Lecture 03 – Programming Languages ~~and Scripting~~

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F20)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

- Survey Results
- C and C++ with comparison to Fortran
- Resuming hands on from last lecture
  - Review Fortran code, write C and C++ examples
- ~~Introduction to Bash Scripting (time permitting)~~

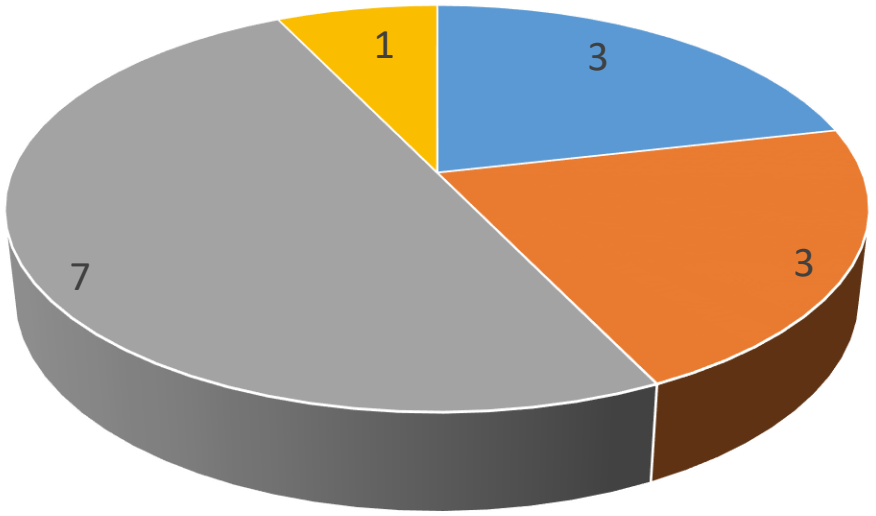
Learning Objectives: By the end of Today's Lecture you should be able to

- *(Knowledge)* clearly understand what we'll be teaching the rest of the semester and the overall previous experience of the class
- *(Skill)* Declare variables that are matrices, and assign or edit their values in Fortran, **C, and C++**
- *(Knowledge)* describe relationships between some language aspects of C/C++ with respect to Fortran
- ~~*(Knowledge)* avoid simple syntax mistakes in Bash~~

