CSE 1325: Object-Oriented Programming

Lecture 17

Exam #2 Review

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

The past, the present, and the future walk into a bar.
It was tense.

Exams Are Graded and Posted

with, of course, a review and suggested solutions!

- No Errata, but...
  - 2 multiple choice questions thrown out as confusing (see review doc)
  - Some points restored on free response because of non-obvious intent
- Your grade *should* have posted to Canvas Monday night
  - Appeal via email or Canvas Inbox ONLY to preserve permanent record
  - 2-week limit to *file* an appeal (decision may take longer)
- The Exam #2 review document with suggested solutions is on Canvas at Modules > Exam #2
  - Complete buildable code for the Free Response questions is on GitHub

Statistics and Such

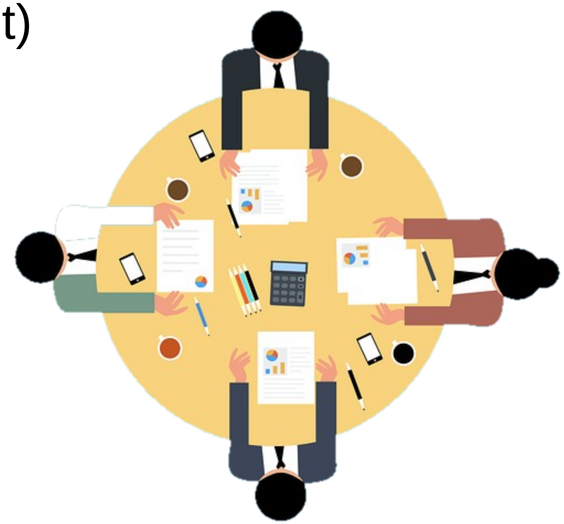
- 85 out of 94 students took the exam
  - Five makeup exam requests are pending
- Scores ranged from 23 to 112 out of 106
  - After question disqualifications and scale
  - Initial median of 62% was *very* disappointing (78% is typical)
  - Questions 3b and 3c had less than 40% of points captured
- The exam timing was nominal to a little long
  - Last semester, 8% finished in the first hour
This semester, 4% finished in the first hour
  - Last semester, 33% finished before the end
This semester, 26% finished before the end

Redo?

- Five students requested a makeup exam
  - This opens the possibility of a “redo” next week
  - I tried this once before, to great initial enthusiasm but relatively little actual participation :(
    - One possible reason is that we’re starting C++ today, which makes studying Java a bit longer more challenging
    - And Friday is the drop deadline
    - But if you’re disappointed, it’s a chance to recover
- If you are *confident* you want a redo anyway, email me ***today*** and I’ll schedule you
  - I’m rooting for you!

Test Markings

- **Vocabulary** – Red “X” marks errors. +2 for each correct definition, 20 points total. Points earned are listed at bottom of page.
- **Multiple Choice** – Red “O” circles corrected answer. +2 for each correct choice, 10 points per page, 30 points total. Points earned are listed at bottom of page. WRITE ANSWER IN THE _____!
- **Free Response** – Corrections *often* marked in detail – this took a LOT of time, but hopefully you’ll READ and CAREFULLY CONSIDER each one! 50 points total.
  - Sum of points per *question* indicated beside each question on the page on which the answer was *asked* (NOT on an additional sheet)
- **Final *Score*** – The sum of all points on every page has been posted **on Canvas *only*** (NOT on the exam)
- **E2_Review.pdf** has been posted on Canvas, and the code used to write the exam is available at **cse1325-prof/Exam2/exam**



Review image by Mohamed_hassan per the Pixabay License

CSE 1325: Object-Oriented Programming

Lecture 17

Introduction to C++

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

The past, the present, and the future walk into a bar.
It was tense.

Today's Topics

- C++ introduction
  - Brief history and context
  - How to compile and run a C++ program
- Console I/O
  - std::cout, std::cerr, and std::cin
  - Formatting without printf
  - Variables, program flow, and such
- Calculator in Java and C++

Francophonie C++ Logo by Oliver H is in the public domain
https://commons.wikimedia.org/wiki/File:Cpp-Francophonie.svg

Thoughts on .gitignore

- The .gitignore file excludes key file extensions from git
  - For Java, .class files are unwanted in git
  - For C++, we also exclude .o, .gch, .exe, .app, and .out
- You may replace your .gitignore with newgitignore to cover both Java AND C++

```
# Java #############

# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java
(J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs
hs_err_pid*
replay_pid*
```

Java

Current .gitignore for Java

ADD to the above to add C++ support

C++

```
# C++ #############

# Prerequisites
*.d

# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll

# Fortran module files
*.mod
*.smod

# Compiled Static libraries
*.lai
*.la
*.a
*.lib

# Executables
*.exe
*.out
*.app
```

C++ : A History

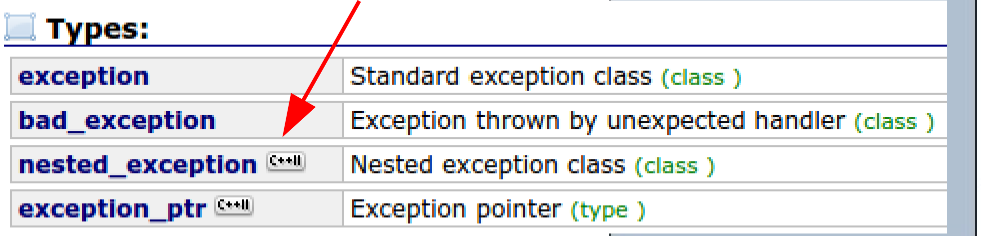
Year	Version	Summary
1982	1	Classes, inheritance, references, constants, // comments, new/delete, operator overloading
1989	2	Multiple inheritance, abstract classes, pointers to members, static and protected members, I/O manipulators
1998	98	Templates, exceptions, namespaces, formal casts, bool, STL (containers, algorithms, iterators, functors)
2003	03	Value initialization
2011	11	Enum class, threads, generic programming, uniform initialization, auto, for-each, lambda, constructor delegation, override keyword, smart pointers, raw and explicit strings, regex, nullptr, user-defined literals (units)
2014	14	Improved auto, binary literals, digit separator (_)
2017	17	Nested namespaces, typename, structured bindings (auto[a,b]=...)
2020	20	Modules, coroutines, concepts, spaceship operator (<=>), ranges (replacing iterator pairs), for-each with initializer, designated initializers

https://en.cppreference.com/w/cpp/language/history

Image of Bjarne Strustrop ©2015 Standard C++ Academic Fair Use Asserted

Where's the Standard C++ Docs?

- C++ has none
- But we'll use cppreference.com & cplusplus.com as our online documentation
  - Watch for version identifiers (up to C++20)

