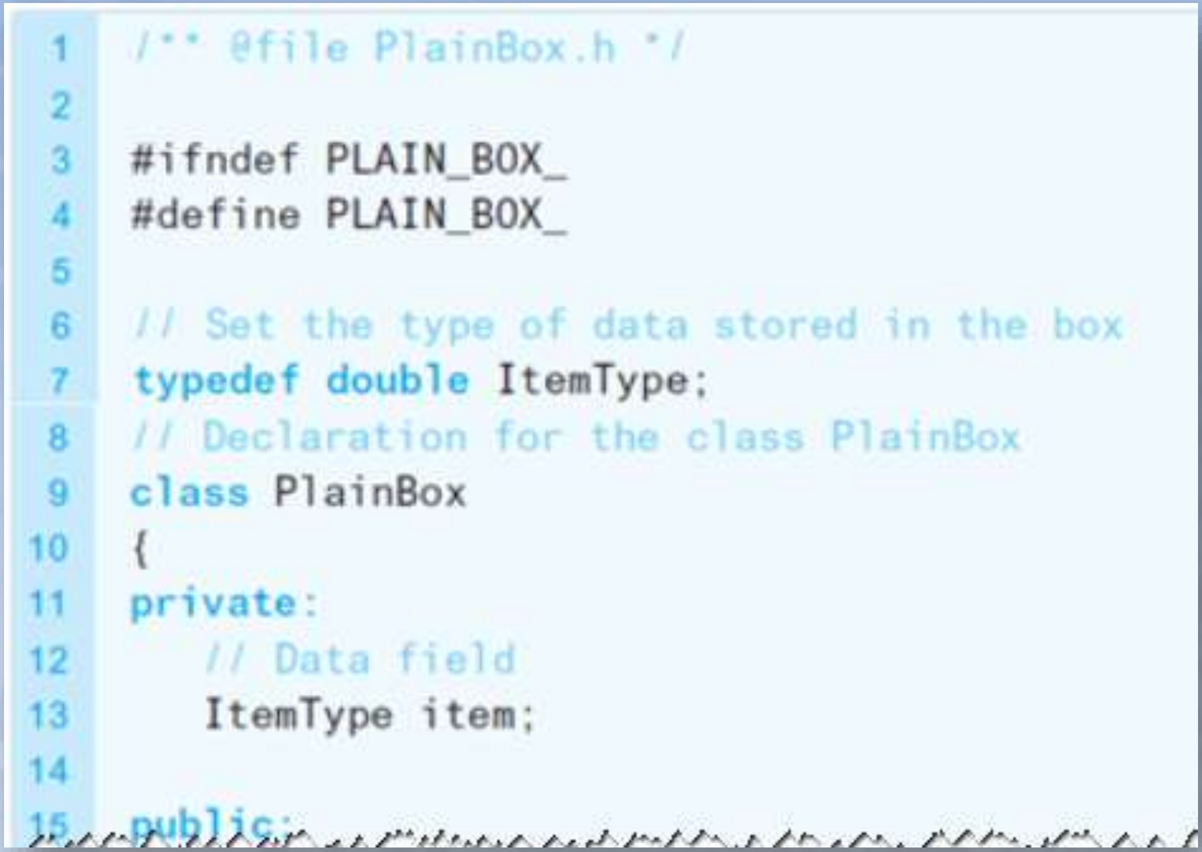
C++ Classes

C++ Interlude 1

A Problem to Solve

- Consider a video game where a character carries three types of boxes
  - Plain box
  - Toy box
  - Magic box
- Plain box design
  - Get and Set public methods

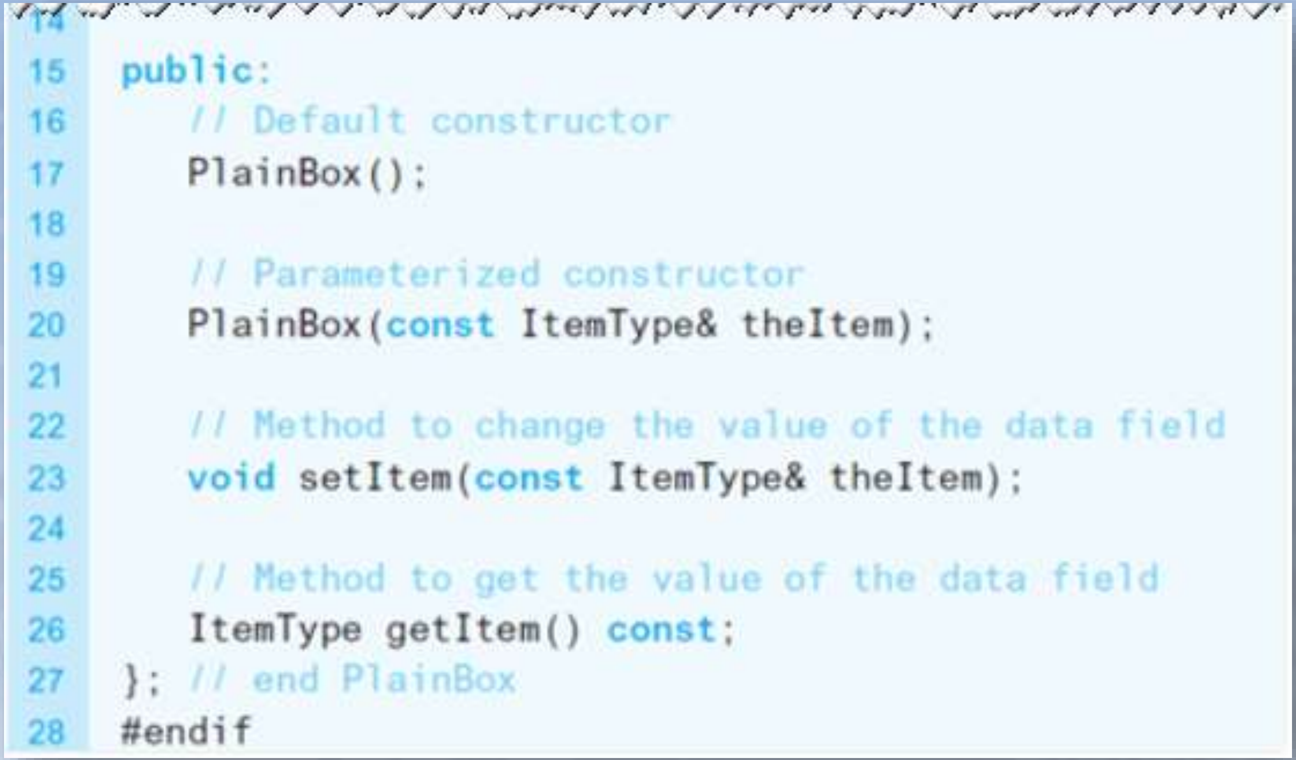
A Problem to Solve



```
/** @file PlainBox.h */

#ifndef PLAIN_BOX_
#define PLAIN_BOX_

// Set the type of data stored in the box
typedef double ItemType;
// Declaration for the class PlainBox
class PlainBox
{
private:
    // Data field
    ItemType item;

public:
```

