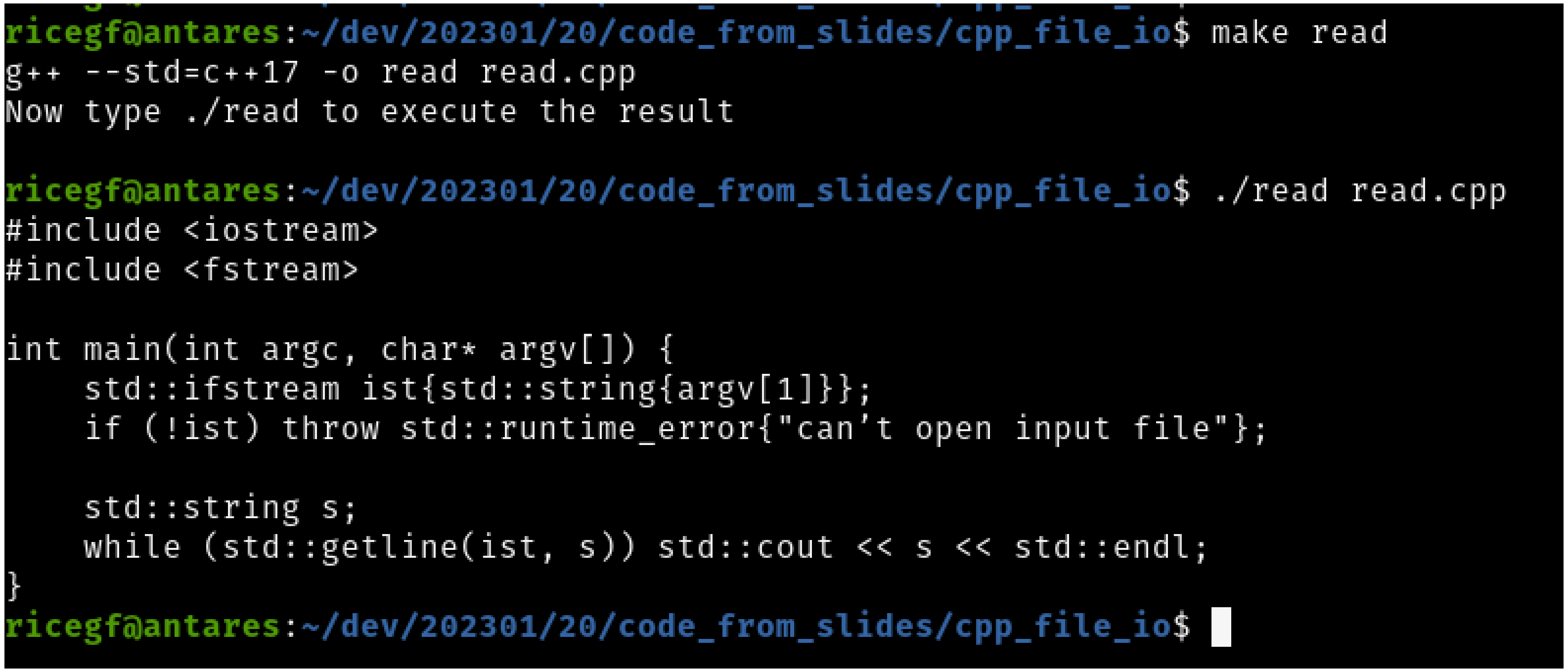
```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Read {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader((args[0])));
        String s;
        while((s=br.readLine())!=null) System.out.println(s);
    }
}
```

read.cpp

**C++**

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

int main(int argc, char* argv[]) {
    std::ifstream ist{std::string{argv[1]}};
    if (!ist) throw std::runtime_error{"can't open input file"};

    std::string s;
    while (std::getline(ist, s)) std::cout << s << std::endl;
}
```

# Opening a Text File for Reading



```
ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$ make read
g++ --std=c++17 -o read read.cpp
Now type ./read to execute the result

ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$ ./read read.cpp
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    std::ifstream ist{std::string{argv[1]}};
    if (!ist) throw std::runtime_error{"can't open input file"};

    std::string s;
    while (std::getline(ist, s)) std::cout << s << std::endl;
}
ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$
```

read.cpp

```cpp
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    std::ifstream ist{std::string{argv[1]}};
    if (!ist) throw std::runtime_error{"can't open input file"};

    std::string s;
    while (std::getline(ist, s)) std::cout << s << std::endl;
}
```

Contrasting C++ with Java

## Opening a Text File for Writing

Write.java

**Java**

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Write {
    public static void main(String[] args) throws IOException {
        BufferedWriter br = new BufferedWriter(new FileWriter(args[0]));
        br.write("Writing this to " + args[0] + "\n");
        br.close(); // flush buffer (automatic with try-with-resources)
    }
}
```

write.cpp

**C++**

**std::ofstream**, equivalent to Java's **BufferedWriter**, works with the **<<** operator.

```cpp
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    std::ofstream ofs {std::string{argv[1]}};
    if (!ofs) throw std::runtime_error{"can't open output file"};
    ofs << "Writing this to " << argv[1] << std::endl;
}
```

Thus, reading and writing files takes only 2 additional lines versus reading std::cin and writing std::cout!
And yes – std::cin and std::cout are simply pre-instanced istream and ostream objects. :-)

## Contrasting C++ with Java

## Symmetry!

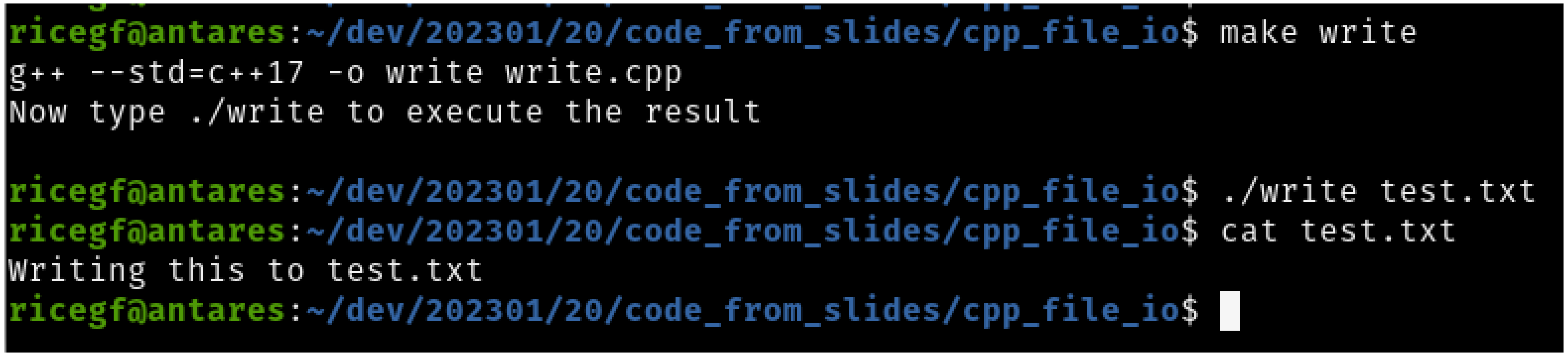
Java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

class WriteFile {
    public static void main(String[] args) throws IOException {
        BufferedWriter br = new BufferedWriter(new FileWriter(args[0]));
        br.write("Writing this to " + args[0] + "\n");
        br.close(); // flush buffer (automatic with try-with-resources)
    }
}
```

C++

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

int main(int argc, char* argv[]) {
    std::ofstream ofs {std::string{argv[1]}};
    if (!ofs) throw std::runtime_error{"can’t open output file"};
    ofs << "Writing this to " << argv[1] << std::endl;
}
```

# Opening a Text File for Writing



```
ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$ make write
g++ --std=c++17 -o write write.cpp
Now type ./write to execute the result

ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$ ./write test.txt
ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$ cat test.txt
Writing this to test.txt
ricegf@antares:~/dev/202301/20/code_from_slides/cpp_file_io$
```

```cpp
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    std::ofstream ofs {std::string{argv[1]}};
    if (!ofs) throw std::runtime_error{"can't open output file"};
    ofs << "Writing this to " << argv[1] << std::endl;
}
```

# Class Survey

- The class survey is now open!
  - I see *no* feedback until *after* your final grades are posted
  - I read and consider *every* comment!
  - **Completely anonymous**
- WARNING: Survey closes … eventually!

