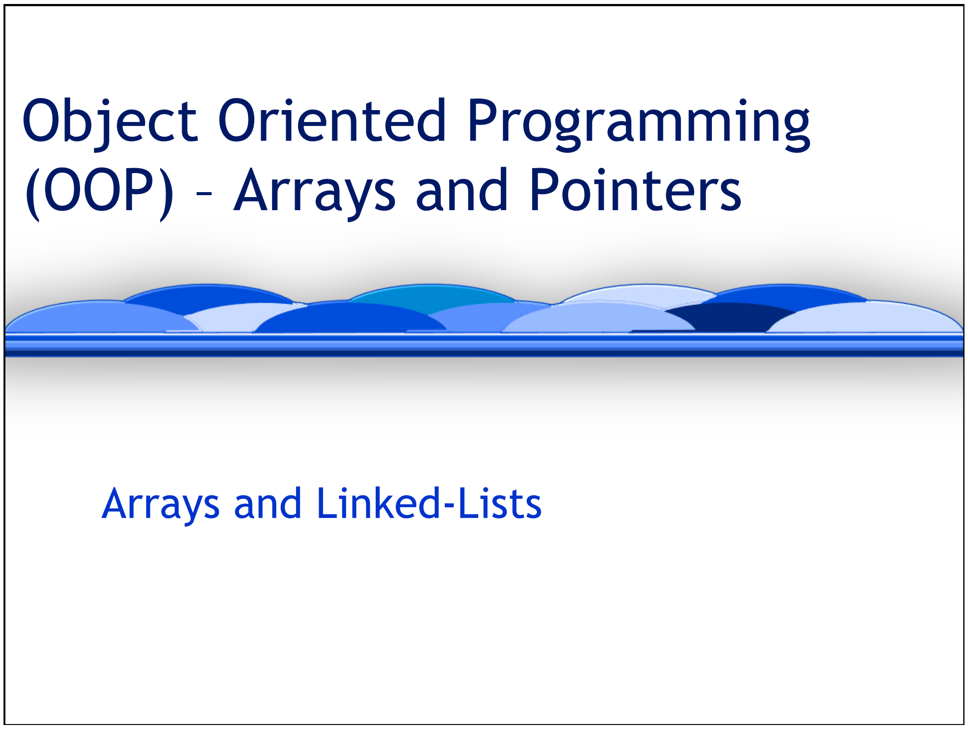
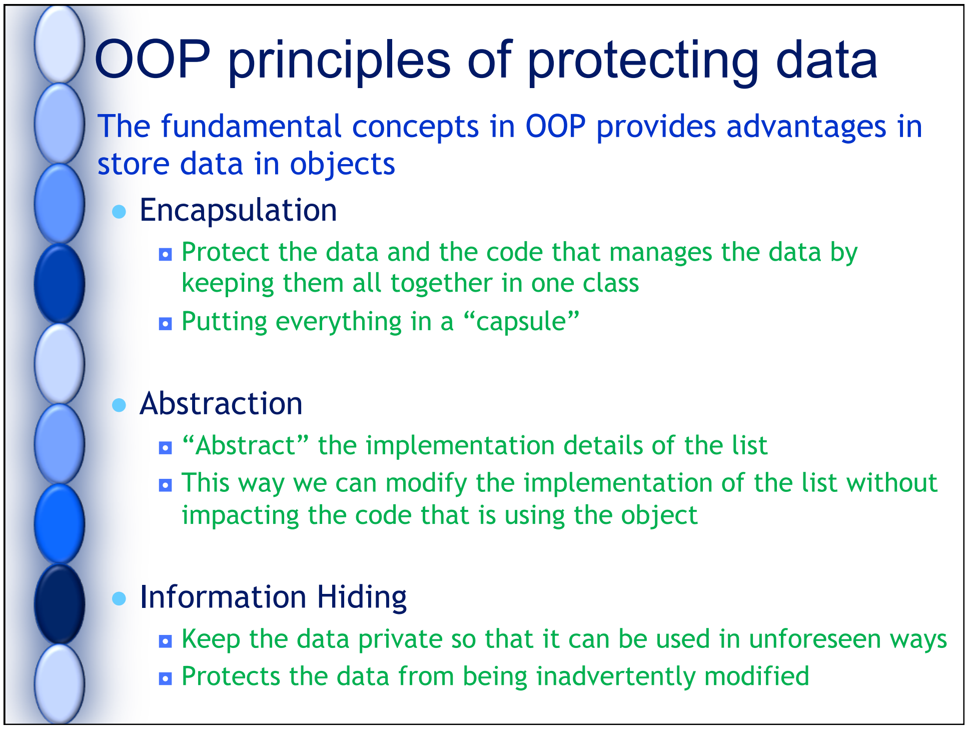


Object Oriented Programming (OOP) – Arrays and Pointers

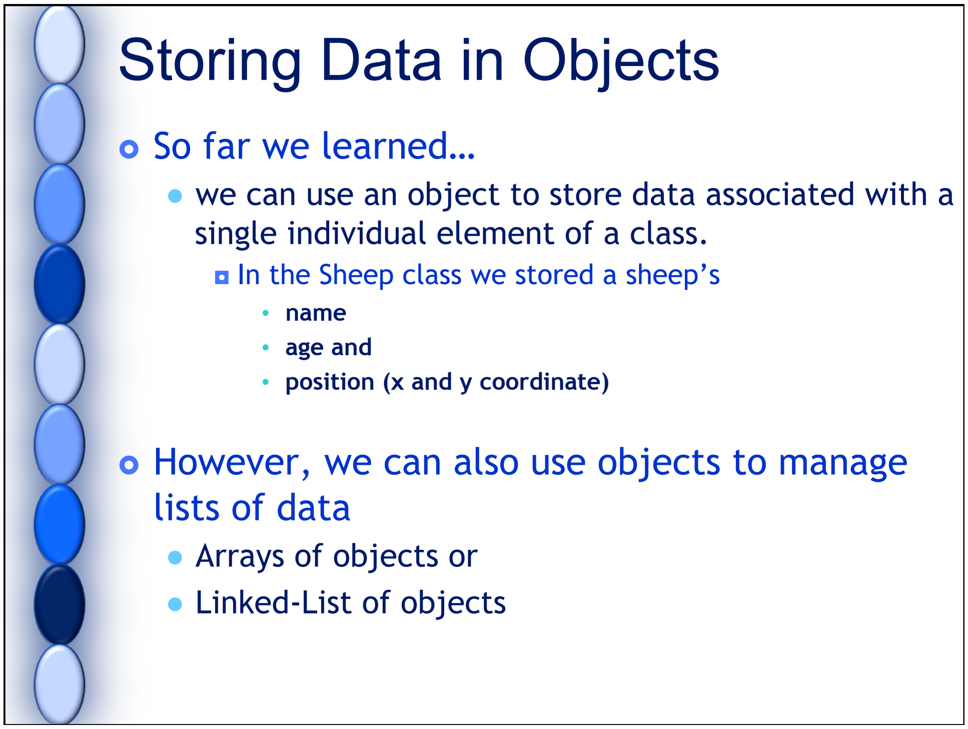
Arrays and Linked-Lists



OOP principles of protecting data

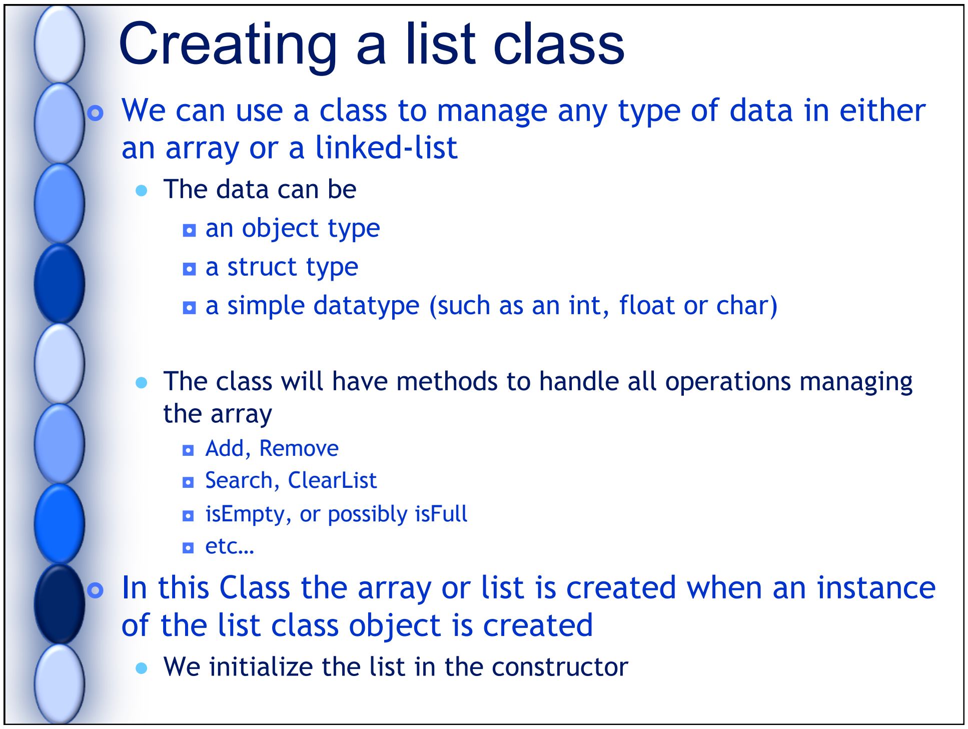
The fundamental concepts in OOP provides advantages in store data in objects