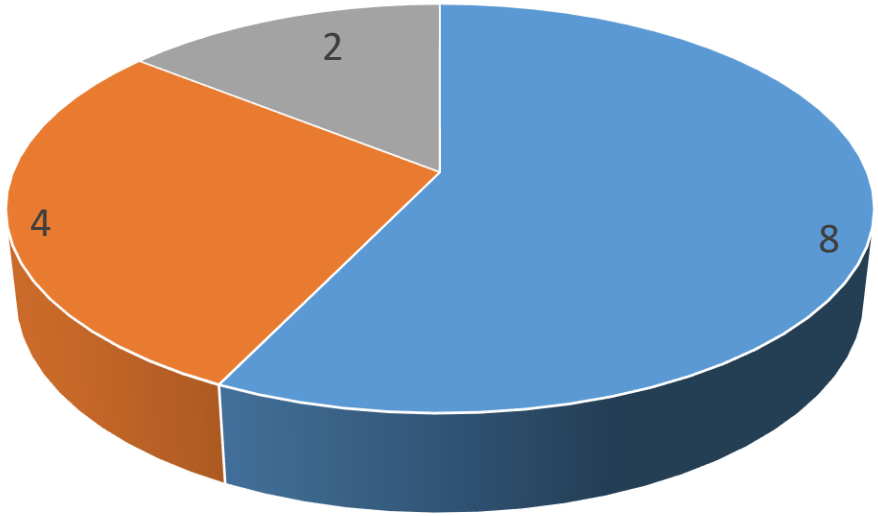
Survey Results

Participation

Format

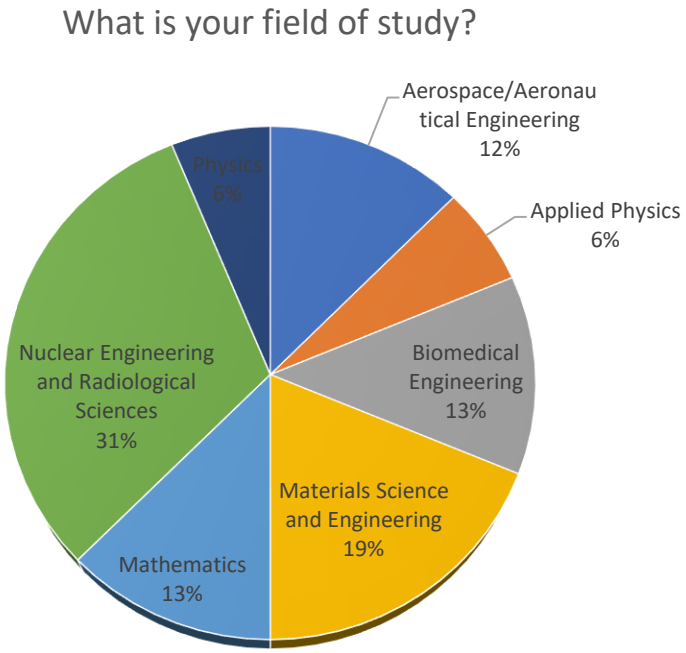


Time zone

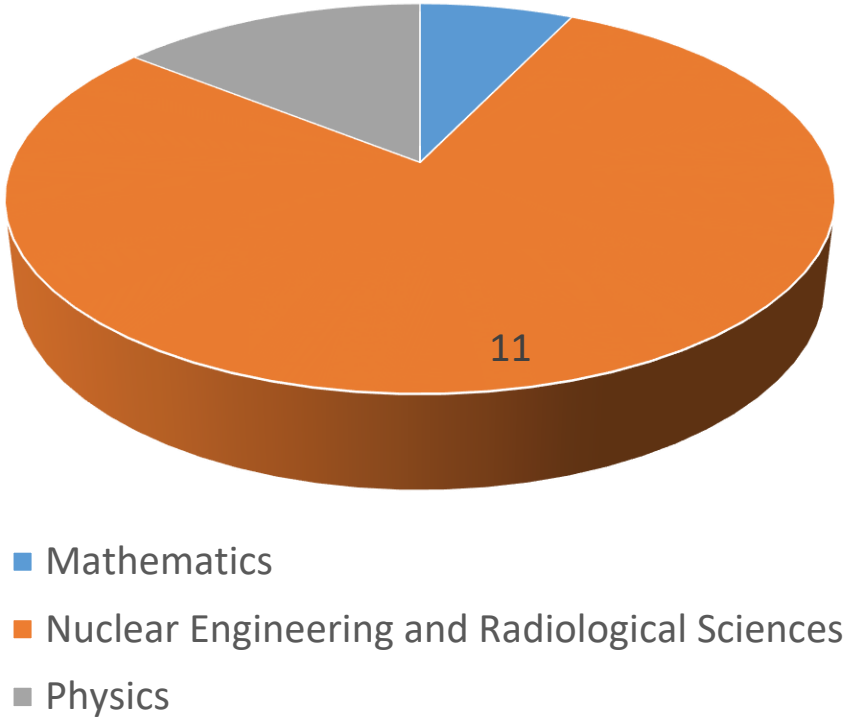


Survey Says…

2019

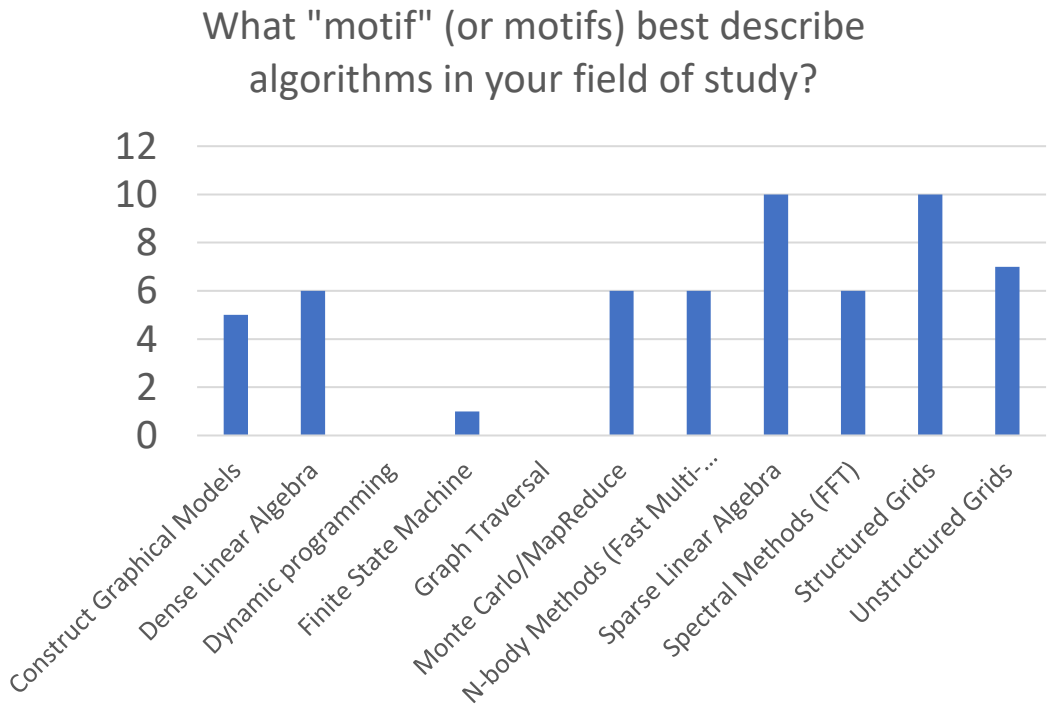


2020

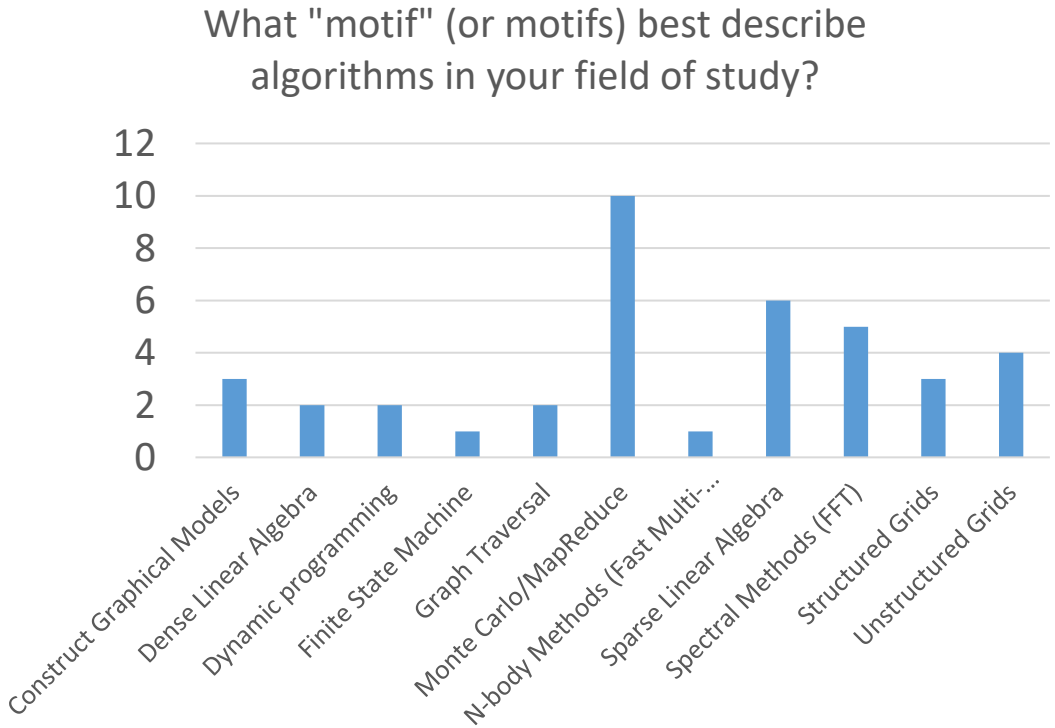


Survey Says…

2019