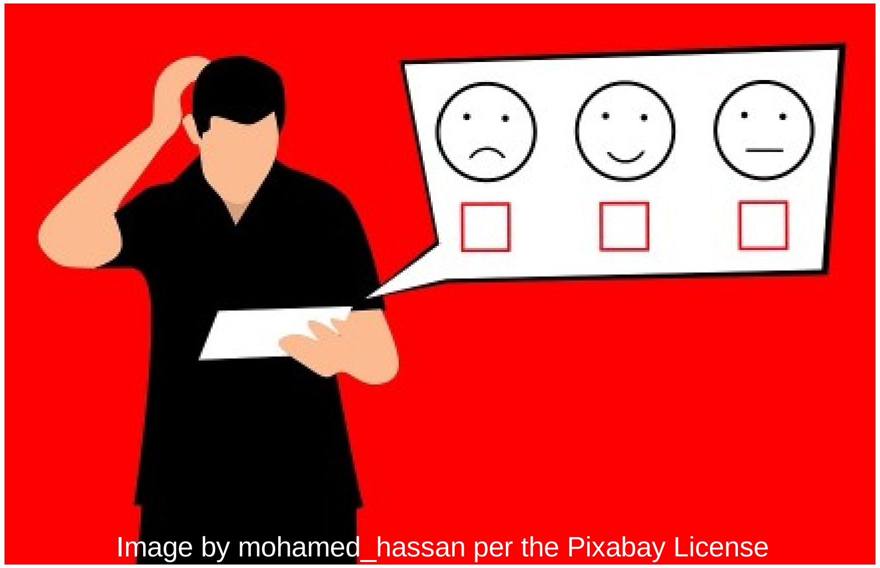


Image by mohamed_hassan per the Pixabay License
https://pixabay.com/en/experience-feedback-survey-customer-3239623/

Benefit for You: The nag screens and reminder emails will cease once you take the survey!

Benefit for Me: Invaluable insight into what worked and what needs to change.

## Class Survey

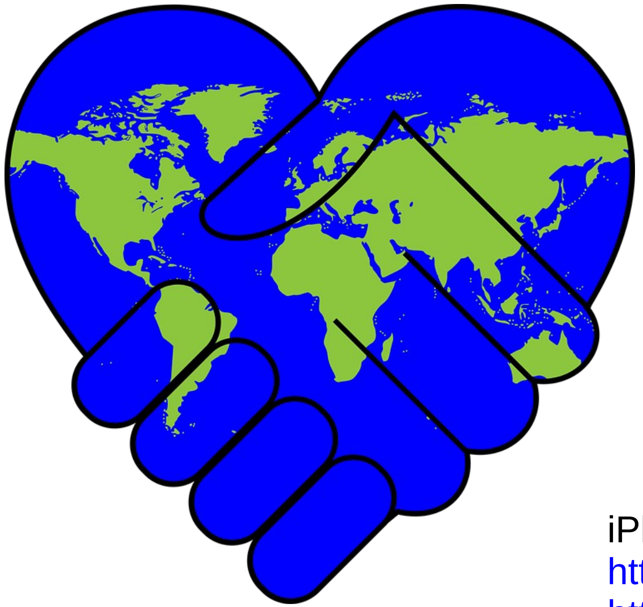
- Improvements that originated from student feedback
  - Suggested Solutions to all homework provided via GitHub and reviewed in class the moment the homework is submitted
  - Suggested solutions licensed under GPL 3 for student reuse
  - Soup-to-nuts in-class project at end of the first/ start of second section
  - "...in 5 Pages" summaries for bash and GitHub at start of class
  - Screencasts for learning bash, and debugging
  - Additional video Lecture 02 on Java Fundamentals
  - (More) practice exams to prepare for the actual exam
  - Providing CSE-VM, a standard homework environment with all tools pre-installed to simplify environment setup and recovery
  - Slides posted *before* the lecture, with all source code on GitHub
  - Replacing CSE-VM with GitHub CodeSpaces (coming Fall 2024)
  - Moving C++ map, set, and iterators earlier than the last lecture
- **What are YOUR suggestions for future classes?**

# Good vs Bad Feedback

- Good feedback is (1) specific, (2) relevant, and (3) actionable
- Good feedback examples:
  - "I needed more menu-driven interface examples using the Menu class, skipping the picocli library that we didn't actually use"
  - "The assignment text was too detailed – I needed a one-page summary at the start of each assignment"
  - "A list of common compiler errors and how to fix them would help" (though I found a useful list to be unattainable)
  - "I missed working on a team for the final project" (no final project atm )
- Bad feedback examples:
  - "I couldn't understand a lot of stuff" (not specific)
  - "We should have learned to write Android apps" (not relevant)
  - "8 am is too early" (probably true, but not actionable)

## Your Action Today

- Help future students
- Help me help future students
- Make the world and UTA better
- Complete the class survey!



iPhone by OpenClipart-Vectors and World Heart by GDJ
https://pixabay.com/vectors/iphone-cell-phone-phone-160307/
https://pixabay.com/vectors/cooperation-friendship-hands-1301790/

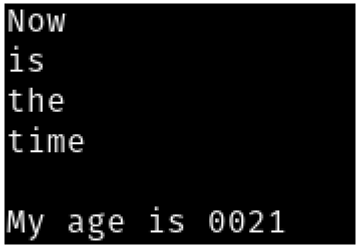
# String Streams

- C++ also has "String Streams" that stream to / from strings as if they are files
- For input, use `std::istringstream iss{s};`
  - String s contains the text to stream in, just like `std::cin`
  - Once s is fully parsed, `iss.eof()` becomes true

  -
    ```
    std::string phrase = "Now is the time";
    std::istringstream iss{phrase};
    std::string s;
    while(iss >> s) std::cout << s << std::endl;
    ```
- For output, use `std::ostringstream oss;`
  - Whatever you stream to oss is saved in a buffer
  - `oss.str()` retrieves all of the text in the buffer
  - `oss.str("")` clears the buffer

  -
    ```
    int age = 21;
    std::ostringstream oss;
    oss << "My age is" << std::setfill('0') << std::setw(4) << age;
    std::string rpt = oss.str(); // rpt = "My age is 0021"
    ```

stringstreams.cpp

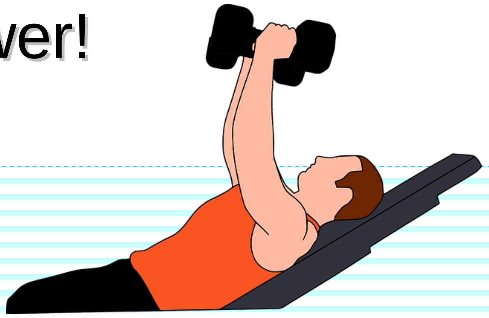


```
Now
is
the
time

My age is 0021
```

# String Stream-Based Converters

A **stringstream** (from <sstream>) adds ALL stream capabilities to your string-editing arsenal!

*Feel* the power!



double_conversions.cpp

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <cmath>

double string_to_double(std::string s) {
    std::istringstream iss{s}; // make an input stream from string s
    double d;
    iss >> d;                  // stream a double from string s
    if (!iss) throw std::runtime_error("double format error");
    return d;
}
std::string double_to_string(double d) {
    std::ostringstream oss;   // make an output stream
    oss << d;                 // stream a double to the output stream
    if (!oss) throw std::runtime_error("string format error");
    return oss.str();         // return std::string containing all text we streamed
}
```

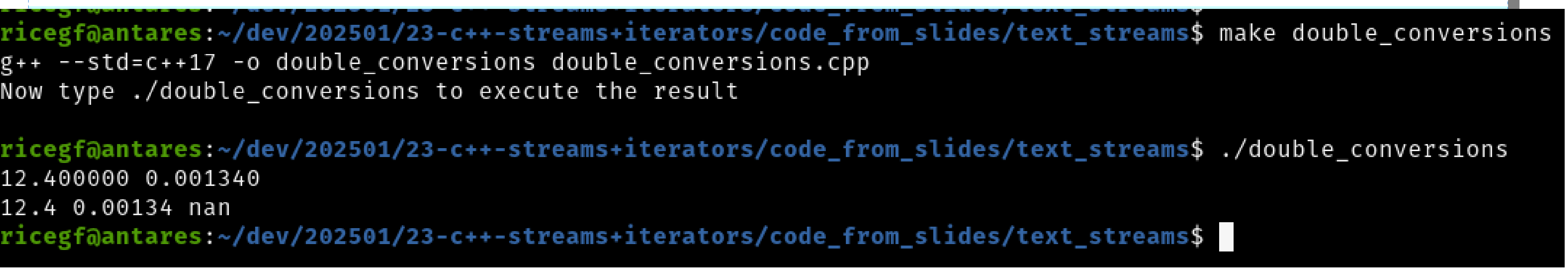
You can clear and reuse an ostringstream with `oss.str("");`

Weight lifter drawing by Hohamed_hassan per the Pixabay License

# Testing isstream Conversions

double_conversions.cpp

```cpp
int main() {
    double d1 = string_to_double("12.4");
    double d2 = string_to_double("1.34e-3");
    try {
        double d3 = string_to_double("twelve point three");
        std::cerr << "No exception from 'twelve point three'!" << std::endl;
    } catch (std::runtime_error e) { // working correctly if thrown!
    }

    std::string s1 = double_to_string(12.4);
    std::string s2 = double_to_string(1.34e-3);
    std::string s3 = double_to_string(NAN);    // "Not A Number"

    std::cout << std::fixed << std::setprecision(6) << d1 << ' ' << d2 << std::endl;
    std::cout << s1 << ' ' << s2 << ' ' << s3 << std::endl;
}
```