LISTING C1-1 The header file for the class PlainBox

A Problem to Solve



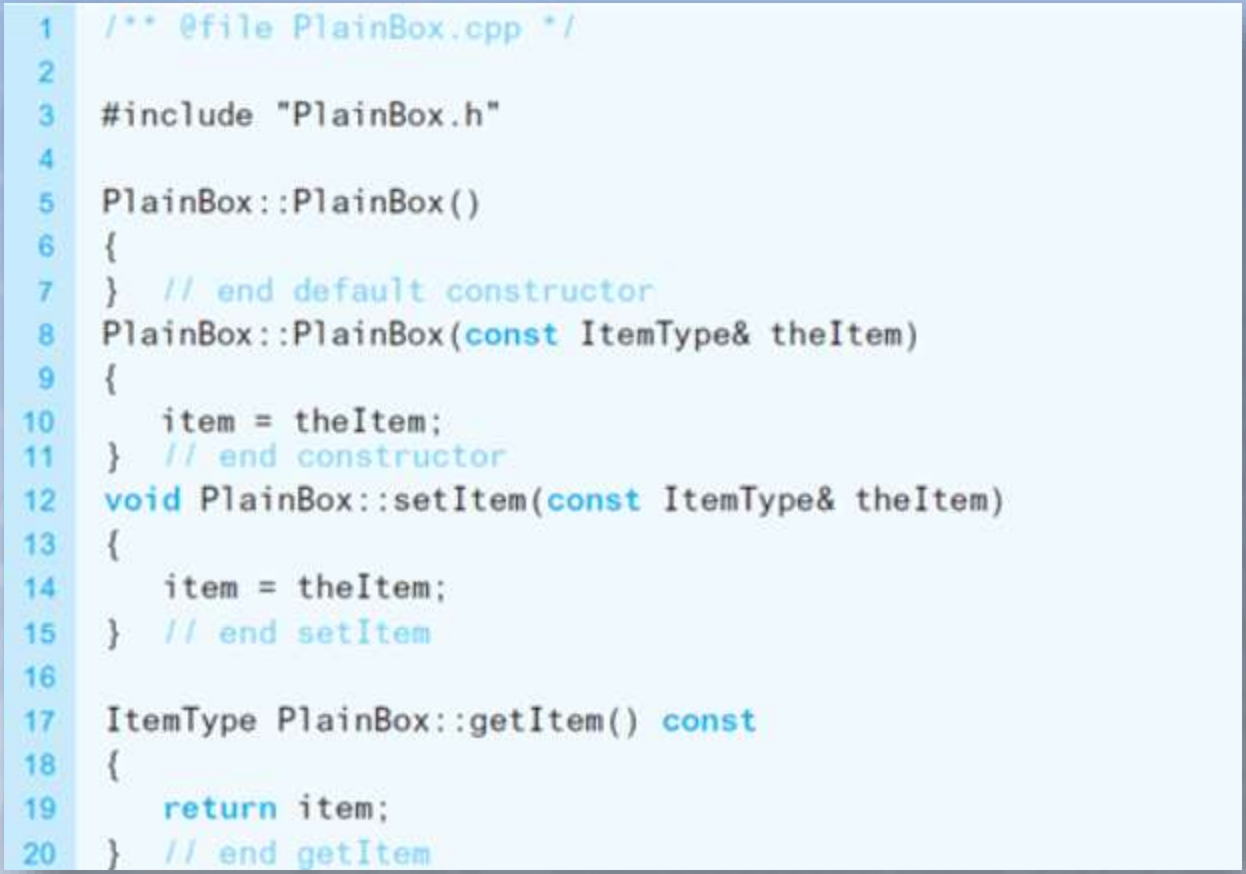
```
public:
   // Default constructor
   PlainBox();

   // Parameterized constructor
   PlainBox(const ItemType& theItem);

   // Method to change the value of the data field
   void setItem(const ItemType& theItem);

   // Method to get the value of the data field
   ItemType getItem() const;
}; // end PlainBox
#endif
```

LISTING C1-1 The header file for the class PlainBox

A Problem to Solve

- Elements of the class
  - Private data fields
  - Constructors, destructors
  - Methods
  - Use of #ifndef , #define , and #endif preprocessor directives
  - Use of initializers
  - Use of typedef
  - Inheritance

