Access levels and friend functions and classes

See code in
https://engineering.purdue.edu/~smidkiff/ECE39595CPP/Code/L7AccessLevelsFriend.zip

C++ access levels

- Access levels allow programmers to control who can access attributes and methods declared in a class
  - *private* attributes and methods are only visible inside the class they are declared in
  - *protected* attributes and methods are only visible inside the class they are declared in, and classes that inherit from the declaring class
  - *public* attributes and methods are visible in any code in the program
  - The default is private, i.e., if no access level is specified, it is *private*.
- Methods and attributes should be declared with the least visibility possible
  - This allows for more flexibility in changing code later on
  - What other classes can't see can be changed, as long as it meets the specification

What we will learn in these slides

- We will learn how access levels allow us to control the visibility of methods and attributes we declare in a class
- We will learn how specifying an access level when inheriting from another class allows us to control the visibility of methods and attributes of that class *when accessed as part of the inheriting class*

Where are objects created in a program?

```
class Base {
public:
  int baseVar;
  Base( );
  virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }
void Base::baseFunc( ) {
  Base b;
}
```

```
class Derived : public Base {
public:
  int derivedVar;
  Derived( );
  virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };
void Derived::derivedFunc( ) {
  Base derivedB;
  derivedB.baseVar = -1;
  derivedB.baseFunc( );
```

```
int main(int argc, char* argv[ ]) {
  Base mainBase;
  Derived mainDerived;
  mainBase.baseVar = -10;
  mainDerived.derivedFunc( );
  mainDerived.derivedVar = -20;
  mainDerived.baseVar = -30;
  mainDerived.derivedVar = -40;
}
```

Example1 code

- Green (italics) indicates places that cause Base objects to be created
- Blue (bold) indicates places that cause Derived objects to be created
- Red (Forte font) indicates places that cause a Base and Derived object to be created

What are affected by access levels?

```
class Base {
public:
  int baseVar;
  Base( );
  virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }
void Base::baseFunc( ) {
  Base baseB;
}
```

```
class Derived : public Base {
public:
  int derivedVar;
  Derived( );
  virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };
void Derived::derivedFunc( ) {
  Base derivedB;
  derivedB.baseVar = -1;
  derivedB.baseFunc( );
```

```
int main(int argc, char* argv[ ]) {
  Base mainBase;
  Derived mainDerived;
  mainBase.baseVar = -10;
  mainDerived.derivedFunc( );
  mainDerived.derivedVar = -20;
  mainDerived.baseVar = -30;
  mainDerived.derivedVar = -40;
}
```