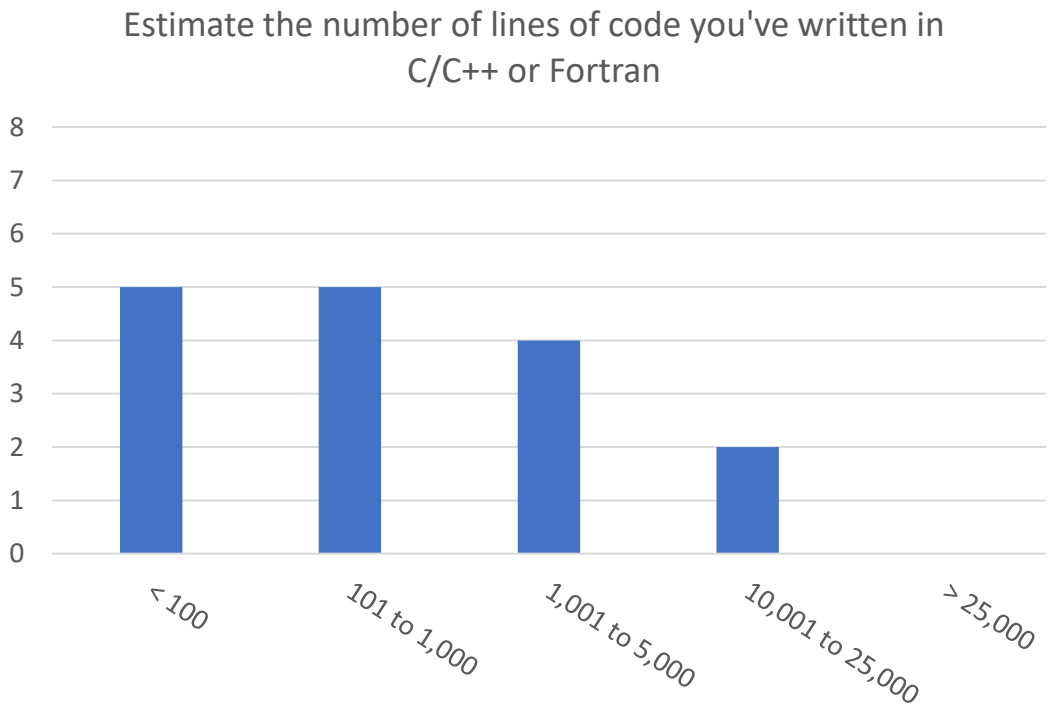


2020

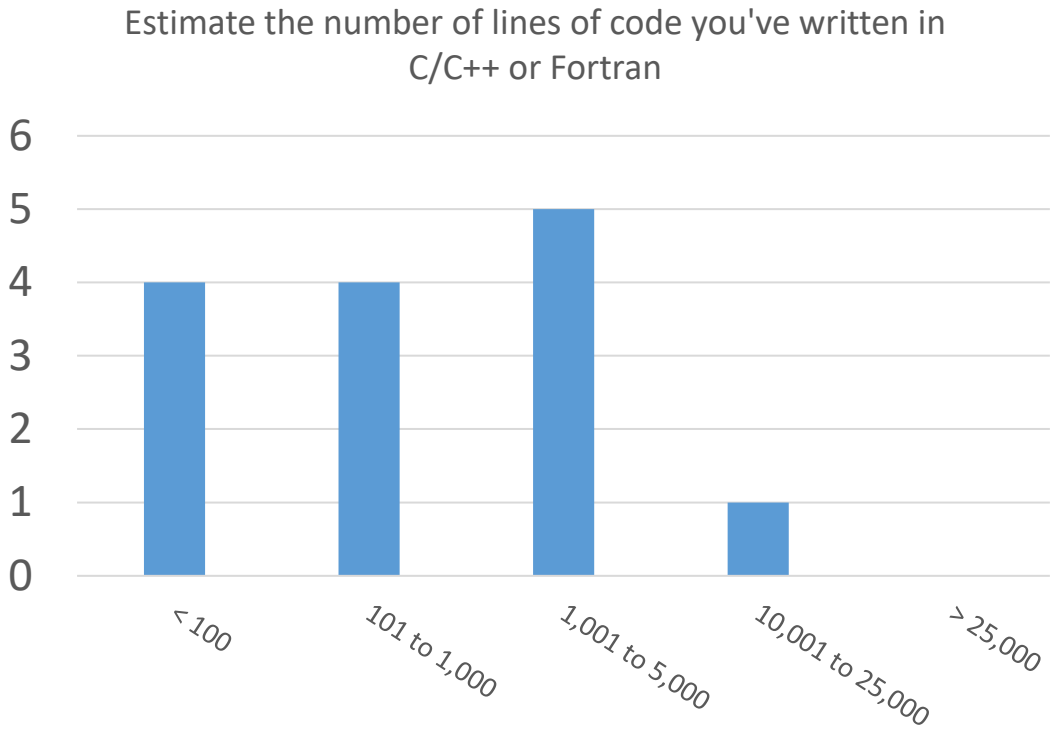


Survey Says…

2019

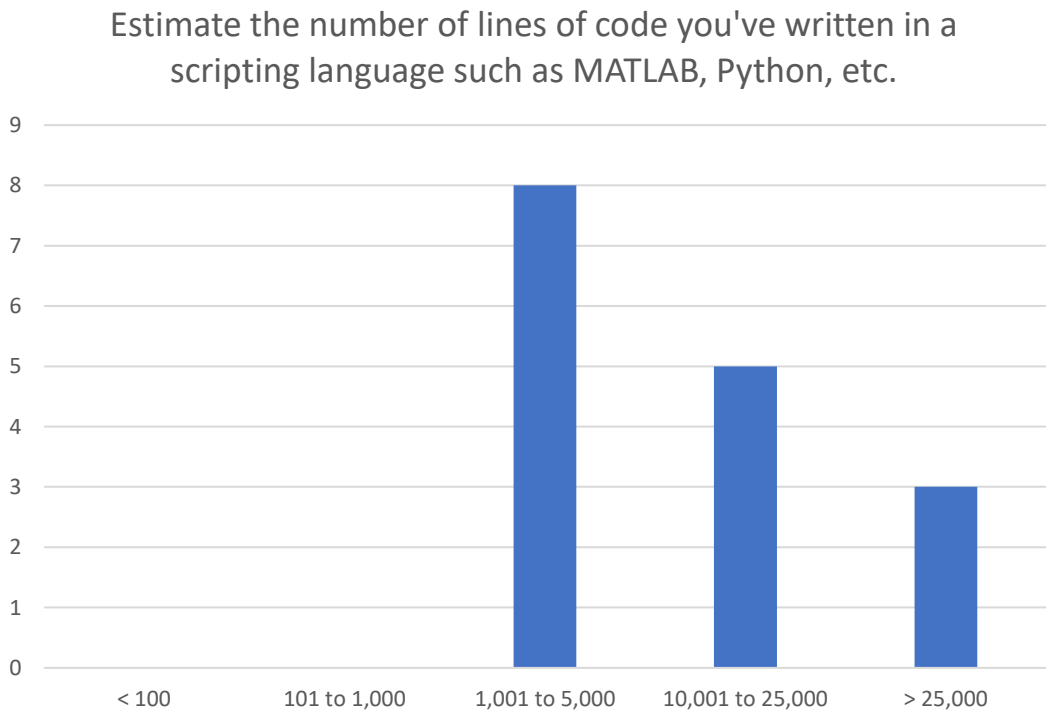


2020

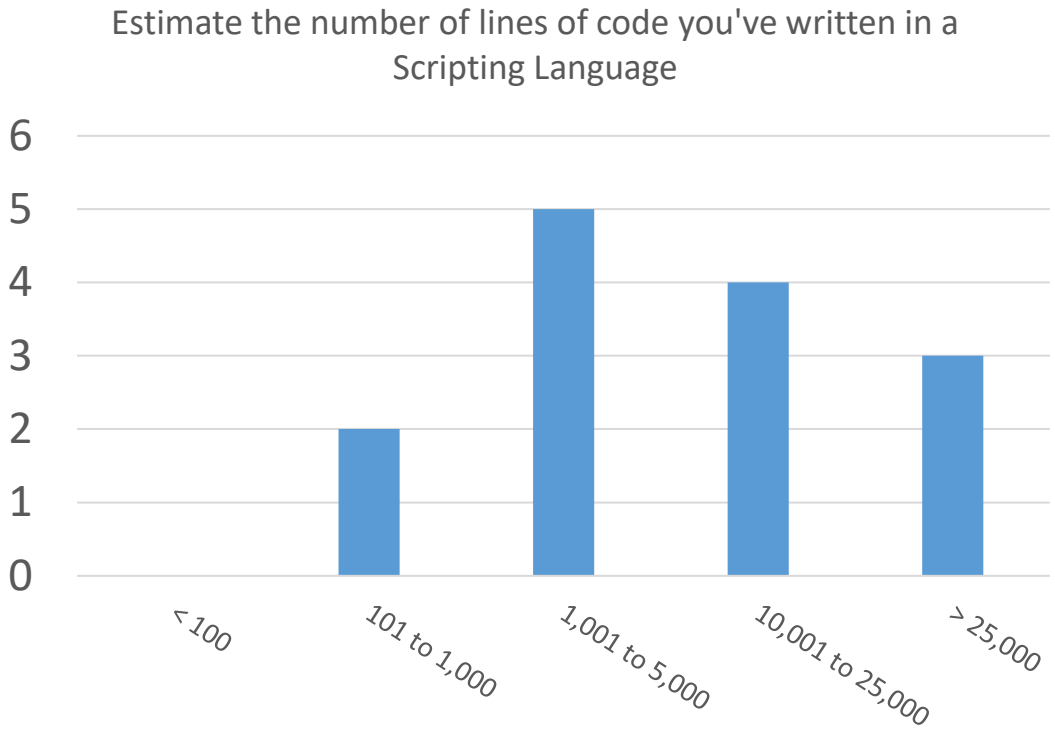


Survey Says…

2019

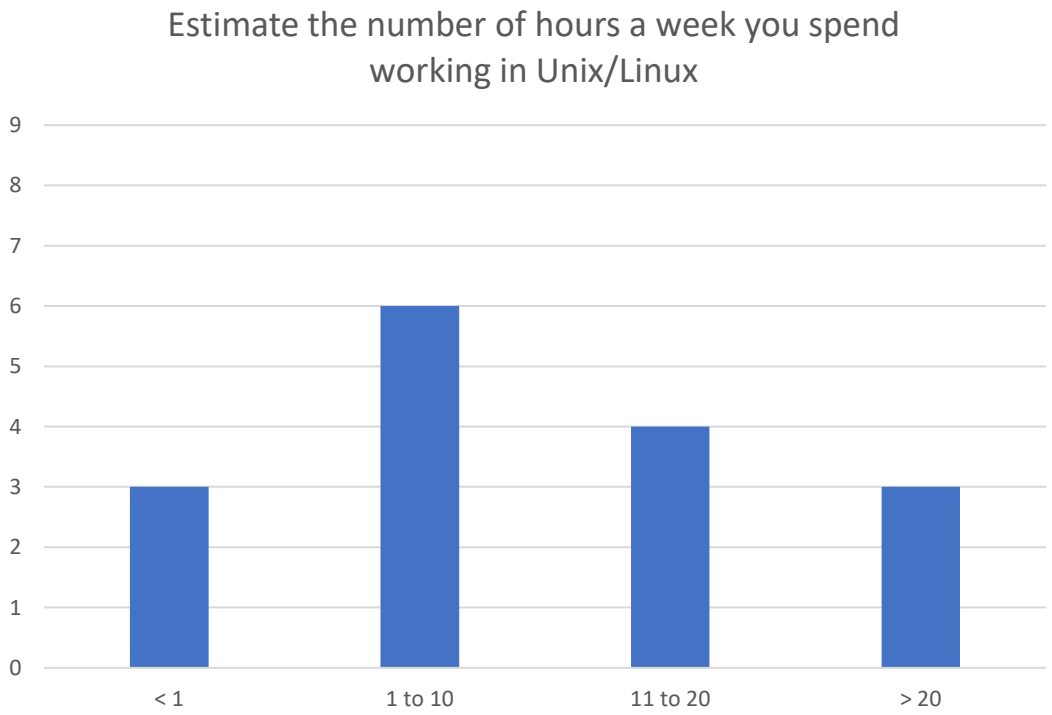


2020

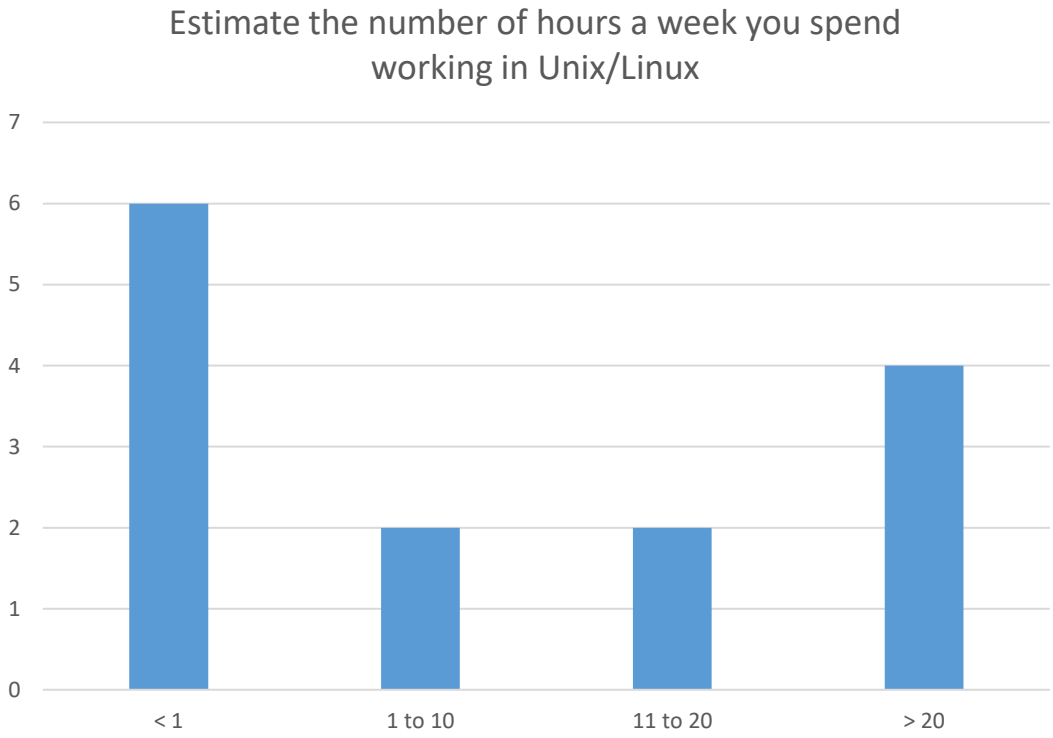


Survey Says…

2019

