Example of classes with pointers and array

Often you create a class that has an array of pointers to objects inside it. The reason to use pointers in this case is so that you have one copy of an item instead of creating multiple instances of it. For example, when playing cards, you only want one copy of each card and you pass that from deck to hand. Or when you have a course with students, you only want one copy of each student so there are not duplicate instances of information.

This example creates a student class, a course class, and then shows how students are added to the course and listed. The code listed here is in Moodle in the folder Course Example in week 4.

Class Student

Here is the header file "student.h". There is no matching .cpp file because I made all methods in-line.

```
#ifndef STUDENT_H
#define STUDENT_H

    // This is included so I can use and define strings
#include <string>

class Student {

  private:

            //The only variables for a student are the name and ID number
        std::string name;
        int idNum;

  public:

          //The constructor creates a new student with a name and ID
          //If no name or ID was entered, an empty string and 0 are used
          //The accessor methods simply return the associated value
        Student(std::string nm = "", int id = 0) {name = nm; idNum = id;}
        int getNum() {return idNum;}
        std::string getName() {return name;}

};
#endif // STUDENT_H
```

As you can see, this class simply contains the information for a student and returns it as needed.

Class Course

The course class is used to hold the list of students. It starts out with no students and allows them to be added one-by-one. This class also has methods to return how many students are enrolled, what their names are, and to reset the course to an empty one for the start of a new term.

Here is the file "course.h".

```
#ifndef COURSE_H
#define COURSE_H

    //Include the header for the student class so you can access objects of type student
#include "student.h"

class Course {
  public:
            //Define a public constant of how many students in a course
        const static int MAX_STUDENTS = 25;

  private:

            //The class variables are an array of students
            // and a count of how many are in the array

            //Note that the array is an array of pointers
            // as we said above, we want to pass a student around, not make a copy of it
        Student * studentsInClass[MAX_STUDENTS];
        int classSize;

  public:
            //The constructor simple initializes the count to zero
        Course();

            //Add student is passed a pointer to a student
            //you must give it a pointer, since that is what we are storing
        bool addStudent(Student * s);

            //Basic accessor methods
        int getClassSize() {return classSize;}
        std::string getStudents();

            //Reset the course to empty and start over
        void resetCourse();
};
#endif // COURSE_H
```

And then the file “course.cpp”:

```
//Include the header, as always
#include "course.h"


 //The constructer, start new course with no students

//Note that I don't bother to initialize the array, I will keep track of
// which slots are filled and only access those.

//Of course, I could initialize the array to NULLs if I wanted to, but
// this implementation would gain nothing by doing so.
Course::Course()
{
  classSize = 0;
}

//The method to add a new student

//Check first to see if there is any room
//If there is, add the student to the array in the next available slot
// increment the count of number of students, and return success

//If no room, do nothing and return failure
bool Course::addStudent(Student * s)
{
  if ( classSize < MAX_STUDENTS )
  {
        studentsInClass[classSize] = s;
        classSize++;
        return true;
  }
  return false;
}

//This method returns a list of the students, one per line
//Note that it only goes through the array as far as there are valid names
std::string Course::getStudents()
{
  std::string theStudents = "";

  for ( int i = 0; i < classSize; i++ )
  {
        theStudents += studentsInClass[i]->getName() + "\n";
  }
  return theStudents;
}

//Start over with new course, setting the number of students to zero

//Here, I chose to delete the students as you would delete the cards
// from BlackJack hand when resetting it.

//Normally, we would not delete students when they finish one course,
// but would keep them around for other classes.
void Course::resetCourse()
{
  for ( int i = 0; i < MAX_STUDENTS; i++ )
  {
        delete studentsInClass[i];
  }
  classSize = 0;
}
```

Main Function

The main function uses the course class. It starts by creating three students, then adds them to a new course, lists the course members, then resets the course and shows that all goes back to the beginning. If you tried to access the students after the course is reset, you would find that they no longer exist since they were deleted in the course reset method.

```
#include <iostream>
#include "course.h"

using namespace std;

int main()
{
        //define a new course, call it dataStructures, it starts empty
    Course dataStructures;

        //create three students. Note the names are of type pointer to student
    Student * mary = new Student("Mary Smith", 99);

    Student * bob = new Student("Bob Jones", 66);

    Student * sam = new Student ("Samwise Gamgee", 42);

        //Output the starting condition of the class
        //It should have no students and an empty list
  cout << "Before starting, course size is " << dataStructures.getClassSize() << endl;
  cout << "and the class list is:" << endl << dataStructures.getStudents() << endl;

        //Add the three students to the course, one at a time
  dataStructures.addStudent(mary);
  dataStructures.addStudent(bob);
  dataStructures.addStudent(sam);

        //Display the course with students
  cout << "After adding, course size is " << dataStructures.getClassSize() << endl;
  cout << "and the class list is:" << endl << dataStructures.getStudents() << endl;

        //reset the course for a new term
  dataStructures.resetCourse();

        //Once again, it has no students
  cout << "After reset, course size is "<< dataStructures.getClassSize() << endl;
  cout << "and the class list is:" << endl << dataStructures.getStudents() << endl;


    return 0;
}
```

Output

Here is a copy of the output of the above program running.

```
Before starting, course size is 0
and the class list is:

After adding, course size is 3
and the class list is:
Mary Smith
Bob Jones
Samwise Gamgee

After reset, course size is 0
and the class list is:


Process returned 0 (0x0)   execution time : 0.005 s
```