- Encapsulation
  - Protect the data and the code that manages the data by keeping them all together in one class
  - Putting everything in a “capsule”
- Abstraction
  - “Abstract” the implementation details of the list
  - This way we can modify the implementation of the list without impacting the code that is using the object
- Information Hiding
  - Keep the data private so that it can be used in unforeseen ways
  - Protects the data from being inadvertently modified



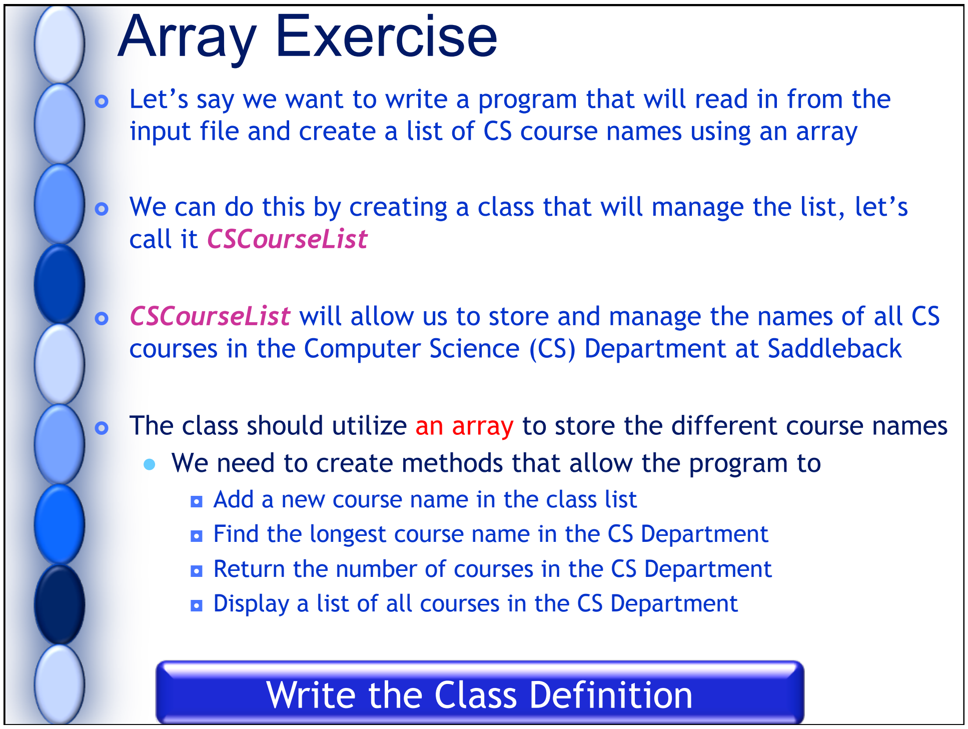
Storing Data in Objects

- So far we learned…
  - we can use an object to store data associated with a single individual element of a class.
    - In the Sheep class we stored a sheep's
      - **name**
      - **age and**
      - **position (x and y coordinate)**
- However, we can also use objects to manage lists of data
  - Arrays of objects or
  - Linked-List of objects



Creating a list class

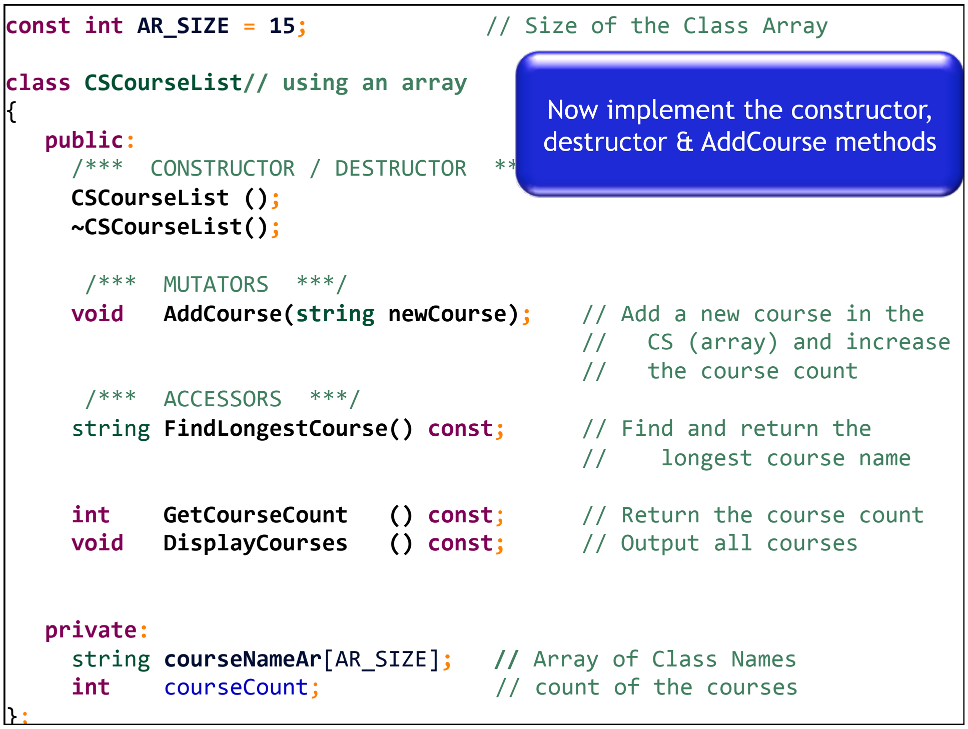
- We can use a class to manage any type of data in either an array or a linked-list
  - The data can be
    - an object type
    - a struct type
    - a simple datatype (such as an int, float or char)
  - The class will have methods to handle all operations managing the array
    - Add, Remove
    - Search, ClearList
    - isEmpty, or possibly isFull
    - etc…
- In this Class the array or list is created when an instance of the list class object is created
  - We initialize the list in the constructor



Array Exercise

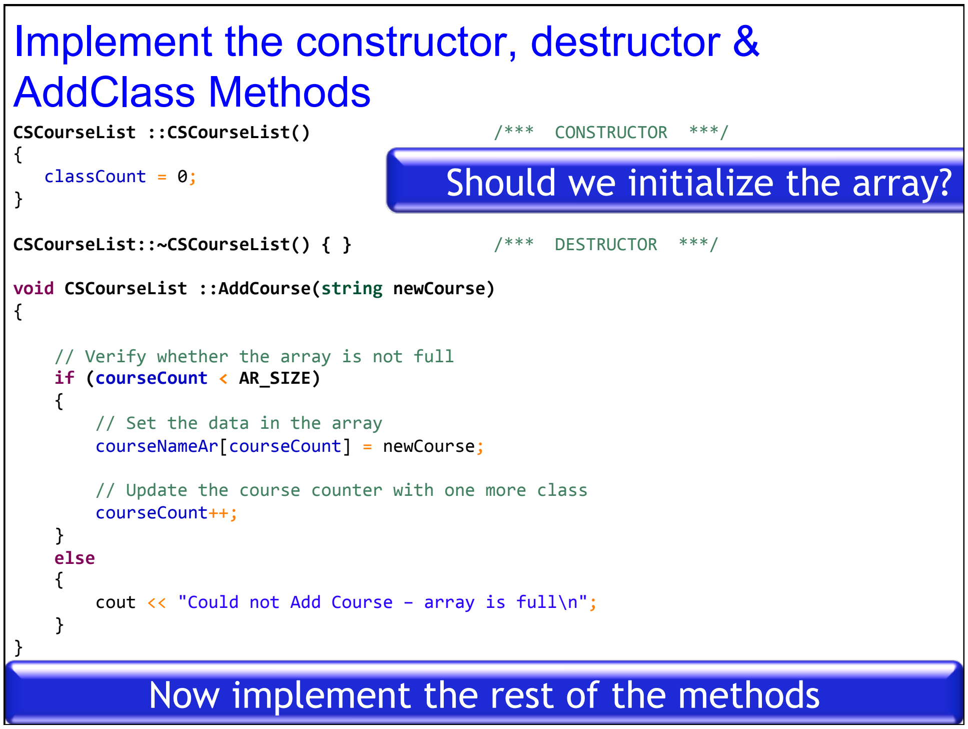
- Let’s say we want to write a program that will read in from the input file and create a list of CS course names using an array
- We can do this by creating a class that will manage the list, let’s call it *CSCourseList*
- *CSCourseList* will allow us to store and manage the names of all CS courses in the Computer Science (CS) Department at Saddleback
- The class should utilize an array to store the different course names
  - We need to create methods that allow the program to
    - Add a new course name in the class list
    - Find the longest course name in the CS Department
    - Return the number of courses in the CS Department
    - Display a list of all courses in the CS Department