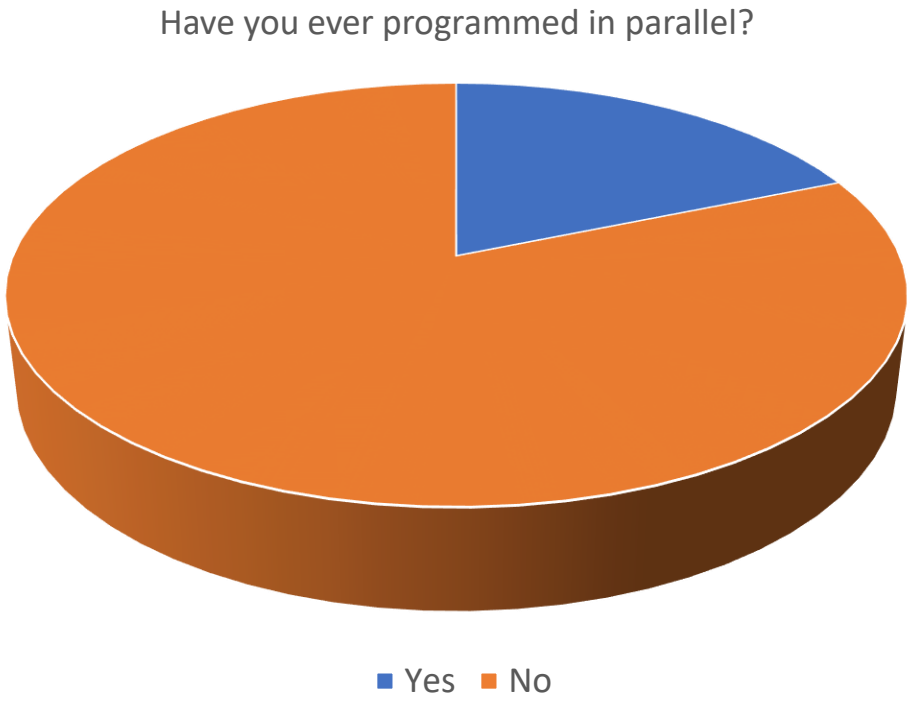


2020

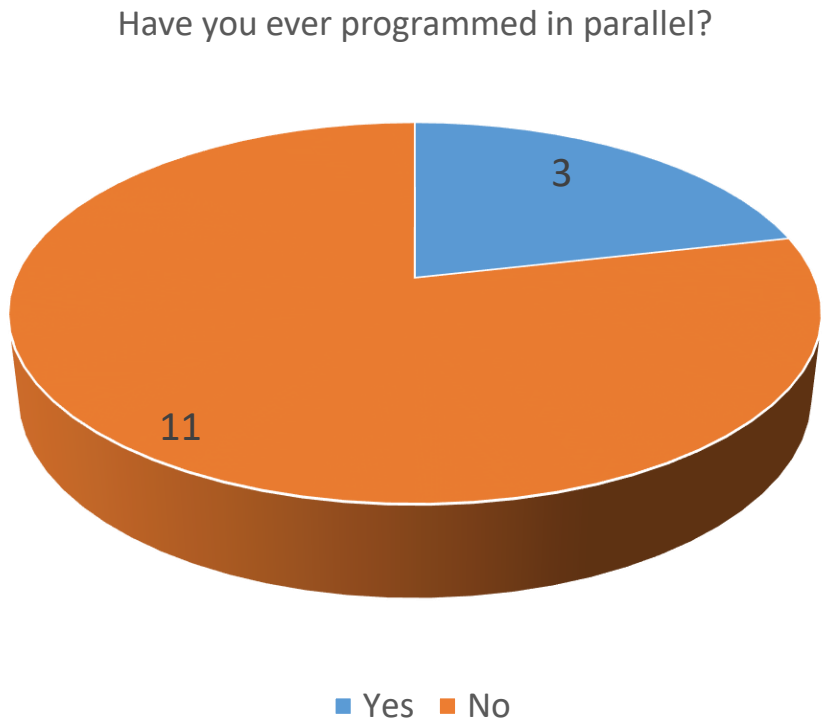


Survey Says…

2019

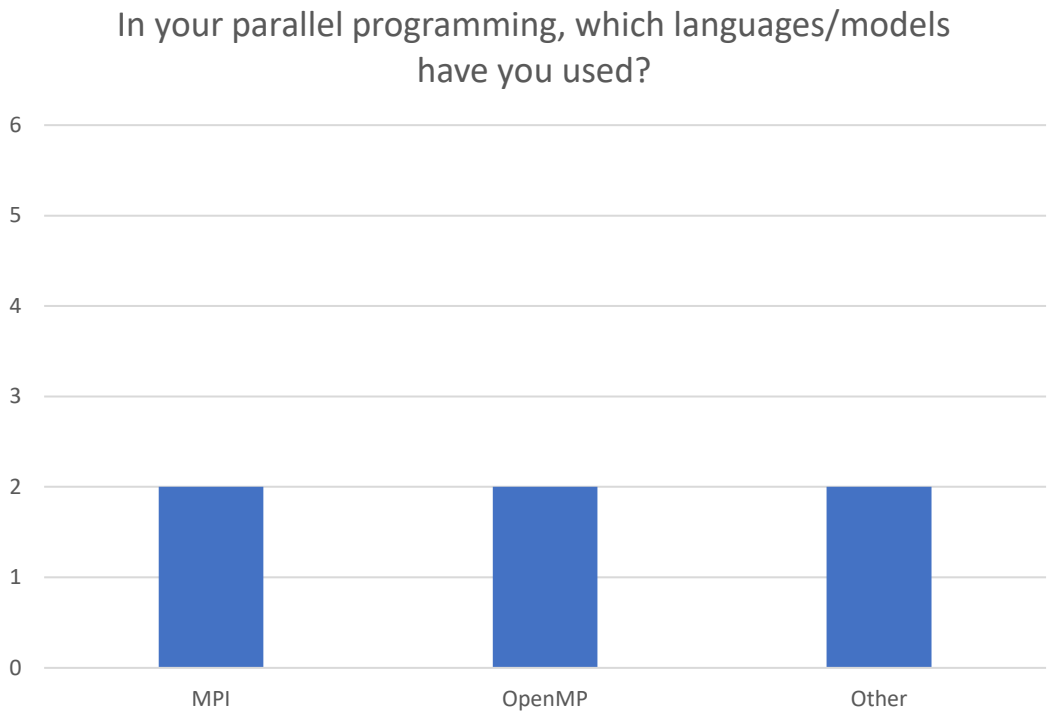


2020

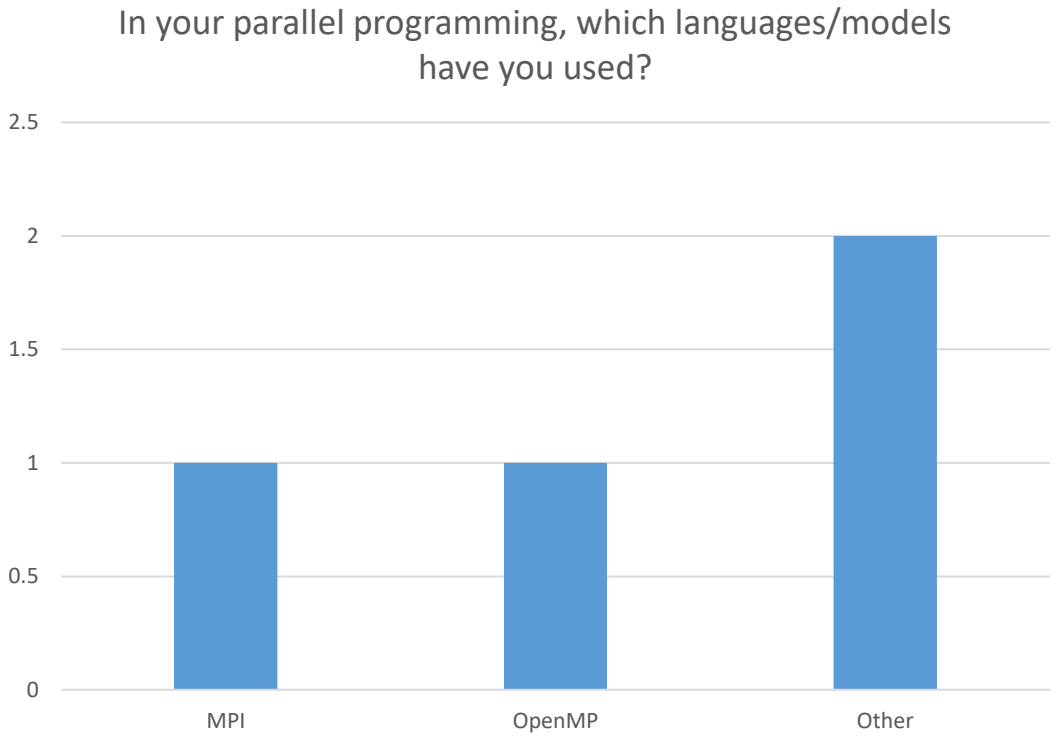


Survey Says...

2019

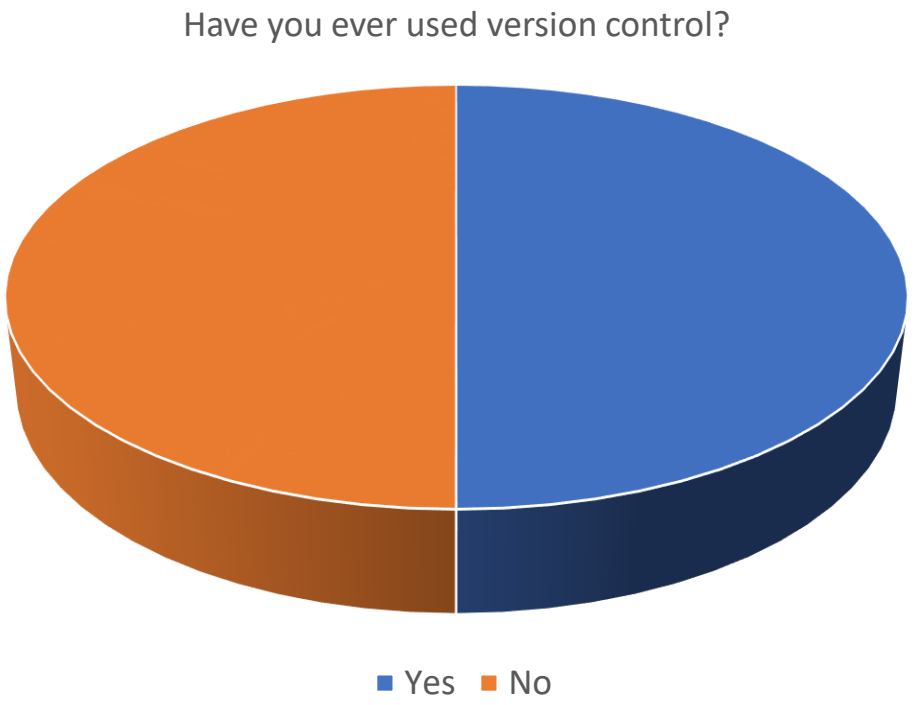


2020

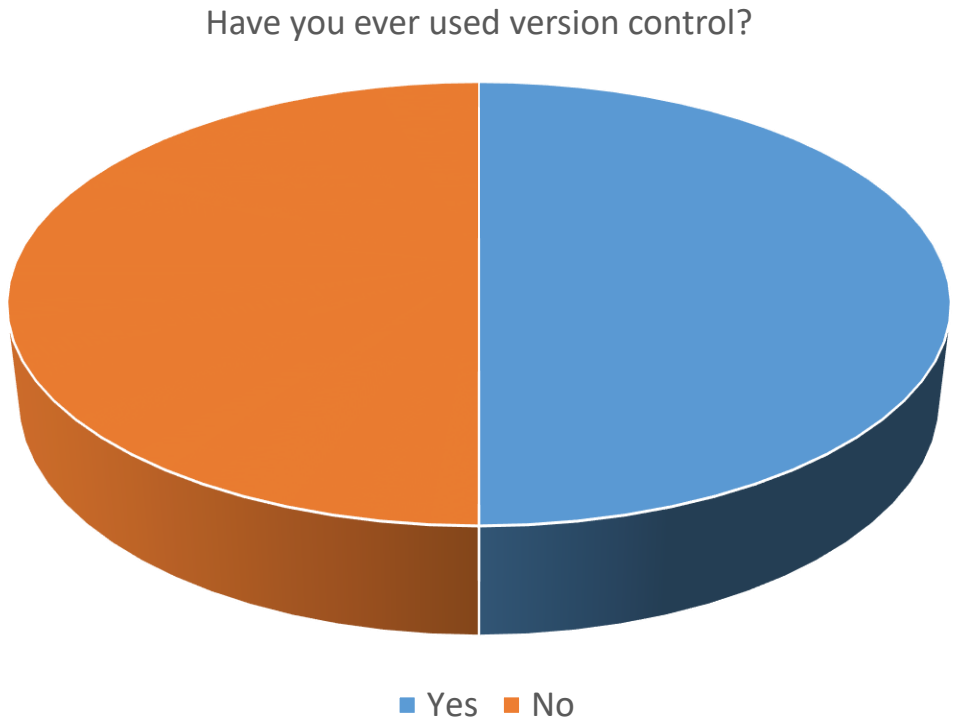


Survey Says...