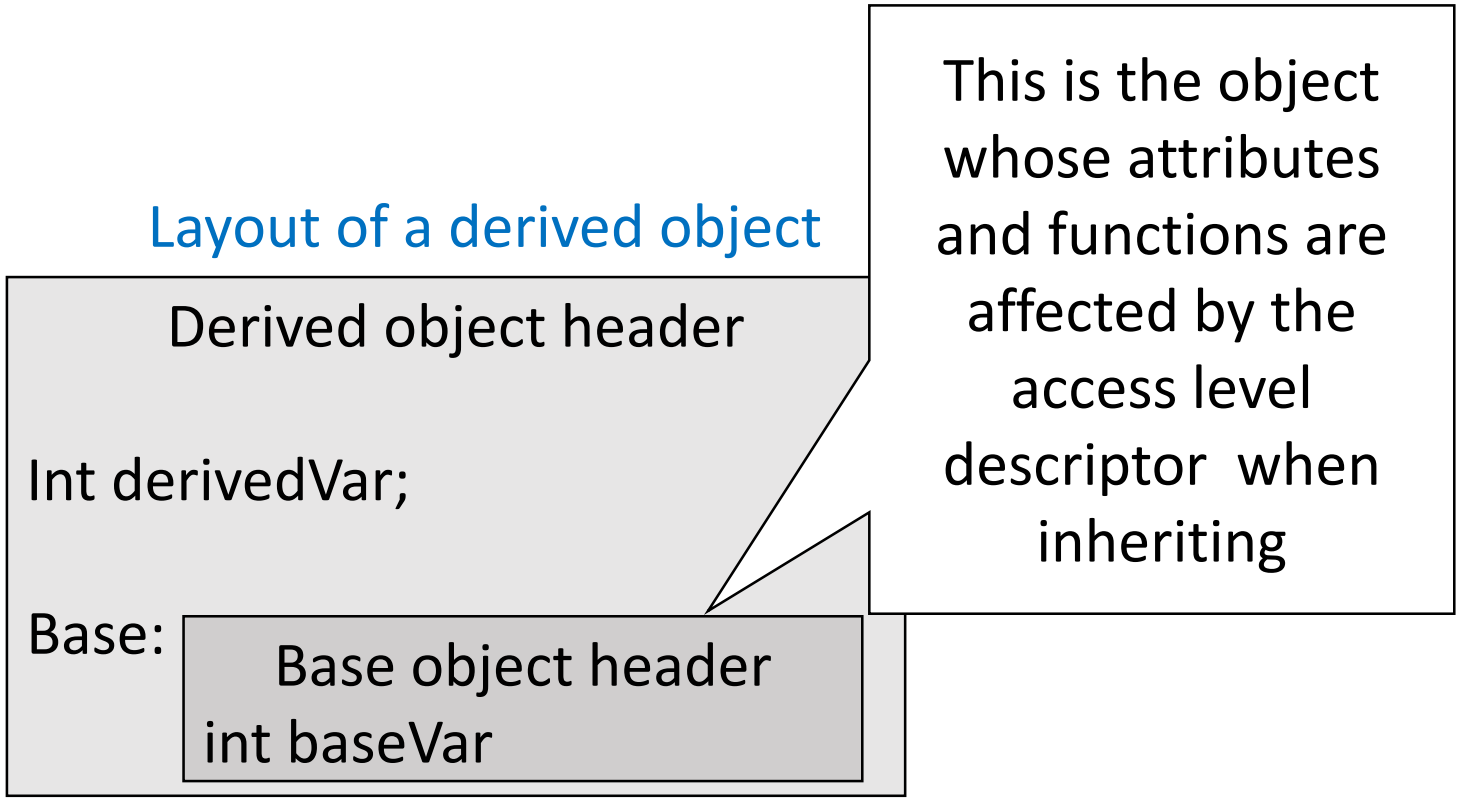
Example1 code

- Attributes or functions that belong to a base object that is part of a derived object because of inheritance are affected by access levels
- No other objects attributes or functions are affected

What are affected by access levels?

```
class Derived : public Base {
public:
   int derivedVar;
   Derived( );
   virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };
void Derived::derivedFunc( ) {
   Base derivedB;
   derivedB.baseVar = -1;
   derivedB.baseFunc( );
```



Declared access level of the base object	Access level specified by derived class when inheriting	Access to included base object attributes in derived class code	Access to included base object attributes in code outside of the derived class	Access to attributes of base object created by derived class code	Access to attributes of base object created by code outside of the derived class
private	private	private	private	private	private
protected	private	protected	private	protected	private
public	private	public	private	public	private

```
class Base {
public:
  int baseVar;

  Base( );
  virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }

void Base::baseFunc( ) {
  Base b;
}
```

```
class Derived : private Base {
public:
  int derivedVar;
  Derived( );
  virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };

void Derived::derivedFunc( ) {
  Base derivedB;
  baseBar = -5;
  baseFunc( );
  derivedB.baseVar = -1;
  derivedB.baseFunc( );
}
```

```
int main(int argc, char* argv[ ]) {
  Base mainBase;
  Derived mainDerived;
  mainBase.baseVar = -10;
  mainDerived.baseFunc( );
  mainDerived.derivedVar = -20;
  mainDerived.baseVar = -30;
  mainDerived.derivedVar = -40;
}
```

Example2 code

Declared access level of the base object	Access level specified by derived class when inheriting	Access to included base object attributes in derived class code	Access to included base object attributes in code outside of the derived class	Access to attributes of base object created by derived class code	Access to attributes of base object created by code outside of the derived class
private	public	private	private	private	private
protected	public	protected	protected	protected	protected
public	public	public	public	public	public

```
class Base {
public:
 Base( );
protected:
   int baseVar;
   virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }

void Base::baseFunc( ) {
   Base b;
}
```

```
class Derived : public Base {
public:
   int derivedVar;
   Derived( );
   virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };

void Derived::derivedFunc( ) {
   Base derivedB;
   baseBar = -5;
   baseFunc( );
   derivedB.baseVar = -1;
   derivedB.baseFunc( );
}
```

```
int main(int argc, char* argv[ ]) {
   Base mainBase;
   Derived mainDerived;
   mainBase.baseVar = -10;
   mainDerived.baseFunc( );
   mainDerived.derivedVar = -20;
   mainDerived.baseVar = -30;
   mainDerived.derivedFunc( );
}
```