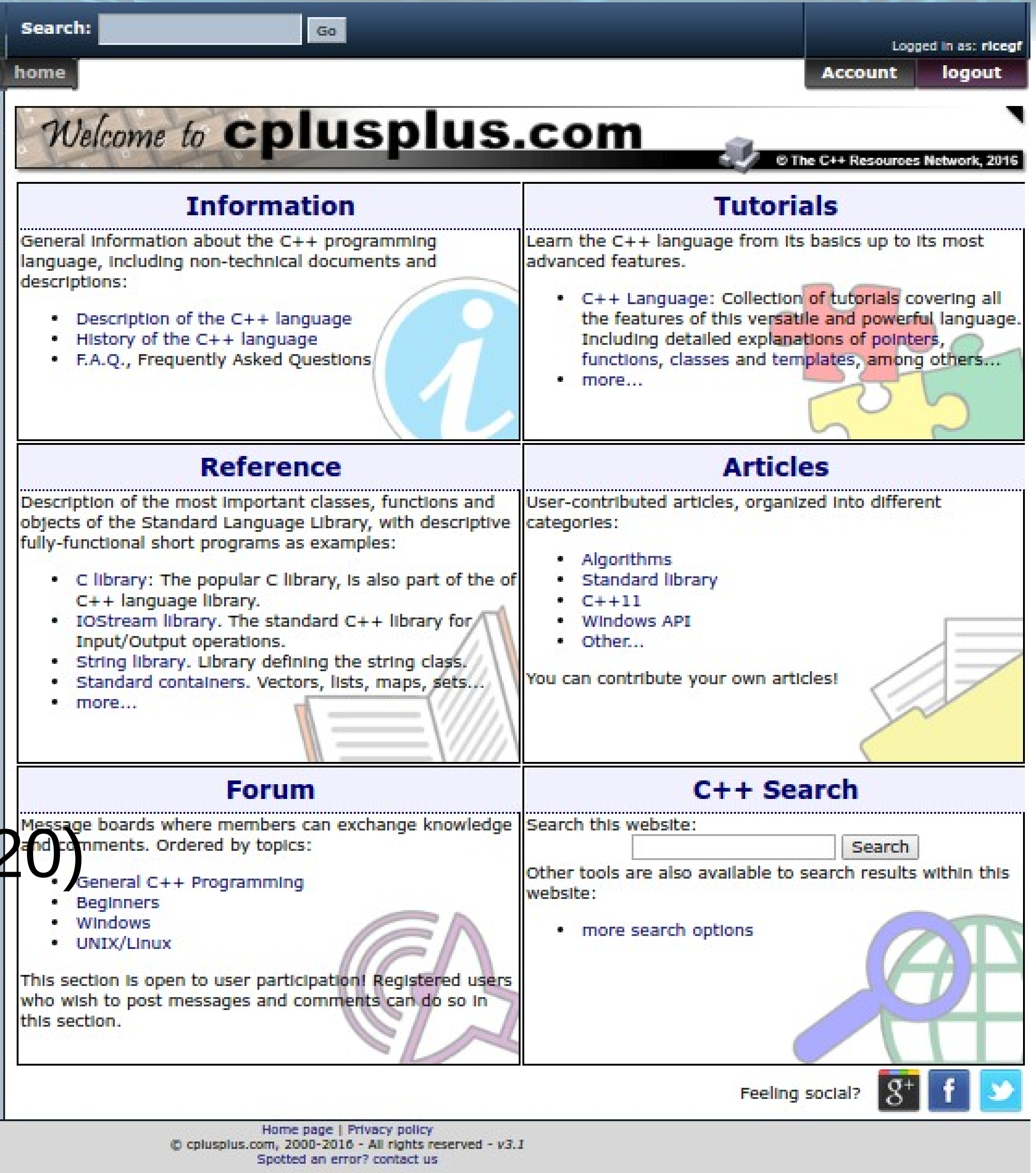


Types:

exception	Standard exception class (class)
bad_exception	Exception thrown by unexpected handler (class)
nested_exception C++11	Nested exception class (class)
exception_ptr C++11	Exception pointer (type)

C++

Information
Tutorials
Reference
Articles
Forum



Information

General information about the C++ programming language, including non-technical documents and descriptions:

- Description of the C++ language
- History of the C++ language
- F.A.Q., Frequently Asked Questions

Tutorials

Learn the C++ language from its basics up to its most advanced features.

- C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others...
- more...

Reference

Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:

- C library: The popular C library, is also part of the of C++ language library.
- IOStream library. The standard C++ library for Input/Output operations.
- String library. Library defining the string class.
- Standard containers. Vectors, lists, maps, sets...
- more...

Articles

User-contributed articles, organized into different categories:

- Algorithms
- Standard library
- C++11
- Windows API
- Other...

You can contribute your own articles!

Forum

Message boards where members can exchange knowledge and comments. Ordered by topics:

- General C++ Programming
- Beginners
- Windows
- UNIX/Linux

This section is open to user participation! Registered users who wish to post messages and comments can do so in this section.

C++ Search

Search this website:

Search

Other tools are also available to search results within this website:

- more search options

Feeling social?

Home page | Privacy policy
© cplusplus.com, 2000-2016 - All rights reserved - *v3.1*
Spotted an error? contact us

Writing the Canonical 1st Program

Python:

Structured
Object-Oriented

```
print("Hello, World")
```

C:

Structured

```
#include <stdio.h>
int main() {
   printf("Hello World");
}
```

Java:

Object-Oriented

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

C++:

Structured
Object-Oriented

```
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
}
```

Writing the Canonical 1st Program

The double colon (`::`) is the *scope resolution* (or *membership*) *operator* in C++.
It is analogous to the period (.) in Java. Java uses a period in place of many C++ operators.

It specifies here that we are using the **`cout`** object from the **`std`** namespace, rather than a **`cout`** object that we instance ourselves in our **global** namespace. If we instance our own **`cout`**, we could specify that explicitly as **`::cout`** instead of **`std::cout`** (Java, by contrast, cannot explicitly access members of the default *package*).

`std::cout` in C++ is conceptually somewhat similar to **`System.out`** in Java. *Global object* **`cout`** belongs to *namespace* **`std`** in C++, while *static field* **`out`** belongs to *class* **`System`** in *package* **`java.lang`** (automatically imported) in Java.

If you add **`using namespace std;`** as the first line, you can omit the membership operator. But since professional C++ developers typically avoid **`using`**, we will as well. Never too early to behave appropriately, as my grandmother always said.

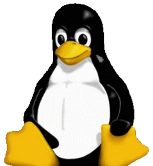
C++:

Structured
Object-Oriented

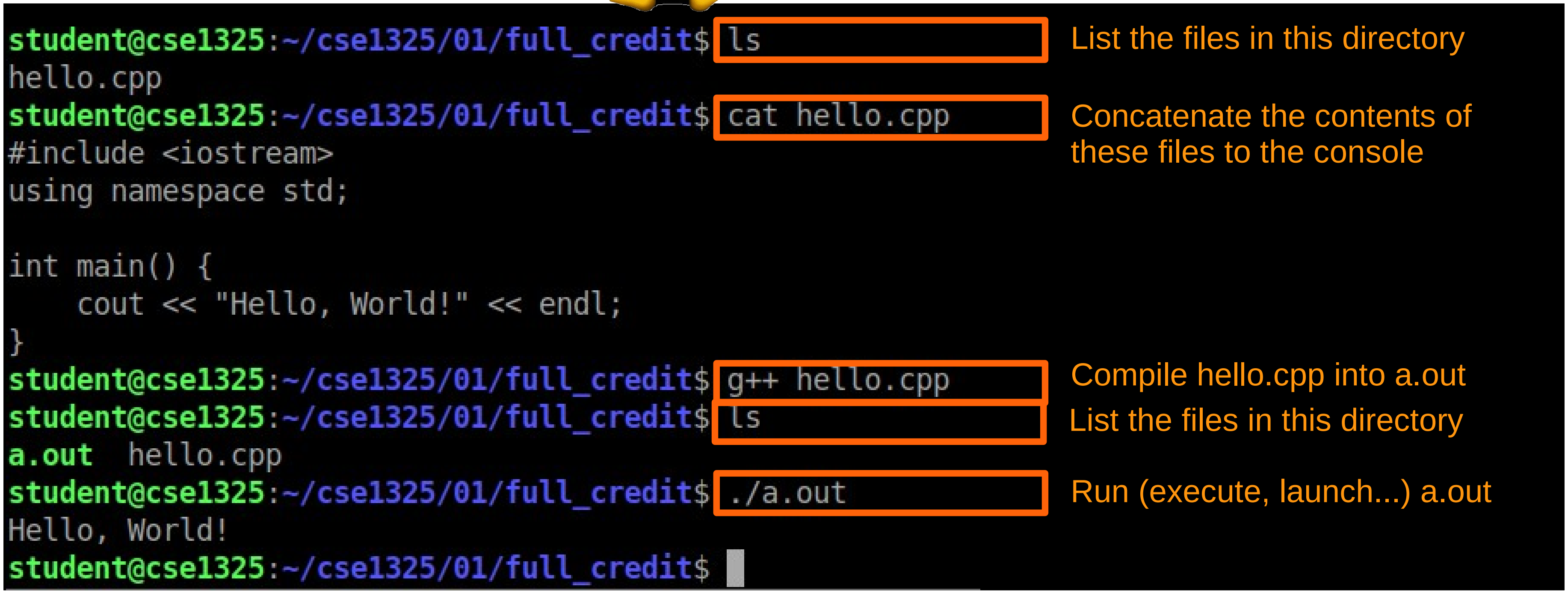
```
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
}
```