2019

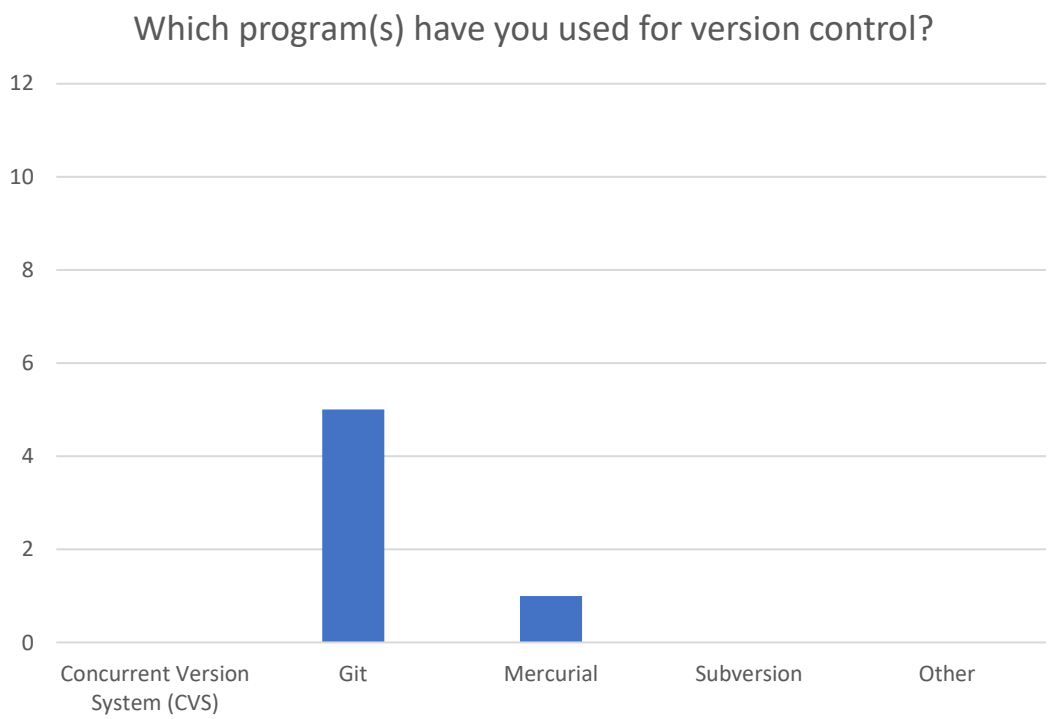


2020

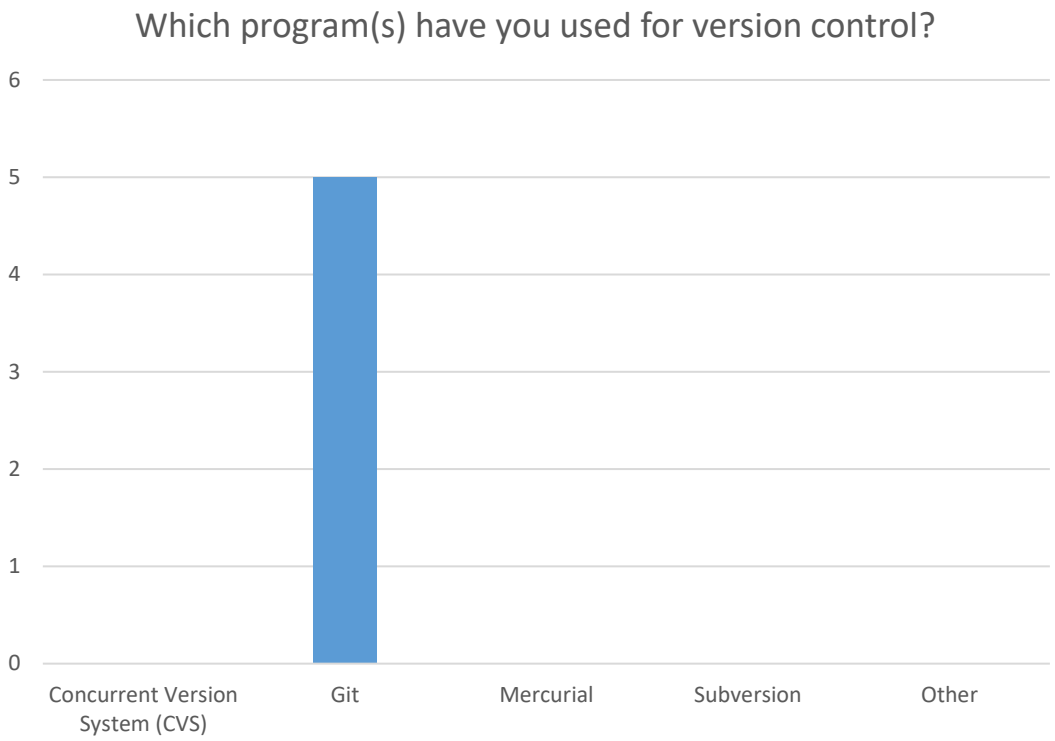


Survey Says..

2019

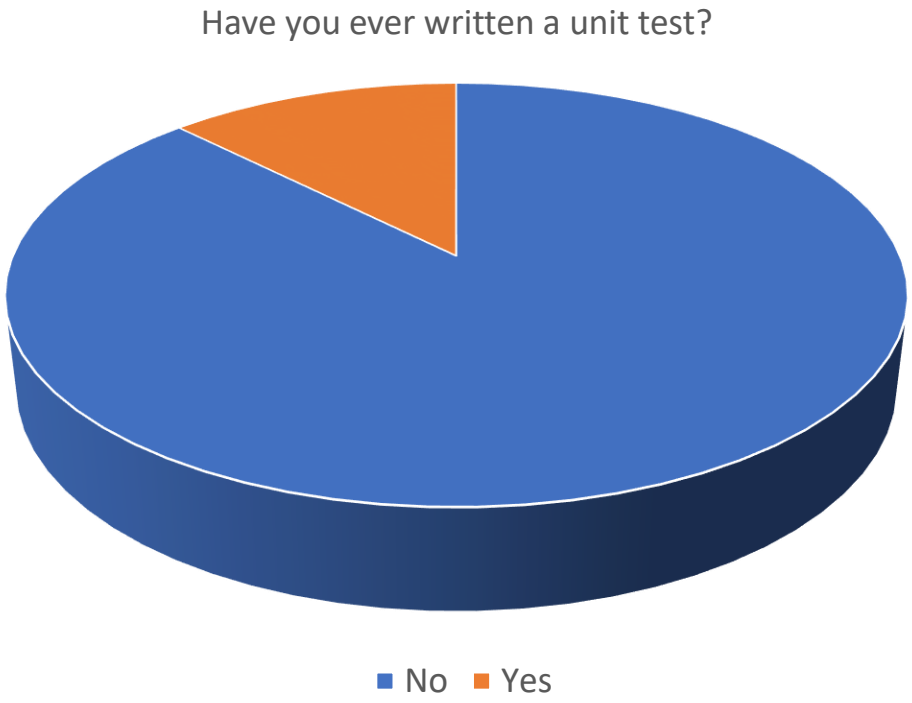


2020

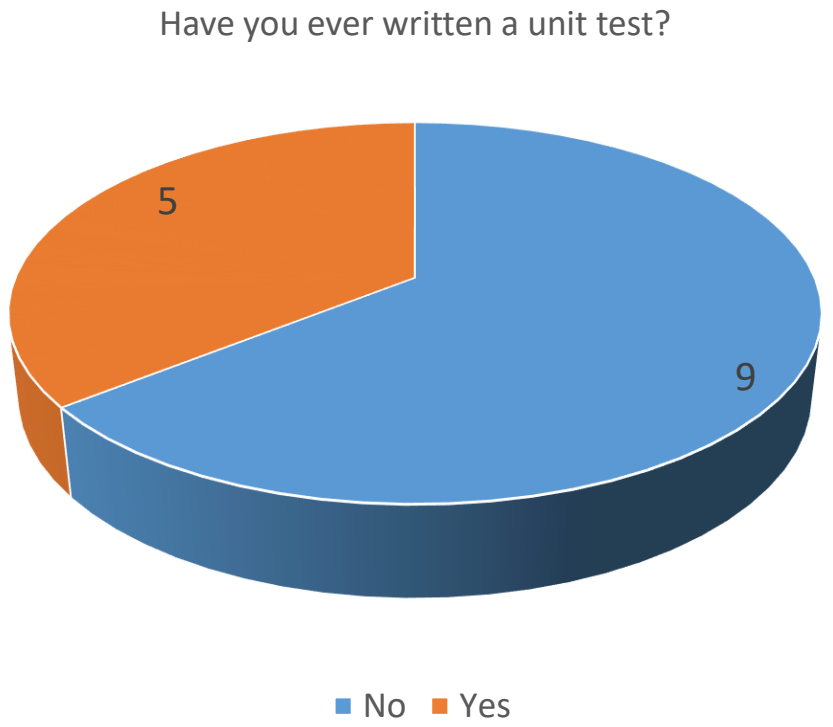


Survey Says…

2019

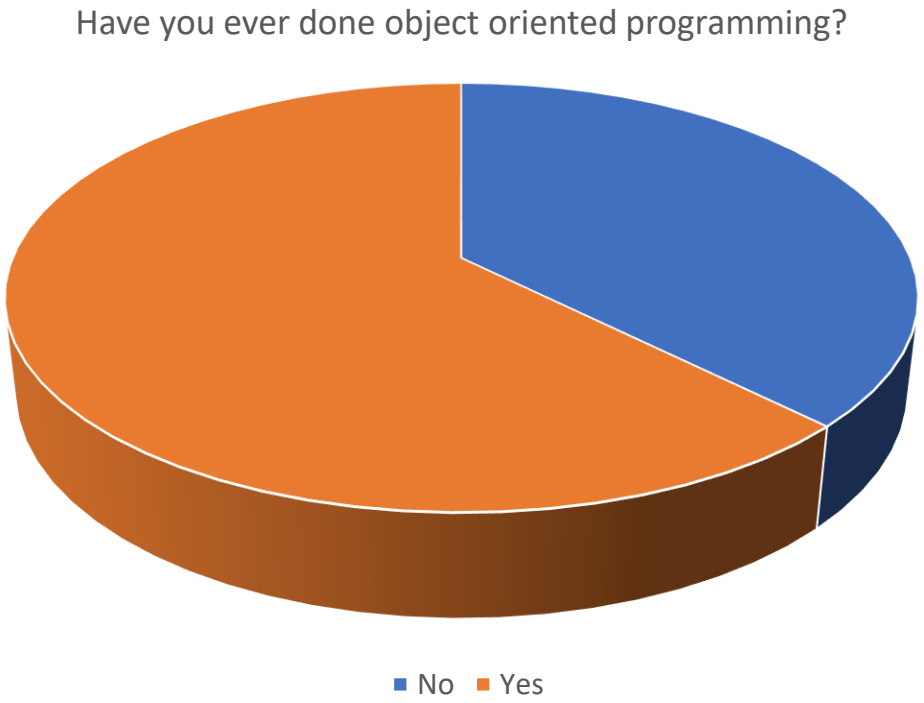


2020

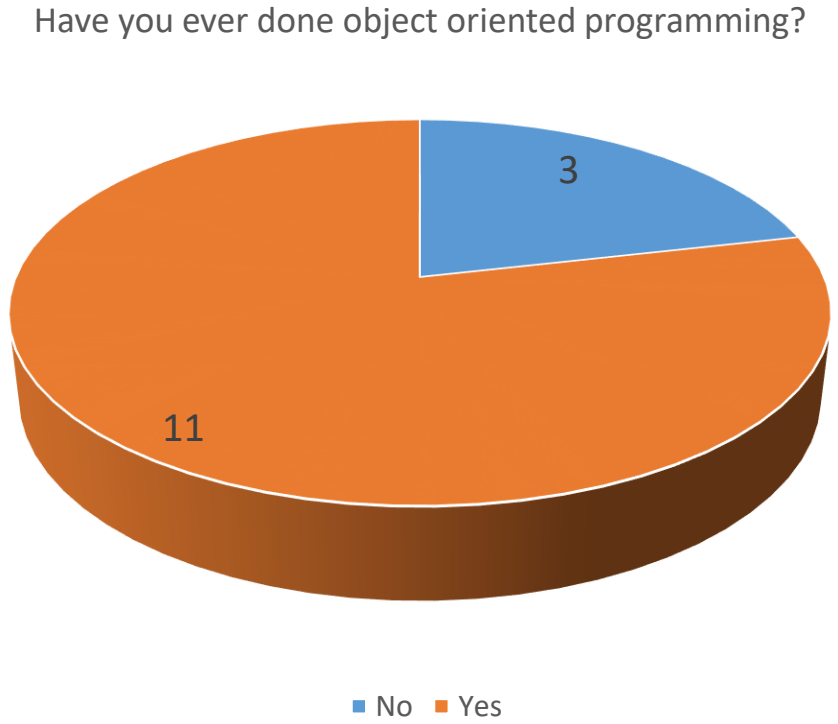


Survey Says...

