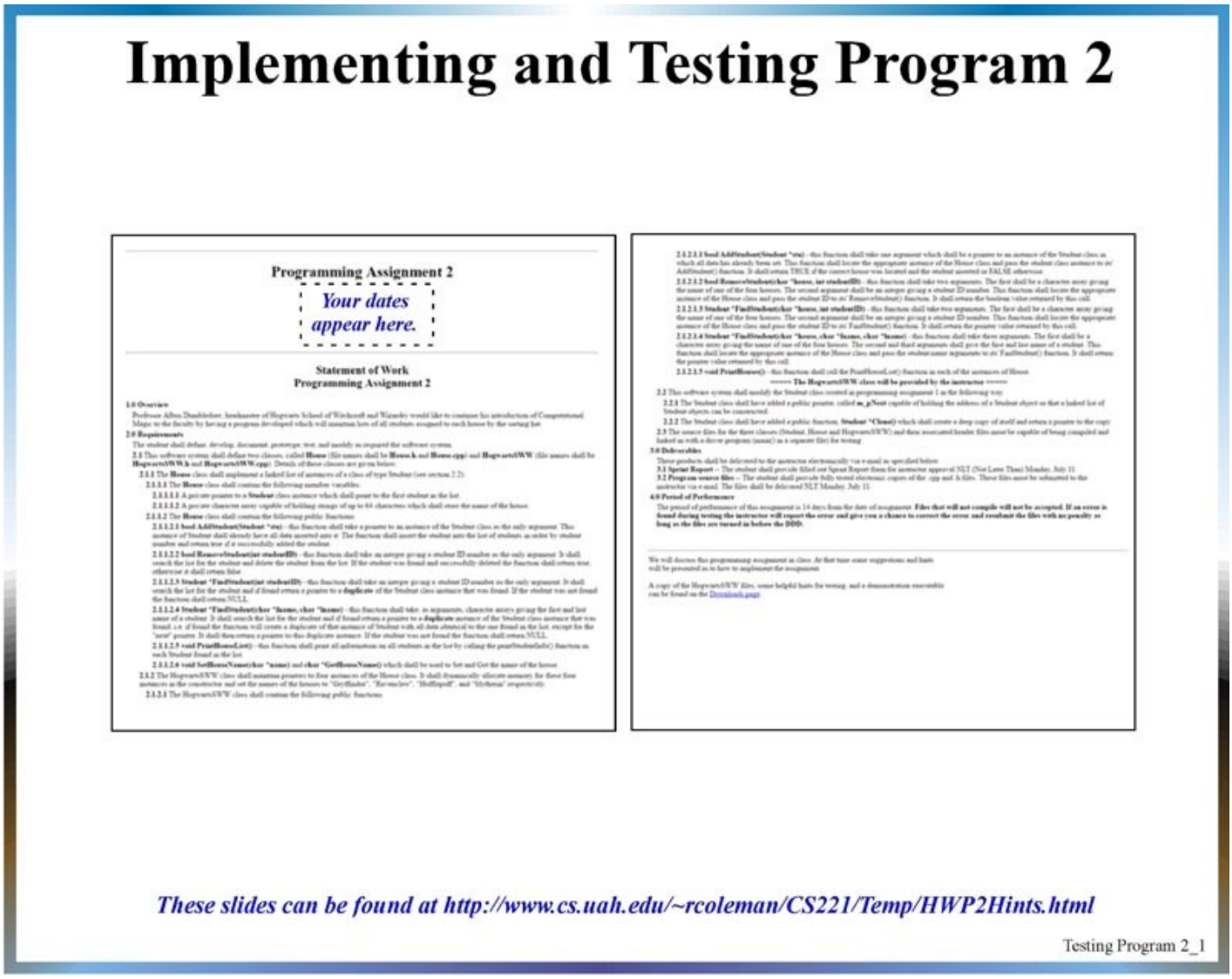
Testing Program 2 Slides

The slides below are extras from a special class presentation.

Note: These slides will be here only for a few days.

Implementing and Testing Programming Assignment 2



Implementing and Testing Program 2

Required Files*

Source file (.cpp) House.cpp
Header file (.h) House.h
Source file (.cpp) HogwartsSWW.cpp
Header file (.h) HogwartsSWW.h
Source file (.cpp) Student.cpp
Header file (.h) Student.h

House Private Member Variables

(1) A pointer to Student called m_pHead
(2) A char array called m_sHouseName able to hold strings of up to 64 characters.

House Public Methods*

(1) House() and ~House() - default constructor and destructor.
(2) bool AddStudent(Student *stu) - add a student to the linked list of Student objects in order by ID.
(3) Student *RemoveStudent(int studentID) - search for a student by ID and remove the student from the linked list of Student objects if found or NULL if not found.
(4) Student *FindStudent(int studentID) - search for a student by ID and return a deep copy of the student if found or NULL if not found.
(5) Student *FindStudent(char *mName, char *fName) - search for a student by name (magical and family) and return a deep copy of the student if found or NULL if not found.
(6) void PrintHouseList() - this function will call the PrintStudentInfo function in each of the Student objects in the list to print all information on all students stored in the list.
(7) void SetHouseName(char *name) and char *GetHouseName() - these functions are used to set and get the name of the House.

