CSCI 103: Introduction to Programming - Lab 13

Goals

1. We would like to first review inheritance and polymorphism by going through the slides
2. Then you can run through the Lab 13 assignment on Codio

Review of Inheritance and Polymorphism

As a Large Group

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

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass (derived class) is:
Public	Public	Public
Protected		Protected
Private		None
Public	Protected	Protected
Protected		Protected
Private		None
Public	Private	Private
Protected		Private
Private		None

```
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
};
```

```
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:

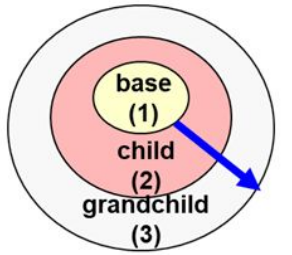
- The order in which constructors of a Base, Child, and Grandchild class run. What about destructors?

- Base class: A
- Child class: B
- Grandchild class: C

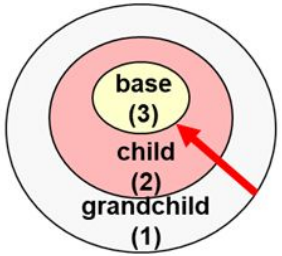
Construction/Destruction ordering

```
Allocating a B object
A:0
B:0
Allocating 1st C object
A:0
B:0
C:0
Allocating 2nd C object
A:0
B:4
C:5
Deleting c1 object
~C ~B ~A
Quitting
~C ~B ~A
~B ~A
```

Output



Constructor call ordering



Destructor call ordering

- Taken from Lecture slide 5d.13

As a Large Group

Can someone explain:

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

The virtual keyword indicates to the compiler that it should choose the appropriate definition of f() not by the type of pointer/reference, but by the type of object that is being referenced.

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

class base {
public:
    virtual void print() { cout << "print base class\n"; }

    void show() { cout << "show base class\n"; }
};

class derived : public base {
public:
    void print() { cout << "print derived class\n"; }

    void show() { cout << "show derived class\n"; }
};

int main()
{
    base* bptr;
    derived d;
    bptr = &d;

    // Virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();

    return 0;
}
```

```
print derived class
show base class
```

```cpp
class Base {
    public:
      virtual void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

int main() {
    Derived derived1;
    Base* base1 = &derived1;

    base1->print();

    return 0;
}
```

print() of Derived class is called because print() of Base class is virtual

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 by the base class. A virtual function is declared to be “pure” using the curious =0 syntax.

```cpp
#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;
}
```

Output:

```
fun() called
```

As a Large Group

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
// C++ program to illustrate the abstract class with pure
// virtual functions
#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

- We provide two exercises in Codio Lab 13 and one exercise in Codio Lab 13b
  - In total one analysis (in Lab 13), two codings (in Lab 13 and Lab 13b)
  - To get credit for the lab, you need to
    - Finish **the analysis exercise**
    - Make an honest attempt of the **coding exercise in Lab 13**
    - **Lab 13b is for your own practice.**
  - 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