2019

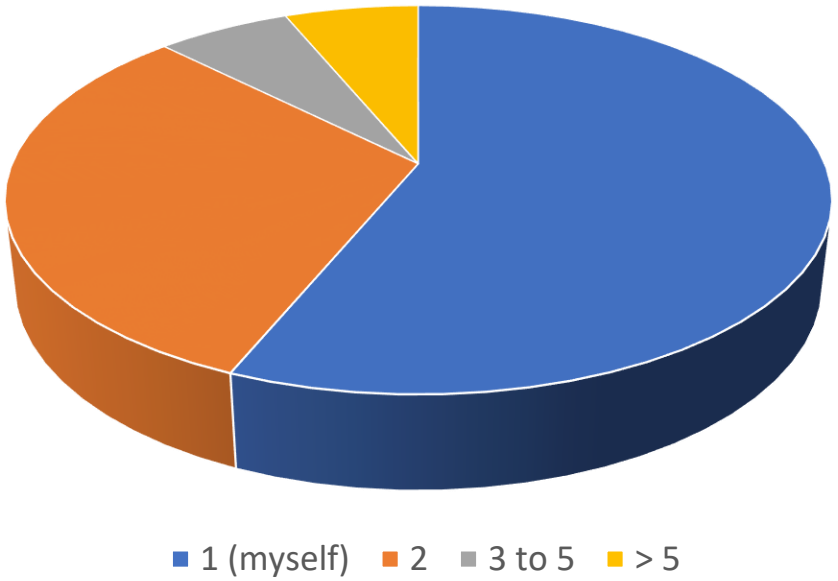


2020

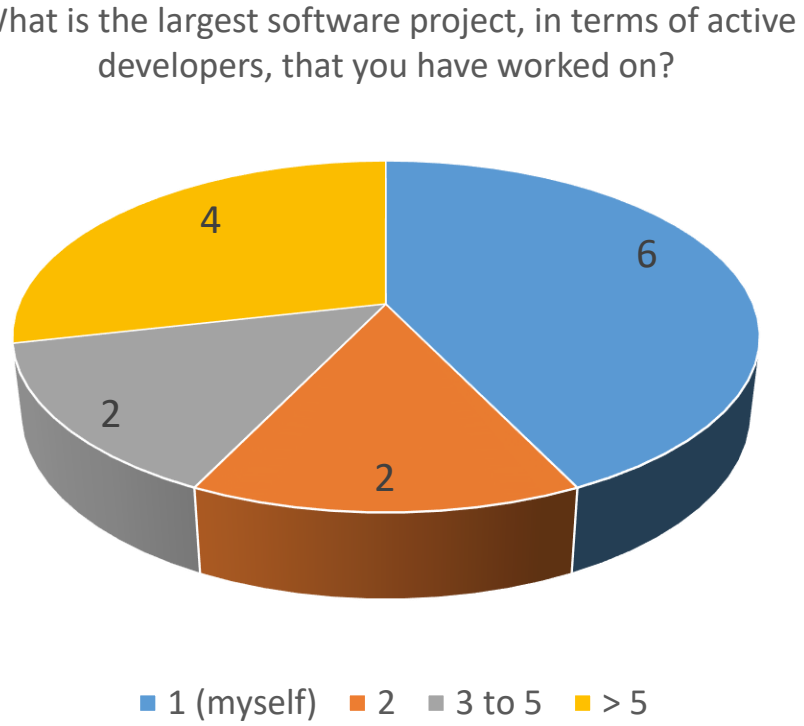


Survey Says…

2019

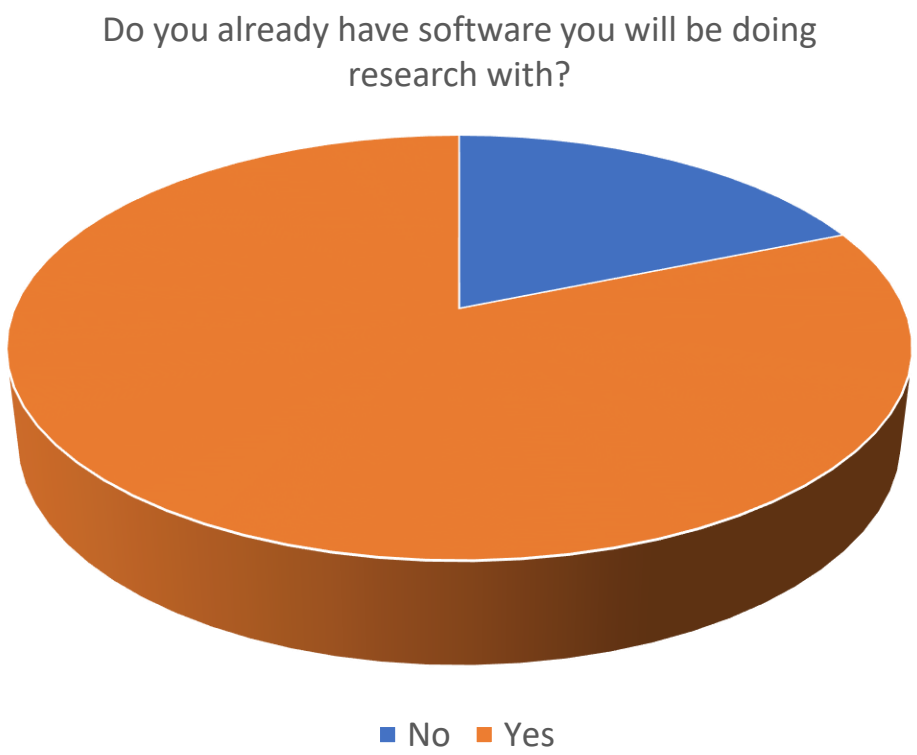


2020

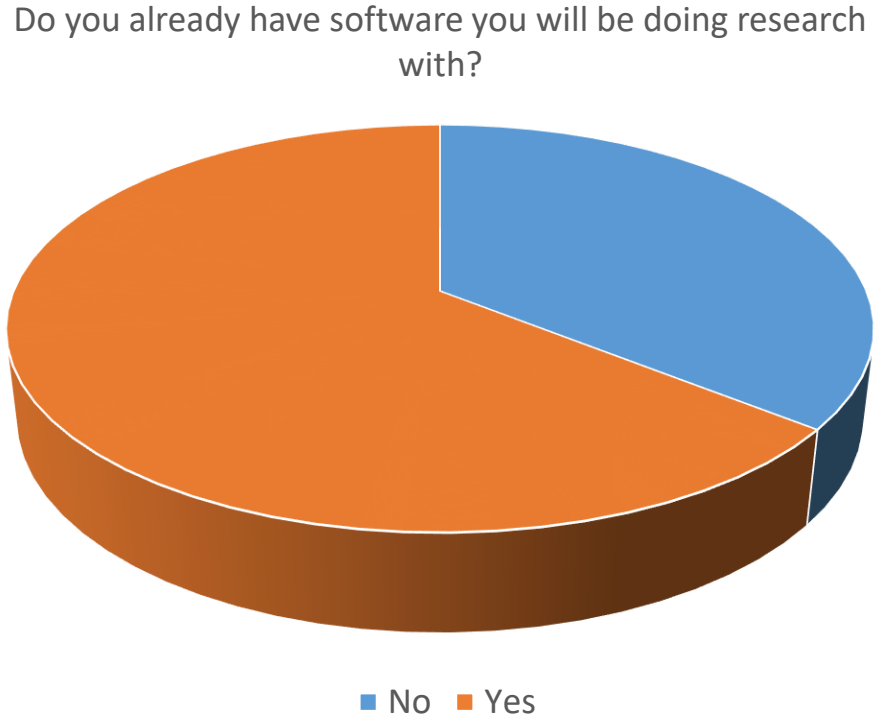


Are You already working with software in your research?

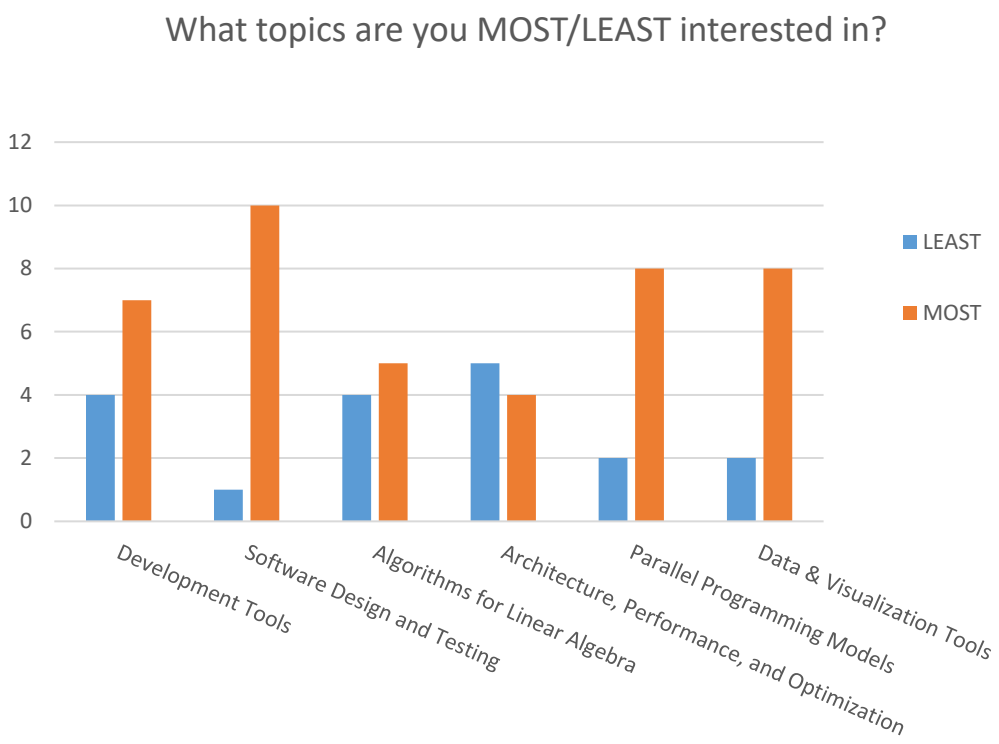
2019



2020



Survey Says...



- What other topics?
  - Machine Learning
  - GPUs
  - Pair Programming
  - Portability
  - Working on an HPC

What do I want out of this course?

