Example programs for Module 4 – Pointers and Linked Lists

Example 1	PlainBoxExample1 folder PlainBox.h PlainBox.cpp MagicBox.h MagicBox.cpp main.cpp	PlainBox defines a class that stores a single item. MagicBox is a sub class of the PlainBox class that restricts the single item such that it can be set only one time. A test client is stored in main.cpp. The main function works with instances of both classes, testing whether or not the instance item can be changed. Below are the 4 tests that the main function conducts: • test1() checks if an item stored in a MagicBox can be changed • test2() assigns an instance of a MagicBox to a PlainBox reference and tests whether or not the item can be changed • test3() passes an instance of a PlainBox or a MagicBox to a function that accepts the instance by reference. The function attempts to change the item • test4() creates an array of pointers to instances of PlainBox and MagicBox objects and attempts to change the item of each instance The compiler will utilize early binding for the code in main.cpp which will result in instances of a MagicBox being bound to the setItem method of the PlainBox class. In other words, the item of a MagicBox can be changed in some settings.
Example 2	PlainBoxExample2 folder PlainBox.h PlainBox.cpp MagicBox.h MagicBox.cpp main.cpp	All files are the same as in folder 1 above, except PlainBox.h. In this version, the setItem method is defined as virtual which will allow the compiler to perform late binding. By binding to the method at runtime, instances of a MagicBox will utilize the setItem method defined in the MagicBox class regardless of how the instance is used. Late binding achieves polymorphism.
Example 3	PlainBoxExample3 folder BoxInterface.h PlainBox.h PlainBox.cpp MagicBox.h MagicBox.cpp main.cpp	In this version, an interface is used, defining all of the methods as pure virtual. The PlainBox class now extends this class. Although the PlainBox class does not include the keyword virtual when defining the methods, the virtual aspect is inherited from the interface.
Example 4	Pointers.cpp	Experiments with pointers and dynamic memory allocation
Example 5	LinkedBag folder BagInterface.h Node.h	The Bag ADT implemented using a linked list

	Node.cpp LinkedBag.h LinkedBag.cpp BagTester.cpp	
Example 6	BagADT folder BagInterface.h Node.h Node.cpp LinkedBag.h LinkedBag.cpp ArrayBag.h ArrayBag.cpp main.cpp	Contains both implementations of the Bag ADT. The main program allows the user to select which implementation to test. The main program utilizes polymorphism by using declarations of the interface (the super class) instead of the sub classes (ArrayBag or LinkedBag).