OBJECT ORIENTED PROGRAMMING

Lab 6

OBJECT ORIENTED PROGRAMMING WITH C++

- Chapter Objectives
  - Base class access control: public, private and protected
  - Constructors, destructors, and inheritance
  - Multiple inheritance
  - Virtual base classes
  - Virtual functions

OBJECT ORIENTED PROGRAMMING WITH C++

- Base class access control: public, private and protected
  - Public
    - Base Class: All public members of base accessible as public
    - Derived Class: All public members of base accessible as public
  - Private
    - Base Class: All private members of base are not accesible
    - Derived Class: All public members of base accessible as private
  - Protected
    - Base Class: All protected members of base accesible as private
    - Base Class: All protected members of base accesible as private

- Constructors, destructors, and inheritance
  - When a base class and a derived class both have constructor and destructor functions:
    - The constructor functions are executed in order of derivation. That is, the base class constructor is executed before the constructor in the derived class.
    - The destructor functions are executed in reverse order. The derived class's destructor is executed before the base class's destructor.

OBJECT ORIENTED PROGRAMMING WITH C++

- Constructors, destructors, and inheritance
- Using an expanded form of the derived class's constructor declaration, you then pass the appropriate arguments along to the base class. The syntax for passing along an argument from the derived class to the base class is shown here:

```
derived_constructor(arg_list): base(arg_list)
{
// body of derived class constructor
}
```

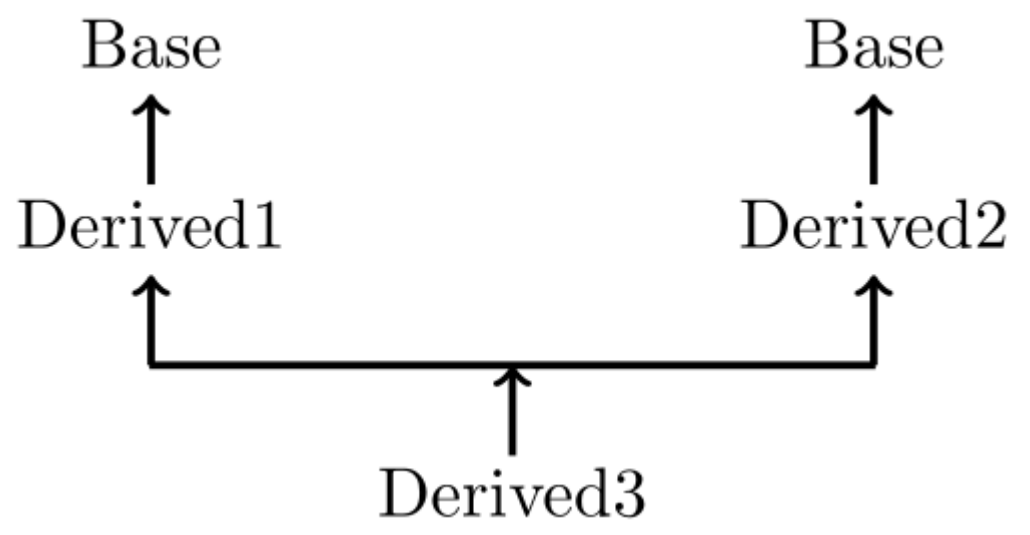
- Multiple inheritance
  - When a derived class directly inherits multiple base classes, it uses this expanded declaration:

```
class derived_class_name : access base1  , access base2 , ..., access baseN
{
// ... body of class
}
```

  - And derived class' constructor function:

```
derived_constructor(arg_list) : base1 (arg_list), base2(arg_list),..., baseN(arg-list)
{
// body of derived class constructor
}
```

- Virtual base classes
  - Here the base class Base is inherited by
  - both Derived1 and Derived2. Derived3
  - directly inherits both Derived1 and Derived2.
  - However, this implies that Base is actually



  - inherited twice by Derived3-first it is inherited through Derived1, and then again through Derived2. This causes ambiguity when a member of Base is used by Derived3.
  - To resolve this ambiguity. C++ includes a mechanism by which only one copy of Base will be included in Derived3. This feature is called a virtual base class.

- Virtual Functions
  - A virtual function is a member function that is declared within a base class and redefined by a derived class.
  - To create a virtual function, precede the function's declaration with the keyword virtual.
  - When a class containing a virtual function is inherited, the derived class redefines the virtual function relative to the derived class.

- Virtual Functions
  - A pure virtual function has no definition relative to the base class. Only the function's prototype is included. To make a pure virtual function, use this general form:

    virtual type func_name(parameter_list)=0;

  - The key part of this declaration is the setting of the function equal to 0. This tells the compiler that no body exists for this function relative to the base class. When a virtual function is made pure, it forces any derived class to override it.