2019

- Work more efficiently
  - Efficient workflow
  - Write code more efficiently
  - Write faster code
  - Spend less time debugging
- Improve Programming Skills
- Working on large projects

2020

- Use HPC
- Become a better/more efficient programmer
- Become more aware of tools
- Work on large projects

Programming Languages

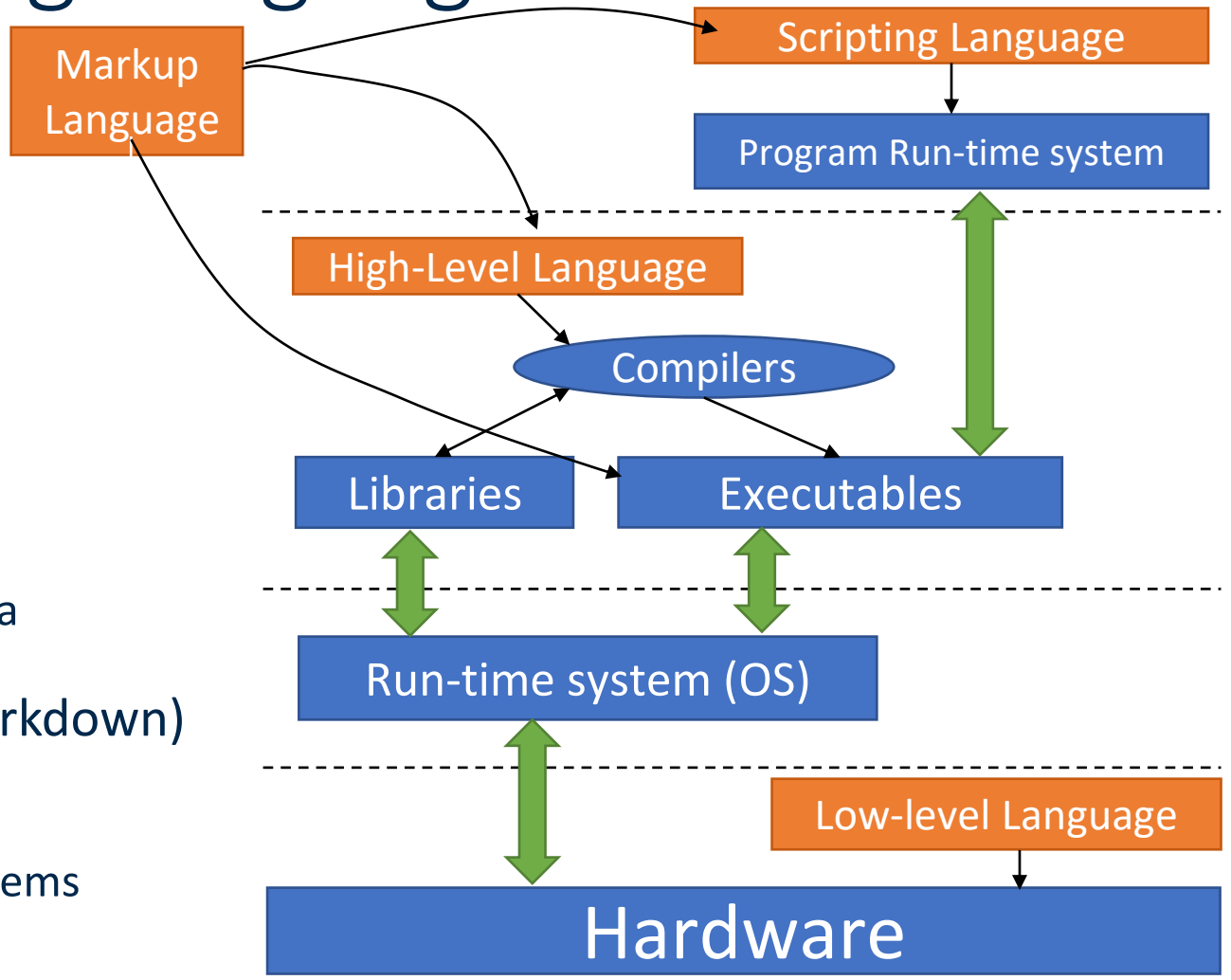
Recap

What defines a programming language?

- Primarily: A formal standard
  - C/C++: C89, C99, C11 and C++98, C++03, C++11, C++14, C++17
  - Fortran: ISO/IEC standard 1539
  - Java
  - Python
- Syntax: e.g. the “grammar”
- Semantics: e.g. the “meaning” which is “vocabulary + grammar”
- Programming languages should define performance of execution
  - Typically syntax/semantics are carefully defined, but not so for performance model.

Types of Programming Languages

- Low Level language (Assembly)
  - Defined by hardware (less portable)
- High Level language (C/C++, Fortran, Java)
  - Defined by run-time system (e.g. Operating System)
  - Portable, depends on compilers
- Scripting language (MATLAB, Python, Bash)
  - Defined by portable run-time system of a program
- Markup language (e.g. XML, YAML, Markdown)
  - Used for annotation
  - Data transfer
  - Input to multiple types of programs/systems



Things a programming language should do

- Variable declaration
  - Scalars, arrays, pointers
  - Intrinsic types, programmer defined types
- Operators
  - Arithmetic: +, -, *, /
  - Relational: <, >, ==
  - Logical: and, or, not, xor
  - Others: e.g. exponentiation
- Execution control constructs
  - Branching constructs
    - e.g. if/else, switch (or case)
  - Looping constructs
    - For or Do
    - While
  - Goto
- Memory management
  - Automatic
  - Or provide programmer with intrinsics

Fortran Recap

Fortran 90/95

- F90 Standard: ISO/IEC standard 1539:1991
- F95 Standard: ISO/IEC 1539-1:1997
- Free Form
  - Forget all the rules about columns
- Dynamic memory allocation
- Intrinsic support for array operations
  - (e.g. a(:)=b(:))
- Pointers: references to memory
- Modules: reusable components of program
- Derived data types: custom data types
- Removed some problematic features of F77

```
module linear_system
  type :: linsys_t
    real,pointer :: a(:,:),b(:),x(:)
  end type

  integer :: n
end module
```

```
program main
use linear_system
implicit none

real,pointer :: sol(:)
type(linsys_t) :: ls1,ls2

read(*,*) n
allocate(ls1%a(n,n))
allocate(ls1%b(n))
allocate(sol(n))
ls1%x => sol
! ...
```

Fortran 2003 & 2008

Fortran 2003: ISO/IEC 1539-1:2004

- Significant addition to F90/95
  - F90/95 programs are F2003 compliant
- Defined Object-Oriented Features
  - Polymorphism and inheritance
  - Type-bound procedures (methods)
- Other
  - Procedure pointers
  - Standardized C-interoperability
  - IEEE floating point arithmetic
  - Enhanced intrinsics for command line processing

Fortran 2008: ISO/IEC 1539-1:2010

- Primarily added language features for enhanced parallelism
- Added submodules
  - Entity facilitating program structure
- Coarray Fortran
  - Simple extension for distributed parallel programming
  - Typically relies on underlying MPI implementation
- DO CONCURRENT
  - Loop iterations have no interdependencies

C and C++

History of C/C++

- C - started in 1969 by Dennis Ritchie at Bell Labs
  - Concurrent with and related to the *development of Unix operating system* (also at Bell Labs)
  - Most widely used programming language – *available across virtually all platforms and architectures*
  - Informal specification "K&R" (Kernighan and Ritchie, the developers) published in 1978
  - ANSI standard since 1989
  - General purpose
  - Relatively fast
  - Current standard is C18
- C++ - also developed at Bell Labs, starting in 1979
  - Standardized in 1998 (International Organization for Standardization)
  - Added features/extensions that make development and management of large software projects much simpler (mainly classes)
  - Newer versions have more advanced functionality (templates, namespaces, abstract classes, and more)
  - Widely used for scientific computing purposes because of powerful Object-Oriented programming capabilities

C and C++ standards

C Standards

- C89
  - original ANSI standard (also C90)
  - still supported by compilers
- C99
  - new features include
    - function inlining,
    - `complex` type,
    - one-line comments with "`//`",
    - variable-length arrays, and more
- C11
  - atomic operations,
  - multi-threading,
  - bounds-checked functions,
  - static assertions
- C18
  - Revision to C11

C++ Standards

- C++98
  - First standard, introduces object-oriented features to C
    - Abstraction, Encapsulation, Inheritance, Polymorphism
  - ***Multiple inheritance***
- C++03
  - Mostly fixed issues with previous standard to improve compiler consistency
- C++11
  - Multi-threading, initializer lists for all classes, type inference (`auto`)
- C++14
  - ***Variable templates***, `deprecated` (for multi-threading), bug fixes, other
- C++17
  - No longer experimental in GNU compilers as of last month

Data Types

- Intrinsic – basic data types that are defined and recognized by standard C
  - Also available as multi-dimensional arrays
- Derived – library or user-defined data type that is made up of some combination of data with intrinsic types
  - Standard library: vector (of any data type), string
  - User-defined: a struct or class that contains several variables of any combination of intrinsic and/or derived types

Intrinsic Data Types	
Keyword	Details
`unsigned char`	8-bit integer 0 to 255
`signed char`	8-bit integer -127 to 128
`unsigned int`	`short` (16-bit), `long` (32-bit or 64-bit), and `long long` (64-bit)
`signed int`	
`float`	32-bit
`double`	64-bit
`complex` (C99)	`complex float` and `complex double`
`bool`	True or False
`enum`	enumeration for a set

Definable (Derived) Data Types	
Keyword	Details
`union`	Define several variables to share same memory space
`struct`	Defined in terms of intrinsic data types (or other `structs`)

C/C++ Variable Declaration

- In C/C++, legal variable declarations can be made *at any point within a code*
  - Unlike Fortran, where all type declarations are made at the top of a routine or program
  - For C90, must be immediately after {
- All variables must be declared as a certain type (and should be initialized) before being used in some expression
- Variables remain defined throughout the scope in which they were declared. When a variable "falls out" of scope, its data is no longer accessible
- Scope is defined in C++ by opening and closing curly braces {}

```
float variable;
int variable = value;
```

C Operators & Special Characters

Logical Operators	
&&	and
||	or
!	not

Arithmetic Operators	
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
=	assignment

Other	
Symbol(s)	Meaning
// or /* */	Comment and comment block
;	End of statement
{ }	Scope declaration
? :	Ternary operator (like a compacted if/else)
[ ]	Array declaration
*	Dereference variable (Unary)
&	Return memory address of variable (Unary)
-> or .	Member selection (-> is for pointers, . is for non-pointers)

Relational Operators	
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Augmented Assignment	
++	a=a+1
--	a=a-1
+=	a=a+b
-=	a=a-b
*=	a=a*b
/=	a=a/b

Bitwise Operators	
&	Bitwise AND
|	Bitwise OR
^	Exclusive OR
~	Toggle (flip) bits
>>	Left Shift
<<	Right Shift

Augmented assignment operators also available for bitwise operations

Program Entity Structure

Preprocessor statements (Header file declarations)

Global Declarations

In C++ Class declarations go here (will cover examples in more detail with OO programming)

Main program (only define once in program)

Functions

Other functions (optional)

```c
/* Preprocessor commands
   and include statements */
#include <stdio.h>

//Global variables and interface definitions
int n;
void sub1(int* n);

/* Functions/Subroutines
   "main" is a special procedure
   in C/C++ corresponding to the
   "main program" */
int main()
{
    sub1(&n);
    return n;
}
/* "void" means return nothing, like a Fortran
   subroutine */
void sub1(int *n)
{
  *n=0;
}
```

C example

Header Files

- Header files are used to define variables and interfaces
  - Generally each source file has a corresponding header file, especially in C++
- The header data is appended to the top of the source code files by the compiler when an #include "header.h" statement is present.
- Header files allow compilers to "resolve dependencies" in your program.
- In C++ header files:
  - Variables are declared not only as a specific type but also public or private
    - Public: any function has access to read, use, and change this data
    - Private: only the object that owns the data can manipulate it
  - Methods of a class are defined by name, type of returned data, and types of all of the input data
  - GOOD practice: "namespace" entitites in your header files.