Don't worry about this yet

Running hello.cpp Manually



Via Bash



```
student@cse1325:~/cse1325/01/full_credit$ ls
hello.cpp
student@cse1325:~/cse1325/01/full_credit$ cat hello.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
}
student@cse1325:~/cse1325/01/full_credit$ g++ hello.cpp
student@cse1325:~/cse1325/01/full_credit$ ls
a.out  hello.cpp
student@cse1325:~/cse1325/01/full_credit$ ./a.out
Hello, World!
student@cse1325:~/cse1325/01/full_credit$ 
```

List the files in this directory

Concatenate the contents of these files to the console

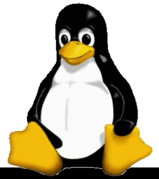
Compile hello.cpp into a.out

List the files in this directory

Run (execute, launch...) a.out

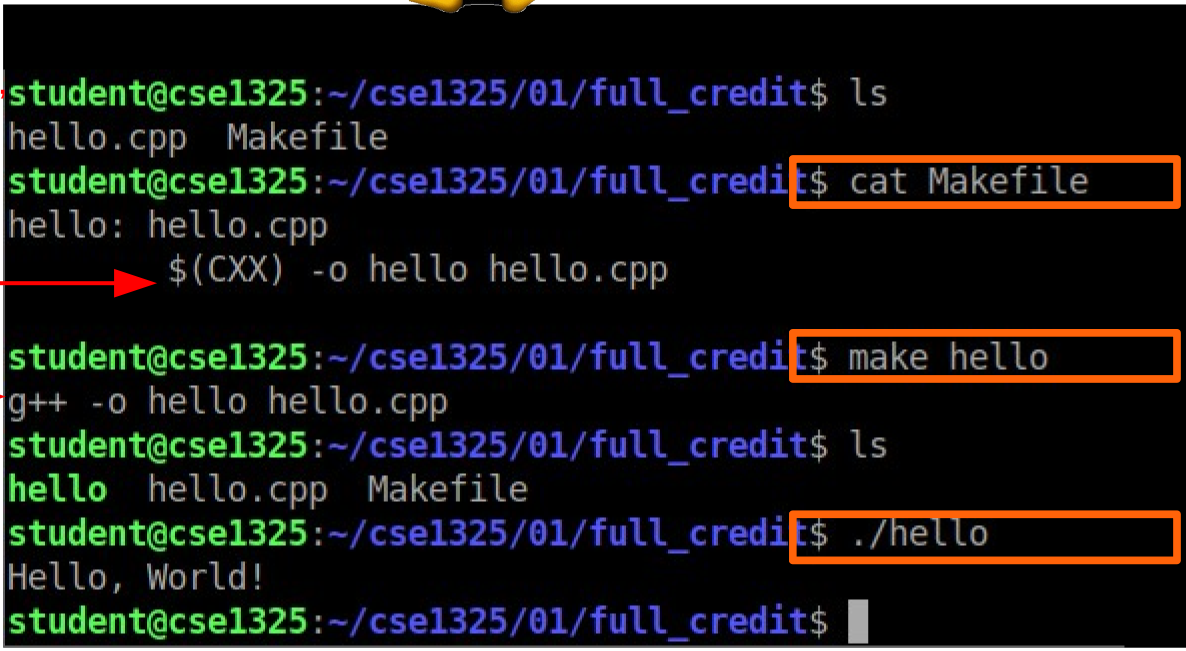
Tux image is ©1996 Larry Ewing. “Permission to use and/or modify this image is granted provided you acknowledge me lewing@isc.tamu.edu and The GIMP if someone asks.”

Running hello.cpp via the Makefile System



Via Bash

“$(CXX)” means “use the default compiler command”; in our case, g++.



```
student@cse1325:~/cse1325/01/full_credit$ ls
hello.cpp  Makefile
student@cse1325:~/cse1325/01/full_credit$ cat Makefile
hello: hello.cpp
	$(CXX) -o hello hello.cpp

student@cse1325:~/cse1325/01/full_credit$ make hello
g++ -o hello hello.cpp
student@cse1325:~/cse1325/01/full_credit$ ls
hello  hello.cpp  Makefile
student@cse1325:~/cse1325/01/full_credit$ ./hello
Hello, World!
student@cse1325:~/cse1325/01/full_credit$ 
```

The Makefile describes how to make the executable. “-o hello” tells g++ to compile to an executable named ”hello” instead of “a.out”.

This make command follows the above rules to make the executable named “hello”.

Now run hello.

Note that by convention, the Makefile filename starts with a capital “M” and has NO EXTENSION (e.g., no “.txt” at the end).

We’ll discuss the Makefile in the next lecture

A Closer Look at Hello World

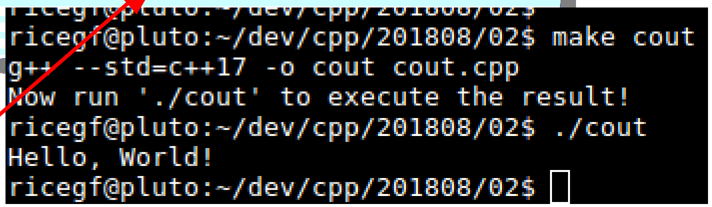
`#include <file>` searches the library.
`#include "file"` *first* searches the local directory, *then* the library.

Main is a function, not a method, and returns an int. Parameters are *optional*.

Stream operators like << can be chained on the same line of code, like 1+2+3

Once found, the file's text is *literally* inserted in place of the #include, exactly as if it were typed there.

```
#include <iostream>

int main() {
  std::cout << "Hello " << "World!" << '\n';
  return 0;
}
```



```
ricegf@pluto:~/dev/cpp/201808/02$
ricegf@pluto:~/dev/cpp/201808/02$ make cout
g++ --std=c++17 -o cout cout.cpp
Now run './cout' to execute the result!
ricegf@pluto:~/dev/cpp/201808/02$ ./cout
Hello, World!
ricegf@pluto:~/dev/cpp/201808/02$ 
```

This is unrelated to Twitter.

We may return 0 to indicate success. Any other int is an error code indicating failure. If no return value is returned, 0 is used.

Single quotes surround single characters (in this case, a newline).
Double quotes surround multiple characters.

The cout method (pronounced "see-out") sends characters* streamed to it to STDOUT. This is usually the console.

A semicolon (';') terminates statements, *but not directives* (which start with #), in contrast to Java's import.

* ASCII characters. And *sometimes* Unicode characters.
But C++ and Unicode go together like apples and airplanes.