Write the Class Definition



```
const int AR_SIZE = 15;                // Size of the Class Array

class CSCourseList// using an array
{
   public:
     /***  CONSTRUCTOR / DESTRUCTOR  **
     CSCourseList ();
     ~CSCourseList();

      /***  MUTATORS  ***/
     void   AddCourse(string newCourse);   // Add a new course in the
                                           //   CS (array) and increase
                                           //   the course count
      /***  ACCESSORS  ***/
     string FindLongestCourse() const;     // Find and return the
                                           //    longest course name

     int    GetCourseCount   () const;     // Return the course count
     void   DisplayCourses   () const;     // Output all courses


   private:
     string courseNameAr[AR_SIZE];   // Array of Class Names
     int    courseCount;             // count of the courses
};
```

Now implement the constructor, destructor & AddCourse methods

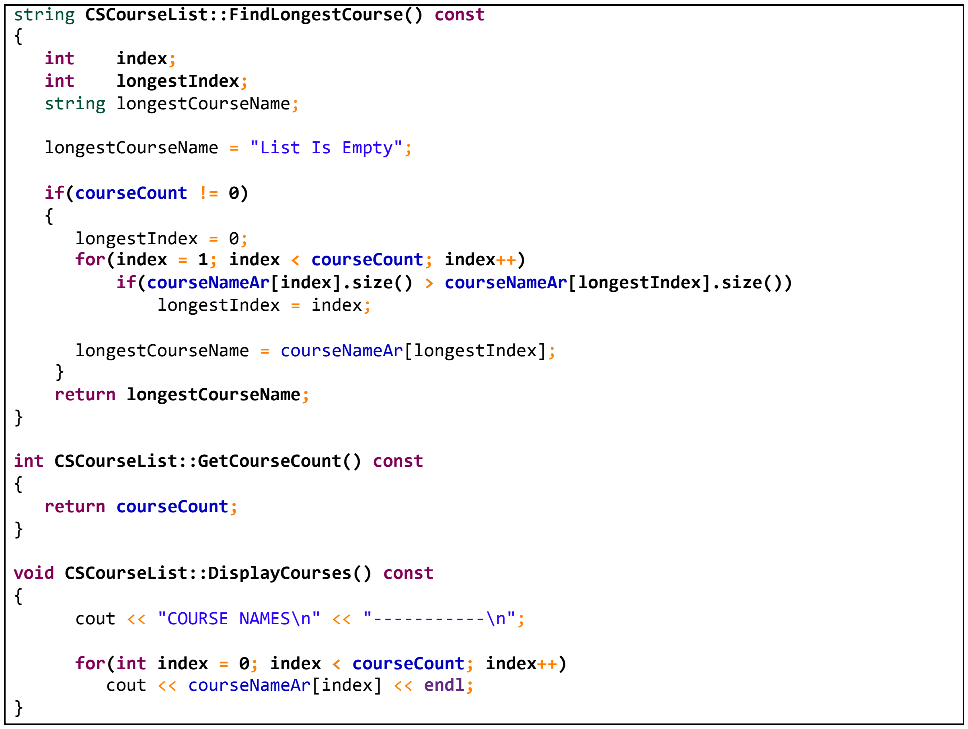


Implement the constructor, destructor & AddClass Methods

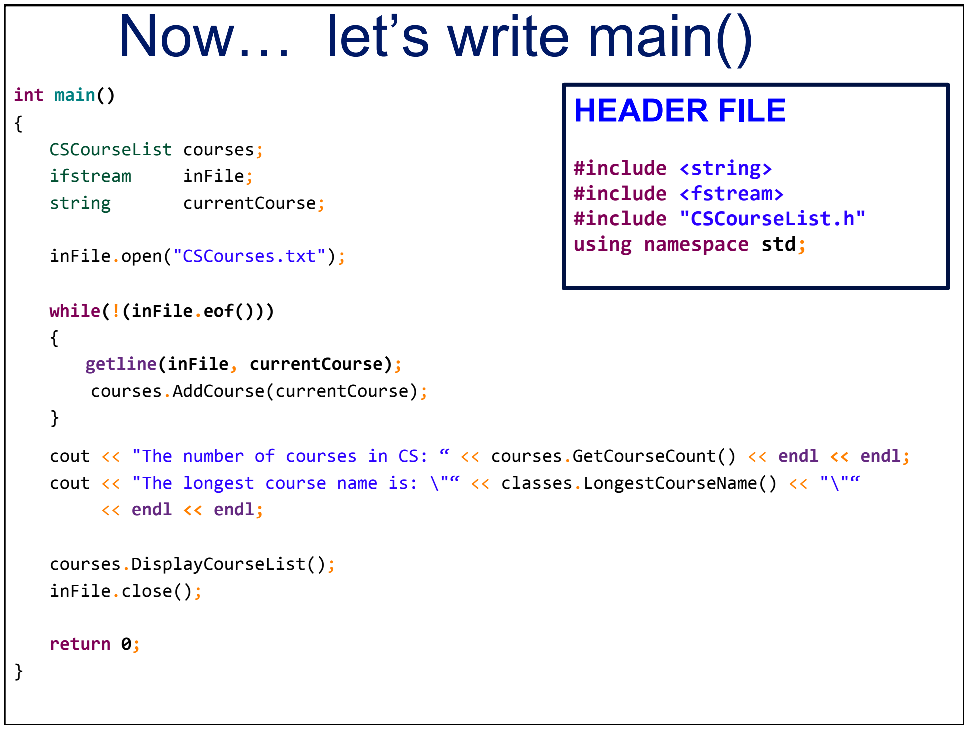
```
CSCourseList ::CSCourseList()                    /***  CONSTRUCTOR  ***/
{
   classCount = 0;
}

CSCourseList::~CSCourseList() { }                /***  DESTRUCTOR  ***/

void CSCourseList ::AddCourse(string newCourse)
{

    // Verify whether the array is not full
    if (courseCount < AR_SIZE)
    {
        // Set the data in the array
        courseNameAr[courseCount] = newCourse;

        // Update the course counter with one more class
        courseCount++;
    }
    else
    {
        cout << "Could not Add Course – array is full\n";
    }
}
```

Should we initialize the array?

Now implement the rest of the methods



```
string CSCourseList::FindLongestCourse() const
{
   int    index;
   int    longestIndex;
   string longestCourseName;

   longestCourseName = "List Is Empty";

   if(courseCount != 0)
   {
      longestIndex = 0;
      for(index = 1; index < courseCount; index++)
          if(courseNameAr[index].size() > courseNameAr[longestIndex].size())
              longestIndex = index;

      longestCourseName = courseNameAr[longestIndex];
    }
    return longestCourseName;
}

int CSCourseList::GetCourseCount() const
{
   return courseCount;
}

void CSCourseList::DisplayCourses() const
{
      cout << "COURSE NAMES\n" << "-----------\n";

      for(int index = 0; index < courseCount; index++)
         cout << courseNameAr[index] << endl;
}
```