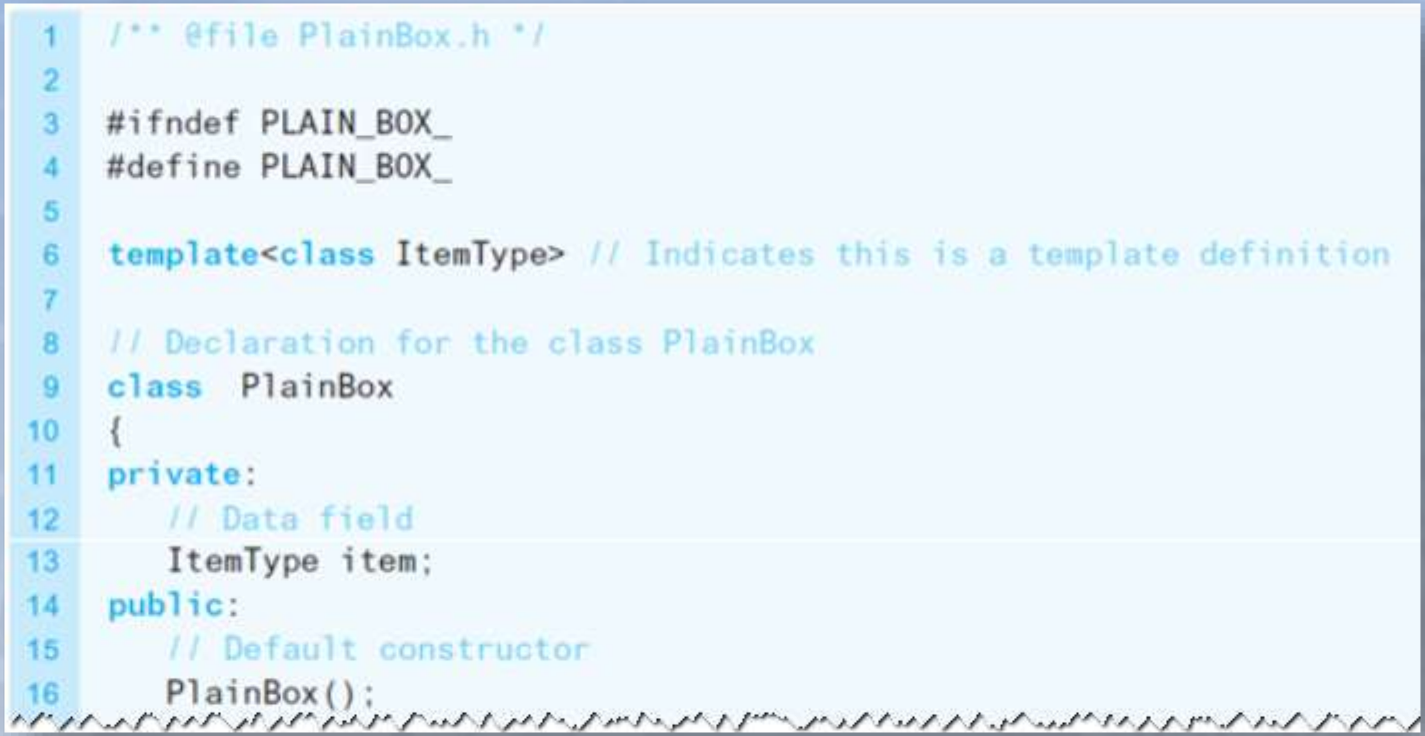
A Problem to Solve



```
/** @file PlainBox.cpp */

#include "PlainBox.h"

PlainBox::PlainBox()
{
}  // end default constructor
PlainBox::PlainBox(const ItemType& theItem)
{
   item = theItem;
}  // end constructor
void PlainBox::setItem(const ItemType& theItem)
{
   item = theItem;
}  // end setItem

ItemType PlainBox::getItem() const
{
   return item;
}  // end getItem
```

LISTING C1-2 Implementation file for the PlainBox class

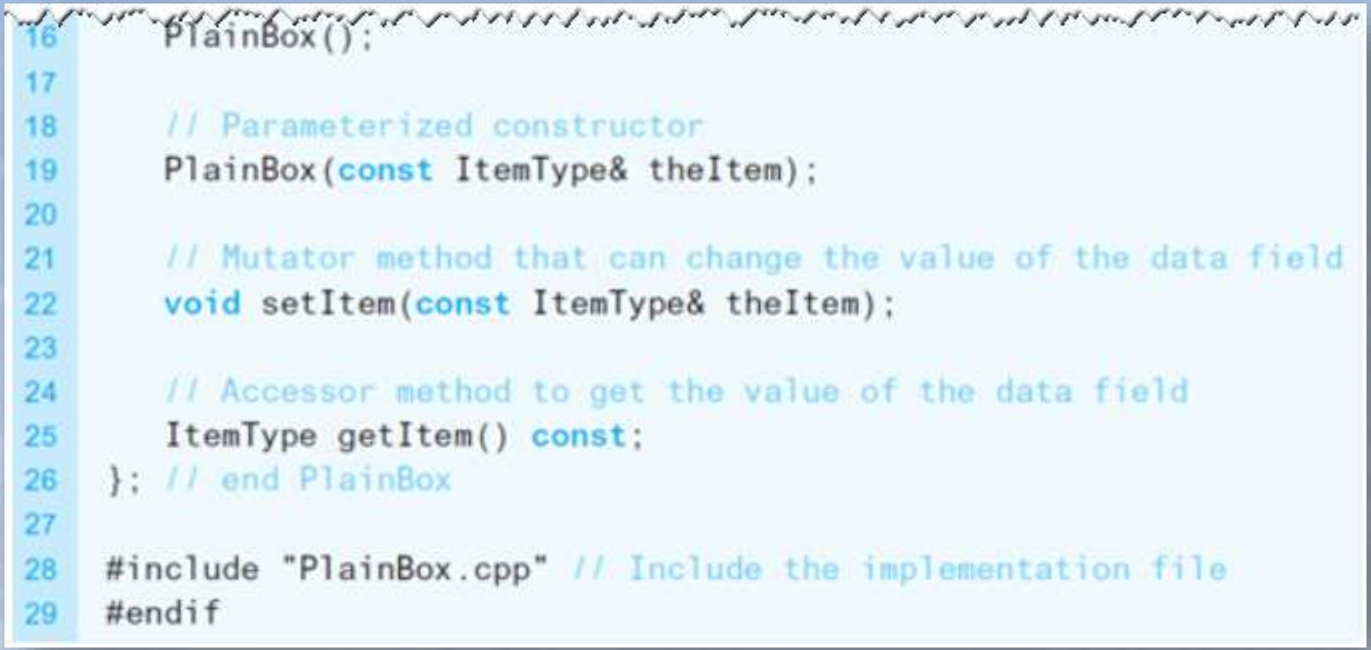
A Problem to Solve



```
/** @file PlainBox.h */

#ifndef PLAIN_BOX_
#define PLAIN_BOX_

template<class ItemType> // Indicates this is a template definition

// Declaration for the class PlainBox
class  PlainBox
{
private:
   // Data field
   ItemType item;
public:
   // Default constructor
   PlainBox();
```