Output formats

- Integer values
  - **1234** (decimal)
  - **2322** (octal)
  - **4d2** (hexadecimal)
- Double values
  - **1234.57** (general)
  - **1.2345678e+03** (scientific)
  - **1234.567890** (fixed)
- Precision (for double values)
  - **1234.57** (precision 6)
  - **1234.6** (precision 5)
- Fields
  - **|12|** (default for **|** followed by **12** followed by **|**)
  - **| 12|** (**12** in a field of 4 characters)

We can achieve these formats (and more) using "I/O Manipulators"

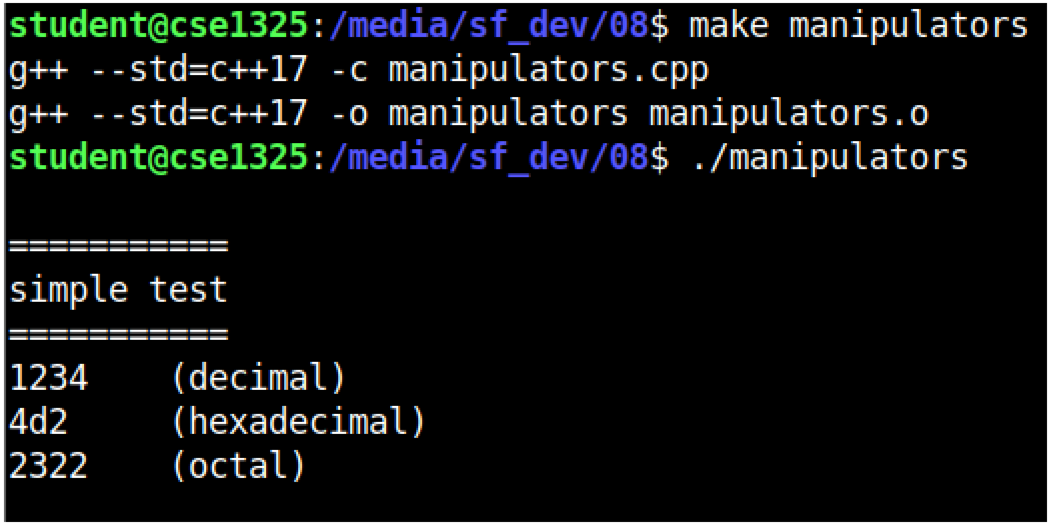
While C++ obviously has the printf *function* (almost identical to Java's System.out.printf *static method)*, streams are preferred. I/O manipulators format streams.

Numerical Base Output

dec hex oct

- You can change “base”
  - Base 10 == **dec**imal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 16 == **hex**adecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f
  - Base 8 == **oct**al; digits: 0 1 2 3 4 5 6 7

```
title("simple test");
std::cout << std::dec << 1234 << "\t(decimal)\n"
          << std::hex << 1234 << "\t(hexadecimal)\n"
          << std::oct << 1234 << "\t(octal)\n";
// The '\t' character is a “tab”

std::cout << std::endl;
```

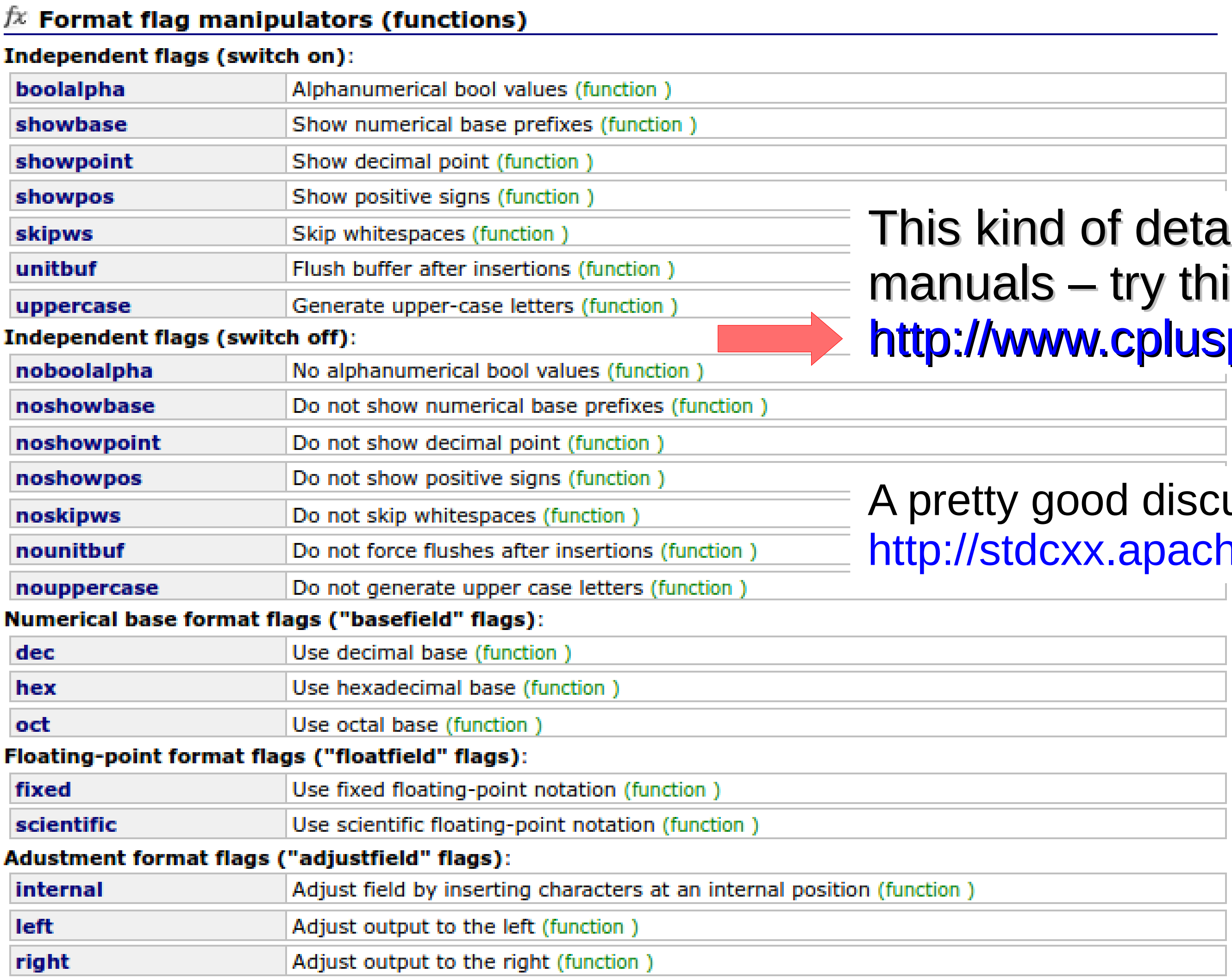


```
student@cse1325:/media/sf_dev/08$ make manipulators
g++ --std=c++17 -c manipulators.cpp
g++ --std=c++17 -o manipulators manipulators.o
student@cse1325:/media/sf_dev/08$ ./manipulators


===========
simple test
===========
1234    (decimal)
4d2     (hexadecimal)
2322    (octal)
```

Other Manipulators

- Integer base
  - std::showbase prepends 0x (for hex, for example) to output integers
- Floating point
  - std::setprecision(5) shows 5 digits past decimal
  - std::defaultfloat, std::hexfloat, std::fixed, and std::scientific set display format
- Field width
  - std::setw(10) sets the width of the next value output to 10 characters (or more if necessary not to lose information)
  - Unlike other manipulators, std::setw is **NOT "sticky"** – you must set it separately for EVERY field

Other Manipulators



fx Format flag manipulators (functions)

Independent flags (switch on):

boolalpha	Alphanumerical bool values (function)
showbase	Show numerical base prefixes (function)
showpoint	Show decimal point (function)
showpos	Show positive signs (function)
skipws	Skip whitespaces (function)
unitbuf	Flush buffer after insertions (function)
uppercase	Generate upper-case letters (function)

Independent flags (switch off):

noboolalpha	No alphanumerical bool values (function)
noshowbase	Do not show numerical base prefixes (function)
noshowpoint	Do not show decimal point (function)
noshowpos	Do not show positive signs (function)
noskipws	Do not skip whitespaces (function)
nounitbuf	Do not force flushes after insertions (function)
nouppercase	Do not generate upper case letters (function)

Numerical base format flags ("basefield" flags):

dec	Use decimal base (function)
hex	Use hexadecimal base (function)
oct	Use octal base (function)