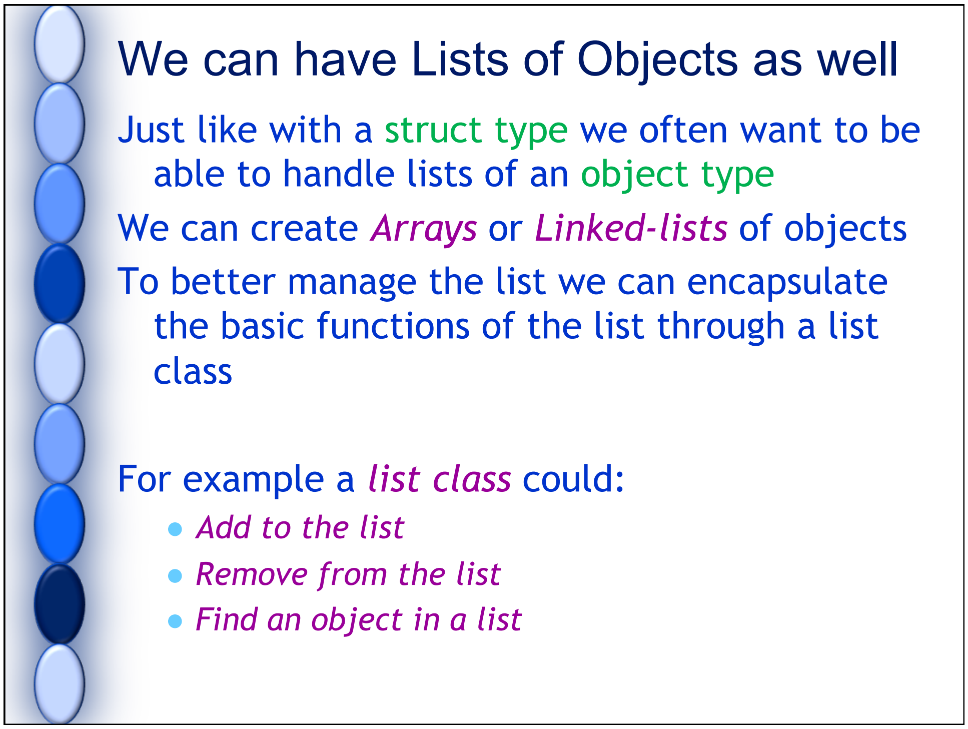


Now… let’s write main()

```
int main()
{
    CSCourseList courses;
    ifstream     inFile;
    string       currentCourse;

    inFile.open("CSCourses.txt");

    while(!(inFile.eof()))
    {
        getline(inFile, currentCourse);
         courses.AddCourse(currentCourse);
    }

    cout << "The number of courses in CS: “ << courses.GetCourseCount() << endl << endl;
    cout << "The longest course name is: \"“ << classes.LongestCourseName() << "\"“
         << endl << endl;

    courses.DisplayCourseList();
    inFile.close();

    return 0;
}
```

HEADER FILE

```
#include <string>
#include <fstream>
#include "CSCourseList.h"
using namespace std;
```



We can have Lists of Objects as well

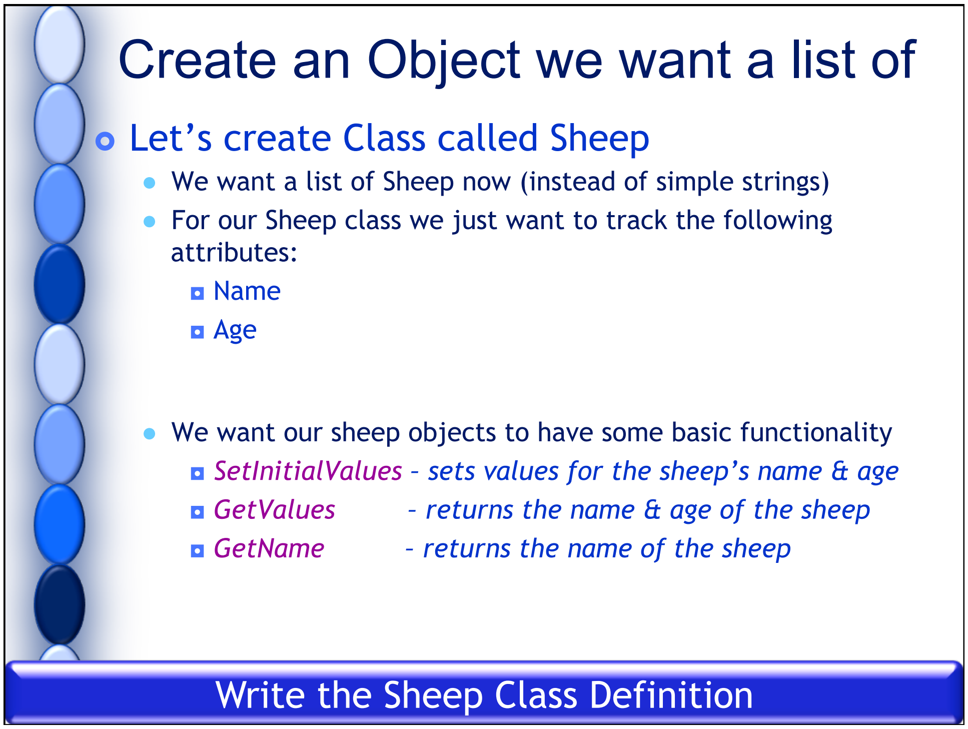
Just like with a struct type we often want to be able to handle lists of an object type

We can create *Arrays* or *Linked-lists* of objects

To better manage the list we can encapsulate the basic functions of the list through a list class

For example a *list class* could:

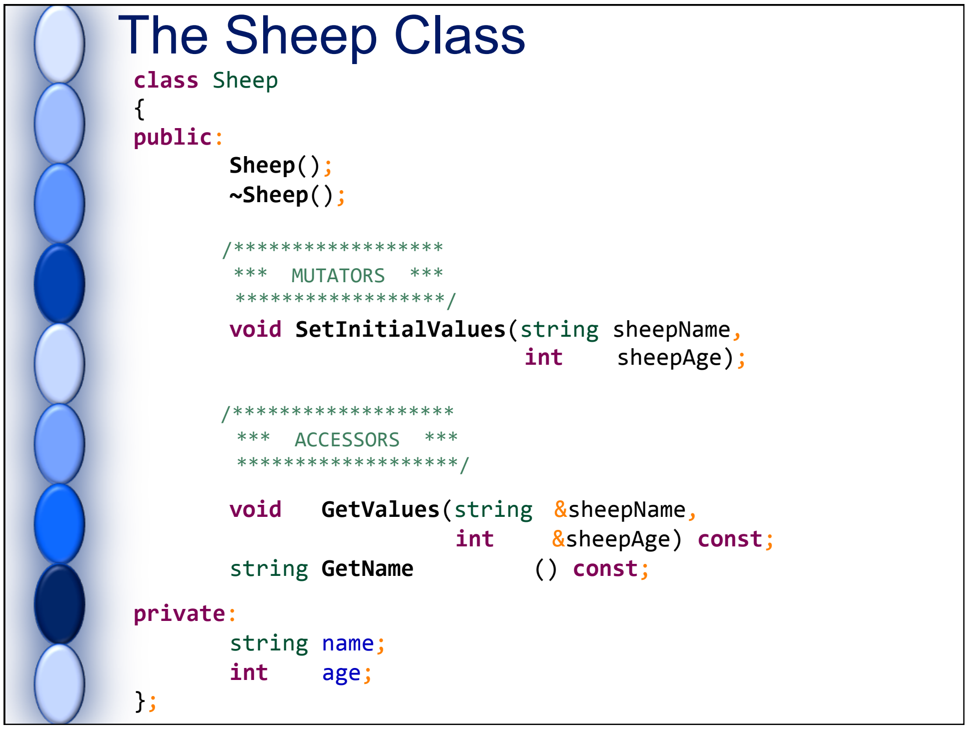
- *Add to the list*
- *Remove from the list*
- *Find an object in a list*



Create an Object we want a list of

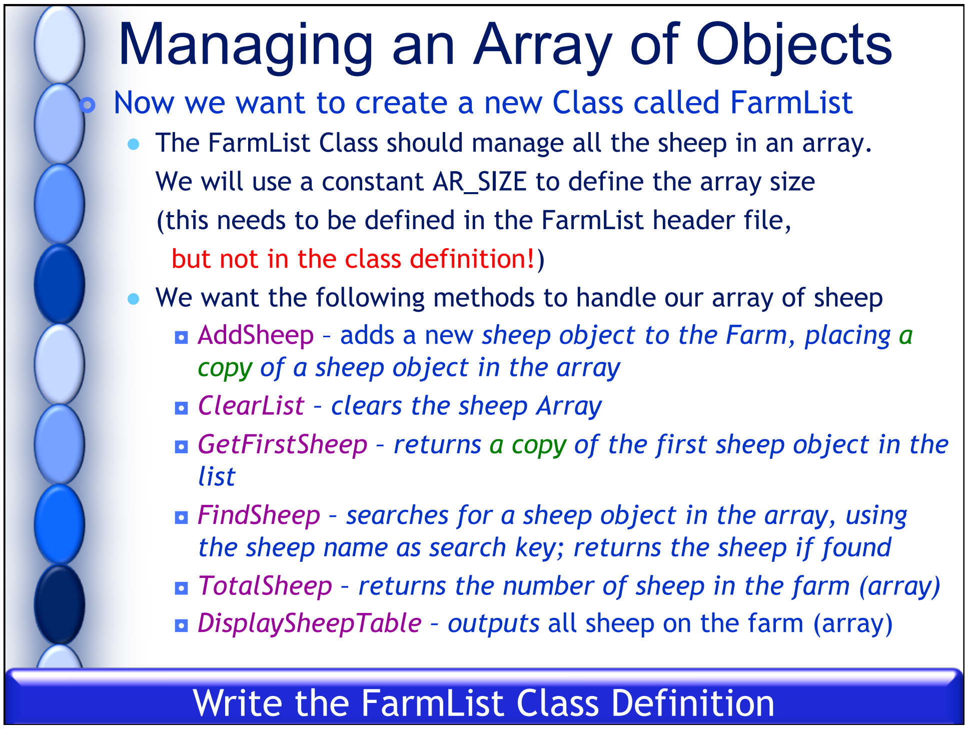
- Let’s create Class called Sheep
  - We want a list of Sheep now (instead of simple strings)
  - For our Sheep class we just want to track the following attributes:
    - Name
    - Age
  - We want our sheep objects to have some basic functionality
    - *SetInitialValues - sets values for the sheep’s name & age*
    - *GetValues - returns the name & age of the sheep*
    - *GetName - returns the name of the sheep*