LISTING C1-3 Template header file for the PlainBox class

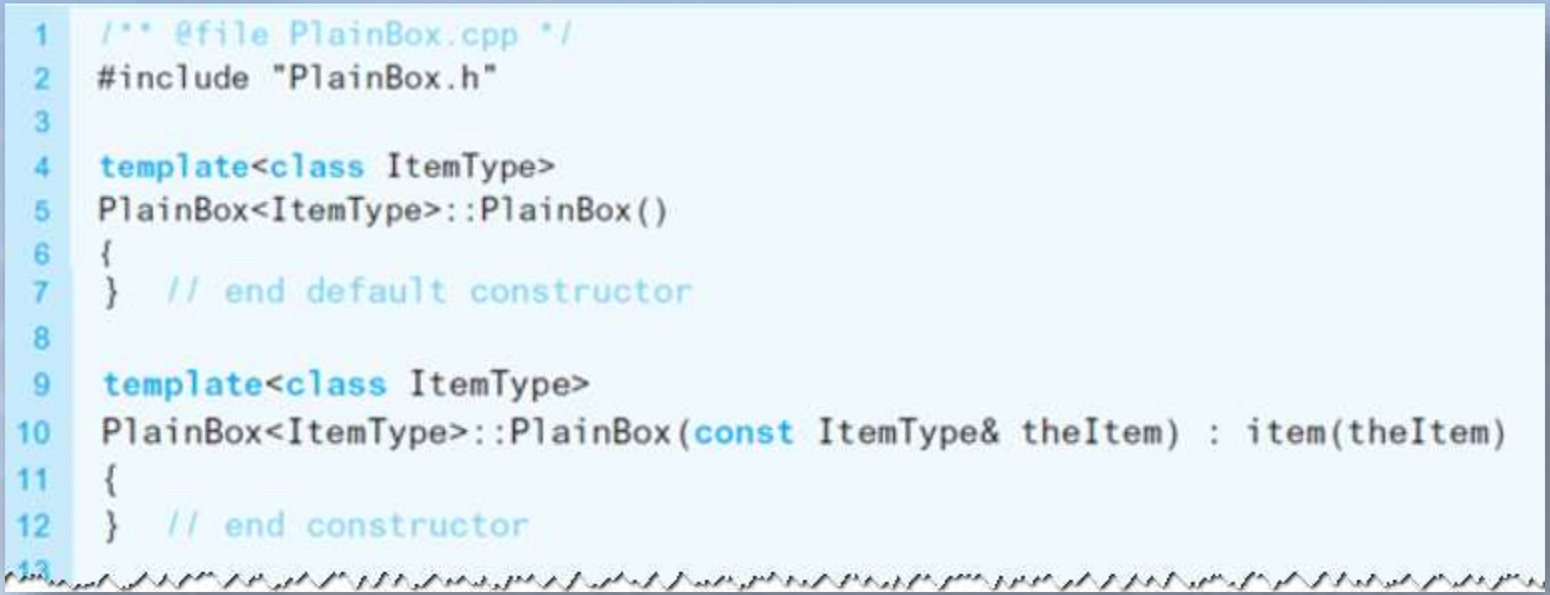
A Problem to Solve



```
    PlainBox();

    // Parameterized constructor
    PlainBox(const ItemType& theItem);

    // Mutator method that can change the value of the data field
    void setItem(const ItemType& theItem);

    // Accessor method to get the value of the data field
    ItemType getItem() const;
}; // end PlainBox

#include "PlainBox.cpp" // Include the implementation file
#endif
```

LISTING C1-3 Template header file for the PlainBox class

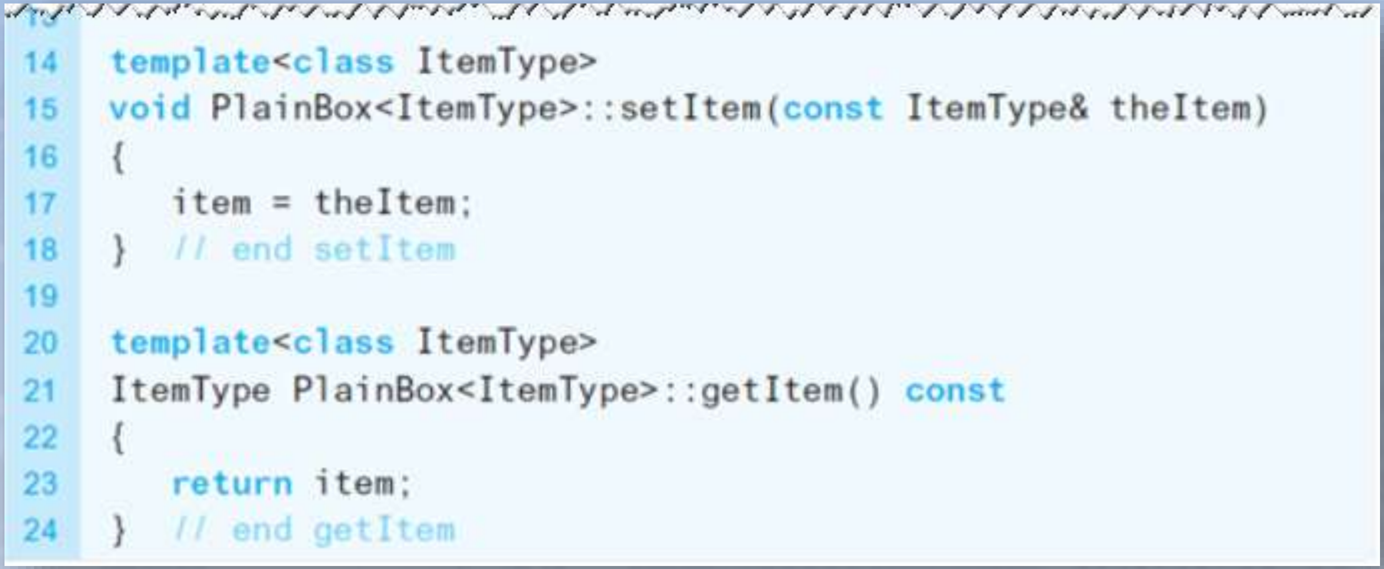
A Problem to Solve



```
/** @file PlainBox.cpp */
#include "PlainBox.h"

template<class ItemType>
PlainBox<ItemType>::PlainBox()
{
}  // end default constructor

template<class ItemType>
PlainBox<ItemType>::PlainBox(const ItemType& theItem) : item(theItem)
{
}  // end constructor
```