Floating-point format flags ("floatfield" flags):

fixed	Use fixed floating-point notation (function)
scientific	Use scientific floating-point notation (function)

Adustment format flags ("adjustfield" flags):

internal	Adjust field by inserting characters at an internal position (function)
left	Adjust output to the left (function)
right	Adjust output to the right (function)

This kind of detail is why you need (online) manuals – try this one: http://www.cplusplus.com/reference/ios/

A pretty good discussion for advanced students is http://stdcxx.apache.org/doc/stdlibug/28-3.html

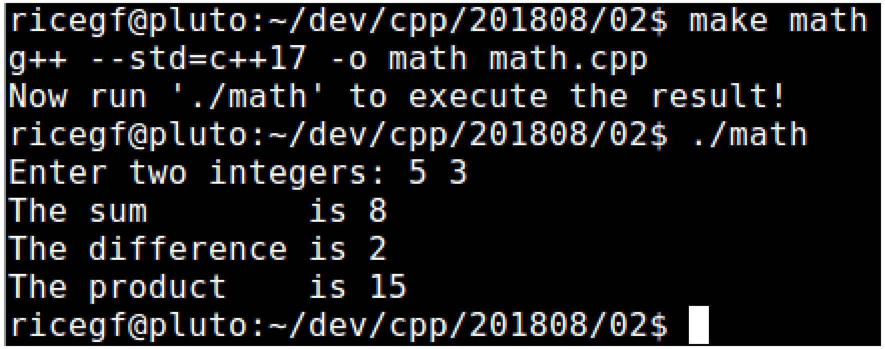
A Closer Look at Input in C++

The cin method (pronounced “see-in”) accepts characters streamed to it from STDIN – usually the keyboard – and converts them into the indicated variables.

Each word (separated by “whitespace” such as spaces, tabs, or newlines) is handled *separately* in the stream.

```
#include <iostream>

int main() {
  std::cout << "Enter two integers: ";
  int num1, num2;
  std::cin >> num1 >> num2;
  std::cout << "The sum        is "<< num1 + num2 << std::endl;
  std::cout << "The difference is "<< num1 - num2 << std::endl;
  std::cout << "The product    is "<< num1 * num2 << std::endl;
  return 0;
}
```

std::ndl (pronounced “end-el”) is a macro that means “start a new line” or “\n”.



```
ricegf@pluto:~/dev/cpp/201808/02$ make math
g++ --std=c++17 -o math math.cpp
Now run './math' to execute the result!
ricegf@pluto:~/dev/cpp/201808/02$ ./math
Enter two integers: 5 3
The sum        is 8
The difference is 2
The product    is 15
ricegf@pluto:~/dev/cpp/201808/02$
```

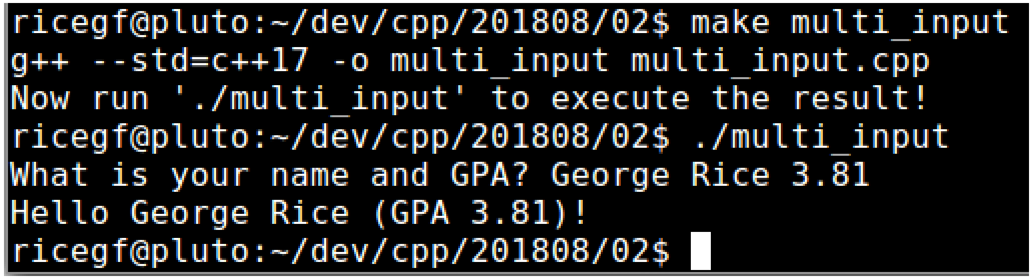
“std::endl” is preferred to “\n” because it also flushes the output buffer.

http://www.cplusplus.com/doc/tutorial/basic_io/

Multi-Text and Integer Input in C++

NEVER char* or char[ ] in CSE1325 unless forced by a library.
ALWAYS use std::string (a mutable equivalent to Java’s String).

```
std::cout << "What is your name and GPA? ";
std::string first;
std::string last;
double gpa;
std::cin >> first >> last >> gpa;
std::string name = first + ' ' + last;
std::cout << "Hello " << name
          << " (GPA " << gpa << ")!\n";
```



```
ricegf@pluto:~/dev/cpp/201808/02$ make multi_input
g++ --std=c++17 -o multi_input multi_input.cpp
Now run './multi_input' to execute the result!
ricegf@pluto:~/dev/cpp/201808/02$ ./multi_input
What is your name and GPA? George Rice 3.81
Hello George Rice (GPA 3.81)!
ricegf@pluto:~/dev/cpp/201808/02$
```

Why << and >>?

- Since C++ allows operators to be overloaded (more on this later), your own class types can be *directly* streamed in and out!

  -
    ```
    Foo f = new Foo{42}; // Note the curly braces
    std::cout << f << std::endl; // Legal AND common!
    ```

  -
    ```
    std::cerr << “Failure to launch!” << std::endl
    ```

  -
    ```
    std::cin >> f; // Also legal but somewhat less common
    ```
- We’ll show you how to define operators like << when we teach you how to write C++ classes next week
- The rough equivalent in Java for << is toString (because we can stream to a std::string!)

Comparing C++ to Java

C++, Java, and Python Types

Type	C++	Java	Python
1-byte integer	**char**	byte	**int** All integers are of arbitrary size
2-byte integer	short, **int** (often 4 bytes)	short	
4-byte integer	long (often int)	**int**, Integer	
8-byte integer	long long	long	
4-byte double	float	float	
8-byte double	**double**	**double**	**float**
8-byte complex			complex
1-byte character	char		bytes
2-byte character	w_char	char	
Boolean	**bool**	**boolean**	**bool**
String	**std::string** char*	**String**	**str**

Comparing C++ to Java

Names, Operators, and Loops

- C++ uses snake case – my_name
  Java uses camel case – myName
  - Autotyping uses **`auto`** instead of **`var`**:
    ```
    for(auto f : foods)
        std::cout << f << std::endl;
    ```
- Same operators (including + for std::string), conditionals (including ternary), and loops (including for-each)
  - No switch *expressions*, though – only statements for which **`break`** is required!

Comparing C++ to Java
Strings and Arguments

- C uses char* - a pointer to an array of char
  - In Java this would be char[ ], but we rarely use it
- C++ uses std::string – similar to Java’s String
  - Except std::string is *mutable* – it can be changed similar to Java’s StringBuilder class
- C++ command line arguments come in an *optional* int argc and array argv of char* - either char** or char*[ ]
  - In Java, with no arguments, you get ZERO arguments – **`args.length==0`**
  - In C++ as in C, with no program arguments, you get **`argc==1`** or ONE argument – the name of the executable

Comparing C++ to Java

Default Parameters

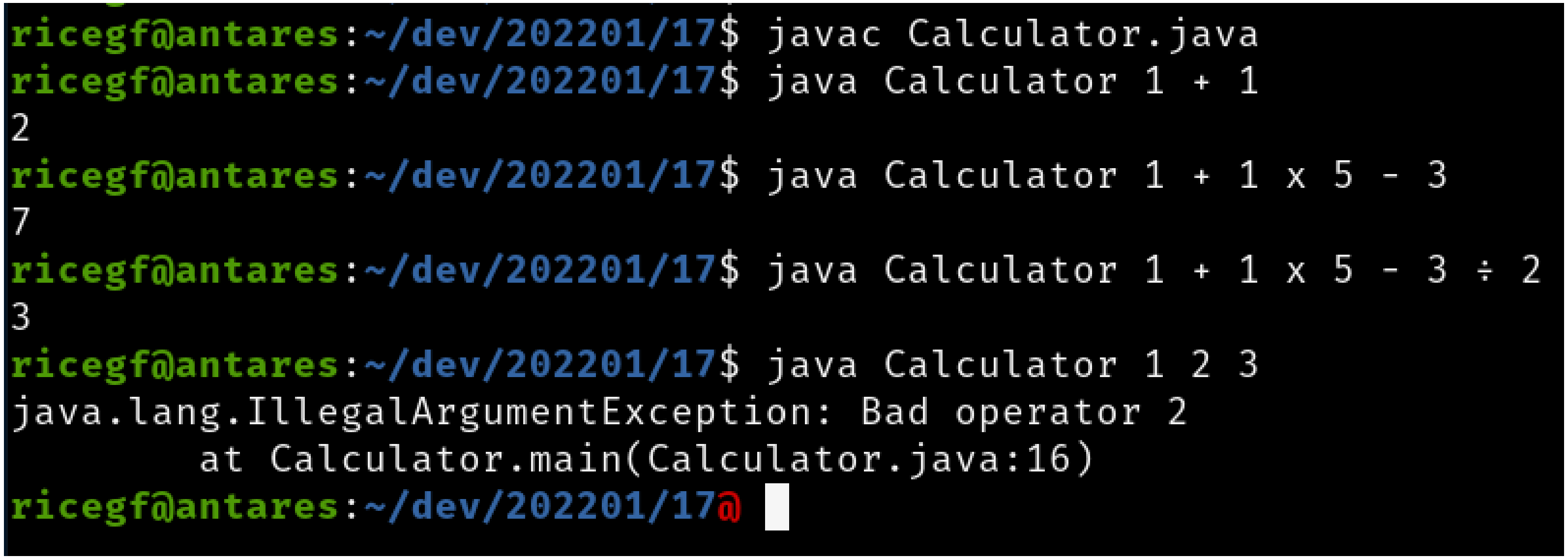
- In Java, if you want 0 or 1 parameters, write