Write the Sheep Class Definition



The Sheep Class

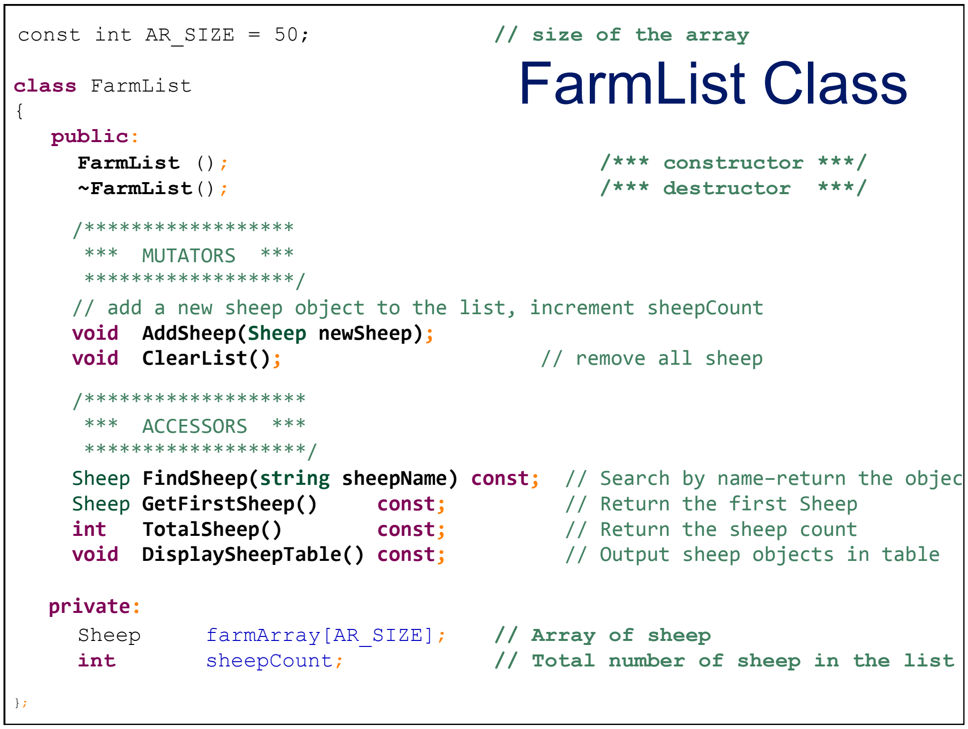
```
class Sheep
{
public:
        Sheep();
        ~Sheep();

       /*****************
        ***  MUTATORS  ***
        *****************/
        void SetInitialValues(string sheepName,
                              int    sheepAge);

       /******************
        ***  ACCESSORS  ***
        ******************/

        void   GetValues(string  &sheepName,
                         int     &sheepAge) const;
        string GetName          () const;

private:
        string name;
        int    age;
};
```



Managing an Array of Objects

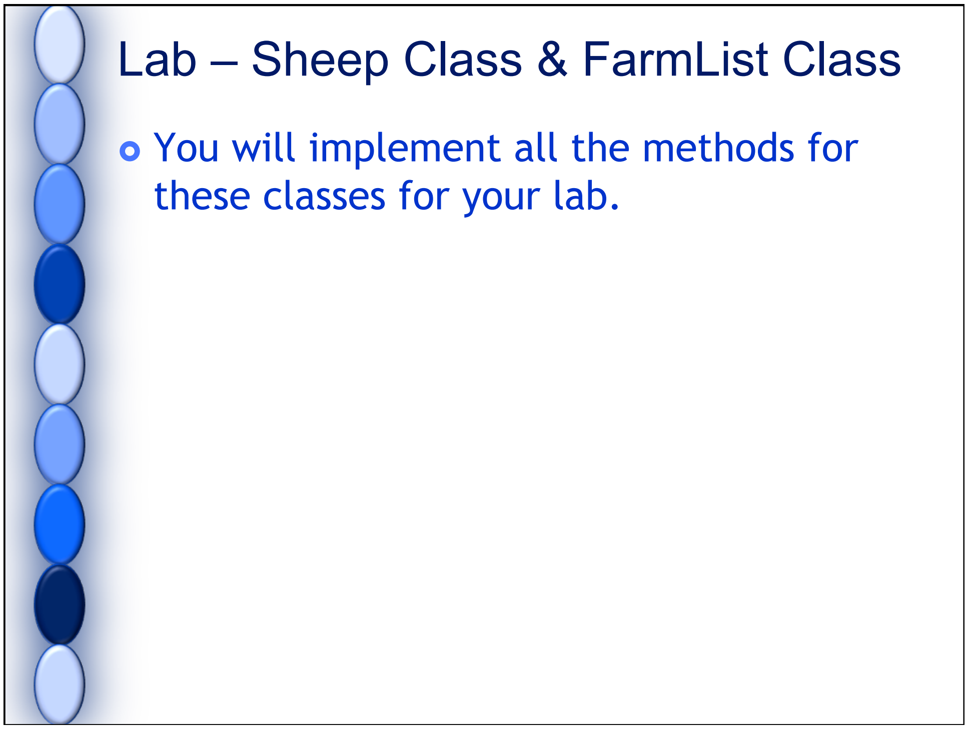
- Now we want to create a new Class called FarmList
  - The FarmList Class should manage all the sheep in an array.
    We will use a constant AR_SIZE to define the array size
    (this needs to be defined in the FarmList header file,
    but not in the class definition!)
  - We want the following methods to handle our array of sheep
    - AddSheep - adds a new *sheep object to the Farm, placing a* copy *of a sheep object in the array*
    - *ClearList - clears the sheep Array*
    - *GetFirstSheep - returns a* copy *of the first sheep object in the list*
    - *FindSheep - searches for a sheep object in the array, using the sheep name as search key; returns the sheep if found*
    - *TotalSheep - returns the number of sheep in the farm (array)*
    - *DisplaySheepTable - outputs* all sheep on the farm (array)

Write the FarmList Class Definition



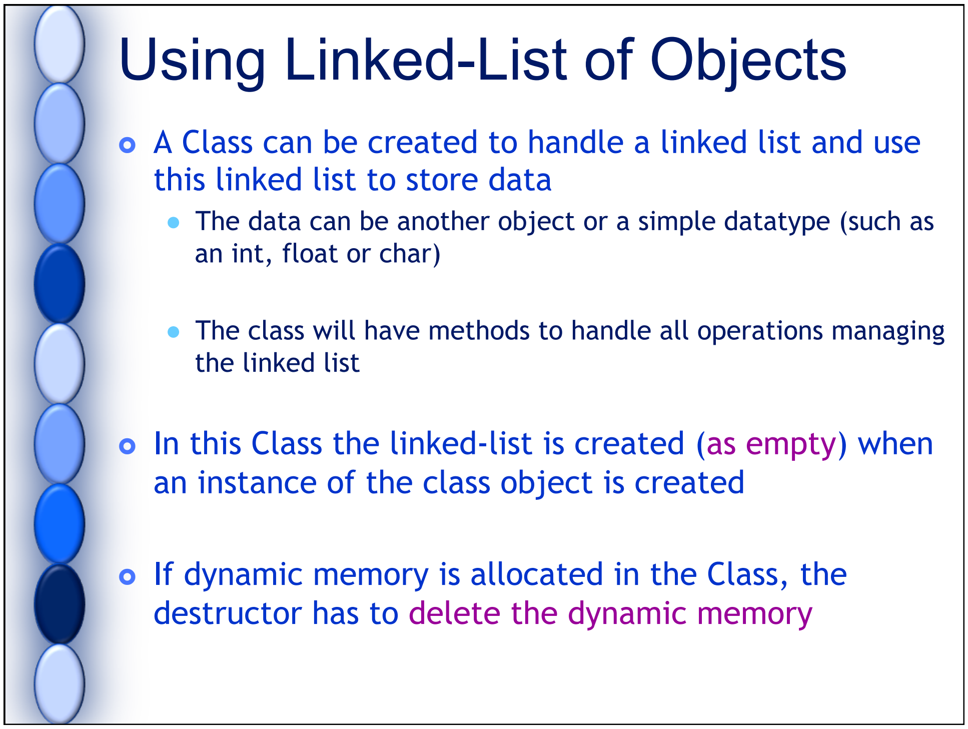
FarmList Class

```
const int AR_SIZE = 50;               // size of the array

class FarmList
{
   public:
     FarmList ();                              /*** constructor ***/
     ~FarmList();                              /*** destructor  ***/

    /******************
     ***  MUTATORS  ***
     ******************/
    // add a new sheep object to the list, increment sheepCount
    void  AddSheep(Sheep newSheep);
    void  ClearList();                     // remove all sheep

    /*******************
     ***  ACCESSORS  ***
     *******************/
    Sheep FindSheep(string sheepName) const;  // Search by name-return the objec
    Sheep GetFirstSheep()     const;          // Return the first Sheep
    int   TotalSheep()        const;          // Return the sheep count
    void  DisplaySheepTable() const;          // Output sheep objects in table

   private:
     Sheep      farmArray[AR_SIZE];    // Array of sheep
     int        sheepCount;            // Total number of sheep in the list

};
```