LISTING C1-4 Implementation file for the PlainBox template class

A Problem to Solve



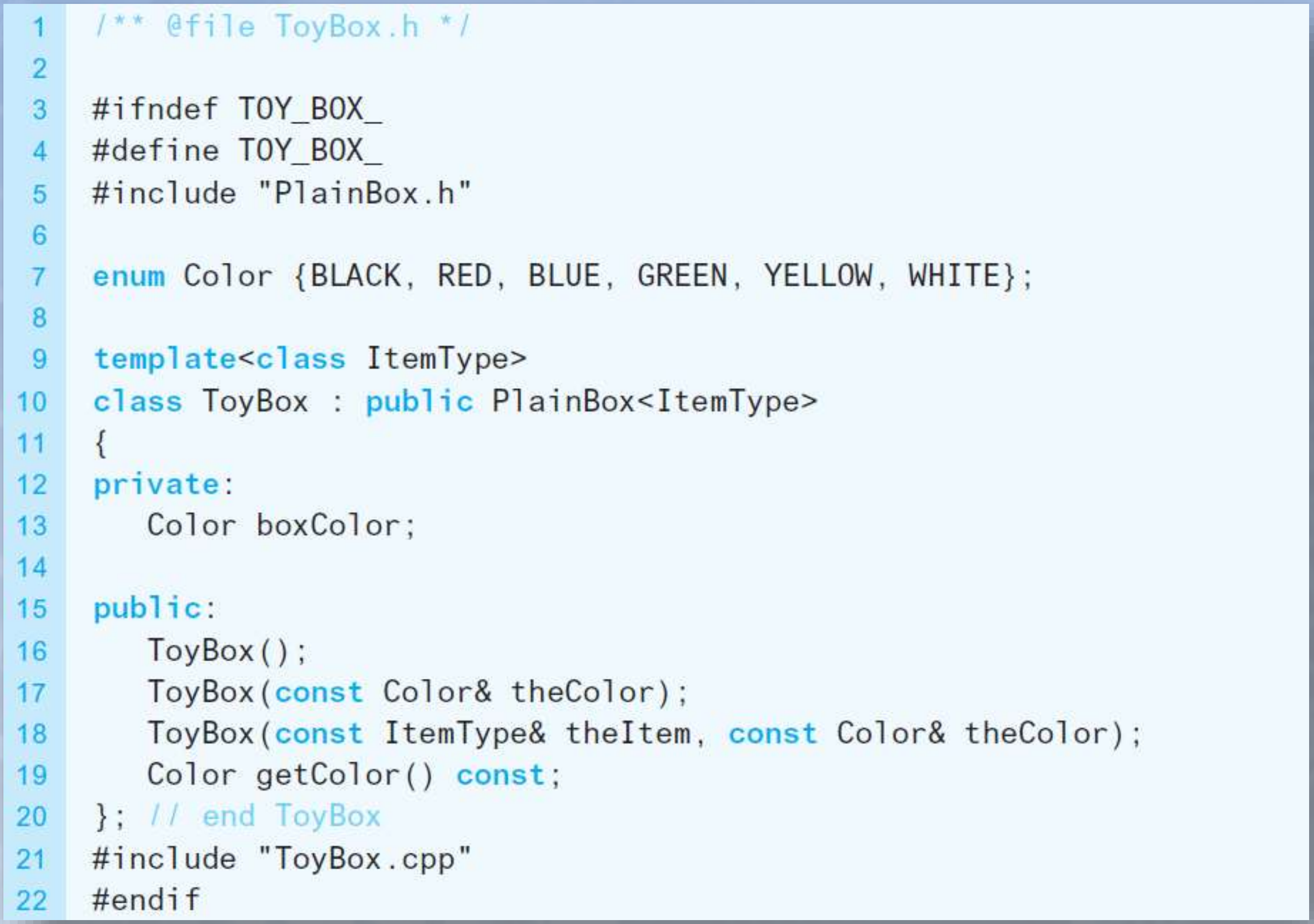
```
template<class ItemType>
void PlainBox<ItemType>::setItem(const ItemType& theItem)
{
   item = theItem;
}  // end setItem

template<class ItemType>
ItemType PlainBox<ItemType>::getItem() const
{
   return item;
}  // end getItem
```

LISTING C1-4 Implementation file for the PlainBox template class

Base Classes and Derived Classes

- Use PlainBox as a base class , or superclass
- The ToyBox class is the derived class , or subclass , of the PlainBox
- Derived class inherits
  - All the members of its base class,
  - (Except the constructors and destructor)

