★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

Introduction to C++

Day 1

13-17 December 2021

Pat Barton.
President/CEO, PES LLC

A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 062419 egd

ANALYTICAL GOVERNMENT SOLUTIONS (AGS), LLC

AGS is a robust, certified Service Disabled Veteran Owned Small Business (SDVOSB)

- Incorporated September of 2009 by Gil Dickens, a 20- year Air Force vet, in Lorton, Virginia.
- AGS takes pride in successful support of Department of Defense and Federal Agency missions
- Our creed: Leave a mark of superior performance that enables the men and women of the DoD and client organizations to accomplish their objectives at the highest performance level possible.

Expanded 2013 to include IT and Cyber technology specialties, full spectrum training curriculum development and delivery, logistics, quantitative sciences and program management.

- Top-notch Large and Small Business partners
- Integrated Business Solutions for IT/Cyber, Training, PPBE/Financial Management, Business Case and Cost Analysis, Systems Acquisition, Logistics and Systems Engineering serving Federal and Private sector clients.

Headquartered in Lorton, VA,

- AGS locations are in Old Town Alexandria, DC, and Maryland areas with major partners supporting clients in Northern Virginia and major metropolitan cities including Nashville, San Diego, and Atlanta.

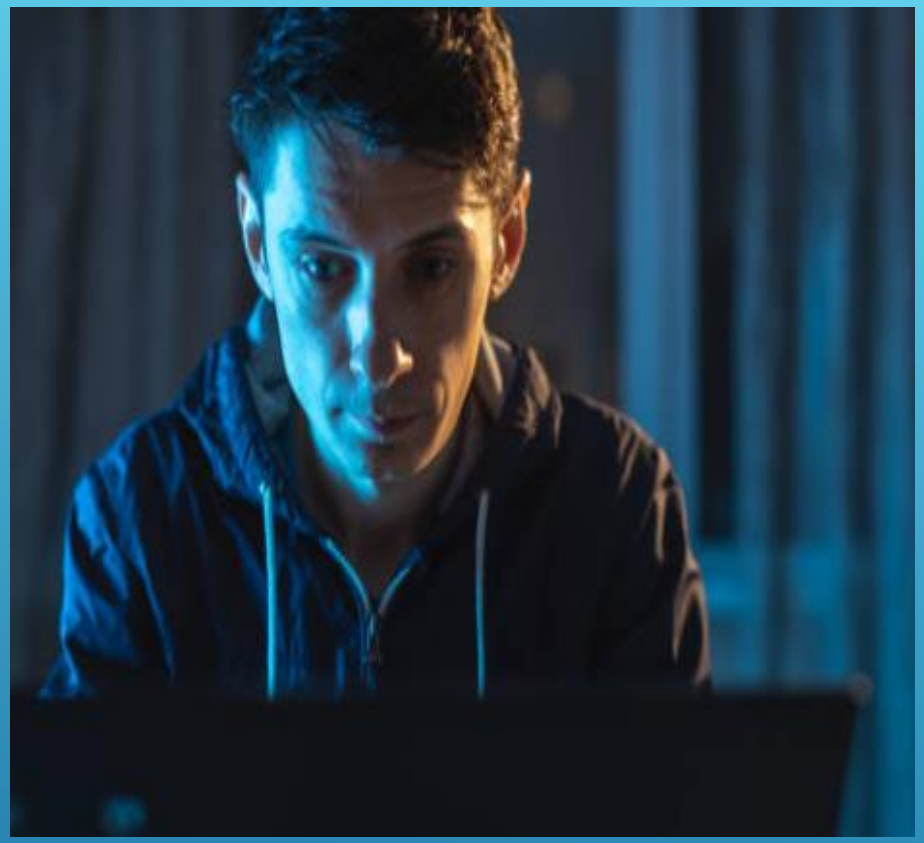
Your Facilitators

Gil Dickens is President of AGS LLC. He was previously Area Director of Acquisition Cost and Logistics for a major defense firm, and has 44 years of acquisition and program management leadership in Service Level Financial Management/ Analytics, IT Program Management for software design/development, Defense Satellite, Air Force Space, Marine Corps ground/ C4ISR, Training Curriculum development/ delivery, Cyber IT, avionics, electronics, environmental remediation, chemical/biological, and Defense systems at the highest DoD levels. He has a Master of Public Administration (Economics) from Harvard with coursework including National Economic Models, International Trade Policy and Law (at Harvard Law).

Pat Barton is a former Energy Scientist at PNNL with expertise in mathematical modeling of complex ecological and human-build systems. He was the lead Python instructor and Dean of Faculty at the O'Reilly School of Technology and has 10 years' experience teaching statistics, programming, and mathematical to mid-career adults.

He is the CEO of PES, LLC, a consulting firm in the software development and mathematical modeling space.

He has a Masters of Engineering from the Thayer School at Dartmouth College and a B.A. in Economics from the University of Michigan.



ADMINISTRATIVE OVERVIEW

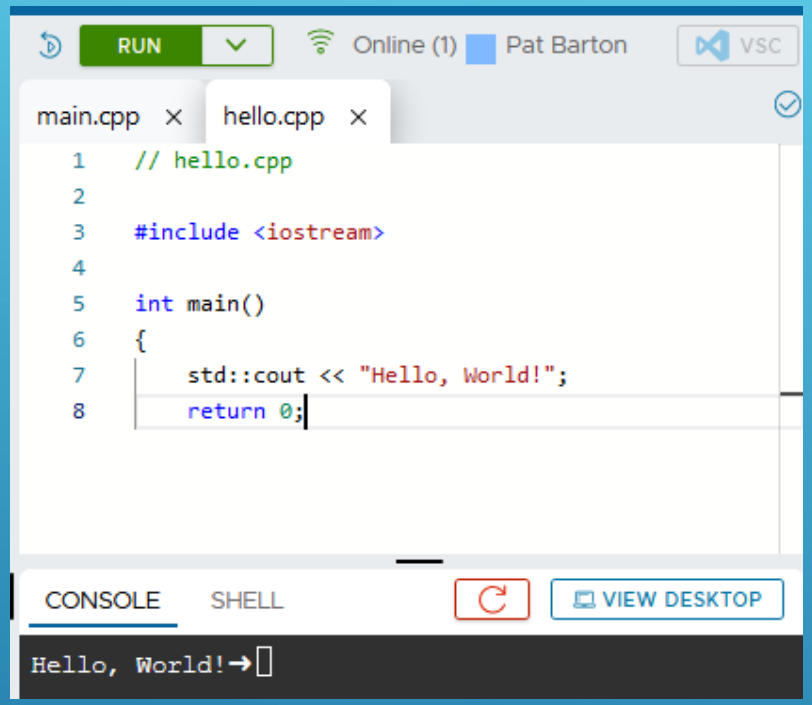
- Five (5) days, 40 hours of training
- Class is 0800-1600, room open 0700 for set up questions/help
- Average lesson blocks of 50 minutes each with exercises; breaks in the morning/ afternoon
- Lunch (generally around 12-1PM) w optional/ flexible continued instruction
- Instructor available during lunch for optional work session/help
- Please mute when you are not speaking
- Information is open source (no classification level)
- Avoid sensitive topics (even Controlled Unclassified Information)

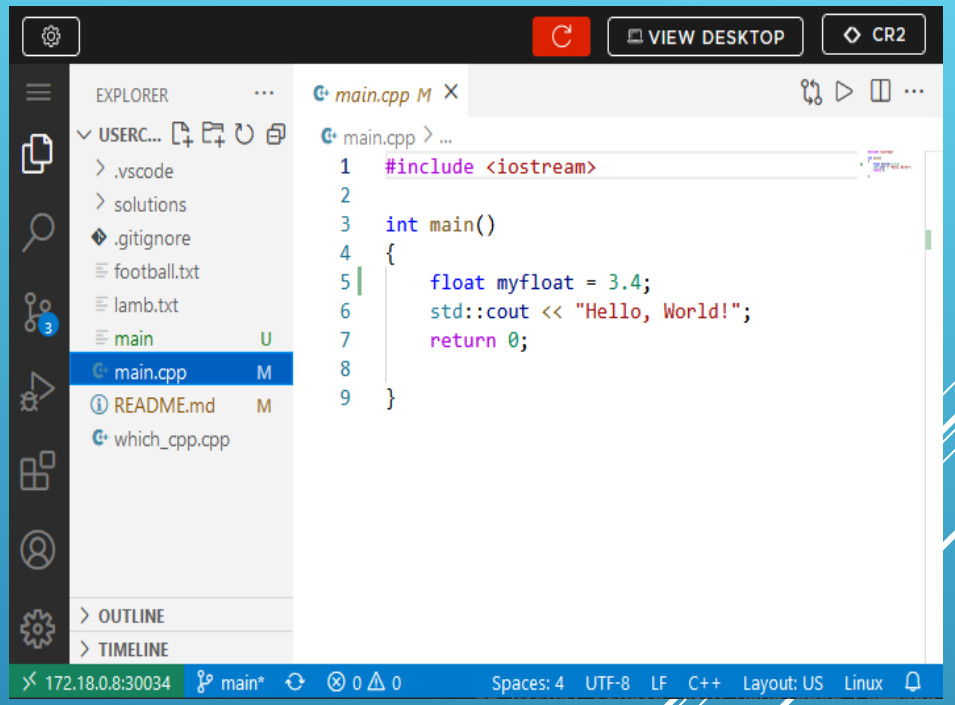
STUDENT INTRODUCTIONS

- What name do you prefer?
- Where do you work (i.e. division, systems, project, etc.)?
- What electronic warfare (EW) experience do you have?
- What other relevant experience (military, educational, etc.)
- Please share a fun fact about yoursekf,

- Welcome !!!
  - Student and faculty introductions
  - Logistics
    - Questions?? YES!! Questions!
    - Platforms and tech
      - CodingRooms (self-contained)
      - BYOD
      - Slack
    - Schedule, breaks, lunch, etc.
  - Overall goals for the class
  - Today's topics

- In your IDE you may see something like one of these:





- These are code editors that sit on a fully-functional virtual computer.
- Or you can use your own. We'll set up these environments a little better shortly

Overall Goals - in this class you will:

- Write dozens of C++ programs
- Learn basic language syntax
- Learn the main variable types
- Utilize collections
- Code with logic loops and branches
- Create functions and classes
- Apply OOP concepts
- Learn how to use pointers
- Apply testing
- Learn to manage exceptions