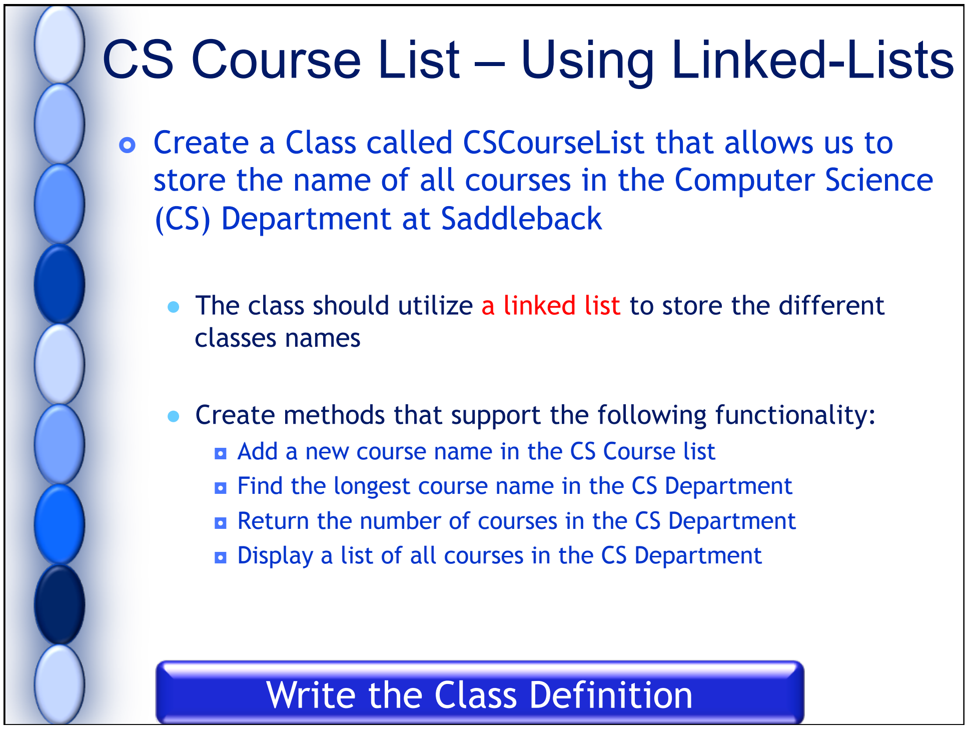
Lab – Sheep Class & FarmList Class

- You will implement all the methods for these classes for your lab.



Using Linked-List of Objects

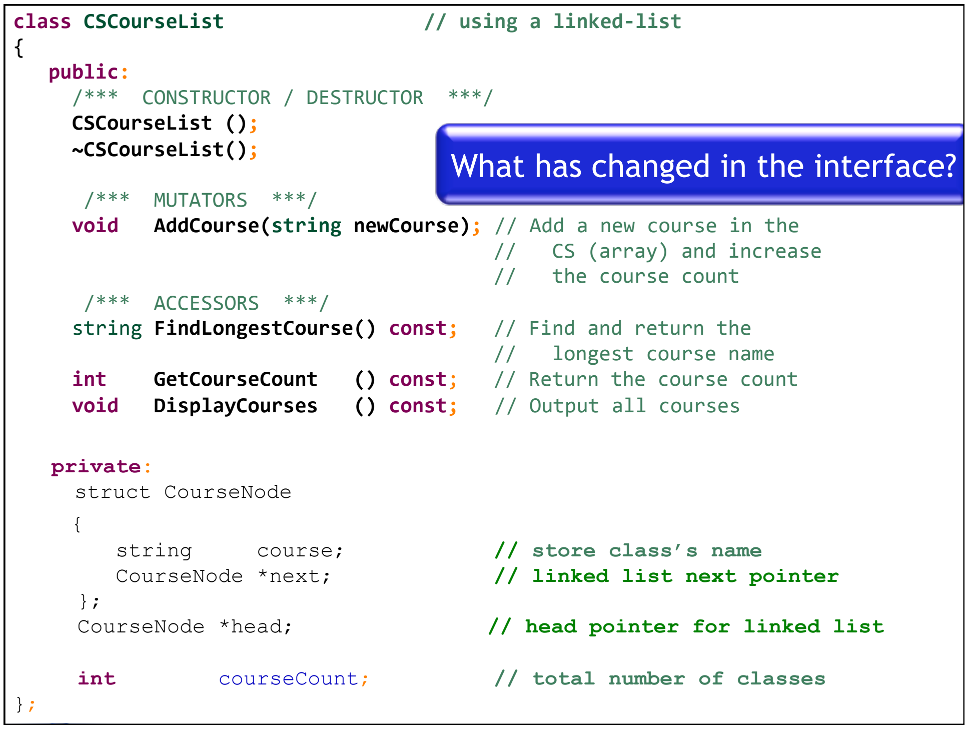
- A Class can be created to handle a linked list and use this linked list to store data
  - The data can be another object or a simple datatype (such as an int, float or char)
  - The class will have methods to handle all operations managing the linked list
- In this Class the linked-list is created (as empty) when an instance of the class object is created
- If dynamic memory is allocated in the Class, the destructor has to delete the dynamic memory



CS Course List – Using Linked-Lists

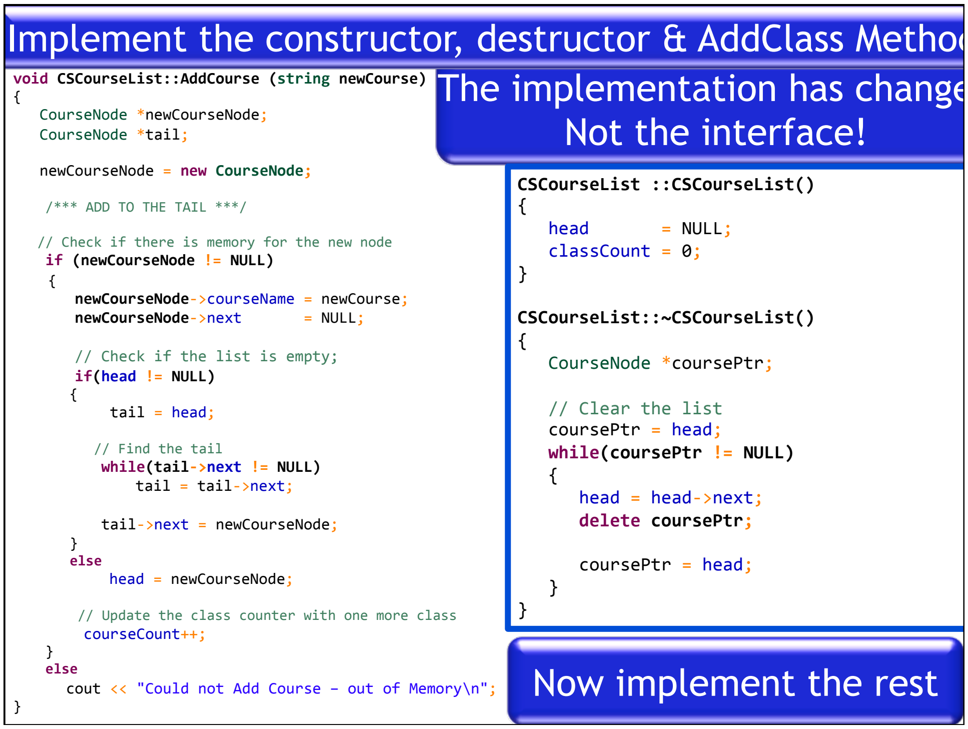
- Create a Class called CSCourseList that allows us to store the name of all courses in the Computer Science (CS) Department at Saddleback
  - The class should utilize a linked list to store the different classes names
  - Create methods that support the following functionality:
    - Add a new course name in the CS Course list
    - Find the longest course name in the CS Department
    - Return the number of courses in the CS Department
    - Display a list of all courses in the CS Department