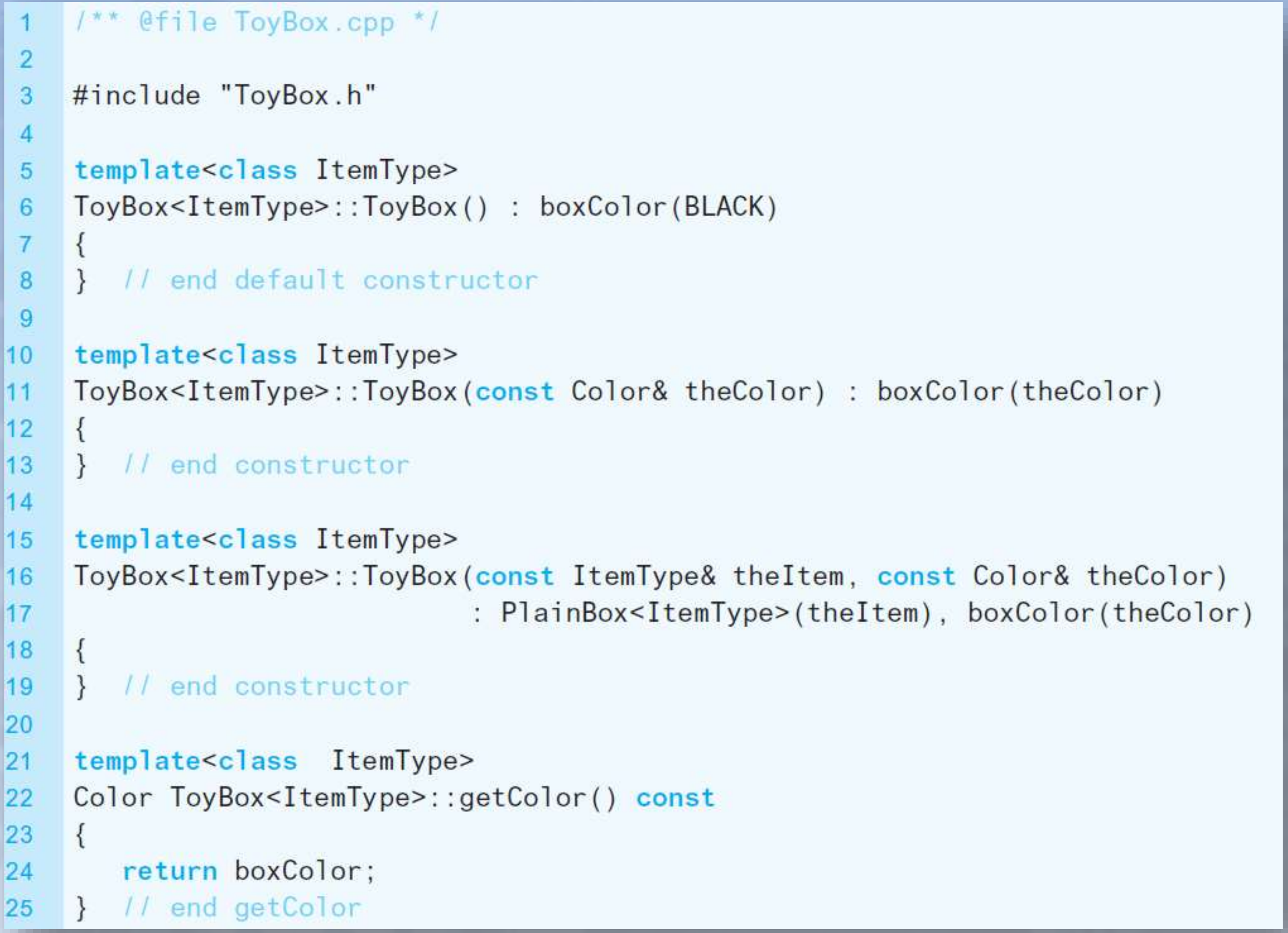
Base Classes and Derived Classes



```cpp
/** @file ToyBox.h */

#ifndef TOY_BOX_
#define TOY_BOX_
#include "PlainBox.h"

enum Color {BLACK, RED, BLUE, GREEN, YELLOW, WHITE};

template<class ItemType>
class ToyBox : public PlainBox<ItemType>
{
private:
   Color boxColor;

public:
   ToyBox();
   ToyBox(const Color& theColor);
   ToyBox(const ItemType& theItem, const Color& theColor);
   Color getColor() const;
}; // end ToyBox
#include "ToyBox.cpp"
#endif
```