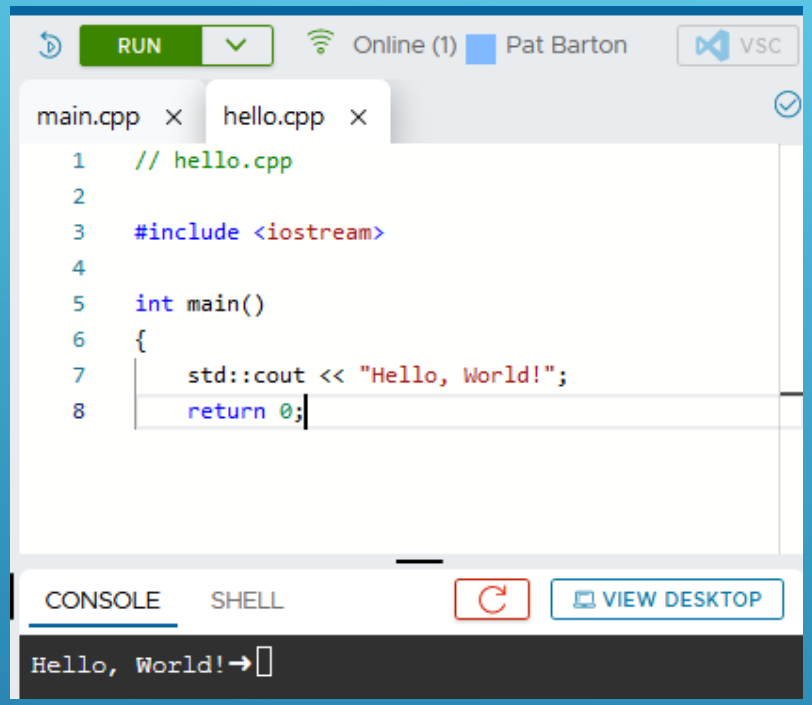
- ... and especially write dozens of C++ programs

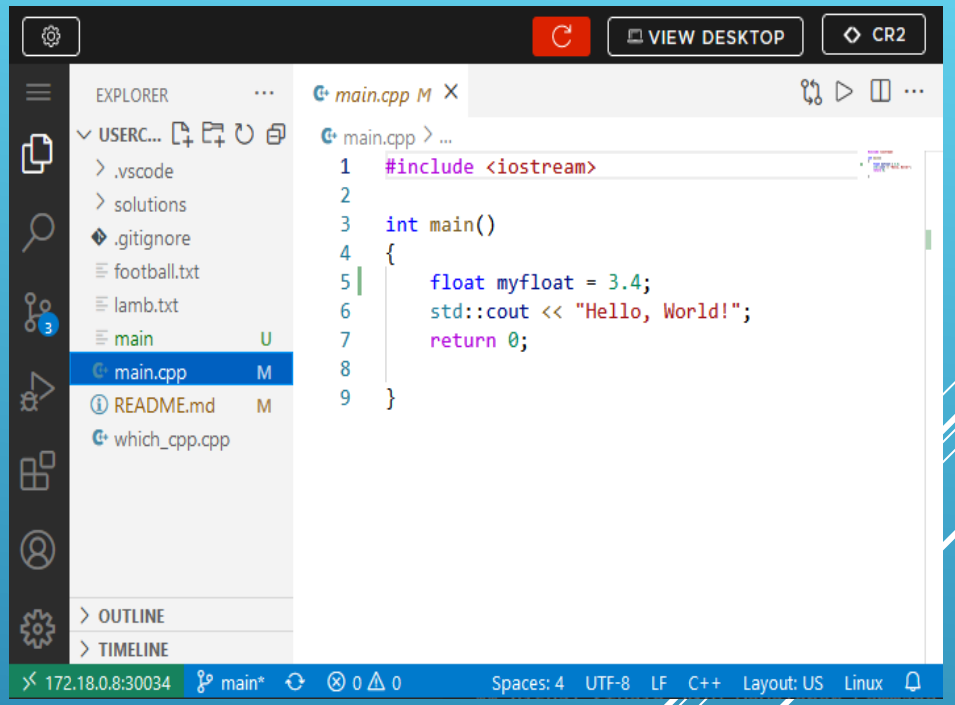
DAY 1

Today's Objectives and Topics

- Introduction to C++
- IDE and command line operations
- Your first C++ application
- Console-based I/O streams
- Manipulators
- Namespace management
- Standard library
- Keywords
- Operators
- Basic variable/data types
- Type-casting
- String manipulation with string
- Vectors
- String manipulation with regex

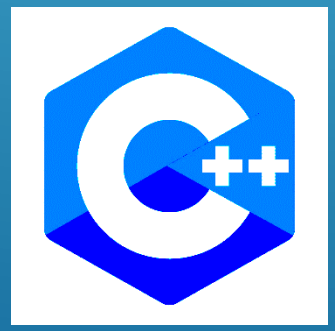
- In your IDE you may see something like one of these:



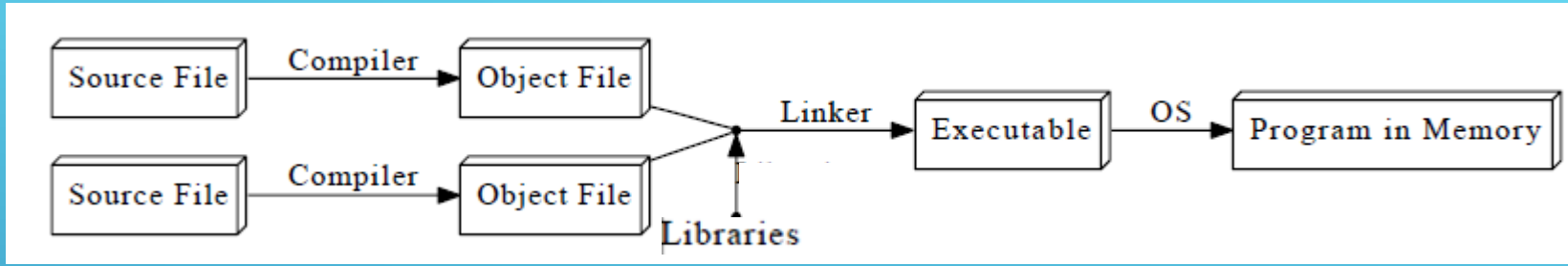


- These are code editors that sit on a fully-functional virtual computer.

- Environment setup
  - In CodeRooms:
    - rm -rf *
    - rm -rf .vscode*
    - git clone https://github.com/pbarton666/student_cpp:
    - default dir is the only persistent one
  - On your own device, you may need to replace the /.VSCode directory with your own.



IDE and command line options (discussion)




```
#  g++ is gnu's c++ compiler

$  g++ -o hello.exe hello.cpp
$  ./hello

"Hello World"
```

```
$  g++ -save-temps -o hello.exe hello.cpp
$  ls

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         11/10/2021   1:32 PM             88 hello.cpp
-a----         11/23/2021   1:26 PM         131063 hello.exe
-a----         11/23/2021   1:26 PM         539060 hello.ii
-a----         11/23/2021   1:26 PM           1839 hello.o
-a----         11/23/2021   1:26 PM           2400 hello.s
```

```
$  g++ -o hello.exe hello.cpp
```

- **Your first C++ application (demo)**

```
// hello.cpp

#include <iostream>

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

- **Flavors of C++ and what you have:**

```
// which.cpp

#include <iostream>

using std::cout;

// Reports which version of C++ you're using

int main(){
    if (__cplusplus == 201703L) cout << "C++17\n";
    else if (__cplusplus == 201402L) cout << "C++14\n";
    else if (__cplusplus == 201103L) cout << "C++11\n";
    else if (__cplusplus == 199711L) cout << "C++98\n";
    else cout << "pre-standard C++\n";
}
```

- Note: you can work with any version e.g.,
  - $ g++ -o hello.exe –std=c++14 hello.cpp

https://caiorss.github.io/C-Cpp-Notes/compiler-flags-options.html

- Console-based I/O text streams
  - Insertion operator <<<
  - extraction operator >>>
  - std::cin
  - std::cout
  - std::endl

Adding dialogue

```
// hello2_anno.cpp

#include <iostream>            // IO opps

using namespace std;           // full importation

int main()    {

    std::cout << "Hello World from hello2!  You good?" << endl;

    std::string resp = "Awesome";     // vars +  types declared

    std::cout << "I'm delighted that you're " + resp + "!";

      return 0;
}
```

- Now you try ...

```
//   file name here

//  # include libraries here (no semicolons)

#include // name_of_included_library

// define your main entrypoint main like this

int main()

// the body of the main() function goes between curly braces
{
    // declare any variables here

    // find out favorite baseball team and store in variable

    // write affirmative message e.g., GO Cubs !!

    // return 0 for success
}
```

Adding calculations

```
// hello4_ints.cpp

#include <iostream>    // part of the standard library
#include <string>      // also part of the standard library

int main()
{
    std::string resp;
    int age;
    int days;

    std::cout << "How are you  ?" << std::endl;

    //std::getline(std::cin >> std::ws, resp);  // ws=whitespace manip <string>
    std::cin << resp;
    std::cout << "I'm delighted that you're " + resp + "!\n";

    std::cout << "How old are you?  ";
    std::cin >> age;

    std::cout << "So you're " << age << "  That's "  << age * 365.25 << " days.\n";

    days = age * 365.25;
    std::cout << "Days is specified as an integer, remembered as:  " << days;
    return 0;
}
```

- Manipulators
  - use against any stream with the << operator
  - you can also set flags manually
  - ... either way, you're creating settings on the base stream class (ios_base)

```
// hello5_manipulators.cpp

#include <iostream>    // all these are in the standard library
#include <iomanip>     // where manipulators live
#include <cmath>       // where a lot of math functions live

int main()
{
     const long double pi = std::acos(-1.L);    //  long int

     std::cout << "default precision (6): " << pi << '\n';
     std::cout << "std::setprecision(10): " << std::setprecision(10) << pi << '\n';


    return 0;
}
```

- Manipulators (continued)
  - lots of options, including formatting and display of Boolean values

```
// hello6_manipulators.cpp

#include <iostream>    // all these are in the standard library
#include <iomanip>     // where manipulators live

int main()
{
     const long a_num = 132323.0;
     std::cout << std::showpos << a_num << std::endl;
     std::cout << std::hex << a_num  << std::endl;
     std::cout << std::oct << a_num  << std::endl;

     std::cout << std::endl;
     std::cout << true << std::endl;
     std::cout << std::boolalpha << true << std::endl;

    return 0;
}
```

https://www.cplusplus.com/reference/library/manipulators/

Manipulators (continued)

```
// hello7_manipulators.cpp

#include <iostream>
#include <iomanip>

int main() {
    std::cout << std::left << std::setw(12) << "Univ. Mich"
    << std::right << std::setw(3) << 42
    << '\n';

    std::cout << std::left << std::setw(12) << "Ohio State"
    << std::right << std::setw(3)
    << 27
    << '\n';
}
```

Now you try - Part 1

- Ask the user how big the largest-ever fish they ever caught was.
- Set a variable in your code to hold that value.
- Report back to the user 75% of the stated length of the fish.
- (snarky comment is optional) .
- return 0 if successful.