Write the Class Definition



What has changed in the interface?

```
class CSCourseList                    // using a linked-list
{
   public:
     /***  CONSTRUCTOR / DESTRUCTOR  ***/
     CSCourseList ();
     ~CSCourseList();

      /***  MUTATORS  ***/
     void    AddCourse(string newCourse); // Add a new course in the
                                          //   CS (array) and increase
                                          //   the course count

      /***  ACCESSORS  ***/
     string FindLongestCourse() const;    // Find and return the
                                          //   longest course name
     int     GetCourseCount   () const;   // Return the course count
     void    DisplayCourses   () const;   // Output all courses

   private:
     struct CourseNode
     {
         string      course;              // store class's name
         CourseNode *next;                // linked list next pointer
      };
      CourseNode *head;                  // head pointer for linked list

      int         courseCount;            // total number of classes
};
```



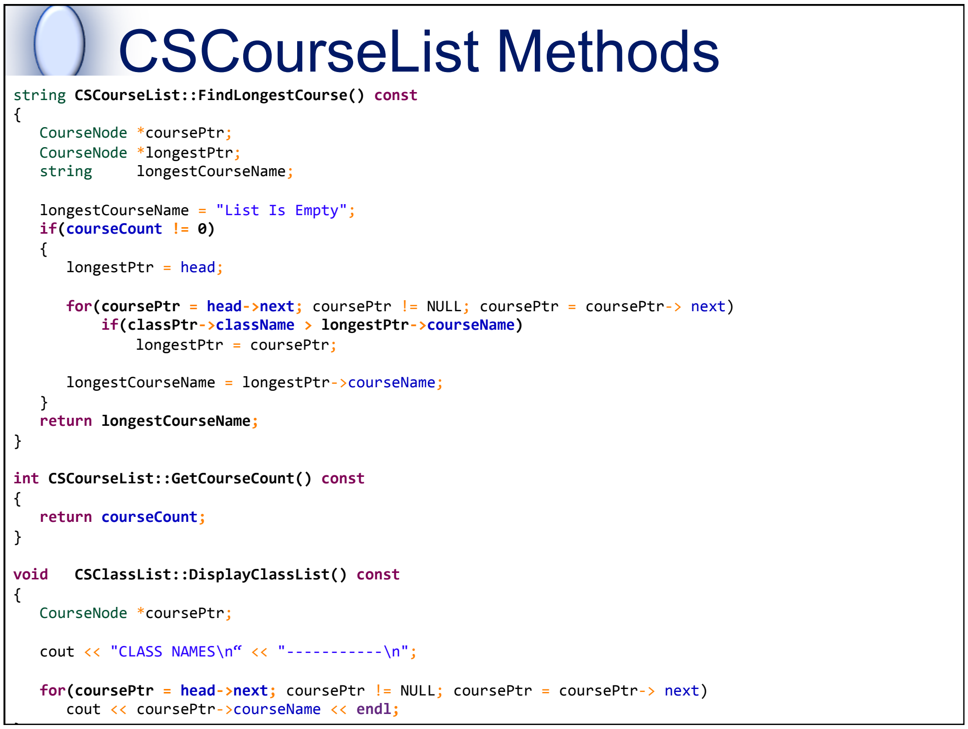
Implement the constructor, destructor & AddClass Methoc

```
void CSCourseList::AddCourse (string newCourse)
{
   CourseNode *newCourseNode;
   CourseNode *tail;

   newCourseNode = new CourseNode;

    /*** ADD TO THE TAIL ***/

   // Check if there is memory for the new node
    if (newCourseNode != NULL)
    {
       newCourseNode->courseName = newCourse;
       newCourseNode->next       = NULL;

       // Check if the list is empty;
       if(head != NULL)
       {
           tail = head;

         // Find the tail
          while(tail->next != NULL)
              tail = tail->next;

          tail->next = newCourseNode;
       }
       else
           head = newCourseNode;

       // Update the class counter with one more class
        courseCount++;
    }
    else
      cout << "Could not Add Course - out of Memory\n";
}
```

The implementation has change
Not the interface!

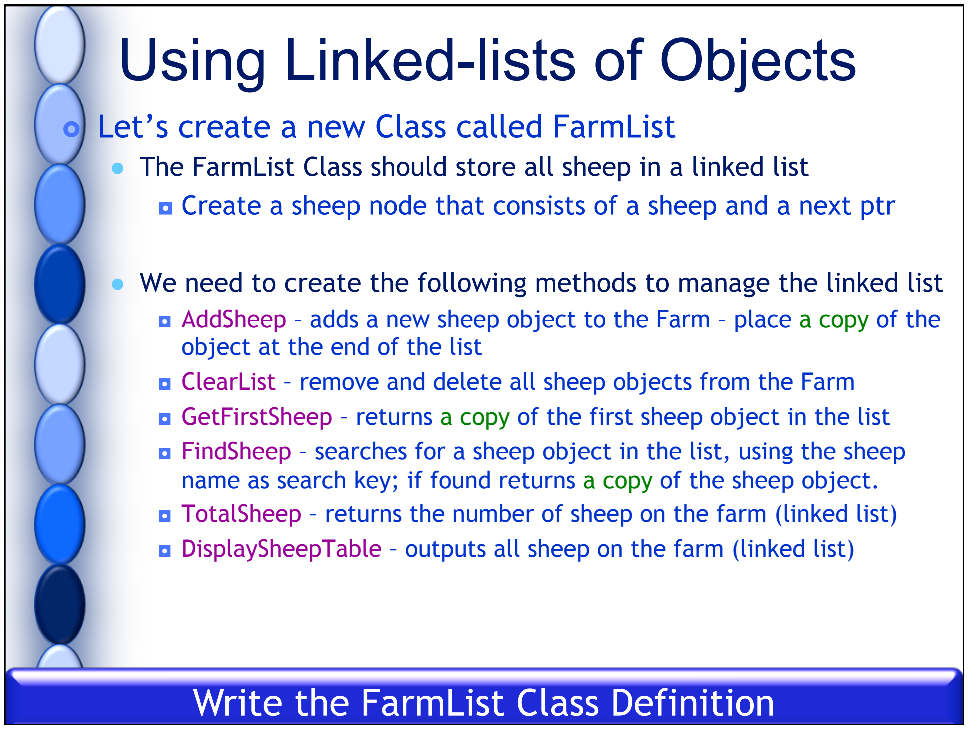
```
CSCourseList ::CSCourseList()
{
   head       = NULL;
   classCount = 0;
}

CSCourseList::~CSCourseList()
{
   CourseNode *coursePtr;

   // Clear the list
   coursePtr = head;
   while(coursePtr != NULL)
   {
      head = head->next;
      delete coursePtr;

      coursePtr = head;
   }
}
```