LISTING C1-5 Template header file for the class ToyBox

Base Classes and Derived Classes



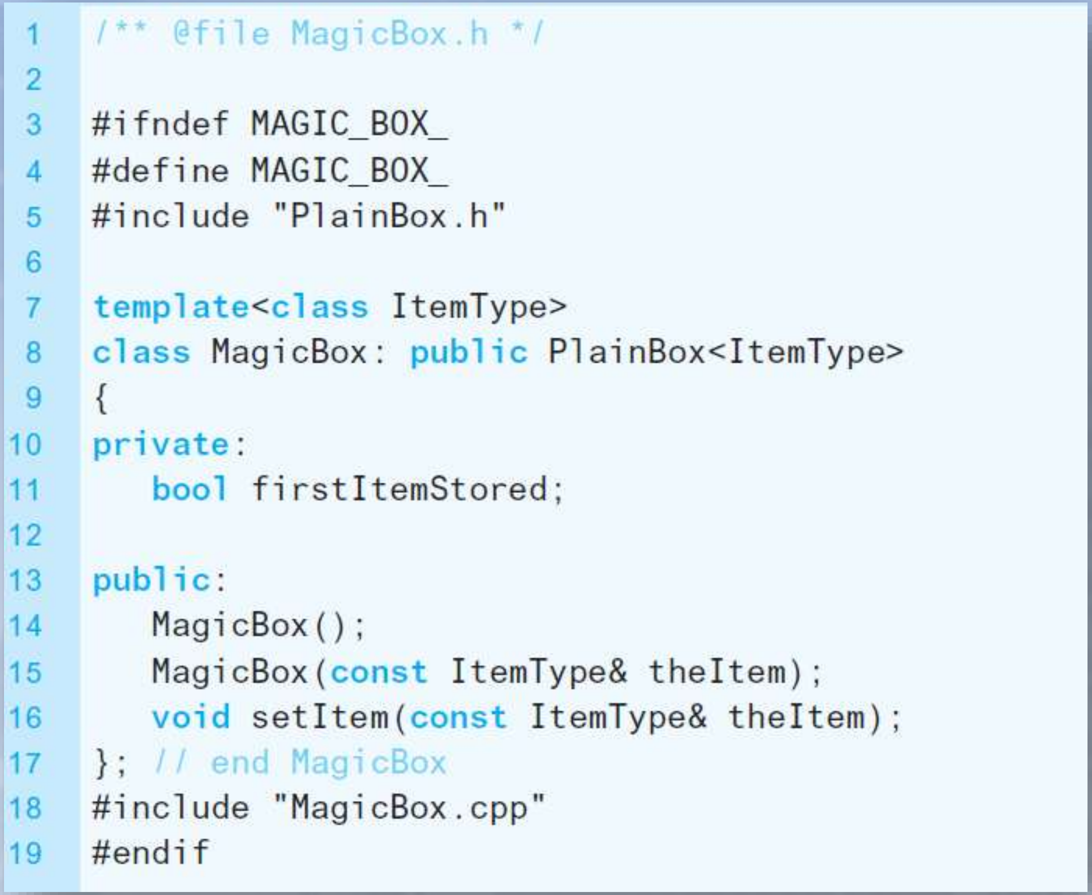
```cpp
/** @file ToyBox.cpp */

#include "ToyBox.h"

template<class ItemType>
ToyBox<ItemType>::ToyBox() : boxColor(BLACK)
{
}  // end default constructor

template<class ItemType>
ToyBox<ItemType>::ToyBox(const Color& theColor) : boxColor(theColor)
{
}  // end constructor

template<class ItemType>
ToyBox<ItemType>::ToyBox(const ItemType& theItem, const Color& theColor)
                        : PlainBox<ItemType>(theItem), boxColor(theColor)
{
}  // end constructor

template<class  ItemType>
Color ToyBox<ItemType>::getColor() const
{
   return boxColor;
}  // end getColor
```

LISTING C1-6 Implementation file for the class ToyBox

Overriding Base-Class Methods

- You can add as many new members to derived class as desired
- You can redefine inherited methods
  - Called overriding a base-class method.
- A method overrides a base-class method when
  - The two methods have the same name and parameter declarations

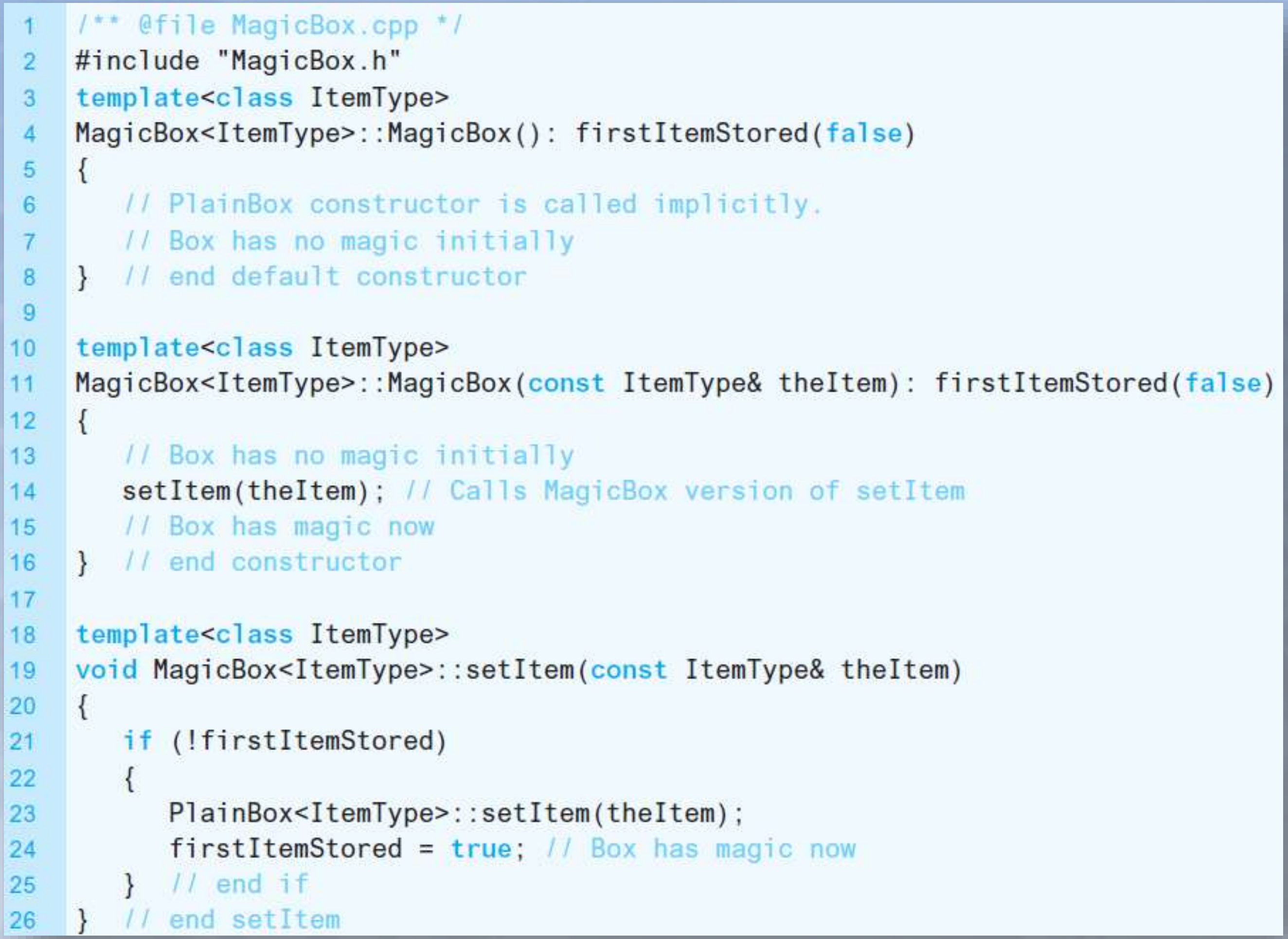
Overriding Base-Class Methods



```cpp
/** @file MagicBox.h */

#ifndef MAGIC_BOX_
#define MAGIC_BOX_
#include "PlainBox.h"

template<class ItemType>
class MagicBox: public PlainBox<ItemType>
{
private:
   bool firstItemStored;

public:
   MagicBox();
   MagicBox(const ItemType& theItem);
   void setItem(const ItemType& theItem);
}; // end MagicBox
#include "MagicBox.cpp"
#endif
```