  -
    ```
    public Foo(int bar) {this.bar = bar;}
    public Foo() {this(0);}
    ```
- In C++, we can assign a default value

  -
    ```
    public: Foo(int bar=0) {this.bar = bar;}
    ```
  - If bar is specified, it is used; if not, 0 is used
  - Default(s) must be the LAST parameter(s):
    ```
    Zed(int a=1, int b); // Illegal!
    ```

Comparing C++ to Java
Example: Int Calculator

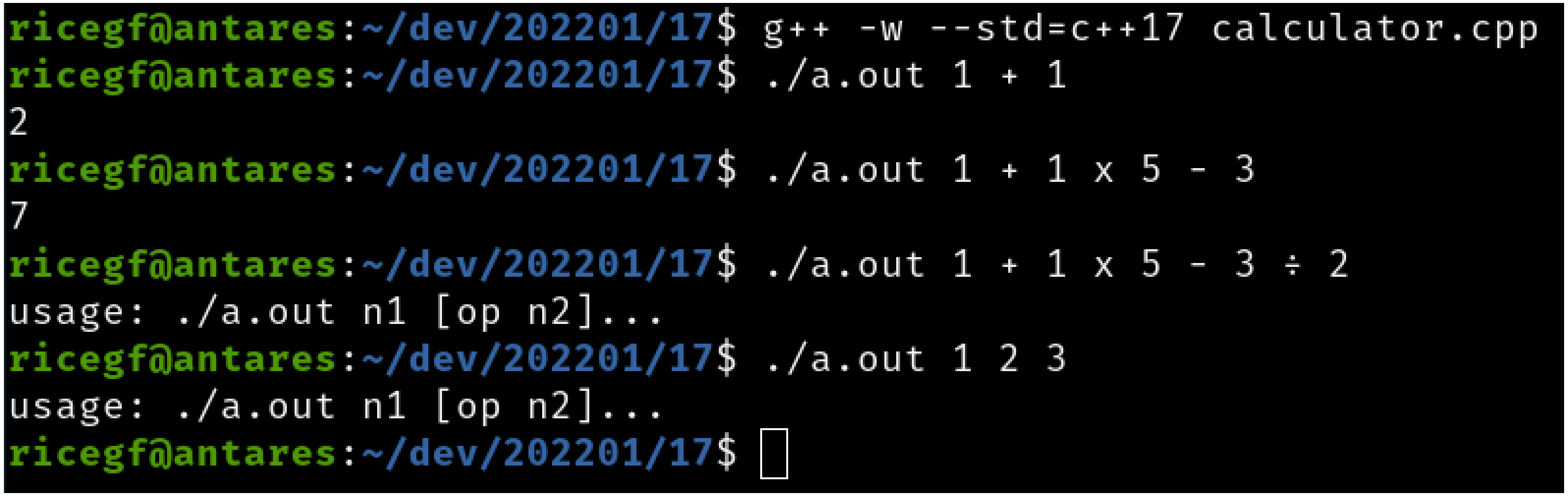
- Let’s compare a simple int calculator in both



```
ricegf@antares:~/dev/202201/17$ javac Calculator.java
ricegf@antares:~/dev/202201/17$ java Calculator 1 + 1
2
ricegf@antares:~/dev/202201/17$ java Calculator 1 + 1 x 5 - 3
7
ricegf@antares:~/dev/202201/17$ java Calculator 1 + 1 x 5 - 3 ÷ 2
3
ricegf@antares:~/dev/202201/17$ java Calculator 1 2 3
java.lang.IllegalArgumentException: Bad operator 2
        at Calculator.main(Calculator.java:16)
ricegf@antares:~/dev/202201/17@ 
```

Java ←

C++ →



```
ricegf@antares:~/dev/202201/17$ g++ -w --std=c++17 calculator.cpp
ricegf@antares:~/dev/202201/17$ ./a.out 1 + 1
2
ricegf@antares:~/dev/202201/17$ ./a.out 1 + 1 x 5 - 3
7
ricegf@antares:~/dev/202201/17$ ./a.out 1 + 1 x 5 - 3 ÷ 2
usage: ./a.out n1 [op n2]...
ricegf@antares:~/dev/202201/17$ ./a.out 1 2 3
usage: ./a.out n1 [op n2]...
ricegf@antares:~/dev/202201/17$ 
```

Comparing C++ to Java

Example: Int Calculator

- Number of arguments, primitives, exceptions

```java
public class Calculator {
    public static void main(String[] args) {
        try {
            if(args.length % 2 != 1) throw new
                IllegalArgumentException(
                    "usage: java Calculator n1 [op n2]...");
            int accumulator = Integer.parseInt(args[0]);
            int index = 1;
```

Java

- C++ supports structured programming, but Java requires OO: a class with a static method
- For arguments, C++ provides the number and a char*[] array, Java requires a String[]
- C++ sets the first argument (a char* NOT a std::string) to the executable program name
- C++ converts chars*[] to int using atoi function, Java uses Integer.parseInt static method

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
    try {
        if(argc % 2 != 0) throw new std::runtime_error("");
        int accumulator = atoi(argv[1]);
        int index = 2;
```

C++

Comparing C++ to Java

Example: Int Calculator

- Switch and char

```
            while(index+1 < args.length) {
                int operand = Integer.parseInt(args[index+1]);
                switch(args[index]) {
                    case "+" -> accumulator += operand;
                    case "-" -> accumulator -= operand;
                    case "x" -> accumulator *= operand;
                    case "÷" -> accumulator /= operand;
```

Java

- Arguments in argc/argv in C++ vs args in Java, with atoi in C++ vs Integer.parseInt in Java
- Java can switch on a String expression (and much more!) but C++ requires a char or int
- Java supports switch *expressions* (with ->) that avoid the break keyword, C++ doesn't
- Java supports 16-bit "long char" (like ÷) but C++ uses only 8-bit char (compiles but breaks)
  - BOTH require extra work for more complex Unicode encodings, unfortunately

```
        while(index+1 < argc) {
            int operand = atoi(argv[index+1]);
            switch((unsigned char)argv[index][0]) {
                case '+': accumulator += operand; break;
                case '-': accumulator -= operand; break;
                case 'x': accumulator *= operand; break;
                case '÷': accumulator /= operand; break;
```