```
//   file name here as a comment

//  # include libraries here (no semicolons)

#include // library name(s), one per line

int main(){

// the body of the main() function goes between curly braces

}
```

- Now you try -Part 2

- See if you can extend the 'football scores' code in hello7_manipulators.cpp to include:
  - a header row
  - extra columns for each quarters points

Output might look like this:

```
               1  2  3  4  T
Univ. Mich     7  7 14 14 42
Ohio State     3 10  0 14 27
```

- Standard library
  - General term for core tools
  - Namespace is “std”
  - Other libraries extend the language, but are not core
    - https://en.wikipedia.org/wiki/C++_Standard_Library
  - Incorporates C standard library w/ mangled names
    - <math.h> --> <cmath>
    - https://en.wikipedia.org/wiki/C_standard_library
  - We can access elements with statements like:
    - #include <iostream>
    - using std::cout;
  - Examples:

<array>	<iterator>
<deque>	<chrono>
<list>	<string>
<map>	<iostream>
<vector>	<valarray>

- Keywords
  - Reserved – can't use for anything else
  - Most are data types and control statements

auto	catch	for	private	template	sizeof	asm
double	throw	while	protected	class		new
const	try	return	public			delete
int		do	extern			
long		else	virtual			inline
unsigned		break				
float		continue				this
enum		if				
char		goto				
short		case				typedef
signed		switch				
void		default				volatile
static						
struct						register
union						

Some important operators

Math	Compare	Boolean	Bitwise
a + b	a == b	!a (not works)	~a
a – b	a != b	a && b (and works)	a & b (bitand works)
+a	a > b	a || b (or works)	a | b (bitor works)
-a	a < b		a ^ b (xor works)
a * b	a >= b		a << b
a / b	a <= b		a >> b
a % b	a <=> b		
++a			
a++			
--a			
a--			

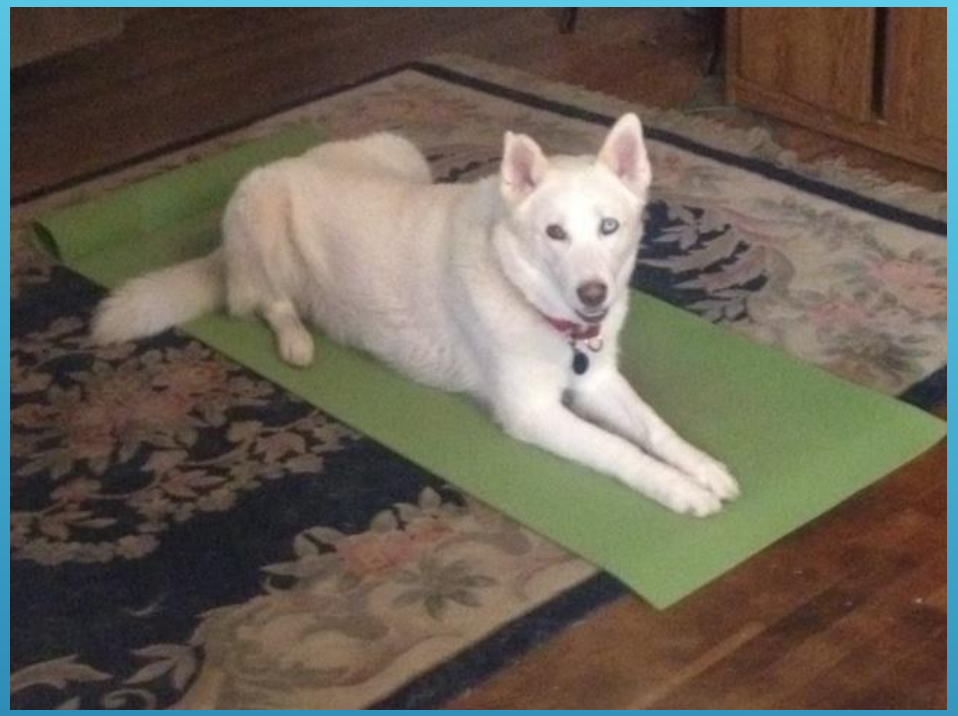
Precedence

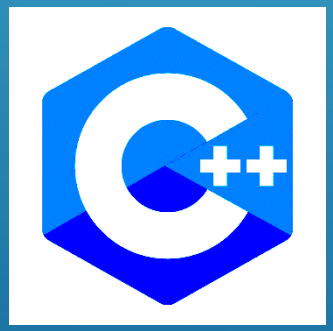
Operator precedence

Complicated. 15 groups.

Generally: scoping, (type assignment, postfix, function calls),(logical, prefix, pointers), (mult and div), (add and subtract)

https://docs.microsoft.com/en-us/cpp/cpp/cpp-built-in-operators-precedence-and-associativity?view=msvc-170




C++

- **Basic variable/data types**

Primitive Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

- Modifiers:
  - signed
  - unsigned
  - short
  - long
  - const
  - volatile
  - static (initialized once; stored in static area)

Size and Range

Type	Typical Bit Width	Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 4,294,967,295
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	Depends on your system
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character

- Discover size (system dependent, sometimes)
  - sizeof()
  - numeric_limits

```
// vars_size.cpp
#include <limits>
using namespace std;

int main(){

  // Query for the size of a specific type
  float my_float;
  cout << "A float is " << sizeof(float) << " bytes.\n";

  // How many digits w/o loss of precision?
  cout << "A float can contain "
       << numeric_limits<float>::digits
        << " digits. \n";

    // What's the largest?
    cout << "Max value: "
         << numeric_limits<float>::max()
         << ". \n";

    // What's the smallest?
    cout << "Min value: "
         << numeric_limits<float>::min()
         << ". \n";
}
```

Creating variables

- must be declared and specified
- both may happen at once:
  - #define PI 3.14 // no ; value stored extremally
  - const float = E 2.718; // value stored in app
- declaration and specification can be separate
  - int dog_years;
  - dog_years=7;
- multiple declarations possible (not necessarily good)
  - int i, j, k;
- data can be specified in terms of literals
  - 120 //base 10 int
  - 033 //base 8 int (octal)
  - 0x3 //base 16 int (hex)
  - 0b11 //baes 2 int (binary)
  - 30u //unsigned int
  - 30L // long
  - 30ul // unsigned long

Type-casting (changing variable type)

- implicit (“standard conversion”)
- explicit
- special functions like std::stoi (string→integers), std::stof (string-> float), std::stod (string-> double),
- dynamic_cast (used with pointers, ensures completeness)
- static_cast (used with pointers between related classes, buyer beware)
- reinterpret_cast (low-level, binary copy or object passed to a new pointer)
- const_cast (make a constant a variable or vice-versa.

Miscellaneous

- typedef allows an alias to a var type
  - typedef int alias_to_int;
- Templates provide a standard set of container variable types via the STL

String manipulation with the string object

- size
- length
- append
- compare
- insert
- find
- replace
- concatenate with the + operator

Vectors

- Ordered sequence of objects
- Part of the STL
- #include <vector>
- methods for loading data, indexing, inserting, removing, etc.
- “self-aware” of key metadata like size
- iterable

Vectors cheat sheet

push_back() adds values to end

pop_back() removes values from end

insert() just what you think

front first element

back last element

access operator is [ ] for individual elements

iterators include begin, end

size() for number of elements

empty() Boolean answer to: is it empty?

https://www.cplusplus.com/reference/vector/vector/

Your turn!

- Please write a program that:
  - creates a empty string vector named parsed
  - adds the first four words of a string, one at a time, to the vector.

```
// starter code

int main(){

  lorem = "Lorem ipsum dolor sit amet consectetur “

  for (int i=0; i<5; i++){
     // your parsing code goes here
  }
  for (int i=0; i<4; i++){
     cout << parsed[i] << ‘\t’;
     }
}
```

- String manipulation with the regex library
  - #include <regex>;
  - regex() to create a regex object
  - regex_search() to search a string
  - regex_replace to find matching position
  - smatch creates a match object
- Requires the use of a “magic decoder ring” built from regex tokens – that’s the regex.
- Examples:
  - [a-z] one lowercase letter
  - [0-9] any digit
  - [a|b] a or b
- Use cases: replace sew, awk, etc. command line tools; validate inputs; evaluate text documents.

- More patterns:
  - any whitespace “[\t\r\n\f]”
  - any non-whitespace “[^ \t\r\n\f]”
  - any character “.”
  - 0 or more X “X*:
  - 1 or more X “X+”
  - 0 or 1 X “X?”
  - exactly 3 X “{3}”
  - at least 3 X “X{3, }“
  - 3 to 5 X “X{3, 5}”
  - X at beginning “^X”
  - X at end “X$”
  - multiline capture regex(“^.*$”, regex::multiline)
  - numbers between 3 and 5 “[3-5]”

Now you try:

- Create a program that accepts user input
- Validate the input as numeric
- Report to the user the value and twice that value.
- Be sure you can validate integers, floating points, positive and negative numbers

★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

Introduction to C++

Day 2

13-17 December 2021

Pat Barton.
President/CEO, PES LLC

A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 062419 egd

DAY 2

Today's Objectives and Topics

- Flow control with if / then / else
- Flow control with switch
- For statement and basic looping
- Looping with while statements
- File system I/O
- Encapsulating code with functions
- Declarations and definitions
- Scope
- Encapsulating data with containers
- More about vectors
- Arrays
- Lists
- Maps

if statement

- basic

```
if (condition) <code here>;
if (condition){
        //code here
}
```

- upgrades
  - if .. else
  - if .. else if .. else if ..
  - if .. else if .. else
  - nesting possible

switch statement

```
switch (int/char condition){
    case <matcher>:
        // code here
        break;
    default:
        // code for 'none of the above;
```

- **conditional expressions (one-liners)**

```
(condition) ? action_if_true : action_if_false ;
```

for statement

```
for ( <loop index init >; < index check>;
<update index>) {
    //    <do stuff>
}
```

- break and continue can intervene
- There are also “range for” and “for_each” options that work on certain collections
- Convenient with a finite number of elements

- Looping with while statements
- Two flavors
  - do .. while

```
do {
        <do this stuff>
    }  while ( <condition is true> )
```

  - while