LISTING C1-7 Header file for the class MagicBox

Overriding Base-Class Methods



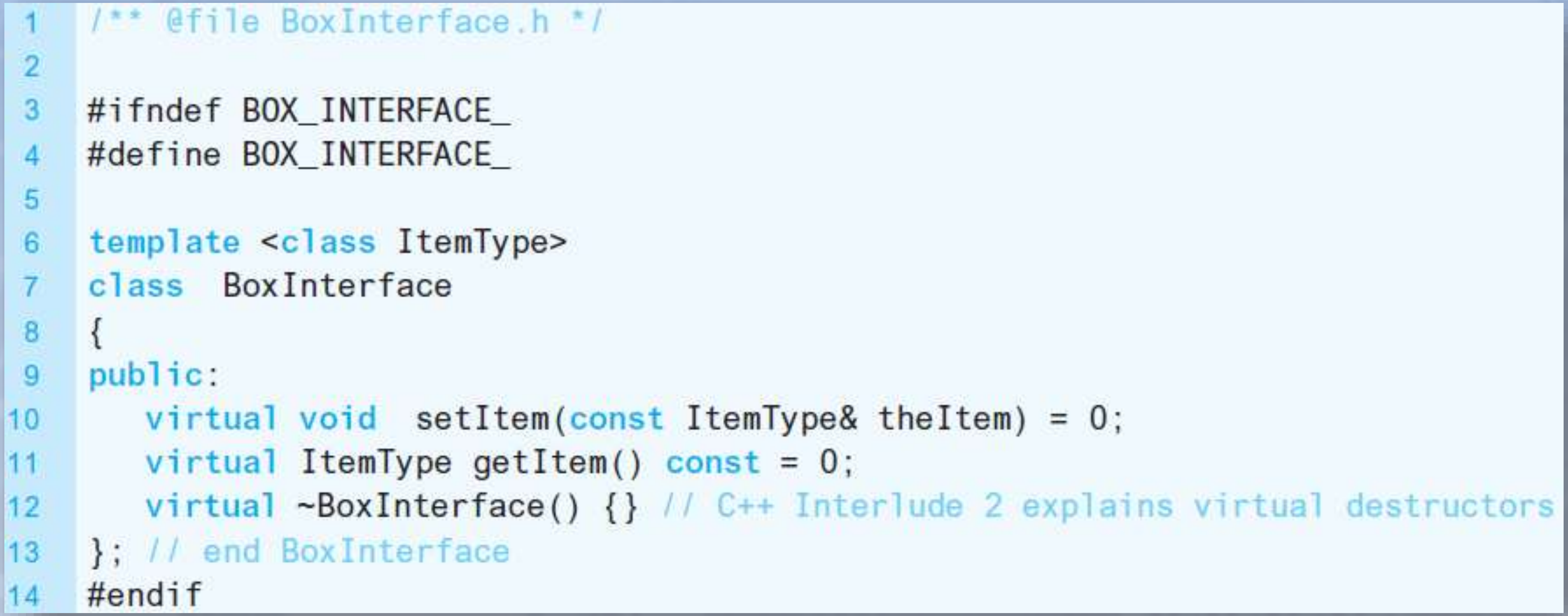
```
/** @file MagicBox.cpp */
#include "MagicBox.h"
template<class ItemType>
MagicBox<ItemType>::MagicBox(): firstItemStored(false)
{
   // PlainBox constructor is called implicitly.
   // Box has no magic initially
}  // end default constructor

template<class ItemType>
MagicBox<ItemType>::MagicBox(const ItemType& theItem): firstItemStored(false)
{
   // Box has no magic initially
   setItem(theItem); // Calls MagicBox version of setItem
   // Box has magic now
}  // end constructor

template<class ItemType>
void MagicBox<ItemType>::setItem(const ItemType& theItem)
{
   if (!firstItemStored)
   {
      PlainBox<ItemType>::setItem(theItem);
      firstItemStored = true; // Box has magic now
   }  // end if
}  // end setItem
```

LISTING C1-8 Implementation file for the class MagicBox

Virtual Methods, Abstract Classes

- Using keyword **virtual** in front of the prototype
  - Tells the C++ compiler that the code this method executes is determined at runtime
- Pure virtual method
  - Virtual method that has no implementation
- Abstract class
  - Has at least one pure virtual method

Virtual Methods, Abstract Classes



```
/** @file BoxInterface.h */

#ifndef BOX_INTERFACE_
#define BOX_INTERFACE_

template <class ItemType>
class  BoxInterface
{
public:
   virtual void  setItem(const ItemType& theItem) = 0;
   virtual ItemType getItem() const = 0;
   virtual ~BoxInterface() {} // C++ Interlude 2 explains virtual destructors
}; // end BoxInterface
#endif
```

LISTING C1-9 An abstract class that is an interface for the ADT box

End

Interlude 1