C++

Comparing C++ to Java
Example: Int Calculator

- I/O, exception handling, main return value

```
            System.out.println(accumulator);
        } catch(Exception e) {
            e.printStackTrace(); // includes usage message in exception
            System.exit(-1);
        }
```

Java

- C++ streams data out to std::cout (STDOUT) and to std::cerr (STDERR) using << while Java uses System.out.println and System.err.println methods
- Java can print a stack trace from an exception object easily, C++ cannot
- Again, C++ provides the executable name as argv[0], Java does not
- C++ optionally returns an int from main (if not, 0 is returned),
  Java has no return type for main, but can return an error code using System.exit
- Java has a base class to all (catchable) exceptions called Exception,
  C++ does not* – but catching "…" will anonymously catch all exceptions

```
        std::cout << accumulator << std::endl;
    } catch(...) {
        std::cerr << "usage: " << argv[0] << " n1 [op n2]..." << std::endl;
        return -1;
    }
```

C++

* Most C++ exceptions are subclasses of std::exception, but you can throw *anything*

Comparing C++ to Java

Example: Int Calculator

- Here’s the full Java version

```java
public class Calculator {
    public static void main(String[] args) {
        try {
            if(args.length % 2 != 1) throw new
                IllegalArgumentException("usage: java Calculator n1 [op n2]...");
            int accumulator = Integer.parseInt(args[0]);
            int index = 1;
            while(index+1 < args.length) {
                int operand = Integer.parseInt(args[index+1]);
                switch(args[index]) {
                    case "+" -> accumulator += operand;
                    case "-" -> accumulator -= operand;
                    case "x" -> accumulator *= operand;
                    case "÷" -> accumulator /= operand;
                    default -> throw new IllegalArgumentException("Bad operator "
                                                + args[index]);
                }
                index += 2;
            }
            System.out.println(accumulator);
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Comparing C++ to Java

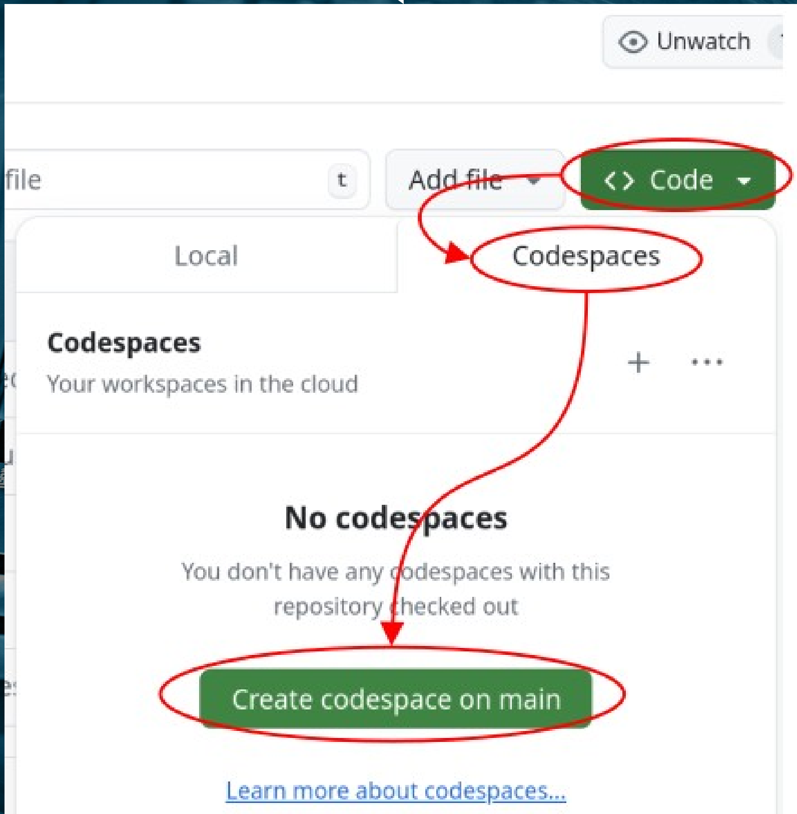
Example: Int Calculator

- Here’s the full C++ version

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
    try {
        if(argc % 2 != 0) throw new std::runtime_error("");
        int accumulator = atoi(argv[1]);
        int index = 2;
        while(index+1 < argc) {
            int operand = atoi(argv[index+1]);
            switch((unsigned char)argv[index][0]) {
                case '+': accumulator += operand; break;
                case '-': accumulator -= operand; break;
                case 'x': accumulator *= operand; break;
                case '÷': accumulator /= operand; break;
                default: throw new
                    std::runtime_error(std::string("Bad operator ")
                                      + argv[index]);
            }
            index += 2;
        }
        std::cout << accumulator << std::endl;
    } catch(...) {
        std::cerr << "usage: " << argv[0] << " n1 [op n2]..." << std::endl;
    }
}
```

[OPTIONAL] Creating a C++ GitHub Codespace

- If you are using a Codespace to do your assignments, launch a new one by selecting Code > Codespaces > Create codespace on main



Codespaces

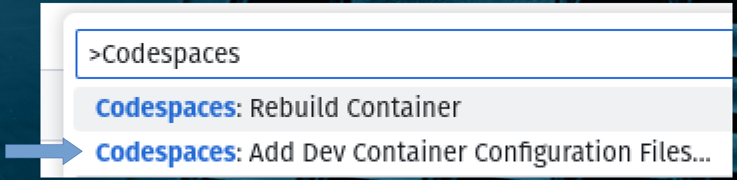
Your workspaces in the cloud

No codespaces

You don't have any codespaces with this repository checked out

Installing the gcc C++ Compiler in your GitHub Codespace (1 of 4)

- You would *think* that installing the C++ Extension would get you the latest version of gcc – but instead, you would get gcc 9, which is too old for C++ 20 . :(
- To install gcc 12, we must *rebuild the configuration*
- Select Ctrl-Shift-P (Cmd-Shift-P on Mac) and run `Codespaces: Add Dev Container Configuration Files…`
  - Select ‘Create new configuration’,
    then ‘C++’,
    then ‘Ubuntu 24.04’,
    then ‘none’,
    then ‘OK’.



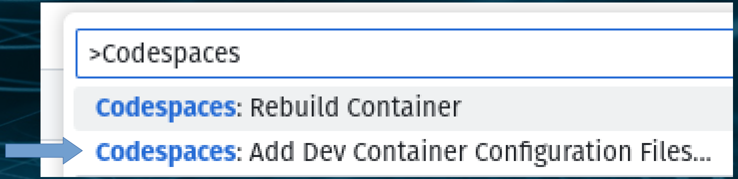
>Codespaces

Codespaces: Rebuild Container

Codespaces: Add Dev Container Configuration Files...

Installing the gcc C++ Compiler in your GitHub Codespace (2 of 4)

- Use the terminal at the bottom to edit the Dockerfile by typing
  `code .devcontainer/Dockerfile`


>Codespaces
Codespaces: Rebuild Container
Codespaces: Add Dev Container Configuration Files...