```
ricegf@antares:~/dev/202501/23-c++-streams+iterators/code_from_slides/text_streams$ make double_conversions
g++ --std=c++17 -o double_conversions double_conversions.cpp
Now type ./double_conversions to execute the result

ricegf@antares:~/dev/202501/23-c++-streams+iterators/code_from_slides/text_streams$ ./double_conversions
12.400000 0.001340
12.4 0.00134 nan
ricegf@antares:~/dev/202501/23-c++-streams+iterators/code_from_slides/text_streams$
```

# String streams

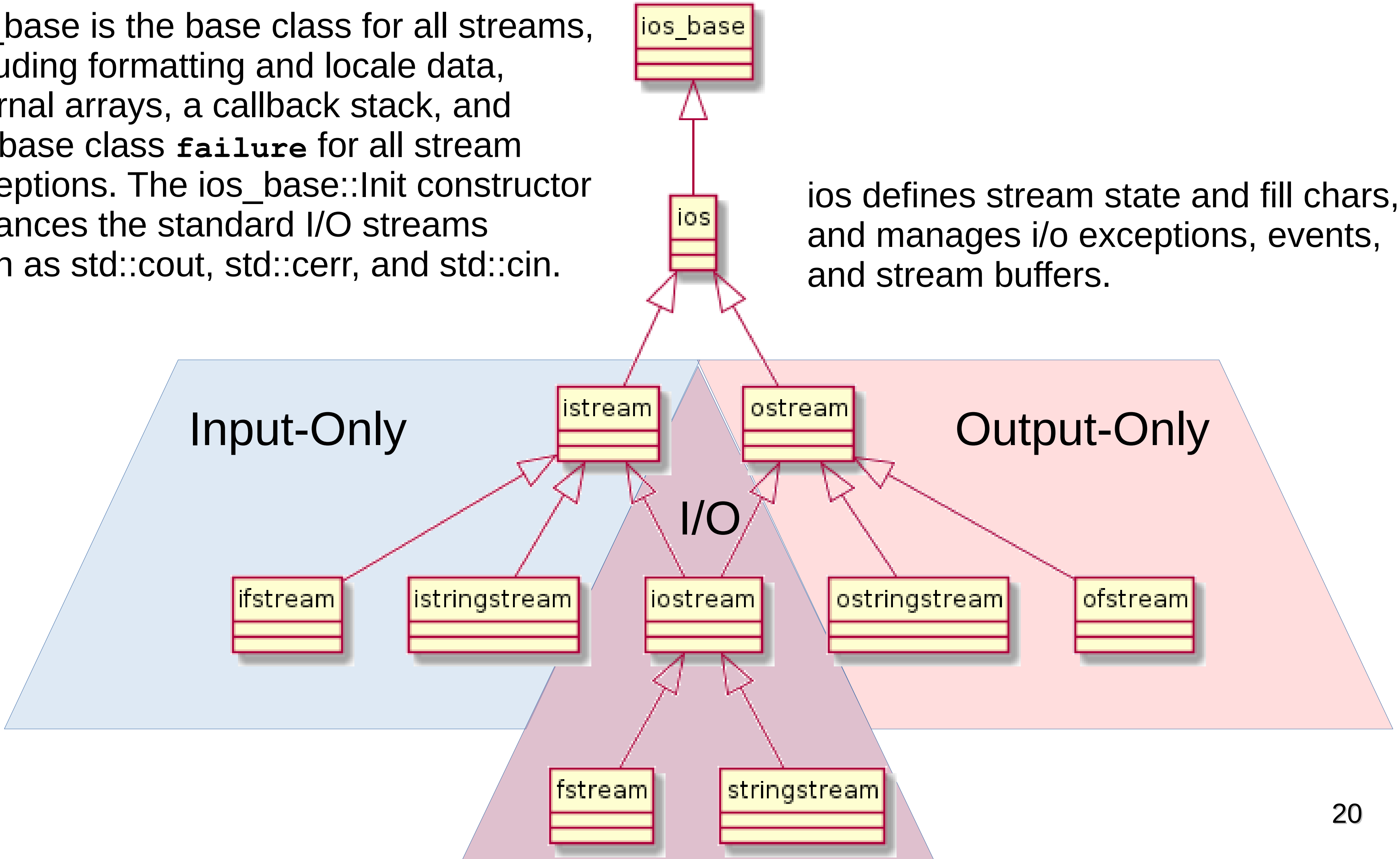
- String streams are very useful for
  - formatting into a fixed-sized space
    - Creating a table or graph on the terminal
    - Formatting data to / from text widgets in a GUI dialog,
      e.g., converting a text entry field into a double
      e.g., formatting a double for display in a label
    - Any time you need to build a well-formatted string representation of an object
  - for extracting typed objects out of a string
    - Sometimes used with getline when you don't know how many elements and what type of each in the input
    - Use **`std::getline(iss, s, ‘,’)`** to parse by commas

http://www.cplusplus.com/reference/sstream/stringstream/

# File and String Streams

ios_base is the base class for all streams, including formatting and locale data, internal arrays, a callback stack, and the base class **`failure`** for all stream exceptions. The ios_base::Init constructor instances the standard I/O streams such as std::cout, std::cerr, and std::cin.

ios defines stream state and fill chars, and manages i/o exceptions, events, and stream buffers.



# I/O Error Handling (for ALL streams)

- Stream errors by default do NOT throw an exception as Java does with IOException
  - You may request exceptions or poll the state of the stream

```
Readings::Readings(std::istream& ist) {
    Reading reading;

    // Throw std::ios_base::failure if ist become bad
    ist.exceptions (ist.exceptions() | std::ifstream::badbit);

    while(ist >> reading) _readings.push_back(reading); // Collect readings

    // Or manually throw the exception based on stream status
    if(!ist.eof()) throw std::ifstream::failure("Error reading temps");
}
```

- ios reduces all errors to 1 of 4 states
  - **good()** *// the operation succeeded*
  - **eof()** *// we hit the end of input ("end of file")*
  - **fail()** *// something unexpected and recoverable*
  - **bad()** *// something unexpected and fatal*

https://cplusplus.com/reference/ios/ios/exceptions/

## Stream State Example
# Reading Integers

- Ended by “terminator character”
  - 1 2 3 4 5 *
  - State is **fail()**
- Ended by format error
  - 1 2 3 4 5.6
  - State is **fail()**
- Ended by “end of file”
  - 1 2 3 4 5 end of file
  - 1 2 3 4 5 Control-Z (Windows)
  - 1 2 3 4 5 Control-D (Mac* / Linux)
  - State is **eof()** (**fail()** is also true)
- Something really bad
  - Disk format error or computer is on fire
  - State is **bad()**

* A second Control-D at the start of a line may also be required

# File open modes

- By default, an ifstream opens its file for reading
- By default, an ofstream opens its file for writing
- Alternatives:
  - `ios_base::app      // append (always add to end of the file)`
  - `ios_base::ate      // “at end” (start at end but seek anywhere)`
  - `ios_base::binary   // binary – beware of system specific behavior`
  - `ios_base::in       // for reading`
  - `ios_base::out      // for writing`
  - `ios_base::trunc    // truncate file to 0-length`
- A file mode is optionally specified after the name of the file:
  - `ofstream of1 {name1};   // defaults to ios_base::out`
  - `ifstream if1 {name2};   // defaults to ios_base::in`
  - `ofstream ofs {name, ios_base::app};   // append, not overwrite`

  -
    ```
    fstream fs {"myfile", ios_base::in | ios_base::out};
                           // both in and out
    ```

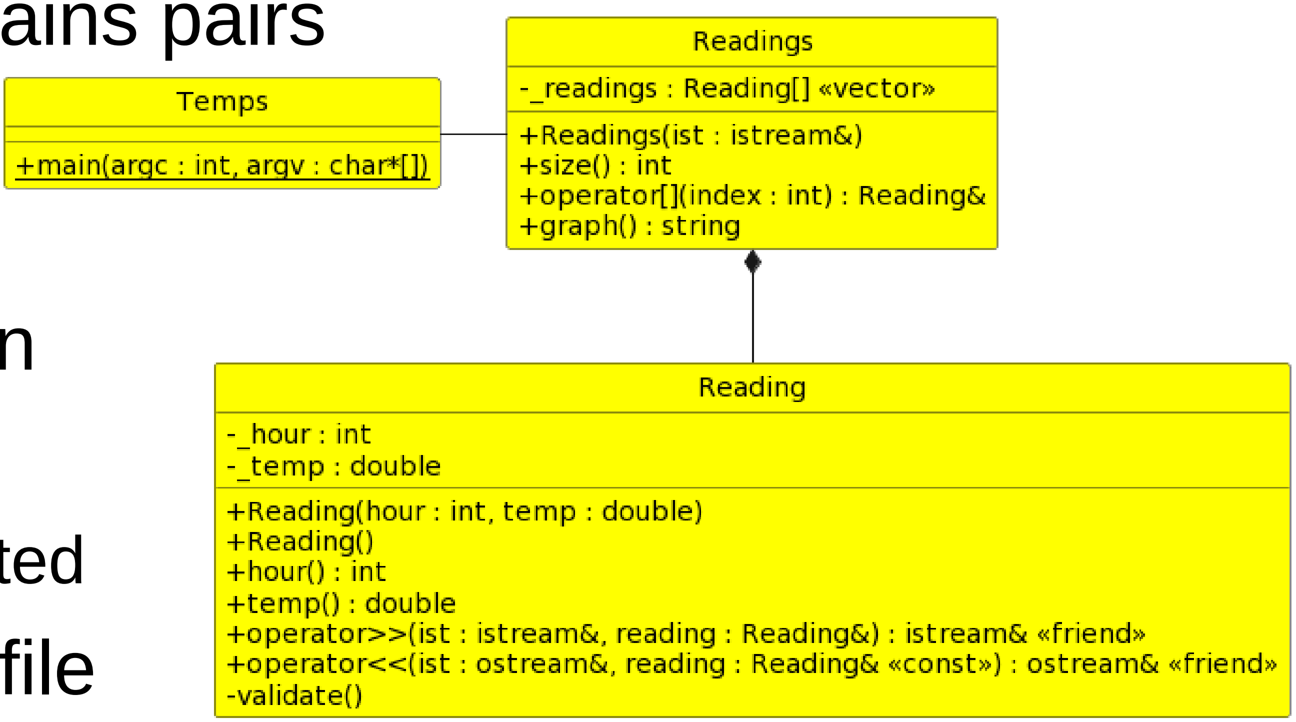
# Text vs. Binary File I/O

- Use text whenever possible
  - You can read it (without a fancy program)
  - You can debug your programs more easily
  - Text is portable across different systems
  - Size (compressed) is typically comparable
  - Most information can be represented reasonably as text
- Use binary when you must
  - E.g. image files, sound files for faster decoding
  - Compressed and / or encrypted files

