Goals

1. We would also like to review inheritance and polymorphism by having you run through the Lab 13 assignment on Codio.
Review of Polymorphism
Can someone explain:

- Protected class members

Private members are only accessible within the class defining them. Protected members are accessible in the class that defines them and in classes that inherit from that class.
# Inheritance

<table>
<thead>
<tr>
<th>When the component is declared as:</th>
<th>When the class is inherited as:</th>
<th>The resulting access inside the subclass is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>public</td>
<td>Public</td>
</tr>
<tr>
<td>protected</td>
<td>public</td>
<td>protected</td>
</tr>
<tr>
<td>private</td>
<td>protected</td>
<td>none</td>
</tr>
<tr>
<td>public</td>
<td>protected</td>
<td>private</td>
</tr>
<tr>
<td>protected</td>
<td>protected</td>
<td>private</td>
</tr>
<tr>
<td>private</td>
<td>protected</td>
<td>none</td>
</tr>
</tbody>
</table>

```java
class A {
    public:
    int x;
    protected:
    int y;
    private:
    int z;
};
class B : public A {
    // x is public
    // y is protected
    // z is not accessible from B
};
class C : protected A {
    // x is protected
    // y is protected
    // z is not accessible from C
};
class D : private A // 'private' is default for classes {
    // x is private
    // y is private
    // z is not accessible from D
};
```
As a Large Group

Can someone explain:

- The order in which constructors of a Base, Child, and Grandchild class run. What about destructors?

Order of Inheritance

```
base  Class C (Base Class 2)
child Class B (Base Class 1)
grandchild Class A (Derived Class)
```

Order of Constructor Call

1. C() (Class C's Constructor)
2. B() (Class B's Constructor)
3. A() (Class A's Constructor)

Order of Destructor Call

1. ~A() (Class A's Destructor)
2. ~B() (Class B's Destructor)
3. ~C() (Class C's Destructor)
Can someone explain:

- What difference does the "virtual" keyword make in a function prototype of a base class member function?

The virtual keyword indicates to the compiler that it should choose the appropriate definition of f() not by the type of lvalue reference, but by the type of object that the lvalue reference refers to.
As a Large Group

Can someone explain:

- What is a pure virtual function?

A pure virtual function is a function that must be overridden in a derived class and need not be defined. A virtual function is declared to be “pure” using the curious =0 syntax.

Output:

```
#include <iostream>
using namespace std;

class Base {
  // private member variable
  int x;

public:
  // pure virtual function
  virtual void fun() = 0;

  // getter function to access x
  int getX() { return x; }
};

// This class inherits from Base and implements fun()
class Derived : public Base {
  // private member variable
  int y;

public:
  // implementation of the pure virtual function
  void fun() { cout << "fun() called"; }
};

int main(void)
{
  // creating an object of Derived class
  Derived d;

  // calling the fun() function of Derived class
d.fun();

  return 0;
}
```
Can someone explain:

- What is an abstract class?

A class is abstract if it has at least one pure virtual function

Ex) Test is an abstract class. We cannot instantiate an abstract class.

If we do not override the pure virtual function in a derived class, then the derived class also becomes an abstract class. For example, if Test2 was a child class of Test, and we did not override the show() function, then Test2 would also be an abstract class.

```cpp
#include <iostream>

using namespace std;

class Test {
    // private member variable
    int x;

public:
    // pure virtual function
    virtual void show() = 0;

    // getter function to access x
    int getX() { return x; }
};

int main(void)
{
    // Error: Cannot instantiate an abstract class
    Test t;
    return 0;
}
```

Output

Compiler Error: cannot declare variable 't' to be of abstract type 'Test' because the following virtual functions are pure within 'Test': note: virtual void Test::show()
Your Task

● Complete the two exercises in Codio Lab 13 (one analysis, one coding)
  – You need to complete this assignment with a 50 to get credit for the lab.
  – Once you get a 50 or above, make sure the TA’s manually change your grade to a 100% on Codio
  – Feel free to ask questions throughout the lab or even after you are done if you are confused on an answer
  – If you are done early, install your own toolchain (info on next slides)
    • This will be especially important in classes like 104, 201, etc
Installing your own Toolchain