```
while <condition is true>{
    <do this stuff>
}
```

- Both forms support break and continue
- Convenient with indeterminate number of elements, polling loops, etc.

Your turn:

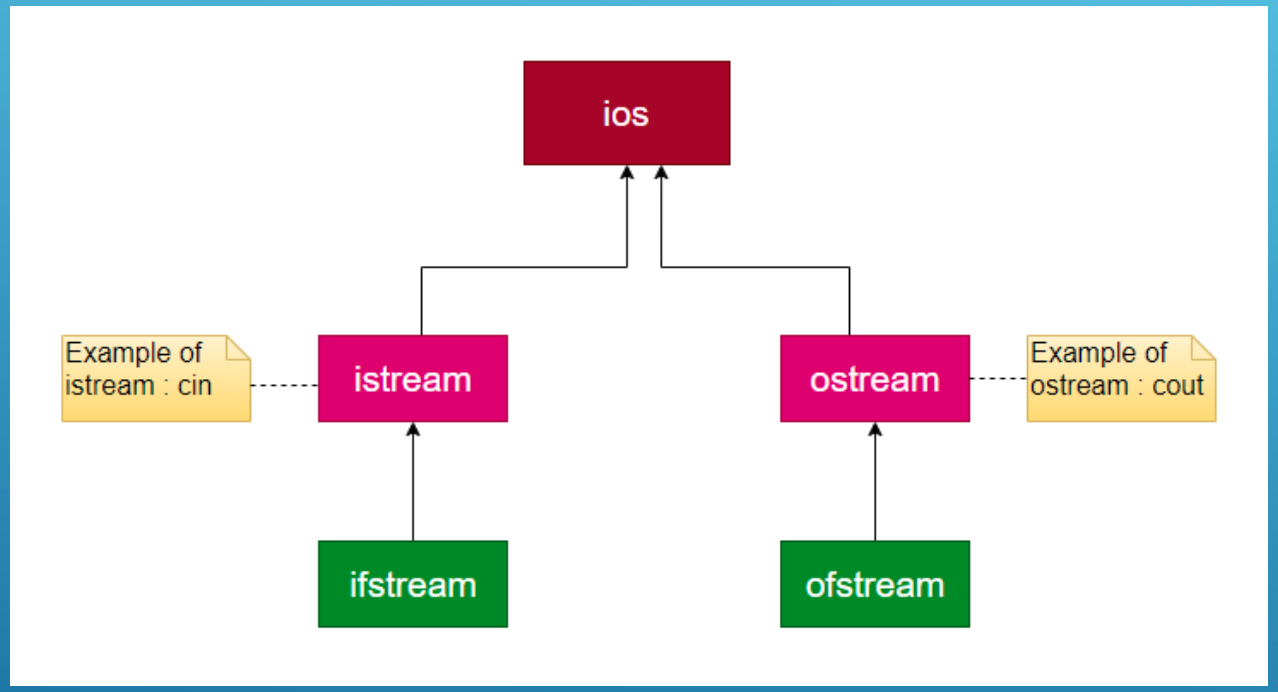
Please write a program that solicits a user's guess of a random number integer. As the user types in their guess, provide feedback on whether they are too high, too low, or spot-on.

When they succeed, offer hearty congratulations and terminate the program.

Hint: Here's how you might get a random number:

```
number = rand() % 1000 + 1;  // a random number
```

... which you probably want to print out or override during development. That should help keep the developer sane ;-)



File system I/O

- #include <fstream>
  - (ofstream(), ifstream(), fstream()
- file streams are the streams
- streams are objects that manage flows of serialized data
- streams can be opened, closed, and filled
- streams include buffers
- buffers may be flushed (but cout() automates that)

Writing to a file:

- Create a stream object
  - <stream> <var_name> (<file name>);
    - std::ofstream mystream (“my_file_name.txt”);
- Check that it worked:
  - mystream.is_open()
- Insertion operator injects content
  - mystream << “Some content”” << std::endl;
- Flush buffer if needed
  - mystream << std::flush;
- Close the stream
  - mystream.close();

Reading from a file:

- Create a stream object
  - <stream> <var_name> (<file name>);
    - std::ofstream mystream (“my_file_name.txt”);
- Check that it worked:
  - mystream.is_open()
- Extract some content
  - content = getline(mystream, line);
  - cout << content;
- Close the stream
  - mystream.close();

Read and write using file stream pointers:

- stream.seekp() pointer for writing (‘put’)
- stream.seekg() pointer for reading (‘get’)
- stream.tellp() location of put pointer
- stream.tellg() logation of get pointer

Your turn !

Please write a program that reads the file football.txt (it's recent AP football standings) and print out all the schools with "State" in their name.

You should get something like this:

5	Oklahoma State	1291	7	11-1
7	Ohio State	1147	2	10-2
11	Michigan State	877	12	10-2
19	San Diego State	416	22	11-1
21	NC State	310	24	9-3

Encapsulating code with functions

- Functions are isolated namespaces with specific entry points
- Functions may take arguments
- Arguments may provide ‘hard coded’ values (“by value”) or pointers (“by reference”)
- Functions may (or may not) return objects
- Why are they cool?
  - they do one thing, well
  - testable
  - easily replaced
  - streamline main code
  - “clean”

Function-related topics

- Syntax
- Declarations versus Definitions
- Scope
- Passing command line args to main()

Function syntax:

<return_type> <function name> (<comma-separated args>) {

<some stuff>

}

```
// A simple function

int addInts(int arg1, int arg2)
{
        int sum = arg1 + arg2;   // isolated variable

        return sum;  // not needed if return type is void
}
```

Function Declaration versus Definition:

```
// addInts() is defined before use in main()

int addInts(int arg1, int arg2){
        int sum = arg1 + arg2;
        return sum;  // not needed if return type is void
}


int multInts(int, int)  // multInts() is DECLARED here

int main(){
    addInts(1, 2);
    multInts(1, 2);
}

// multInts() is DEFINED here

int multInts(int arg1, int arg2){
    return arg1 * arg2;
}
```

Passing data into functions

- By value
  - (usually) the default, exception is arrays
  - passes a new object – a hard copy – to function
  - function operates in isolation
  - original is safe, but at a cost
- By reference
  - pass the address of the object
  - no new object is created
  - values at the address are directly manipulated by function
  - original is changed

Passing command line args to main()

argc passed in by value

argv passed in by reference

- argc is argument count (args + program name)
- argv is the argument vector (an array of char pointers)
- argv[0] is the name of the program

```
int main(int argc, char** argv) {
    cout << "Number of args: " << argc<<endl;

    for (int i = 0; i < argc; ++i)
        cout << "argv[" <<i<< "] : "<<argv[i] << "\n";

    //argv[0] is name of program
    //argv[1] is the first arg provided at command line
    //argv[2] is the second arg ... (etc.)

    return 0;
}
```

Scope

- Global
  - external to functions, classes
  - available within functions, classes
  - can be changed from within functions, classes
- Local
  - internal to functions, classes, etc.
  - variables declared within: while, for, if, and switch are local to block
  - sort of a “private copy” of the value
- Namespaces
  - lookup tables matching names to values
  - can be nested
  - use the :: (scope resolution) operator to drill down through nested layers

```
// declare functions here
// define these functions
  main
      countArgs
      pickOperations
          doModulo
          doMult
```

Your turn:

Here, the 'ask' is that you write a modular program using short, single-purpose functions, working in concert. Please:

- Capture the user's input in the main() function. Input should have two integers followed by the word mod or mul ... something like:
  - ./myprogram 2 3 mul OR ./myprobram 2 4 mod
- In main() call function countArgs() with all the user inputs. countArgs():
  - returns true if 3 args were provided
  - yells at the user and exits if exactly 3 arguments were not provided
  - exit(2);
- If countArgs() returns true, main() should call the function pickOperation() , passing along all the user input as arguments.
- pickOperations():
  - calls function doModulus() if 'mod' is specified; or calls function doMult() if 'mul' is specified;
  - yells at the user and exits if the operation is neither mod nor mul
  - returns the results of doModulus() or doMult to the calling function.
- functions doModulus() and doMult() take two input arguments and return the required result to the calling function.

Possible user experience:

```
$ ./myprogram 3 4 mul
result is 12
```

```
$ ./myprogram 3 4 gak!!
Dude, I don't do gak.
```

Containers

- C++ has several container objects
- Most (by default) can be passed by value.
- Many are part of the C++ Standard Template Library (STL)
- The STL provides compile-time polymorphism

Important container objects

- Sequences
  - vector
  - array
  - list, forward_list
  - deque (stack for LIFO, queue for FIFO, priority_queue)
- Dictionary-like (associative)
  - set (multiset for non-unique)
  - map (multimap)

What container to use:

- vector: easiest sequential container, generally
- array: if the size is fixed and you want speed
- deque: if you usually add elements at either end
- list: if you usually add elements in the middle (no random access, though)
- string: if you just have characters.
- map: best for key:value pairs if order matters
- unordered map: more efficient, but random

- Memory nuances:
  - contiguous: vector, array, string
  - chunks: deque
  - nodes: list
    - every element knows its value and where the next is

More about vectors

- All-purpose container for homogenous data
- Flexible, mutable
- Contain data and metadata (size, etc.)
- Part of the STL, so generic constructors possible
- We can use a template class to report values
- Support their own version of iterators
- Extensible to higher dimensions:
  - #include <vector>
  - vector<int> my_vector; //1-d integer vector
  - vector< vector<int> > my2dv; // 2-d vector of integer vectors
- Index operations scale thusly:
  - myvector[0] my2dv[1][1]

Your turn ...

- Please use the vector object to create the classic Pascal's Triangle. You'll want to use:
  - a 2-d array of integer vectors (one per row)
  - push_back() and size()
- Your output might look like one of these:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