## Reading from Files

# A More Realistic Example

- File temps.txt contains pairs
  - Hour (0-23)
  - Temperature (⁰F)
- Stop reading file on
  - End of file
  - Anything unexpected
- Read & graph the file
  - Store each reading in a Reading object
  - Store readings and generate a bar graph using a Readings object
- The main function is in main.cpp


Temps
+main(argc : int, argv : char*[])
Readings
-_readings : Reading[] «vector»
+Readings(ist : istream&)
+size() : int
+operator[](index : int) : Reading&
+graph() : string
Reading
-_hour : int
-_temp : double
+Reading(hour : int, temp : double)
+Reading()
+hour() : int
+temp() : double
+operator>>(ist : istream&, reading : Reading&) : istream& «friend»
+operator<<(ist : ostream&, reading : Reading& «const») : ostream& «friend»
-validate()

temps.txt

```
0 42.8
1 40.9
2 38.6
3 35.9
4 34.1
5 33.7
6 32.0
7 32.1
8 33.3
9 35.1
```

# Reading Class

reading.h

```
#ifndef __READING_H
#define __READING_H

#include <iostream>

class Reading { // a temperature reading
    public:
        Reading(int hour = 0, double temp = 0); // or use chaining
        int hour();
        double temp();
        friend std::istream& operator>>(std::istream  &ist, Reading& reading);
        friend std::ostream& operator<<(std::ostream& ost, Reading& reading);
    private:
        void validate(); // Throw exception if _hour is invalid
        int _hour;       // 0 to 23 GMT
        double _temp;    // Celsius
};
#endif
```

# Reading Class

reading.cpp

```cpp
#include "reading.h"
#include <iomanip>

Reading::Reading(int hour, double temp)
       : _hour{hour}, _temp{temp} {validate();}

int Reading::hour() {return _hour;}
double Reading::temp() {return _temp;}

void Reading::validate() {
    if (_hour < 0 || 23 < _hour)
         throw std::runtime_error{"Invalid hour: " + std::to_string(_hour)};
}

std::istream& operator>>(std::istream& ist, Reading& reading) {
  ist >> reading._hour >> reading._temp;
  reading.validate();
  return ist;
}

std::ostream& operator<<(std::ostream& ost, Reading& reading) {
    ost << std::setw(4) << reading._hour << ": "
        << std::setw(6) << std::right << std::fixed
                        << std::setprecision(1) << reading._temp;

    return ost;
}
```

## Readings Class

**readings.h**

```
#ifndef __READINGS_H
#define __READINGS_H

#include "reading.h"
#include <vector>

class Readings {
  public:
    Readings(std::istream& ist);
    int size();                        // Number of readings
    Reading& operator[](int index);    // Access a reading with subscript
    std::string graph();               // Return char graph of data
  private:
    std::vector<Reading> _readings;
};
#endif
```

Here’s an example of constructing an object from an istream (a file, a string stream – even std::cin!)

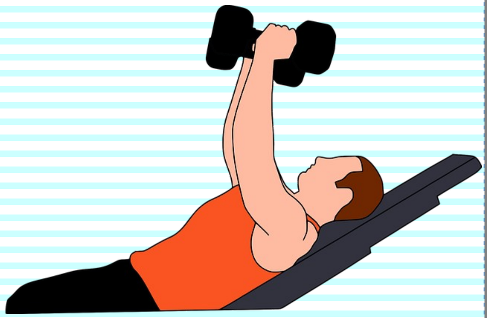
Here’s an example of overloading the subscript operator [ ]

# Readings Class

readings.cpp

```
#include "readings.h"
#include <sstream>
#include <fstream>

Readings::Readings(std::istream& ist) {
    Reading reading;
    while(ist >> reading) {  // Using our overloaded >> operator for Reading
        _readings.push_back(reading);
    }
    if(!ist.eof()) throw std::ifstream::failure("Error reading temperatures");
}
int Readings::size() {return _readings.size();}

Reading& Readings::operator[](int index) {  // Defining our [] operator
    return _readings.at(index); // throws std::out_of_range if needed
}
std::string Readings::graph() {
    std::ostringstream oss; // Use a stream to format a string
    for(auto r : _readings) {
        oss << r << ' ';
        for (int j=0; j<r.temp()/2; j++) oss << '#';
        oss << '\n';
    }
    return oss.str();
```

Here’s a string stream example



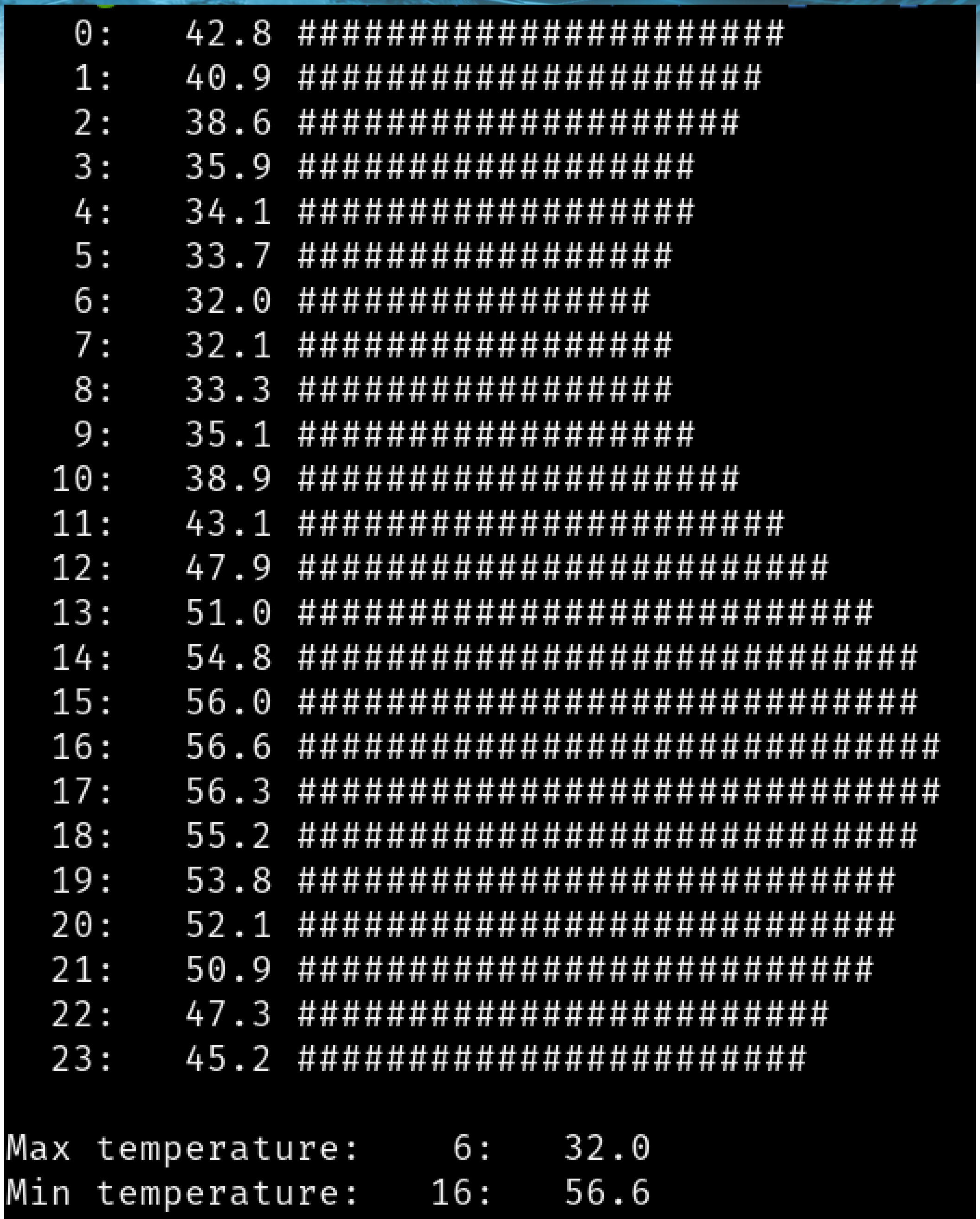
Weight lifter drawing by Hohamed_hassan per the Pixabay License