Goals

1. While we ALL love Codio, it will be important to be able to develop, compile, and test code on your laptop, natively. Thus, we want to help you install a working C++ compiler, editor, and other tools on your laptop and have you write and run a simple "Hello world" application.
Tutorial of MS VS Code
MS Visual Studio Code

• Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

• It is ONLY an editor though but can be linked to a compiler and debugger if they are installed on your machine — We will have you try to install those as well.
If you already have your own toolchain (IDE or separate editor/compiler) installed, show us that you can compile and run a simple C++ program.
Windows OS Instructions
Using GCC with MinGW on Windows
(MacOS go to slide 12)

• In this tutorial, you’ll configure Visual Studio Code to use the GCC C++ compiler (g++) and GDB debugger from mingw-w64 to create programs that run on Windows.

• After configuring VS Code, you will compile and debug a simple Hello World program in VS Code.

• If you have any problems, feel free to check the VS Code documentation repository.
Prerequisites

• Install Visual Studio Code
• Install the C/C++ extension for VS Code. You can install the C/C++ extension by searching for 'c++' in the Extensions view.

![Visual Studio Code Extensions View]

• Get the latest version of Mingw-w64 via MSYS2, which provides up-to-date native builds of GCC, Mingw-w64, and other helpful C++ tools and libraries. You can download the latest installer from the MSYS2 page or use this link to the installer.
• Follow the Installation instructions on the MSYS2 website to install Mingw-w64. Take care to run each required Start menu and pacman command.
Prerequisites

- Install the Mingw-w64 toolchain (pacman -S --needed base-devel mingw-w64-x86_64-toolchain). Run the `pacman` command in a MSYS2 terminal. Accept the default to install all the members in the `toolchain` group.

- Add the path to your Mingw-w64 bin folder to the Windows PATH environment variable by using the following steps:
  - In the Windows search bar, type 'settings' to open your Windows Settings.
  - Search for **Edit environment variables for your account**.
  - Choose the **Path** variable in your **User variables** and then select **Edit**.
  - Select **New** and add the Mingw-w64 destination folder path to the system path. The exact path depends on which version of Mingw-w64 you have installed and where you installed it. If you used the settings above to install Mingw-w64, then add this to the path: `C:\msys64\mingw64\bin`.
  - Select **OK** to save the updated PATH. You will need to reopen any console windows for the new PATH location to be available.
Check your MinGW

• To check that your Mingw-w64 tools are correctly installed and available, open a new Command Prompt (do NOT use the previous one you had open) and type:
  - `gcc --version`
  - `g++ --version`
  - `gdb --version`

• If you don’t see the expected output or `g++` or `gdb` is not a recognized command, make sure your PATH entry matches the Mingw-w64 binary location where the compilers are located. If the compilers do not exist at that PATH entry, make sure you followed the instructions on the MSYS2 website to install Mingw-w64.
• If `gcc` has the correct output but not `gdb`, then you need to install the packages you are missing from the Mingw-w64 toolset.
  - Missing the `mingw-w64-gdb` package is one cause of the "The value of miDebuggerPath is invalid." message upon attempted compilation if your PATH is correct.
Create Hello World

• Follow Instructions on [this link](#)
Mac OS Instructions
Using GCC with MinGW on MacOS

- In this tutorial, you configure Visual Studio Code on macOS to use the Clang/LLVM compiler and debugger.
- After configuring VS Code, you will compile and debug a simple C++ program in VS Code.
- If you have any problems, feel free to check the VS Code documentation repository.
Prerequisites

• Install Visual Studio Code on macOS.
• Install the C++ extension for VS Code. You can install the C/C++ extension by searching for 'c++' in the Extensions view.

Ensure Clang is installed

• Clang may already be installed on your Mac. To verify that it is, open a macOS Terminal window and enter the following command:
  -- clang --version
• If Clang isn't installed, enter the following command to install the command line developer tools:
  -- xcode-select --install
Create Hello World

• Follow Instructions on this link
Debugging

- If you get the error message
  - `expected ; at the end of declaration vector<string> msg`
- Visual Studio Code is using C++03 by default for g++
- To fix this, go to .vscode folder → tasks.json and…
  - add “-std=c++11” under args
- Reopen the workspace/file in VSCode