```
     1
    1  1
   1  2  1
  1  3  3  1
 1  4  6  4  1
1  5  10  10  5  1
1  6  15  20  15  6  1
```

Arrays

- Two types:
  - `std::array<int, 2> myarr {3, 4};  //STD`
  - `int cstyle_arr[2] = {3, 4);  // C-style`
- Both for sequential data of fixed size
- std::array much easier to wrangle
- Both widely used
- searchable
- sortable

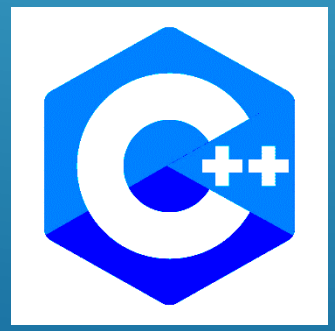
https://en.cppreference.com/w/cpp/container/array

Lists

- elements stored non-contiguously
- resizable on the fly
- two flavors: “regular” and forward_list (latter only iterates forward)
- ordered containers
- searchable
- insertions can be made anywhere
- iterable
- sortable
- part of the STL
- “go to” object if additions likely will be made in the middle

Now you're at bat.

- Please write a program that:
  - Creates a list with the integers 0 .. 9
  - Adds 10 to each element
  - Prints the changed list to the console


C++

Maps (dictionaries)

- key:value pairs
- one-way lookups
- keys and values separately declared
- flexibly-sized
- Templatized
- immutable keys
- unordered or ordered
- may be nested – good for settings

Try it out for yourself

- Populate a map with the first few letters of the NATO alphabet (alpha, bravo, charlie ...)
- Keys should be letters e.g., “a”, values should be the pronunciation e.g., “alpha”.
- Create a second map where the keys and values are reversed e.g., the key of “alpha” should return the value of “a”.

Your solution might look like this:

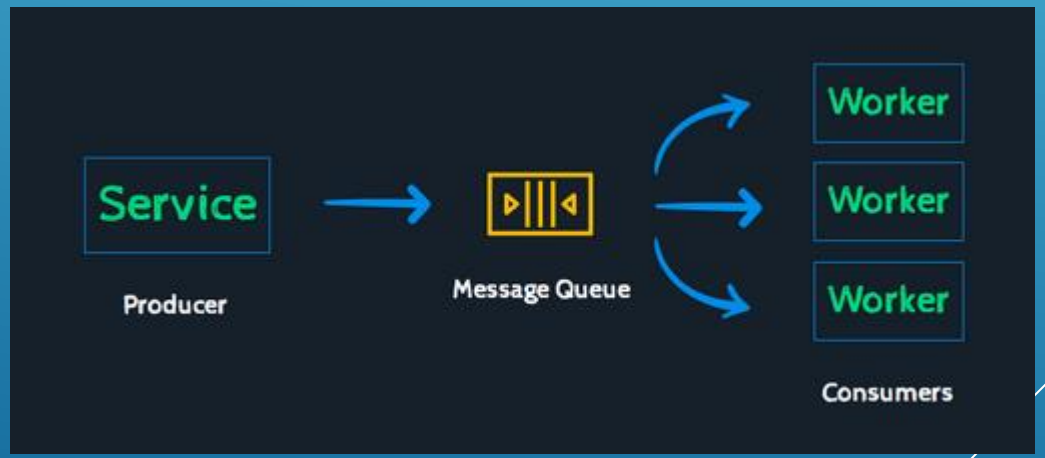
bravo:b

alpha:a

charlie:c

Queues (queue and deque)

- Sequences of like items
- Flexible – expand as needed
- Items can be added or removed from one end (queue) or both ends (deque)
- Work like vectors using the STL
- Use case: message queues e.g., Celery that might have traffic simulated by using a Poisson distribution.



Cue up a queue exercise:

- Create a deque and initialize it 5 elements of 201..205
- Create a second queue
- Assign the second queue the 3 middle elements of the first queue
- Create a third queue and populate it with three elements of an integer array

Your output might look like this:

201 202 203 204 205

Size of first: 5

202 203 204

Size of second 3

2000 2 29

Size of third: 3

Introduction to Data Structures

- "Roll your own" data type
- Contain composites of other data types
- Known to C++ as the struct data type
- Use like it any other sort of data
- Access internal elements with "dot notation"
- Some data structures e.g., complex numbers, already available. But here's how you make your own:

```
// Create a general data type called complexNumber

typedef struct _complexNumber {
    float real;
    float imag;
} complexNumber;
```

This unit will be a bit “complex”

- Learn about the built-in complex number object
  - Use a special library for the imaginary object
  - Work with polymorphic operators
- Learn how to create our own structure
  - Define a custom structure
  - Build a constructor
  - Create custom methods to support it
  - Create a human-friendly reporting function
  - Test custom methods
- And in the process …
  - Work with templates
  - Introduce the assert statement

OK ... your turn! Starting with this code ...

solution_complex_struct_starter_kit.cpp

... please create a method that will multiply two complex numbers. Test your work.

Hint, you can use the FOIL method.

Introduction to Templates

- We've used them already e.g.,
  - vector<int>> my_vector;
  - unordered_map<string, string> nato;
- Provide a way to make generic functions. The templated vector and unordered_map classes are agnostic to variable type...
- ... until we actually use them.
- We can make the completely or selectively generic)
- This is only specified upon compilation (when the compiler figures how it will be called)
- Generic functions and classes supported