# Temps
## (main function)

**temps.cpp**

```cpp
#include "readings.h"
#include <iostream>
#include <fstream>

int main(int argc, char* argv[]) {
    if(argc != 2) {                                  // Argument validation
        std::cerr << "usage: " << argv[0] << " temps.txt" << std::endl;
        return -1;
    }
    std::string filename{argv[1]};                       // Open the file
    std::ifstream ifs{filename};
    if (!ifs) {
        std::cerr << "Invalid filename: " << filename << std::endl;
        return -2;
    }

    Readings readings{ifs};                           // Load the data
    std::cout << readings.graph() << std::endl; // Graph the data

    Reading min = readings[0];                        // Calculate min & max
    Reading max = readings[0];                        // Yes, std::minmax exists …
    for(int i=1; i<readings.size(); ++i) {
        if(min.temp() < readings[i].temp()) min = readings[i];
        if(max.temp() > readings[i].temp()) max = readings[i];
   }
   std::cout << "Max temperature: " << max << std::endl; // Print min & max
   std::cout << "Min temperature: " << min << std::endl;
}
```

# Graphing Temperatures



```
   0:   42.8 ######################
   1:   40.9 #####################
   2:   38.6 ####################
   3:   35.9 ##################
   4:   34.1 ##################
   5:   33.7 #################
   6:   32.0 ################
   7:   32.1 #################
   8:   33.3 #################
   9:   35.1 ##################
  10:   38.9 ####################
  11:   43.1 ######################
  12:   47.9 ########################
  13:   51.0 ##########################
  14:   54.8 ############################
  15:   56.0 ############################
  16:   56.6 #############################
  17:   56.3 #############################
  18:   55.2 ############################
  19:   53.8 ###########################
  20:   52.1 ###########################
  21:   50.9 ##########################
  22:   47.3 ########################
  23:   45.2 #######################

Max temperature:    6:   32.0
Min temperature:   16:   56.6
```

# Pointers in C++ are just like C

pointers.cpp

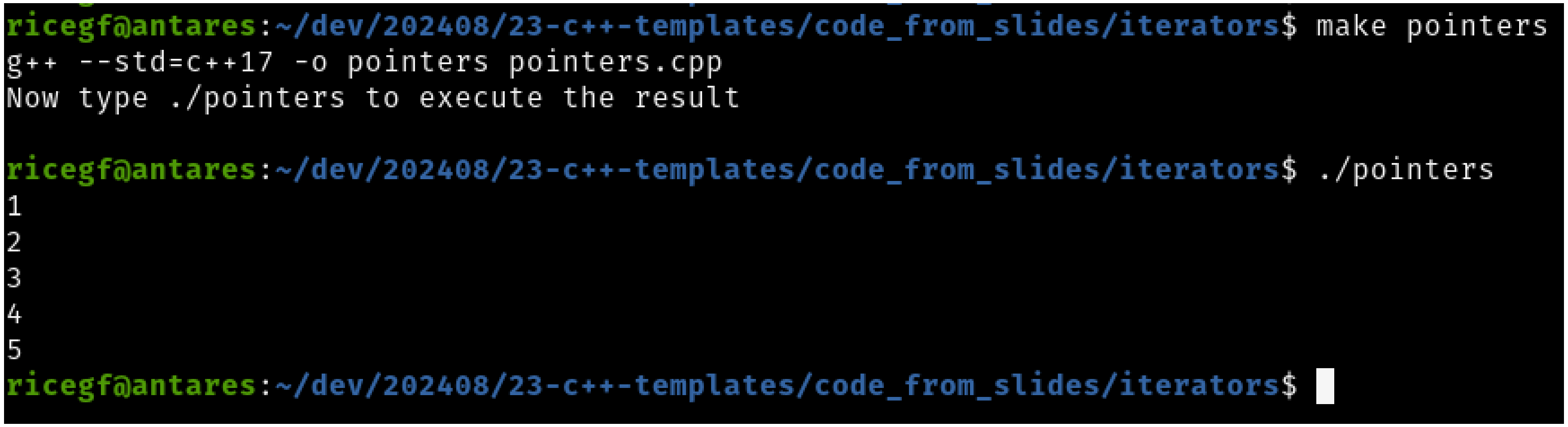
```cpp
#include <iostream>

int main() {
  int v[] = {1, 2, 3, 4, 5};
  int* pv = v;
  int* v_end = v+5;
  do {
      std::cout << *pv << std::endl;
  } while(++pv != v_end);
}
```

The pointer is an address that points inside the array.

Pointer assignment sets pv to point to the first element.

v_end is assigned the address one *past* the last element of the vector.

*pv dereferences the pointer, returning the value to which it points.



```
ricegf@antares:~/dev/202408/23-c++-templates/code_from_slides/iterators$ make pointers
g++ --std=c++17 -o pointers pointers.cpp
Now type ./pointers to execute the result

ricegf@antares:~/dev/202408/23-c++-templates/code_from_slides/iterators$ ./pointers
1
2
3
4
5
ricegf@antares:~/dev/202408/23-c++-templates/code_from_slides/iterators$ 
```

# Iterators

- **Iterator**: A pointer-like instance of a nested *class* used to access items managed by the outer class instance
- A* nested class is typically provided by a container such as **`std::vector`**
  - **`iterator`** is the class name nested inside the container
- The container itself provides methods to obtain iterators to its first and (one past the) last methods
  - Get two* **`iterator`** instances with **`begin()`** and **`end()`**
- Use iterators much like pointers
  - An iterator can always be incremented via **`++`**, dereferenced with **`*`**, and compared to other iterators

* A second class, const_iterator, with getter methods cbegin() and cend() are also provided for handling const values.

# Iterating Through a Vector

## With C++ Iterators

iteration.cpp

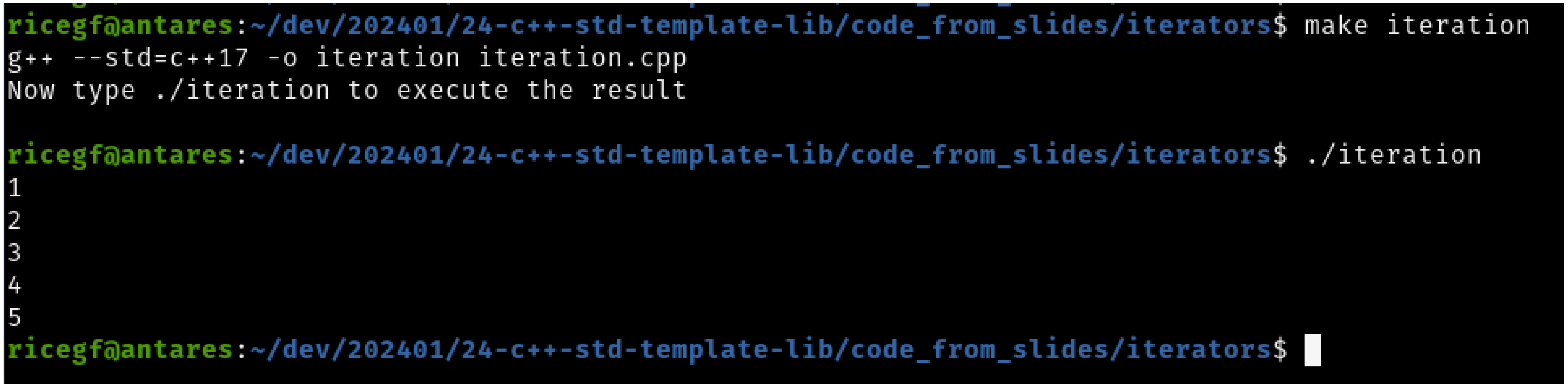
```
#include <vector>
#include <iostream>

int main() {
  std::vector<int> v = {1, 2, 3, 4, 5};

  std::vector<int>::iterator it = v.begin();

  do {
      std::cout << *it << std::endl;
  } while(++it != v.end());
}
```

The iterator is a nested class inside the container into which it points.

The begin() method returns an iterator pointing to the first element.

You may treat it almost exactly like a pointer!

The end() method returns an iterator pointing one *past* the last element.



```
ricegf@antares:~/dev/202401/24-c++-std-template-lib/code_from_slides/iterators$ make iteration
g++ --std=c++17 -o iteration iteration.cpp
Now type ./iteration to execute the result

ricegf@antares:~/dev/202401/24-c++-std-template-lib/code_from_slides/iterators$ ./iteration
1
2
3
4
5
ricegf@antares:~/dev/202401/24-c++-std-template-lib/code_from_slides/iterators$ 
```

# Comparing Pointers to Iterators

## Pointers

pointers.cpp

```
#include <iostream>

int main() {
   int v[] = {1, 2, 3, 4, 5};

   int* pv = v;
   int* v_end = v+5;

   do {
       std::cout << *pv << std::endl;
   } while(++pv != v_end);
}
```

## Iterators

iteration.cpp

```
#include <vector>
#include <iostream>

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};

    std::vector<int>::iterator it = v.begin();

    do {
        std::cout << *it << std::endl;
    } while(++it != v.end());
}
```

## Comparing Pointers to Iterators

### Pointers

pointers.cpp

```cpp
#include <iostream>

int main() {
  int v[] = {1, 2, 3, 4, 5};

  auto pv = v;
  auto v_end = v+5;

  do {
      std::cout << *pv << std::endl;
  } while(++pv != v_end);
}
```

### Iterators

iteration.cpp

```cpp
#include <vector>
#include <iostream>

int main() {
   std::vector<int> v = {1, 2, 3, 4, 5};

// std::vector<int>::iterator it = v.begin();
   auto it = v.begin();

   do {
      std::cout << *it << std::endl;
   } while(++it != v.end());
}
```

And this is why we love `auto` so much!