- Append EXACTLY this text at the bottom:

```
RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
    && apt-get -y install --no-install-recommends software-properties-common \
    && add-apt-repository ppa:ubuntu-toolchain-r/test \
    && apt-get update \
    && apt-get -y install --no-install-recommends gcc-12 g++-12 \
    && update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-12 100 \
    && update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-12 100
```

- Save the Dockerfile with Control-s.
- Verify that the file is correct using this terminal command to list it:
  `cat .devcontainer/Dockerfile`

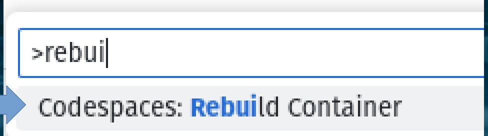
Installing the gcc C++ Compiler in your GitHub Codespace (3 of 4)

- Here’s my complete Dockerfile, but it’s **OK** for yours to differ!
  Only the ***last*** RUN command needs to be exactly what is shown below.

```
FROM mcr.microsoft.com/devcontainers/cpp:1-ubuntu-24.04

ARG REINSTALL_CMAKE_VERSION_FROM_SOURCE="none"

# Optionally install the cmake for vcpk
COPY ./reinstall-cmake.sh /tmp/

RUN if [ "${REINSTALL_CMAKE_VERSION_FROM_SOURCE}" != "none" ]; then \
        chmod +x /tmp/reinstall-cmake.sh && /tmp/reinstall-cmake.sh $
{REINSTALL_CMAKE_VERSION_FROM_SOURCE}; \
    fi \
    && rm -f /tmp/reinstall-cmake.sh

# [Optional] Uncomment this section to install additional vcpkg ports.
# RUN su vscode -c "${VCPKG_ROOT}/vcpkg install <your-port-name-here>"


# [Optional] Uncomment this section to install additional packages.
# RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
#     && apt-get -y install --no-install-recommends <your-package-list-here>

RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
    && apt-get -y install --no-install-recommends software-properties-common \
    && add-apt-repository ppa:ubuntu-toolchain-r/test \
    && apt-get update \
    && apt-get -y install --no-install-recommends gcc-12 g++-12 \
    && update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-12 100 \
    && update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-12 100
```

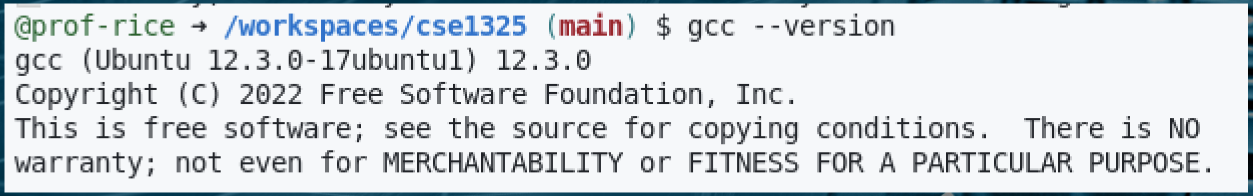
Installing the gcc C++ Compiler in your GitHub Codespace (4 of 4)

- Close all files using the ‘x’ on each editor tab near the top of the screen.
- Select Ctrl-Shift-P (Cmd-Shift-P on Mac), type "rebuild", then select **`Codespaces: Rebuild Container`**
  - Select “Full Rebuild”. This will take awhile.



```
>rebui
Codespaces: Rebuild Container
```

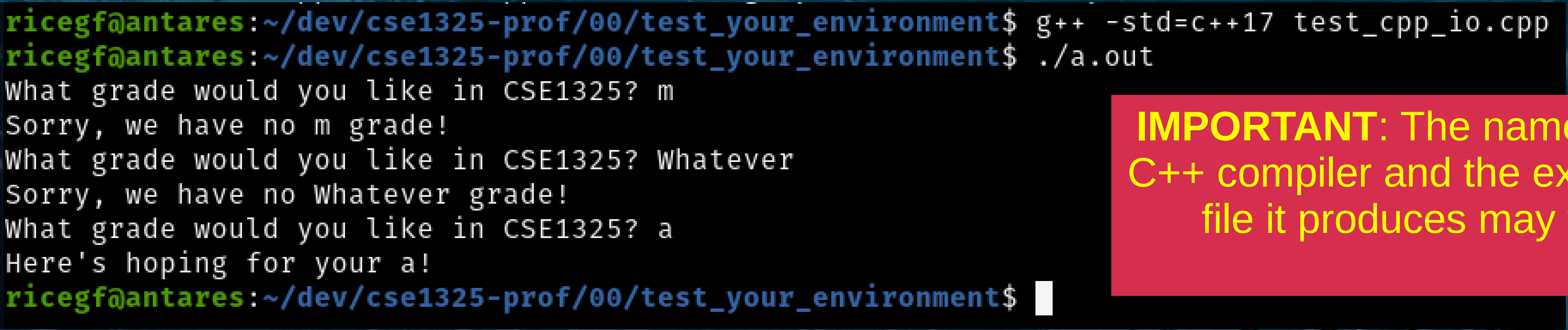
- Once the CodeSpace has restarted, run **`gcc --version`** on the command line – it should be version 12



```
@prof-rice → /workspaces/cse1325 (main) $ gcc --version
gcc (Ubuntu 12.3.0-17ubuntu1) 12.3.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

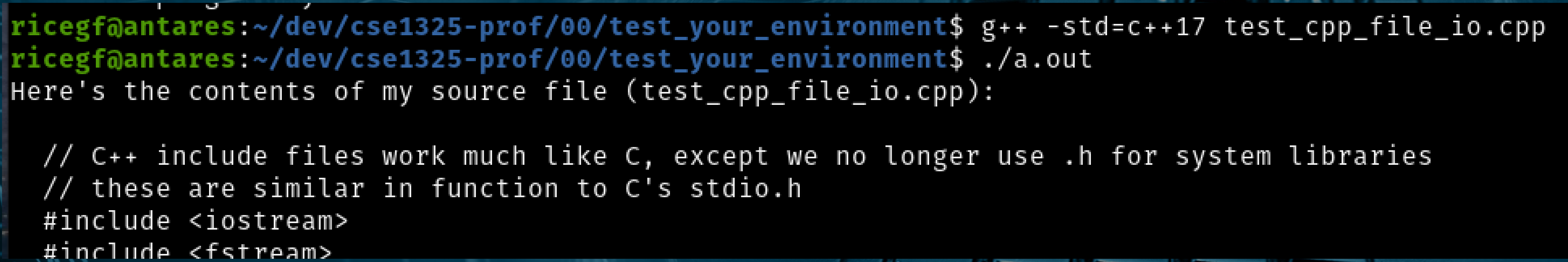
Testing your C++ Environment

- test_cpp_io.cpp
  - Console input / output
- test_cpp_file_io.cpp
  - Read text file



```
ricegf@antares:~/dev/cse1325-prof/00/test_your_environment$ g++ -std=c++17 test_cpp_io.cpp
ricegf@antares:~/dev/cse1325-prof/00/test_your_environment$ ./a.out
What grade would you like in CSE1325? m
Sorry, we have no m grade!
What grade would you like in CSE1325? Whatever
Sorry, we have no Whatever grade!
What grade would you like in CSE1325? a
Here's hoping for your a!
ricegf@antares:~/dev/cse1325-prof/00/test_your_environment$ 
```

IMPORTANT: The name of your C++ compiler and the executable file it produces may vary!



```
ricegf@antares:~/dev/cse1325-prof/00/test_your_environment$ g++ -std=c++17 test_cpp_file_io.cpp
ricegf@antares:~/dev/cse1325-prof/00/test_your_environment$ ./a.out
Here's the contents of my source file (test_cpp_file_io.cpp):

  // C++ include files work much like C, except we no longer use .h for system libraries
  // these are similar in function to C's stdio.h
  #include <iostream>
  #include <fstream>
```

Note: Source code from the lectures is always provided to you at https://github.com/prof-rice/cse1325-prof.git!

Summary

- C++ and Java are both K&R-style languages
  - Kernighan and Ritchie, inventors of C
  - Thus their syntax is quite similar in many respects
- Expression syntax is very close
  - Use std::string rather than java.lang.String
  - Use auto instead of var for automatic types
  - No switch expressions or enum members
- I/O is very different but also very consistent
  - std::cout << “Hi!” instead of System.out.print(“Hi!”)
- Building and running programs is very different
  - **`g++ --version=20 main.cpp`** instead of **`javac Main.java`**