```
// General form

template <typename some_label>  return_type  function_name (parameters) { // some_code }
```

You're up. A bit more on your (tem)plate

- Please write a program that creates the function tres_func() that takes three arguments of any type …
- … that also supplies specific templates to handle the following signatures:
  - tres_func(int, float, float)
  - tres_func(float, float, float)
- … and tests to ensure that all perform as expected.
- Hint: you might set the function up to return a string that reflects the input types e.g., "fff" or "iff".

★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

Introduction to C++

Day 3

13-17 December 2021

Pat Barton.
President/CEO, PES LLC

A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 062419 egd

DAY 3

Today's Objectives and Topics

- Queues
- Data structures
- Introduction to templates
- Introduction to pointers
- Array management and navigation
- Handling user input arrays
- Exceptions

Introduction to pointers

- Pointers are ordinary variables with a little built-in magic. Like ordinary variables:
  - have names
  - hold values
  - have variable type
  - can be elements of containers like arrays
- The magic:
  - know how to retrieve the value at the address
  - know how to stride through memory
  - allow unequivocal reference to any amount of data stored starting at the address contained.

Array navigation with pointers

- Arrays and pointers are almost the same thing
- Pointers can be used to traverse an array
- Pointers can be used to reference and change the value in memory.

Now, you're on point. Can you please write separate programs that:

- Create a function that accepts three pointers as arguments. Two of the pointers refer to integers, the last refers to the sum. The function returns nothing but adds two numbers.
- Create a function that accepts two pointers as arguments, each referring to an integer. The function returns nothing but swaps the two numbers.
- Create a five-element integer array. Using only pointer arithmetic, calculate the sum of the array.

Exceptions

- Gracefully handle problems with your code or inputs.
- Syntax:
- try{
  - // some protected code
  - throw() // what to do if there's a problem
  - }
- catch() {
  - // just like any function

★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

Introduction to C++

Day 4

13-17 December 2021

Pat Barton.
President/CEO, PES LLC

A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 062419 egd

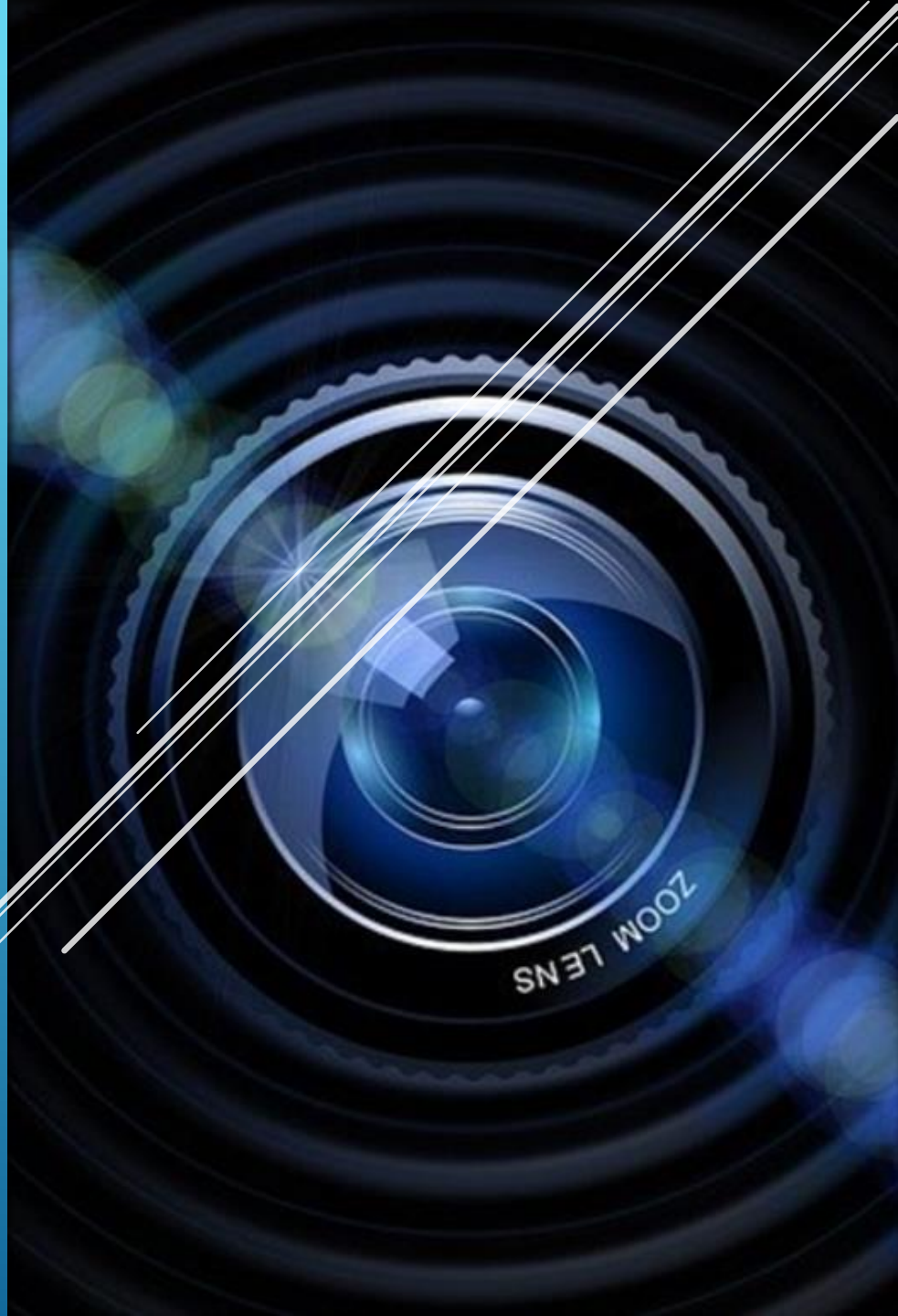
DAY 4

Today's Objectives and Topics

- Classes
- Functions to class methods
- Variables to class members
- Declarations, definitions, instances
- Constructors
- Access specifiers
- Manage instance namespaces

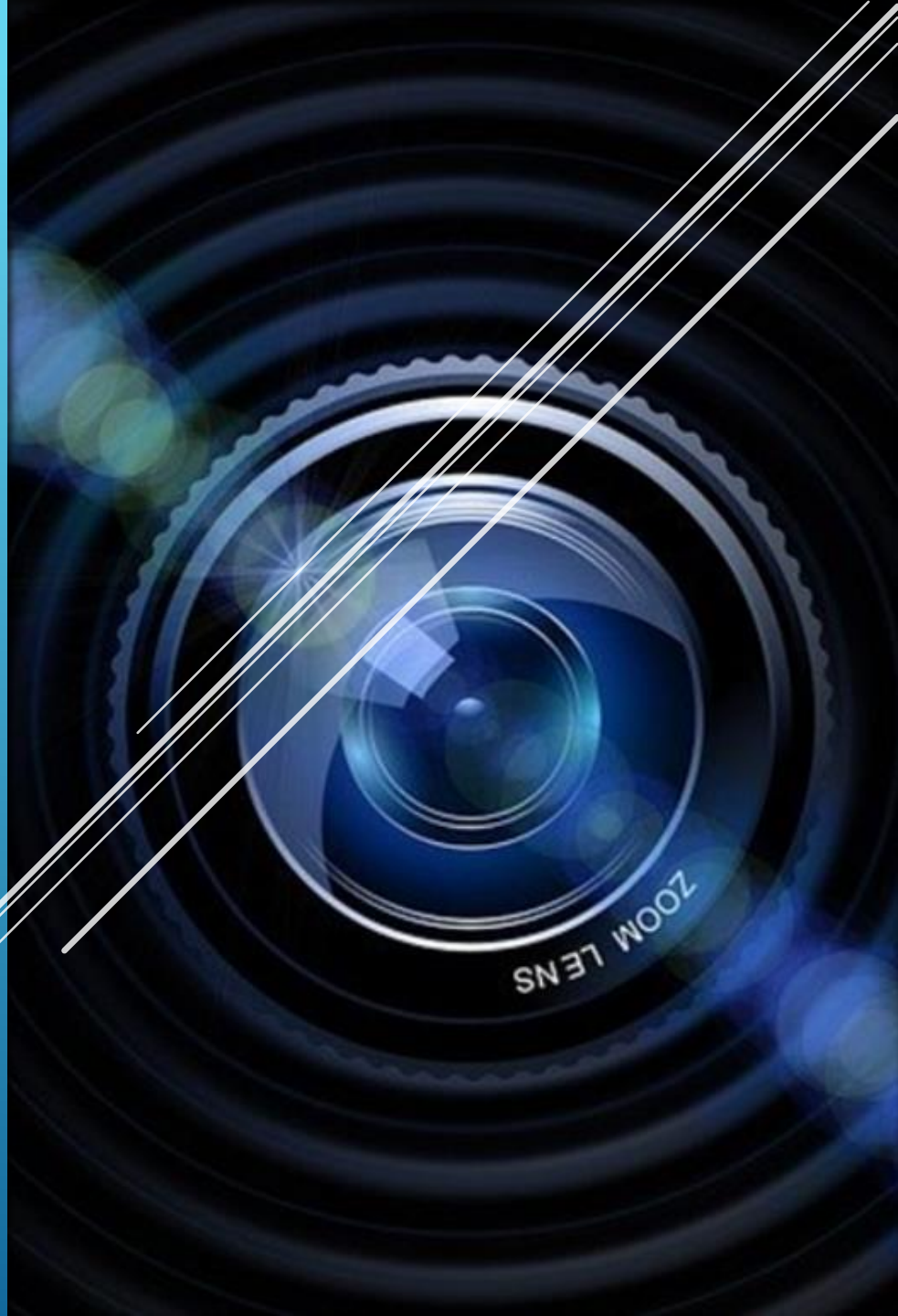
Introduction to classes

- Classes are another way to gather and encapsulate functionality.
- They have properties (nouns) and methods (verbs). Methods are functions that belong to a single class.
- Methods are used to provide related functionality.
- Properties are used to store variables in the class scope (each class has its own namespace).
- You can think of a class object as a general template. When you use it to make a specific thing, you make an "instance".



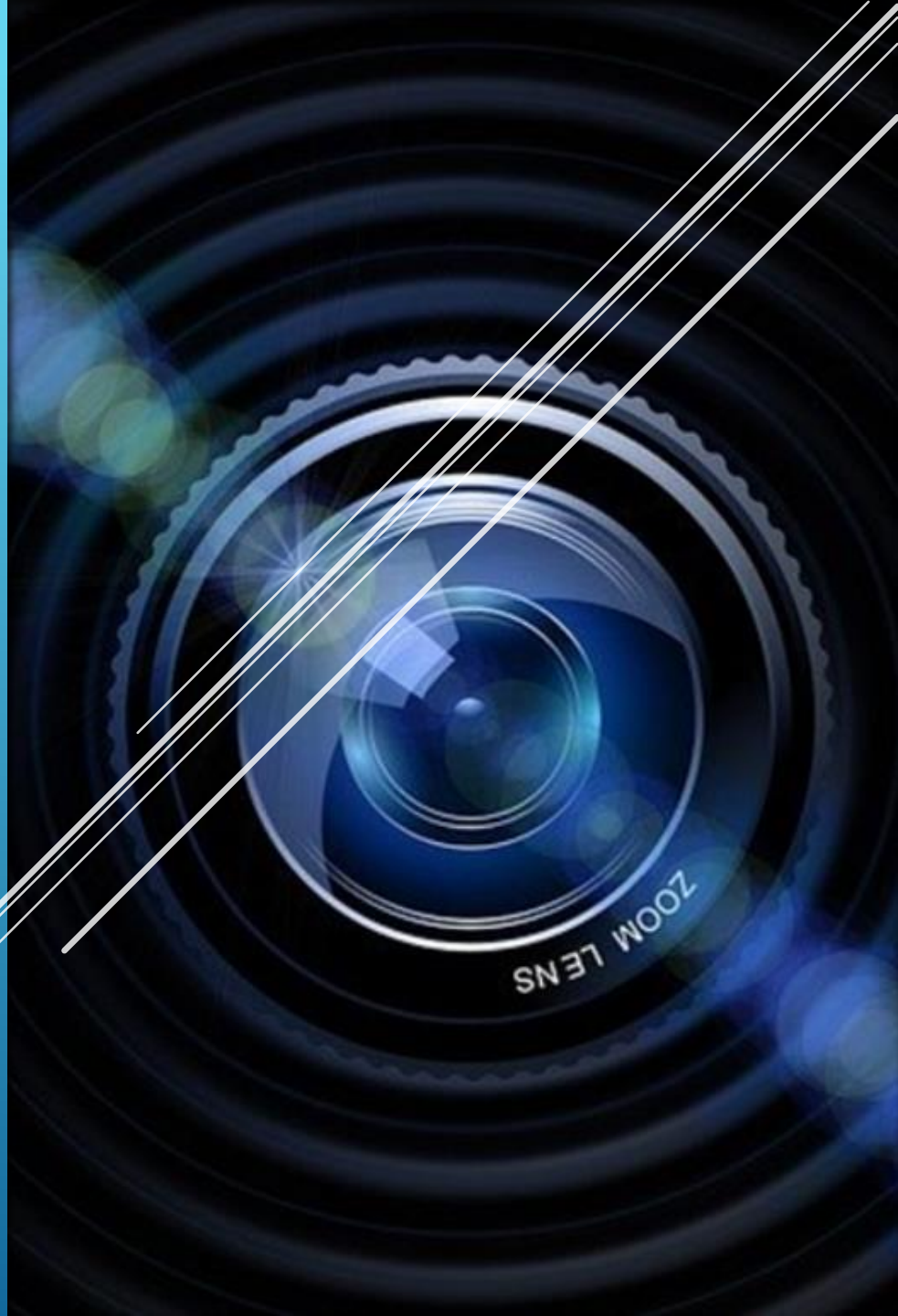
Functions -> class methods

- Class methods are internal functions.
- In classes these can be private or public
- Note that classes still have access to functions declared or included in the scope of the program file itself.



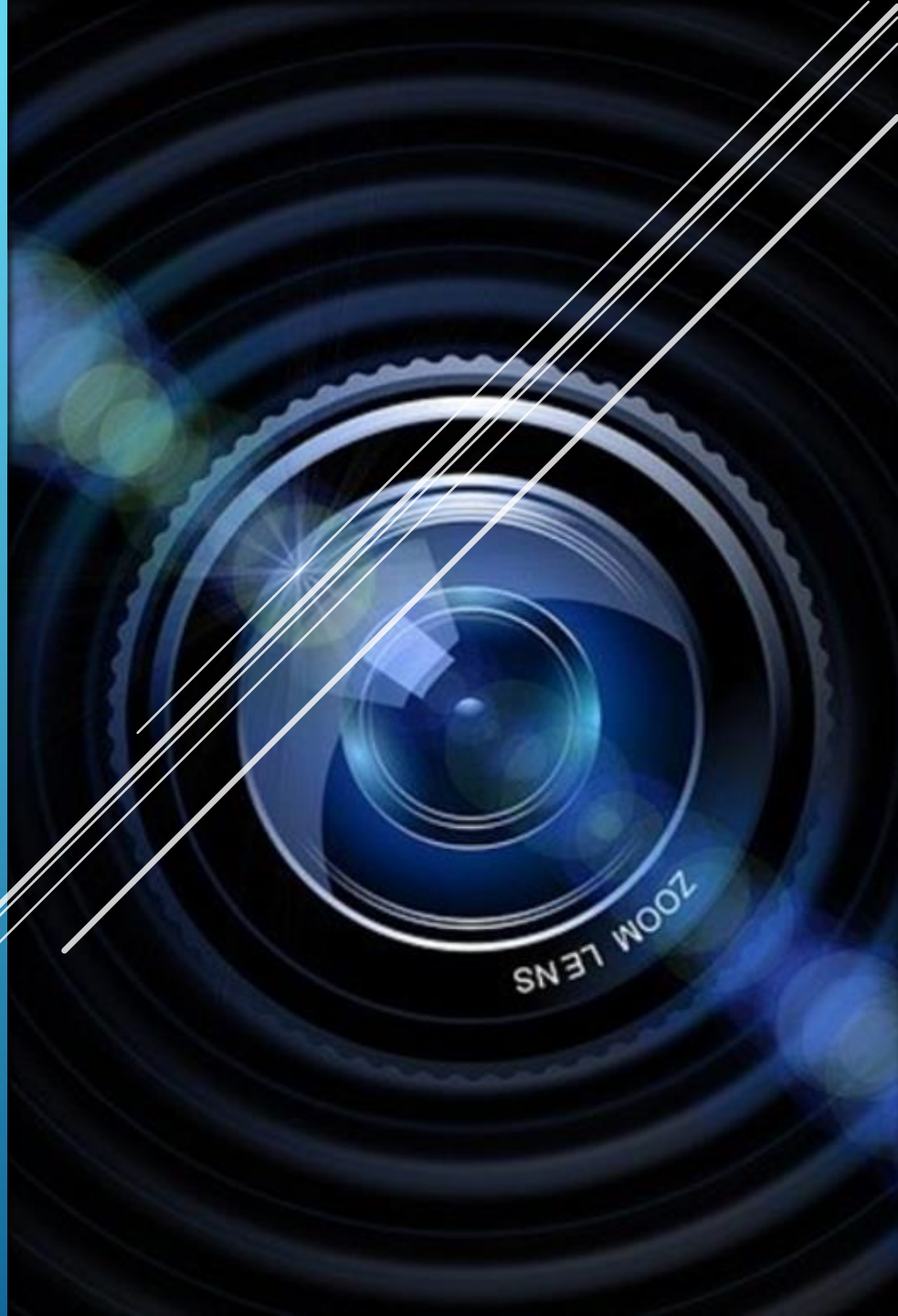
Variables -> class members

- Ordinary variables created in class belong to the class.
- These may be public or private
- Classes have access to variables exposed to the global namespace.



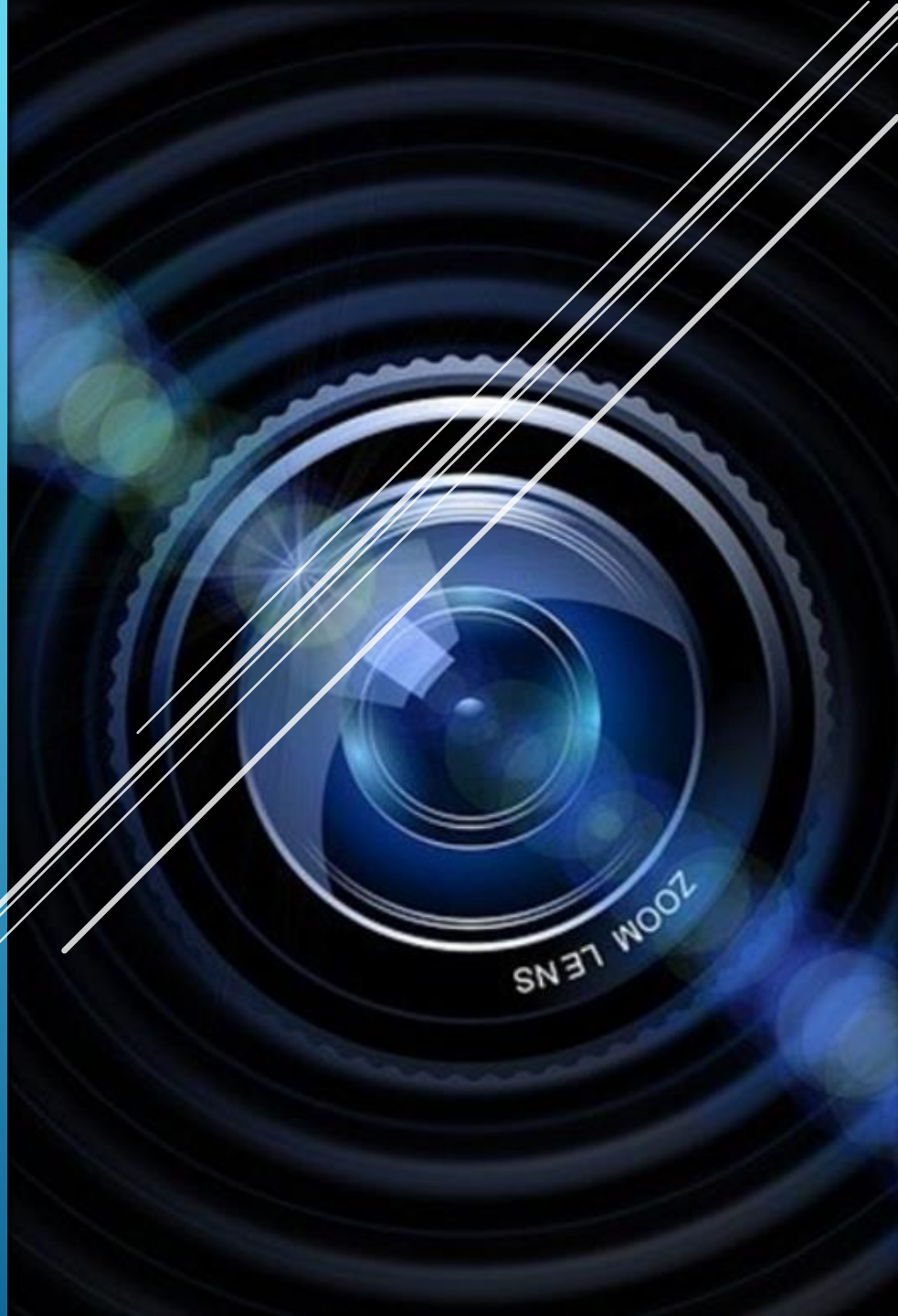
Class declarations vs. definitions vs. instances

- Declaration – adds name to namespace only:
  - class Dog;
- Definition – this actually allocates memory and will typically include a constructor.
  - class Dog{};
- Instantiation – this creates an actual thing
  - Dog myDog;



Constructors

- When you create an instance of a class, what do you need to provide? What does the class have to set up?
- The class might have input requirements – these have the same rules as functions
  - can have default values
  - provided as positional variables
  - have types
- If there's an attempt to build a class instance without meeting the constructor's rules, the operation will fail.



Access specifiers

- Manage visibility among instances and to unrelated processes.
- Declared as codeblocks offset thuly:

```
class MyClass{
    private:
        //private properties and methods here
    public:
        //public properties and methods here
    protected
        //private properties and methods here
        //  ... except when inherited
```

Managing instance namespaces

- Common practice to only use private members for important work.
- Can't be changed from the outside, but internal methods work with them just fine.
- So ... we can have externally-facing methods that filter potential incoming information then change values ("setters")
- .... and externally-facing methods that provide information back "getters".

Now it's your day in the sum.

- Please write a class that:
  - has a private integer vector for relative humidity values; and a private integer variable.
  - a public setRh method that takes a string argument ...
  - ... and rejects invalid values (good values can be converted to integers between 0 and 100).
  - ... and stores good values in the vector
  - Hint: use stoi to convert string-> integer like this:
    - int (stoi (string_argument))
  - Hint: steal code from one of the vector sessions to print out progress.
  - Hint: you can use a try/catch block to weed out strings that won't convert to integers.

★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

Introduction to C++

Day 5

13-17 December 2021

Pat Barton.
President/CEO, PES LLC

A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 062419 egd

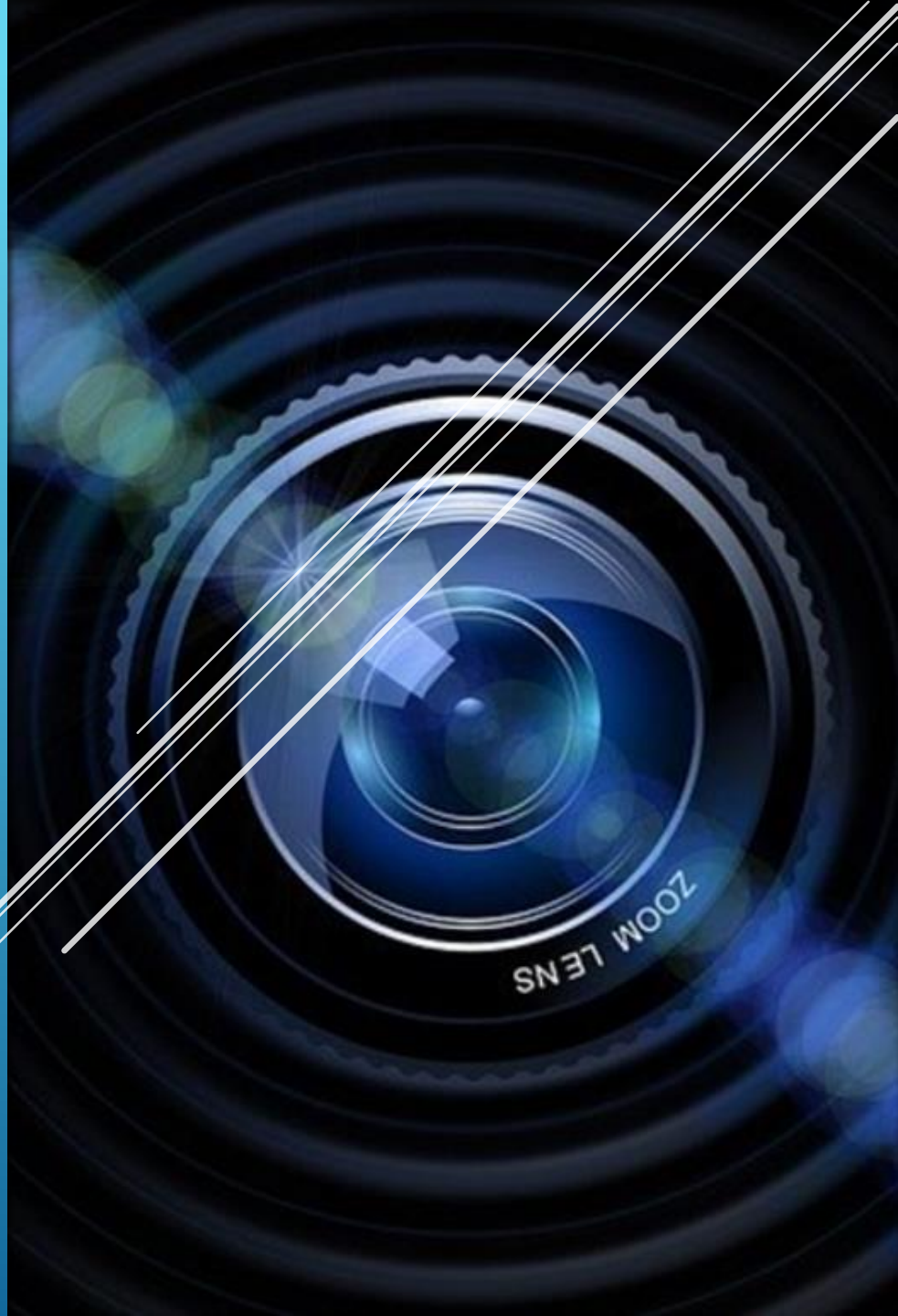
DAY 5

Today's Objectives and Topics

- Inheritance
- Multiple inheritance
- Virtual classes
- External class libraries
- Constructors and destructors
- More on pointers
  - Arithmetic with pointers
  - Double pointers and arrays
  - "By reference" versus "By value" ops
- Testing and test-driven development
- Assertions

Inheritance

- One of the three pillars of OOP (with encapsulation and polymorphism)
- DRY – create dendritic, hierarchy of increasingly specific classes with no repeating code
- Class methods and attributes are both heritable
- Multiple inheritance (more than one parent) is possible.



Inheritance Example:

- Mammal
  - warm blooded and live young
- Dog
  - Mammal + 4 legs, loyal, has name
- Husky
  - Dog + sings blues

A specific creature can be built from this hierarchal template.

My husky is named Quinn, can sing blues, has 4 legs, is loyal, warm-blooded, and can bear live young.

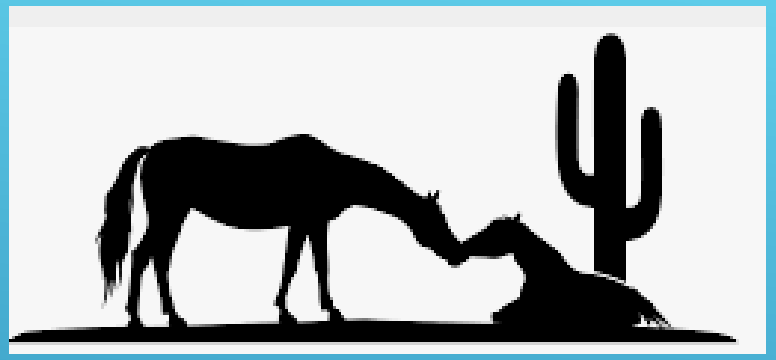
Your turn: warm and fuzzy Dog classes

Please write a program that extends the mammalian classes we've explored:

- add a reportFood() method to the DogClass
- add a reportFood() method to the HuskyClass
- create a new SiberianHuskyClass
- add a reportColor() method to the SiberianHuskyClass

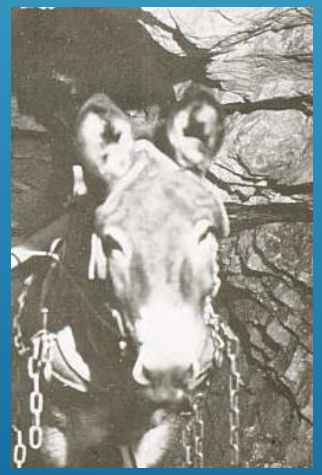
... and demonstrate that it works as expected.

Multiple Inheritance and Virtual Classes



equine

donkey



mule

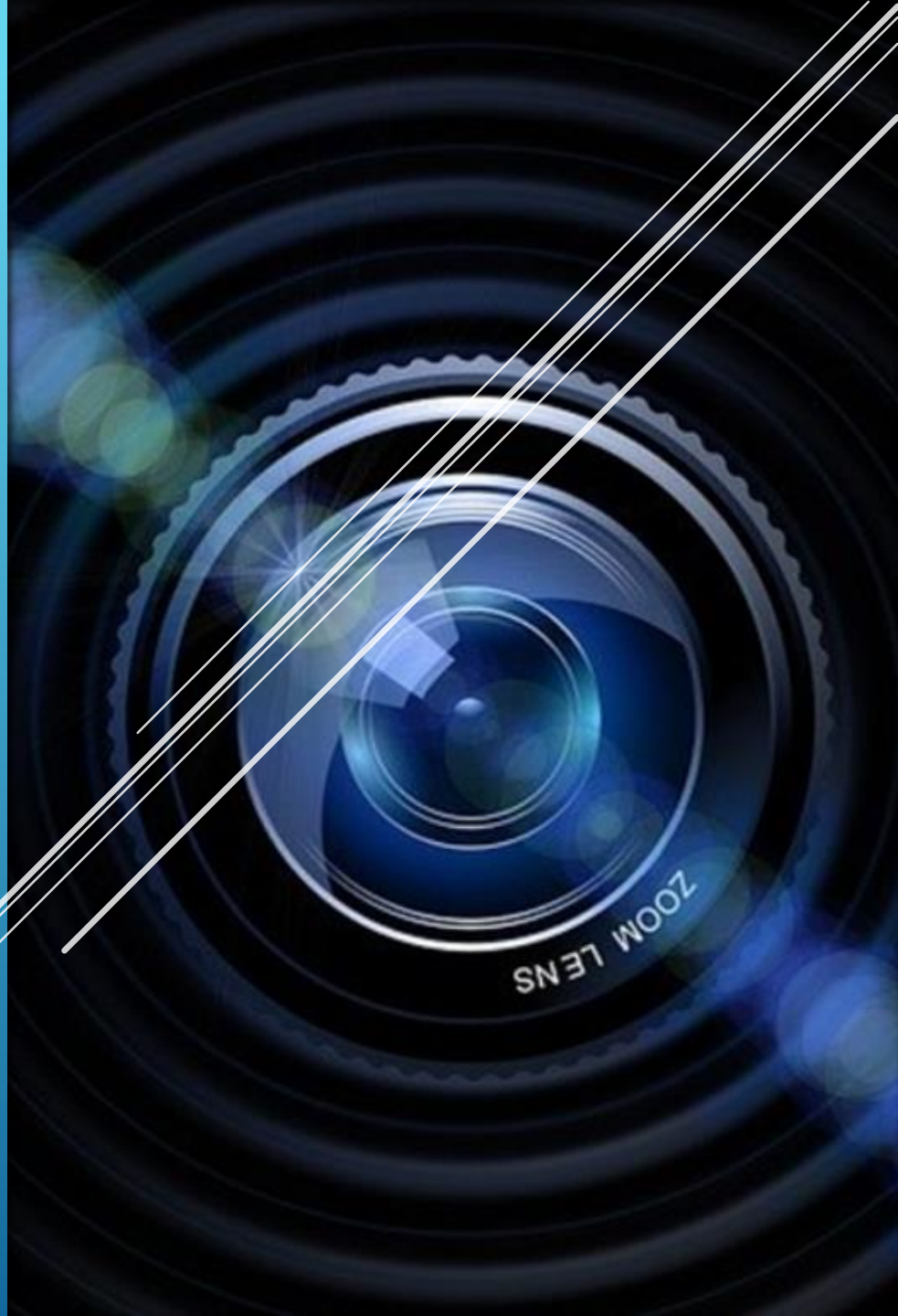
horse

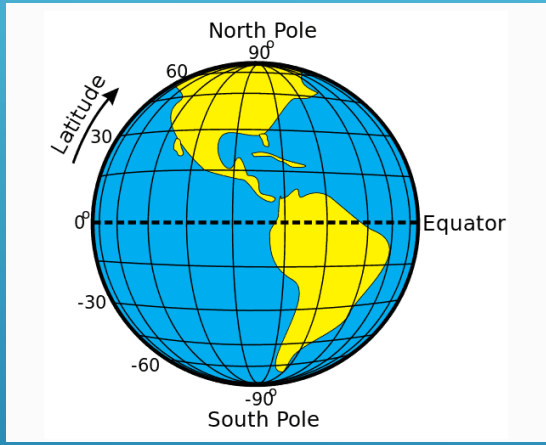
Multiple Inheritance and Virtual Classes

- Use virtual classes to allow direct inheritance from grandparent w/o “diamond problem”
- Methods from parent class can be overridden
- No MRO in C++. Unambiguous or error.

External classes – from whence comes the magic?

- Most capability comes from libraries external to our programs
  - standard library elements
    - #include <iostream> // use angle brackets
  - outside libraries like BOOST
- To build custom external capability:
  - class declarations in a header file myclass.hpp
  - class definitions in myclass.cpp
    - #include “myclass.hpp” //use double quotes
  - to use in an independent program:
    - #include “myclass.cpp”





Now you try it:

Create a file with a SleighClass and an appropriate header file. The class should have attributes for sleighSpeed and destinationLatitude. The class should have a method transitTime() to calculate the time required to travel from the North Pole to any destination in the northern hemisphere.

Create a second file that accesses the SleighClass and uses it to calculates how long Santa will take to get to your house.

Assumptions:

The earth is a perfect sphere with a diameter of 8,000 miles.

Santa's trip originates at the true North Pole, and proceeds at a nominal altitude with his first stop at your house.

Hint: Santa travels along the arc of a circle of diameter 8000

Hint: <include cmath> pi = 2 * acos(0.0);

Enumerations

- user-defined constants
- can be used like
- enum creates constant names with an implicit index
  - great for self-documenting code
  - sets finite number of values
  - enum enum_var {element1, element2);

- Constructors and destructors
  - A constructor is a class that's executed when we make new objects.
  - A destructor is just the opposite. It's called when a class is no longer needed.
  - Syntax:

```
class MyClass{
    MyClass();   // declares a constructor
    ~MyClass(); // declares a destructor
}
```

  - MyClass::MyClass(void){ // define constructor here}
  - MyClass::~MyClass(void) // define destructor here}
  - Destructors are called automatically as "last rites"
  - C++ has no garbage collection, but it is possible to allocate and deallocate memory.

- More on pointers
  - Pointers can be advanced or decrement ted using the ++/--/+/- etc.
  - A “one unit” move of a pointer goes sizeof(pointer) memory units.
  - For a container of elements, an increment moves to the address of the next element.
    - container[0] = *ptr
    - container[1] = *(ptr +1)
  - We can use pointer arrays (arrays of pointers) …
  - … So we can also have pointers to pointers using a double-asteric **ptr

- More on pointers
  - When we pass a pointer to a function, we're passing the address of the an object. Within the function we can dereference then change the value held in that address.
  - When we pass a value to a function, we're passing a "hard copy" of the value as its own object.
  - Passing by reference is much more efficient for large objects ...
  - but that means multiple process can operate on the same object
  - Passing by value is less efficient ...
  - ... but it's "safer" and better encapsulated.

- Testing and test-driven development
  - Uncle Bob's manifesto
  - Unit tests (back end)
  - Acceptance (business) tests
  - Integration tests ("plumbing")
  - Functional tests (full-scale with mocks)
  - End to end tests (integrate front end)

- Assertions are built into core C++
- More, and easier to use testing environments are available in outside libraries. E.g.,
  - Google Test
  - Google Mock
  - CppUnit (based on Junit)
  - Boost.Test

- Standard Template Library
- Generic objects, tools, and utilities
  - Algorithms https://en.cppreference.com/w/cpp/algorithm
  - Containers (vector, queue, etc.)
  - Functions (push_front, insert, begin, size, etc.)
  - Iterators (input, output, forward, bidirectional, random access)

★ AGS LLC ★

ANALYTICAL GOVERNMENT SOLUTIONS

Defensible, highly accurate analytical products and decision analysis for Decision Makers

The AGS Staff and Instructors
Pat Barton and Gil Dickens
Wish to express our thanks to our Dahlgren SWC and Navy Community
For Joining Our Presentation of

AGS APPLIED PYTHON COURSE

July 19th – July 23rd 2021

Pat Barton.
President/CEO, PES LLC
A Certified Service Minority Owned Small Business

E. G. "Gil" Dickens, Jr.
President/CEO, AGS LLC
A Certified Service Disabled Veteran Owned Small Business (SDVOSB)

V 072321 egd