# What We Learned Today

- **Iterators** are nested classes in collections that behave just like pointers
  - Get them using **`auto it = v.begin();`** (points to first element) and **`it = v.end();`** (points to end+1)
  - For an iterator into a **`const`** collection, use **`v.cbegin()`** and **`v.cend()`** instead!
- We **stream text files** by instancing **`std::ofstream{filename}`** or **`std::ifstream{filename}`**
  - Then use **`<<`**, **`>>`**, and **`std::getline`** as usual
- We **stream strings** by instancing **`std::ostringstream`** and **`std::istringstream{std::string}`**
  - Then use **`<<`**, **`>>`**, and **`std::getline`** as usual
  - Use **`std::ostringstream::str()`** to get / set the text and **`std::ostringstream::str("")`** to clear the text
- Streams have 4 states for **stream error management**: **`good()`**, **`eof()`**, **`fail()`** (recoverable), and **`bad()`** (non-recoverable)
  - **`while(ifs >> s)…`** or **`while(std::getline(ifs, s))…`** are common idioms!
  - **`if(iss.eof()) std::cerr << "Unexpected end of stream!";`** is, too!
- Prefer text files (compressed if needed) instead of binary
  - But if you want to do binary files in C++, keep going below for a *brief* intro!

The End

## Optional (NOT on the Exam)
## C++ Binary File I/O

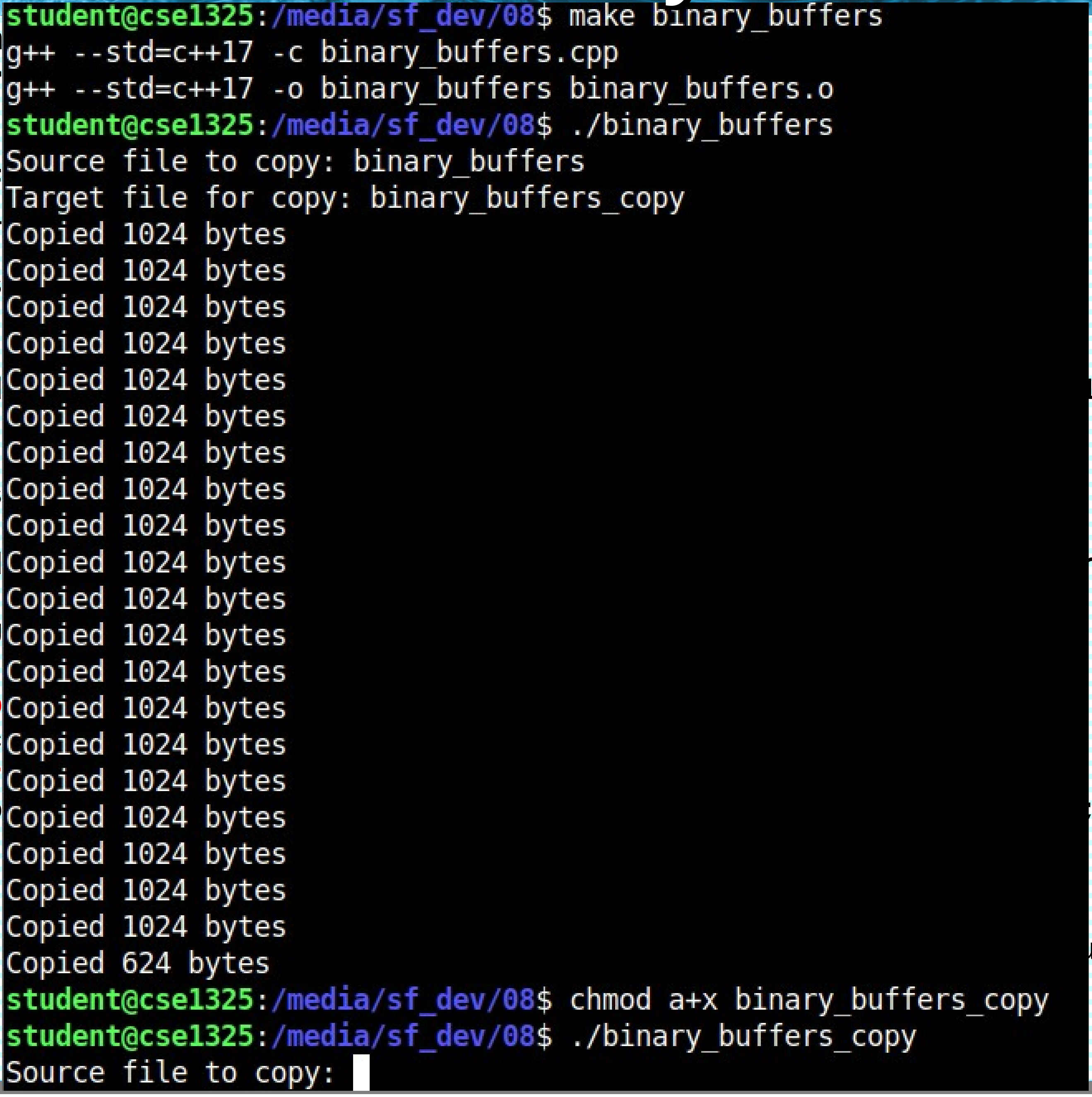
- C++ can also read and write binary files (of course)
  - Read bytes into a pre-allocated buffer
  - Write bytes from a buffer
- Unlike Java, C++ has two file pointers, read and write
  - This makes directly moving bytes in a file much simpler
- Here's a few simple examples to get you started

# Buffered Binary File I/O

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

int main() {
    const int BUFFER_SIZE = 1024;
    std::string filename;
    std::cout << "Source file to copy: ";
    std::getline(std::cin, filename);

    std::ifstream ifs {filename,std::ios_base::binary};  // note: binary
    if (!ifs) {std::cerr << "Can't open " << filename << std::endl; return -1;}

    std::cout << "Target file for copy: ";
    std::getline(std::cin, filename);
    std::ofstream ofs {filename, std::ios_base::binary};    // note: binary
    if (!ofs) {std::cerr << "Can't open output file: aborted" << std::endl; return -2;}

    char buffer[BUFFER_SIZE];
    while(ifs) {
        ifs.read(buffer, BUFFER_SIZE);
        if (ifs.gcount()) {
            ofs.write(buffer, ifs.gcount());
            if (!ofs) {std::cerr << "File write error: aborted" << std::endl; return -4;}
        }
        std::cout << "Copied " << ifs.gcount() << " bytes" << std::endl;
    }
    if (!ifs.eof()) {std::cerr << "Source file read error" << std::endl; return -3;}
    return 0;
}
```

# Buffered Binary File I/O

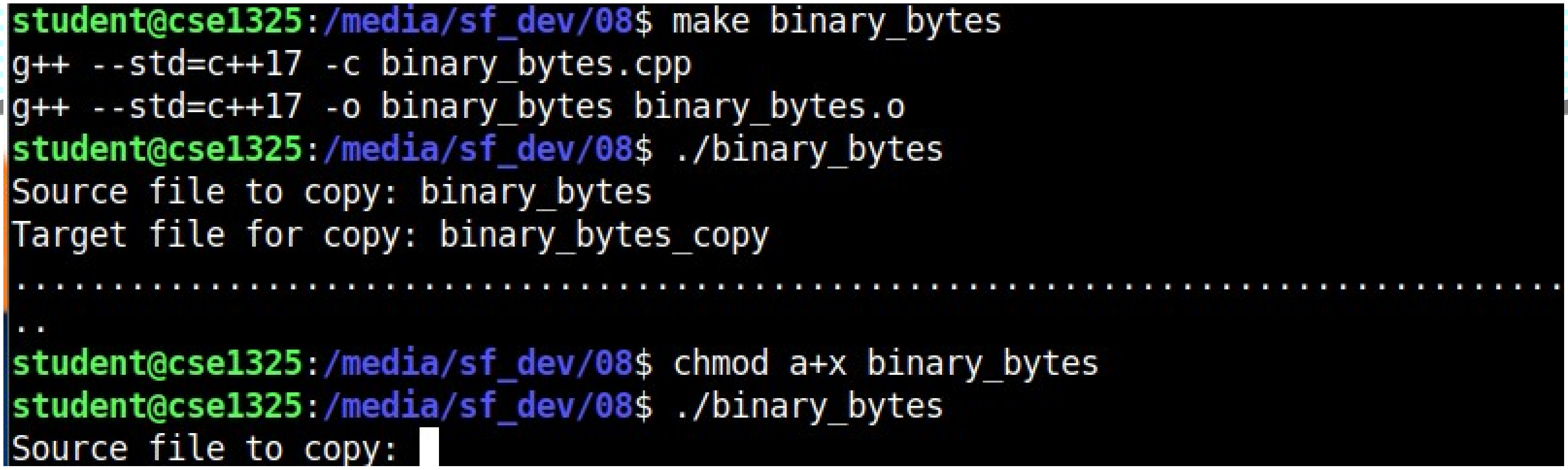
```
#include <iostream
#include <fstream>

int main() {
    const int BUFF
    std::string fi
    std::cout << "
    std::getline(s

    std::ifstream
    if (!ifs) {std                                                  1;}

    std::cout << "
    std::getline(s
    std::ofstream
    if (!ofs) {std                                                  return -2;}

    char buffer[BU
    while(ifs) {
        ifs.read(b
        if (ifs.gc
            ofs.wr
            if (!o                                                  ; return -4;}
        }
        std::cout
    }
    if (!ifs.eof()                                                  urn -3;}
    return 0;
}
```