C++ example header

```
namespace ners570
{
  class timer {
    public:
        tic();
        toc();
    private:
      double startTime_;
      double totalTime_;
}
```

Usage

```
#include "ners570_timer.hpp"

//Declare variable "t" as a ners570
//timer, other headers might define
//classes named timer
ners590::timer t;
```

Execution Control Constructs

- IF/ELSE
  - Followed by curly braces { }, executes code within braces depending on result of conditional checks
  - `if (condition) { }`
- Switch
  - A logical branch, achieves the same logical result as IF/ELSE with much simpler syntax in some cases, and a different sequence of logical checks
    - `switch(variable):`
      - `case(value1)`
      - `case(value2)`
- `for(initialization, condition, step) {}`
  - Repeat code and execute step (usually increment/decrement) after each loop until the condition is no longer met

C and C++ I/O

C

- Routines are defined in "standard library", accessible through header files.
  - Primary library: `<stdio.h>`
- Some useful routines:
  - `printf`, `fprintf`
  - `fopen`, `fclose`, `fget`, `fscanf`

C++

- C++ I/O libraries: `<fstream>`, `<iostream>`
- `cout << "Hello, world!" << endl;`
  - Hello, world!
- `cout` outputs to screen, is in standard namespace (e.g. equivalent to `std::cout`, "`std::`" is implied)
- `cin <<` waits for input from the keyboard

Standard Library

- Different for C and C++
- ***A HUGE advantage over Fortran***
- Contains many useful types and functions
  - Example: `Vector`
    - a dynamic data container class with many methods for accessing more effectively than a standard array
    - `.push_back()`, `.pop_back()`, `.at()`, `.size()`
  - Dynamic memory management (`malloc`, `dmalloc`, `free`)
  - Random number generator (`rand`, `srand`, `RAND_MAX`)
  - `abs()` – absolute value
- Many other useful C libraries
  - `cstring`, `cmath`, `cstdio`

Angled brackets imply header is supplied by system or compiler

```
#include <vector>
#include "ners570_timer.hpp"
```

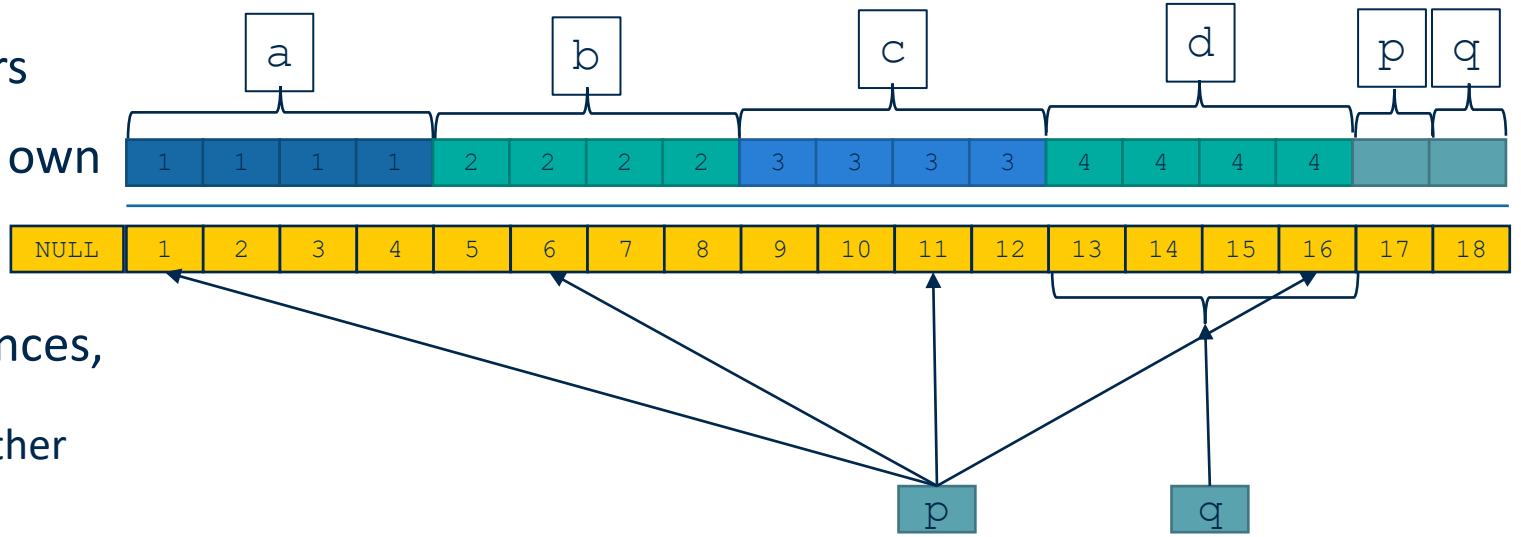
Quotes imply header file is source file defined by programmer

Pointers

- Difficult to grasp if new to programming.
- In high-level programming languages variable names are associated with memory.
  - value of variable is associated with memory location
- Variables that are defined as pointers can “point” to different memory locations and often don’t have their own associated memory.
- If a block of memory loses all references, this is a memory leak.
  - These are bad. More on these in another lecture.

```
integer a[4], b[4], c[4], d[4]
pointer p, q[]

p => a(1)
p => b(2)
p => c(3)
p => d(4)
q => d(:)
```



Reference Counted Pointers (RCP)

- Also known as a smart pointer.
  - Automates “garbage cleanup” e.g. frees unused memory
- Keeps track of (e.g. counts) how many pointers are referencing a particular block of memory
  - When no more pointers are referencing the block of memory, it can be automatically freed.
  - Useful machinery for helping to prevent memory leaks in C++

- Part of C++11 Standard: `std::shared_ptr`
- Other implementations available from third party libraries: boost, Trilinos

Templating (C++)

- ***Very powerful*** feature in C++
  - Idea is like meta-programming.
- Write a program for a "template" type.
- Compiler generates machine code for all data types matching the template.

```
class calc
{
  public:
    int multiply(int x, int y);
    int add(int x, int y);
};
int calc::multiply(int x, int y) { return x*y; }
int calc::add(int x, int y) { return x+y; }
```

```
template <class A_t> class calc
{
  public:
    A_t multiply(A_t x, A_t y);
    A_t add(A_t x, A_t y);
};

template <class A_t> A_Type calc<A_t>::multiply(A_t x,A_t y)
{
  return x*y;
}

template <class A_t> A_t calc<A_t>::add(A_t x, A_t y)
{
  return x+y;
}
```

Creating a Template means you have written this routine once for integers, floats, doubles, bools, and any classes of the templated type!

Example from: http://www.cprogramming.com/tutorial/templates.html

Further Reading

C

- The Language
- Standard Library

C++

- The Language
- Standard Library

Working with Fortran and C/C++ together

Interoperable Intrinsic Datatypes (Fortran 2003 and later)

Description	Fortran type declaration	C type declaration
Character	`CHARACTER(LEN=1,KIND=C_CHAR)`	`char`
True/False	`LOGICAL(C_BOOL)`	`_Bool`
default integer	`INTEGER(C_INT)`	`int`
floating point (32-bit)	`REAL(C_FLOAT)`	`float`
double precision (64-bit)	`REAL(C_DOUBLE)`	`double`
Integer 8-bit	`INTEGER(C_INT8_T)`	`int8_t`
Integer 16-bit	`INTEGER(C_INT16_T)`	`int16_t`
Integer 32-bit	`INTEGER(C_INT32_T)`	`int32_t`
Integer 64-bit	`INTEGER(C_INT64_T)`	`int64_t`
Long integer	`INTEGER(C_LONG)`	`long int`
Long long integer	`INTEGER(C_LONG)`	`long long int`

See documentation for ISO_C_BINDING module for complete listing.

Modules and Headers

- Fortran modules are like compiler generated header files
  - Makes Fortran a little bit more tricky to compile because modules must be compiled in the correct order to resolve dependencies.
  - Makes C/C++ a little more cumbersome because you are always having to maintain header files and source files.
- Header files are portable.
- *Compiled* module files are not.
- Fortran can also include "header files", but the `INCLUDE` keyword in Fortran implies a direct insertion (e.g. copy-paste) of the contents of an `INCLUDE` file.
  - Contents must be valid Fortran code
  - Typically used for defining types or global variables, sort of deprecated as a bad programming practice.

```
program main
  use module1
  use module2

  include 'file1.h'

  !Rest of program

end program
```

Modules "module1" and "module2" must have been compiled prior to compiling the program.

Other Things to keep in mind

Item	Fortran	C	C++
Case Sensitive variable names	No	Yes	Yes
Statements end with “;”	Optional	Required	Required
Assumed Starting Index	1	0	0
Multi-dimensional array ordering in memory	In-out	Out-in	Out-in
Strings	Fixed length	End with null character	

Memory layout of multi-dimensional arrays

Fortran

i ⟶

j ↓

a(i,j)	1	2
1	1	**2**
2	**3**	4

C/C++

i ⟶

j ↓

a[i][j]	0	1
0	1	**3**
1	**2**	4

Other things you'll encounter

Programming Language Extensions

- Includes things like
  - Syntax or Semantics not defined in the standard
    - An example is Co-array Fortran
      - Until Fortran 2008 standard
  - Additional functions
    - Interfaces to OS
- BEWARE
  - Extensions are usually not portable between compilers

Directives and Preprocessor

- Primarily applicable to high level languages with compilers
- Preprocessor
  - Happens before compilation
  - Can control what source code is compiled
  - Mechanism for portability
- Directives
  - can appear as comments or preprocessor commands
  - Allows for safe compilation when feature is not available.

Upcoming Assignments

Lab 2 – Bash and Python Scripting

- Searching (on your own outside of lab)
  - Use of grep and find
- Parametric Study (work through half of this in lab)
  - Steam Table Evaluation
  - Write same script in Bash and Python

Homework 1

- Write a LaTeX document
  - Suggest you use www.overleaf.com
- Develop the same program in C/C++ and Fortran
  - Index mapping and numbering with a space filling curve
- C/C++/Fortran Interoperability
  - Take previous program and call Fortran from C and C from Fortran