Now implement the rest



CSCourseList Methods

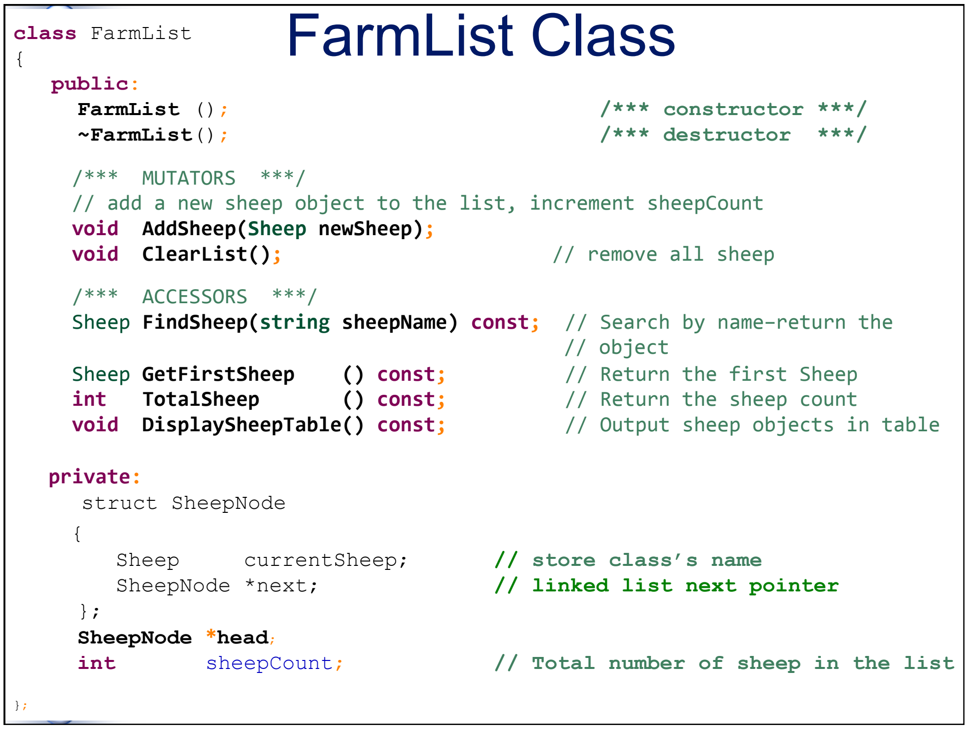
```
string CSCourseList::FindLongestCourse() const
{
   CourseNode *coursePtr;
   CourseNode *longestPtr;
   string      longestCourseName;

   longestCourseName = "List Is Empty";
   if(courseCount != 0)
   {
      longestPtr = head;

      for(coursePtr = head->next; coursePtr != NULL; coursePtr = coursePtr-> next)
          if(classPtr->className > longestPtr->courseName)
              longestPtr = coursePtr;

      longestCourseName = longestPtr->courseName;
   }
   return longestCourseName;
}

int CSCourseList::GetCourseCount() const
{
   return courseCount;
}

void   CSClassList::DisplayClassList() const
{
   CourseNode *coursePtr;

   cout << "CLASS NAMES\n" << "-----------\n";

   for(coursePtr = head->next; coursePtr != NULL; coursePtr = coursePtr-> next)
      cout << coursePtr->courseName << endl;
```



Using Linked-lists of Objects

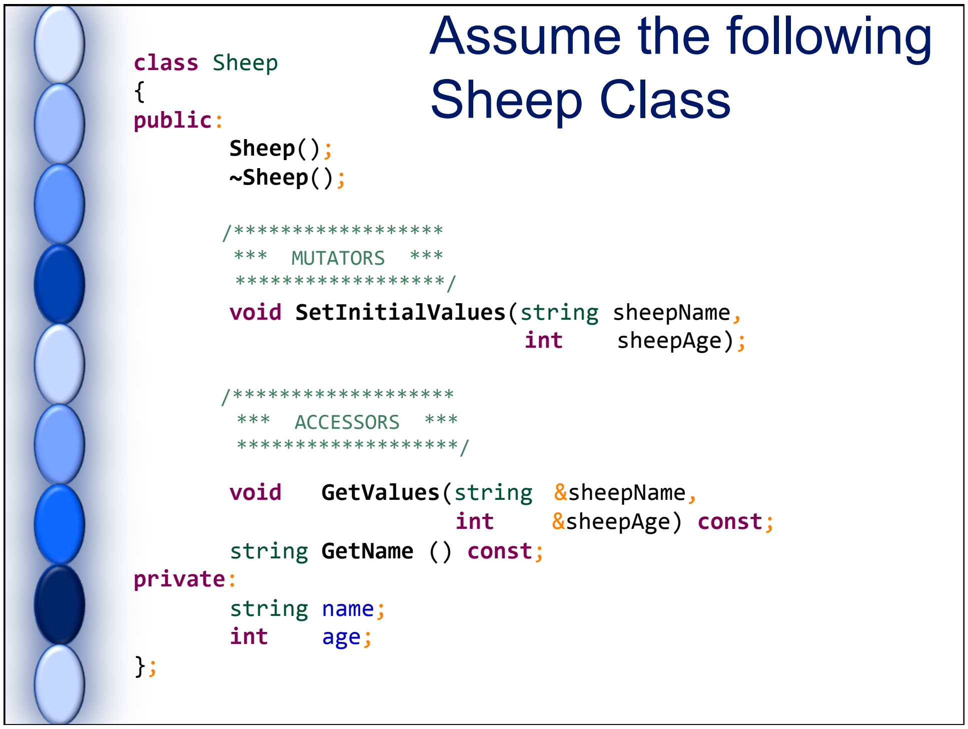
- Let’s create a new Class called FarmList
  - The FarmList Class should store all sheep in a linked list
    - Create a sheep node that consists of a sheep and a next ptr
  - We need to create the following methods to manage the linked list
    - AddSheep - adds a new sheep object to the Farm - place a copy of the object at the end of the list
    - ClearList - remove and delete all sheep objects from the Farm
    - GetFirstSheep - returns a copy of the first sheep object in the list
    - FindSheep - searches for a sheep object in the list, using the sheep name as search key; if found returns a copy of the sheep object.
    - TotalSheep - returns the number of sheep on the farm (linked list)
    - DisplaySheepTable - outputs all sheep on the farm (linked list)

Write the FarmList Class Definition



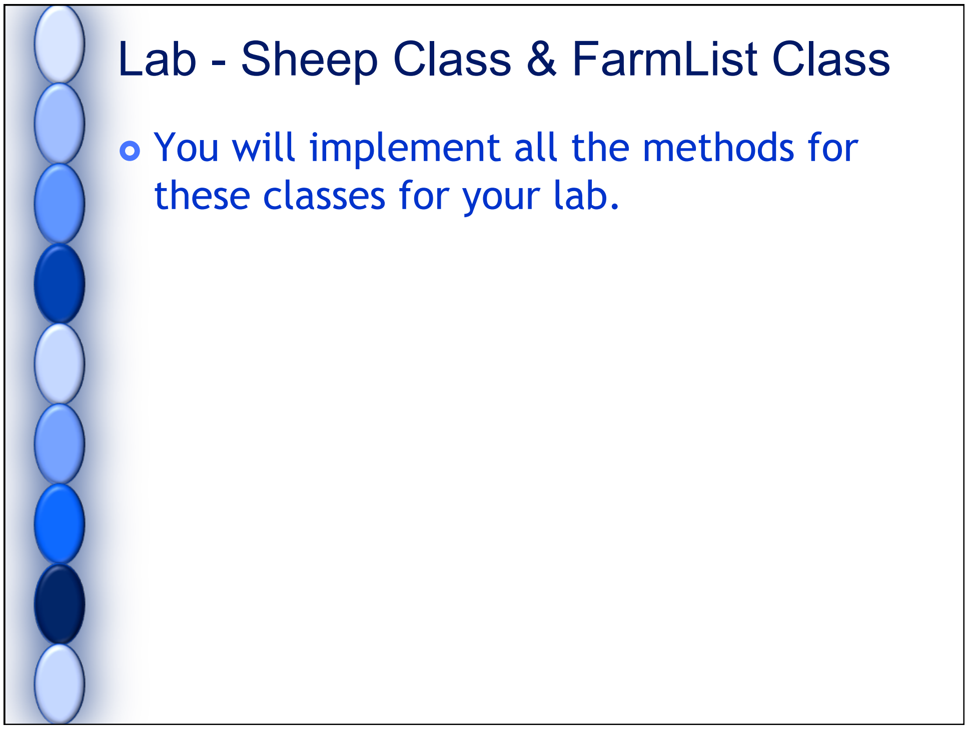
FarmList Class

```
class FarmList
{
   public:
     FarmList ();                                /*** constructor ***/
     ~FarmList();                                /*** destructor  ***/

    /***  MUTATORS  ***/
    // add a new sheep object to the list, increment sheepCount
    void  AddSheep(Sheep newSheep);
    void  ClearList();                         // remove all sheep

    /***  ACCESSORS  ***/
    Sheep FindSheep(string sheepName) const;  // Search by name-return the
                                              // object
    Sheep GetFirstSheep    () const;          // Return the first Sheep
    int   TotalSheep       () const;          // Return the sheep count
    void  DisplaySheepTable() const;          // Output sheep objects in table

   private:
     struct SheepNode
    {
        Sheep     currentSheep;        // store class's name
        SheepNode *next;               // linked list next pointer
     };
     SheepNode *head;
     int        sheepCount;            // Total number of sheep in the list

};
```



Assume the following Sheep Class

```
class Sheep
{
public:
        Sheep();
        ~Sheep();

       /******************
        ***  MUTATORS  ***
        ******************/
        void SetInitialValues(string sheepName,
                              int    sheepAge);

       /******************
        ***  ACCESSORS  ***
        ******************/

        void   GetValues(string  &sheepName,
                         int     &sheepAge) const;
        string GetName () const;
private:
        string name;
        int    age;
};
```



Lab - Sheep Class & FarmList Class

- You will implement all the methods for these classes for your lab.