```
student@cse1325:/media/sf_dev/08$ make binary_buffers
g++ --std=c++17 -c binary_buffers.cpp
g++ --std=c++17 -o binary_buffers binary_buffers.o
student@cse1325:/media/sf_dev/08$ ./binary_buffers
Source file to copy: binary_buffers
Target file for copy: binary_buffers_copy
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 1024 bytes
Copied 624 bytes
student@cse1325:/media/sf_dev/08$ chmod a+x binary_buffers_copy
student@cse1325:/media/sf_dev/08$ ./binary_buffers_copy
Source file to copy:
```

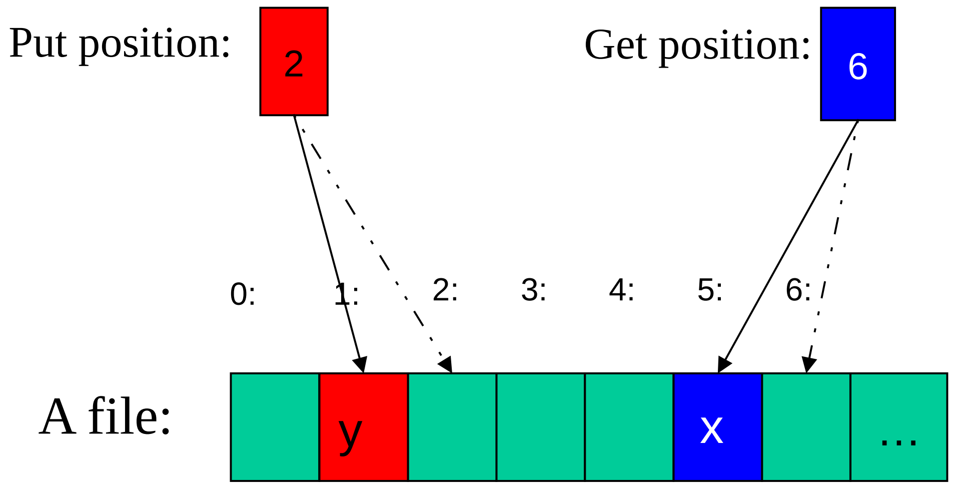
# Binary File I/O by Bytes

```
// Same as before

    char byte;
    int counter = 0;
    while(ifs) {
        ifs.get(byte);
        if (ifs) {
            ofs.put(byte);
            if (!ofs) {std::cerr << "File write error: aborted" << std::endl; return -4;}
        }
        if (!(++counter % 256)) std::cout << ".";
    }
    std::cout << std::endl;
    if (!ifs.eof()) {std::cerr << "Source file read error" << std::endl; return -3;}
    return 0;
}
```



```
student@cse1325:/media/sf_dev/08$ make binary_bytes
g++ --std=c++17 -c binary_bytes.cpp
g++ --std=c++17 -o binary_bytes binary_bytes.o
student@cse1325:/media/sf_dev/08$ ./binary_bytes
Source file to copy: binary_bytes
Target file for copy: binary_bytes_copy
..............................................................................
..
student@cse1325:/media/sf_dev/08$ chmod a+x binary_bytes
student@cse1325:/media/sf_dev/08$ ./binary_bytes
Source file to copy: 
```

# Positioning in a filestream



```
fstream fs {name};    // open for input and output (C++ 11 and later)

fs.seekg(5); // move reading position (‘g’ for ‘get’) to 5 (the 6th character)
char ch;
fs.get(ch);        // read the x and increment the reading position to 6
cout << "sixth character is " << ch << '(' << int(ch) << ")\n";

fs.seekp(1); // move writing position (‘p’ for ‘put’) to 1 (the 2nd character)
fs.put('y');       // write and increment writing position to 2
```

## Positioning

- Whenever you can
  - Use simple streaming
    - Streams/streaming is a very powerful metaphor
    - Write most of your code in terms of "plain" **istream** and **ostream**
    - Default backups for file modifications are fairly easy to implement, e.g., rename the old file with a trailing '~' or '.bak', and write the updated file to the original filename
  - Positioning is far more error-prone
    - Handling of the end of file position is system dependent and basically unchecked
    - A subtle bug can destroy the file being edited

## Optional (NOT on the Exam)
## C++ Threads

- C++ also supports threads (of course)
  - The code to execute is any *function*
    - To run a method as a thread, simply call it using a lambda function
  - The thread automatically starts on instance – no start() method
  - As with Java, Thread::join() merges to the main thread
- C++ lacks synchronized methods and scopes
  - But it has a std::mutex class with lock() and unlock() methods
- The g++ compiler requires the -pthread flag to generate thread-compatible code
- Given those hints, you can probably understand the C++ version of the Kentucky Derby simulator already

C++ Version of Kentucky Derby

## Horse Interface (including thread)

horse.h

Thread!

```
#pragma once
#include <string>
#include <mutex>

class Horse {
    public:
        Horse(std::string name, int speed);
        std::string name();
        bool running();
        std::string view(); // String showing its position
        void gallop();      // The thread that moves the horse
        static std::string winner();  // Racing to grab the mutex
                                      // and insert _name here!
    protected:
        std::string _name;  // Name by which horse is known
        bool _running;      // True while thread is running
        int _position;      // Distance from the finish line
        int _speed;         // Rough time between gallops (ms)
        static std::mutex m;  // Controls write access to _winner
        static std::string _winner; // Name of the winning horse
}; // NOTE: Static fields declared here must also be defined in the .cpp file
```

C++ Version of Kentucky Derby

## Horse Implementation sans Thread

horse.cpp
(1 of 2)

```
#include "horse.h"
#include <chrono>
#include <thread>

Horse::Horse(std::string name, int speed)
    : _name{name}, _speed{speed}, _position{30} { }

// The horses race to be first to grab the mutex
//    and insert their name in the static _winner string!
std::mutex Horse::m;                  // Static fields declared in .h must also
std::string Horse::_winner = "";      //    be defined in the .cpp file
std::string Horse::winner() {return _winner;}

// Getters
std::string Horse::name() {return _name;}
bool Horse::running() {return _running;}

// Representation of the horse on the racetrack
std::string Horse::view() {
    std::string result;
    for (int i = 0; i < _position; ++i) result += (i%5 == 0 ? ':' : '.');
    result += " " + _name;
    return result;
}
```

C++ Version of Kentucky Derby

## Horse Thread Implementation

horse.cpp
(2 of 2)

Thread!

Synchronize!

```
// The thread
void Horse::gallop() {
    _running = true;
    while (_winner.empty()) {
        std::this_thread::sleep_for(
            std::chrono::milliseconds(_speed + std::rand() % 200));
        if (_position > 0)
            --_position;
        else {
            m.lock();
            if (_winner.empty()) _winner = _name;
            m.unlock();
        }
    }
    _running = false;
}
```

# C++ Version of Kentucky Derby
# Main Function

horserace.cpp
(1 of 2)

```
#include <iostream>
#include <thread>
#include <chrono>
#include <ctime>
#include <algorithm>
#include <vector>
#include <array>
#include "horse.h"

const int HORSES = 20;

int main() {
    // Randomize the pseudorandom number generator
    srand(time(NULL));

    // Pick random names for the horses (based mostly on Kentucky Derby winners)
    std::vector<std::string> names {"Legs of Spaghetti","Ride Like the Calm",
        "Duct-taped Lightning","Flash Light","Speedphobia","Cheat Ah!",
        "Go For Broken","Whining Racer","Spectacle","Cannons a'Boring",
        "Plodding Prince","Lucky Snooze","Wrong Way","Fawlty Powers","Broken Tip",
        "American Zero","Exterminated","Great Regret","Manual","Lockout",
    };
    std::random_shuffle(names.begin(), names.end());
    names.push_back("2 Biggaherd"); // Default name for "too many"
```

## C++ Version of Kentucky Derby

# Main Function

horserace.cpp
(2 of 2)

```
    // Our competitors, to be assigned random names and speeds (smaller is faster)
    std::vector<Horse> horses;
    for (int i=0; i<HORSES; ++i)
        horses.push_back(Horse{names[std::min(i,
                   static_cast<int>(names.size())-1)], 100 + std::rand() % 100});

    // Instance a thread for each horse
    std::array<std::thread, HORSES> threads;
    for (int i=0; i<HORSES; ++i)
        threads[i] = std::thread{[&,i] {horses[i].gallop();}};