Example3 code

Declared access level of the base object	Access level specified by derived class when inheriting	Access to included base object attributes in derived class code	Access to included base object attributes in code outside of the derived class	Access to attributes of base object created by derived class code	Access to attributes of base object created by code outside of the derived class
private	protected	private	private	private	private
protected	protected	protected	protected	protected	protected
public	protected	protected	protected	protected	protected

```
class Base {
public:
  Base( );
  int baseVar;
  virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }

void Base::baseFunc( ) {
  Base b;
}
```

```
class Derived : protected Base {
public:
  int derivedVar;
  Derived( );
  virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };

void Derived::derivedFunc( ) {
  Base derivedB;
  baseBar = -5;
  baseFunc( );
  derivedB.baseVar = -1;
  derivedB.baseFunc( );
}
```

```
int main(int argc, char* argv[ ]) {
  Base mainBase;
  Derived mainDerived;
  mainBase.baseVar = -10;
  mainDerived.baseFunc( );
  mainDerived.derivedVar = -20;
  mainDerived.baseVar = -30;
  mainDerived.derivedFunc( );
}
```

Example4 code

```
class Base {
protected:
  Base( );
  int baseVar;
  virtual void baseFunc( );
};

Base::Base( ) : baseVar(1) { }

void Base::baseFunc( ) {
  Base b;
}
```

```
class Derived : public Base {
public:
  int derivedVar;
  Derived( );
  virtual void derivedFunc( );
};

Derived::Derived( ) : derivedVar(2) { };

void Derived::derivedFunc( ) {
  Base derivedB;
  baseBar = -5;
  baseFunc( );
  derivedB.baseVar = -1;
  derivedB.baseFunc( );
}
```

```
int main(int argc, char* argv[ ]) {
  Base mainBase;
  Derived mainDerived;
  mainBase.baseVar = -10;
  mainDerived.baseFunc( );
  mainDerived.derivedVar = -20;
  mainDerived.baseVar = -30;
  mainDerived.derivedFunc( );
}
```

Example5 code

Friend functions and classes – a way to selectively get around access levels

- A class can declare a function not in the class as a friend.
  - This makes all of the functions and attributes of the class accessible by the friend function – even private functions and attributes.
  - Friend functions are often functions that don't belong to any class
- A class can also declare another class as a friend.
  - this makes all of the functions and attributes of the class accessible by the friend class – even private functions and attributes.
- *friend* will prove very handy when we do operator overloading, but it's basically a way to allow helper classes and functions to access our code

Friend functions (See Example6/Function code)

```
class Complex {
private:
  double real;
  double imag;
public:
  Complex(double, double);
  friend void printComplex(Complex);
};

Complex::Complex(double re, double im) {
  real = re;
  imag = im;
}
```

```
void printComplex(Complex c) {
  std::cout << "(" << c.real << ", “;
  std::cout << c.imag << ")" << std::endl;
}

int main(int argc, char* argv[ ]) {
  Complex c(1.0, 1.0);
  printComplex(c);
}
```

(1, 1)

Friend classes

(See Example3/Class code)

```
class Complex {
private:
  double real;
  double imag;
public:
  Complex(double, double);
  friend class ComplexHelper;
};

Complex::Complex(double re, double im) {
  real = re;
  imag = im;
}

class ComplexHelper {
public:
  void print(Complex);
  Complex add(Complex, Complex);
  Complex subtract(Complex, Complex);
};
```

```
void ComplexHelper::print(Complex c) {
  std::cout << "(" << c.real << ", " << c.imag << ")" << std::endl;
}

Complex ComplexHelper::add(Complex c1, Complex c2) {
  return Complex(c1.real + c2.real, c1.imag + c2.imag);
}

Complex ComplexHelper::subtract(Complex c1, Complex c2) {
  return Complex(c1.real - c2.real, c1.imag - c2.imag);
}

int main(int argc, char* argv[ ]) {
  Complex c1(1.0, 1.0);
  Complex c2(1.0, 1.0);
  ComplexHelper h;
  Complex cSum = h.add(c1, c2);
  Complex cDifference = h.subtract(c1, c2);
  h.print(c1);
  h.print(c2);
  h.print(cSum);
  h.print(cDifference);
}
```

(1, 1)
(1, 1)
(2, 2)
(0, 0)

What we learned in these slides

- Access levels specify the visibility of a method or attribute outside of the class
  - private says the symbol is only visible in the class it is declared in
  - protected says the symbol is visible in the class it is declared in and classes that inherit from the class that it is defined in
  - public says that the symbol is visible anywhere
- Inheritance access levels specify the visibility of inherited symbols by code outside of the inheriting class