    // Display the horse track as the race runs
    bool running = true;
    while (running) {
        std::cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
        for (Horse& h : horses) {
            running &= h.running();  // Stop when any horse stops running
            std::cout << h.view() << std::endl;  // Display this horse's position
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    // Join the threads (let all of the horses come to a stop)
    for (std::thread& t : threads) t.join();

    // Announce the winner!
    std::cout << "\n### The winner is " << Horse::winner() << std::endl;
}
```

C++ Lambda! Copy i, reference other vars.

Thread!

Thread!

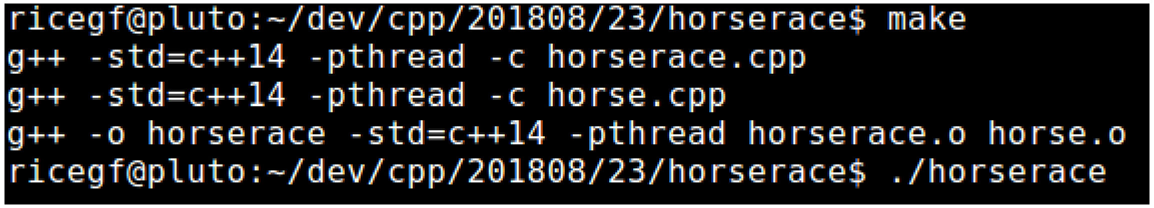
# The Obligatory Makefile

Makefile

```
CXXFLAGS += -std=c++17 -pthread

all: main

debug: CXXFLAGS += -g
debug: main

rebuild: clean main

main: horserace.o horse.o
	g++ -o horserace $(CXXFLAGS) horserace.o horse.o
horserace.o: horserace.cpp horse.h
	g++ $(CXXFLAGS) -c horserace.cpp
horse.o: horse.cpp horse.h
	g++ $(CXXFLAGS) -c horse.cpp
clean:
	-rm -f *.o *~ horserace
```

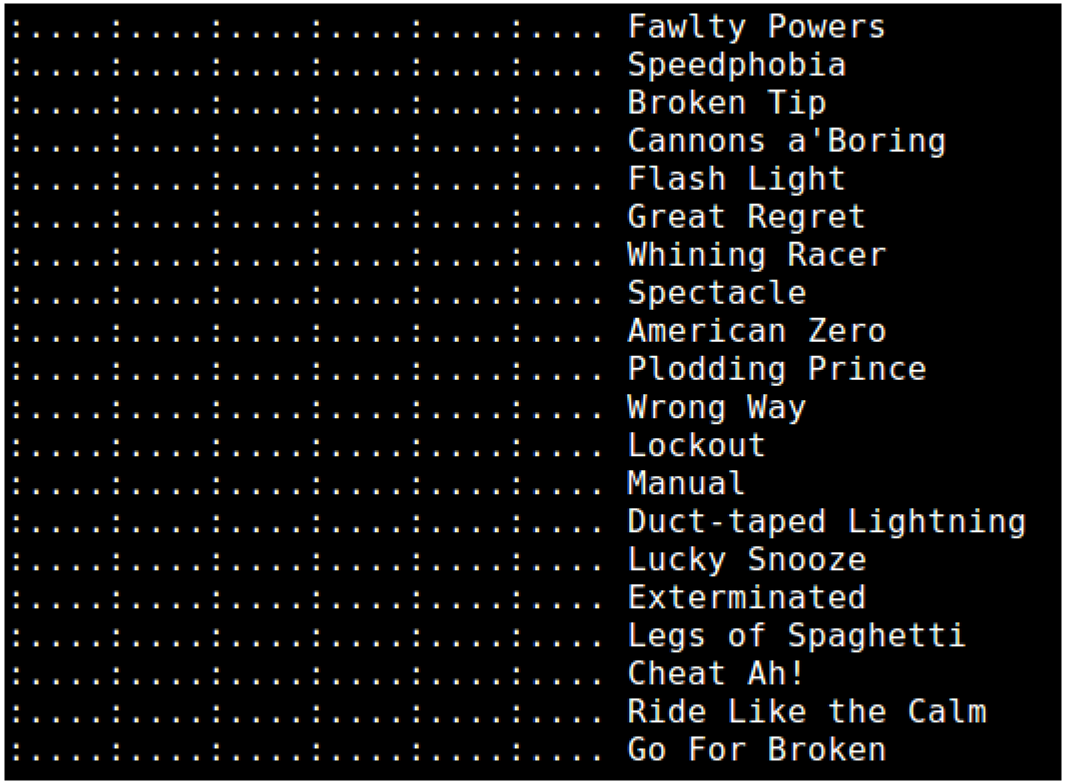
The **`-pthread`** flag is required to enable threads in the gcc C++ compiler

# Running the Kentucky Derby

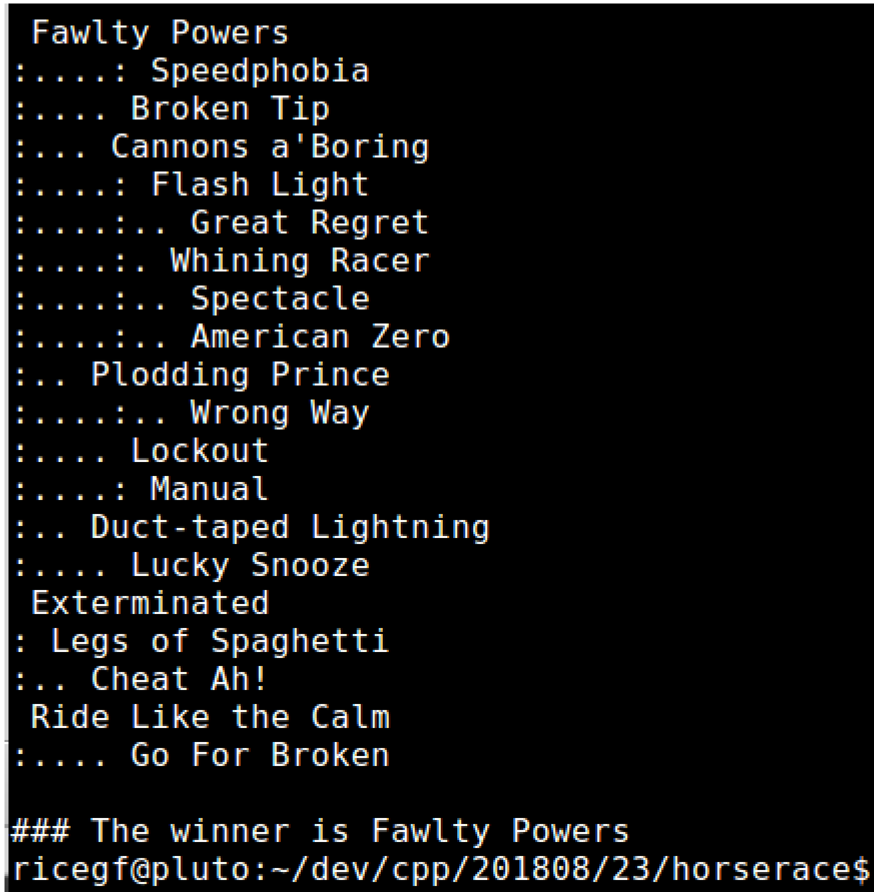


```
ricegf@pluto:~/dev/cpp/201808/23/horserace$ make
g++ -std=c++14 -pthread -c horserace.cpp
g++ -std=c++14 -pthread -c horse.cpp
g++ -o horserace -std=c++14 -pthread horserace.o horse.o
ricegf@pluto:~/dev/cpp/201808/23/horserace$ ./horserace
```

(It's a lot easier to follow live!)



```
:....:....:....:....:....:.... Fawlty Powers
:....:....:....:....:....:.... Speedphobia
:....:....:....:....:....:.... Broken Tip
:....:....:....:....:....:.... Cannons a'Boring
:....:....:....:....:....:.... Flash Light
:....:....:....:....:....:.... Great Regret
:....:....:....:....:....:.... Whining Racer
:....:....:....:....:....:.... Spectacle
:....:....:....:....:....:.... American Zero
:....:....:....:....:....:.... Plodding Prince
:....:....:....:....:....:.... Wrong Way
:....:....:....:....:....:.... Lockout
:....:....:....:....:....:.... Manual
:....:....:....:....:....:.... Duct-taped Lightning
:....:....:....:....:....:.... Lucky Snooze
:....:....:....:....:....:.... Exterminated
:....:....:....:....:....:.... Legs of Spaghetti
:....:....:....:....:....:.... Cheat Ah!
:....:....:....:....:....:.... Ride Like the Calm
:....:....:....:....:....:.... Go For Broken
```



```
 Fawlty Powers
:....: Speedphobia
:.... Broken Tip
:... Cannons a'Boring
:....: Flash Light
:....:.. Great Regret
:....:. Whining Racer
:....:.. Spectacle
:....:.. American Zero
:.. Plodding Prince
:....:.. Wrong Way
:.... Lockout
:....: Manual
:.. Duct-taped Lightning
:.... Lucky Snooze
 Exterminated
: Legs of Spaghetti
:.. Cheat Ah!
 Ride Like the Calm
:.... Go For Broken

### The winner is Fawlty Powers
ricegf@pluto:~/dev/cpp/201808/23/horserace$
```