*Spelling must be exactly as specified.

Testing Program 2_2

Implementing and Testing Program 2

HogwartsSWW.h and .cpp will be provided by the Instructor.

HogwartsSWW Private Member Variables

(1) House *m_pGryffindor;
(2) House *m_pRavenclaw;
(3) House *m_pHufflepuff;
(4) House *m_pSlytherin;
Pointers to four instances of House.

HogwartsSWW Public Methods*

(1) HogwartsSWW() and ~HogwartsSWW() - default constructor and destructor.
(2) bool AddStudent(Student *stu) - locate the correct house based on the name of the house set in the instance of Student and call the AddStudent() function in that house returning the bool value returned by that call.
(3) Student *RemoveStudent(char *house, int studentID) - locate the correct house and call the RemoveStudent() function in that house returning the pointer returned by that call.
(4) Student *FindStudent(char *house, int studentID) - locate the correct house and call the FindStudent() function in that house returning the pointer returned by that call.
(5) Student *FindStudent(char *house, char *mName, char *fName) - locate the correct house and call the FindStudent() function in that house returning the pointer returned by that call.
(6) void PrintHouseList() - this function will call the PrintHouseList function in each of the House objects.

HogwartsSWW.h and .cpp will be provided by the Instructor.

Testing Program 2_3

Implementing and Testing Program 2

HogwartsSWW implementation.

```
//-----------------------------------------------------------------------------------------
// HogwartsSWW.h
// Inteface definition file for the HogwartsSWW class.
// Author: Dr. Rick Coleman
// This file is provided by the instructor for use in programming assignment 2.
//-----------------------------------------------------------------------------------------
#pragma once

#include "House.h"
#include "Student.h"

class HogwartsSWW
{
	private:
		House	*m_pGryffindor;
		House	*m_pRavenclaw;
		House	*m_pHufflepuff;
		House	*m_pSlytherin;

	public:
		HogwartsSWW();                                          // Default constructor
		~HogwartsSWW();                                         // Destructor
		bool AddStudent(Student *stu);                          // Add a student to a house
		Student *RemoveStudent(char *house, int studentID); // Remove a student given an ID
		Student *FindStudent(char *house, int studentID);// Find a student given the student ID
		Student *FindStudent(char *house, char *fname,   // Overloaded find function.  Find a student
		                     char *lname);               //  given the first and last names
		void PrintHouses();
};
```

Testing Program 2_4

Implementing and Testing Program 2

HogwartsSWW implementation.

```
//-----------------------------------------------------------------------------
// HogwartsSWW.cpp
// Implementation file for the HogwartsSWW class.
// Author: Dr. Rick Coleman
// This file is provided by the instructor for use in programming assignment 2.
//-----------------------------------------------------------------------------
#include "HogwartsSWW.h"
#include <iostream>
#include <stdio.h>

using namespace std;

//---------------------------------------------------
// Default constructor - Create the 4 houses
//---------------------------------------------------
HogwartsSWW::HogwartsSWW()
{
        m_pGryffindor = new House();
        m_pGryffindor->SetHouseName("Gryffindor");
        m_pRavenclaw = new House();
        m_pRavenclaw->SetHouseName("Ravenclaw");
        m_pHufflepuff = new House();
        m_pHufflepuff->SetHouseName("Hufflepuff");
        m_pSlytherin = new House();
        m_pSlytherin->SetHouseName("Slytherin");
}

//---------------------------------------------------
// Destructor - Delete the 4 houses
//---------------------------------------------------
HogwartsSWW::~HogwartsSWW()
{
        delete m_pGryffindor;
        delete m_pRavenclaw;
        delete m_pHufflepuff;
        delete m_pSlytherin;
}
```

The destructor in House is responsible for deleting all instances of Student still in the list.

Testing Program 2_5

Implementing and Testing Program 2

HogwartsSWW implementation.

```
//-------------------------------------------------------------
// AddStudent()  - Adds a student to a house
// Args: stu - pointer to a Student object
// Returns: bool - true if add was successful
//-------------------------------------------------------------
bool HogwartsSWW::AddStudent(Student *stu)
{
        char house[64];
        stu->getHouse(house);
        if(strcmp(house, "Gryffindor") == 0)
                return m_pGryffindor->AddStudent(stu);
        else if(strcmp(house, "Ravenclaw") == 0)
                return m_pRavenclaw->AddStudent(stu);
        else if(strcmp(house, "Hufflepuff") == 0)
                return m_pHufflepuff->AddStudent(stu);
        else if(strcmp(house, "Slytherin") == 0)
                return m_pSlytherin->AddStudent(stu);
        else
                return false;
}
//-------------------------------------------------------------------
// RemoveStudent() - Removes a student from a house
// Args: house - Name of the student's house
//       stu - pointer to a Student object
// Returns: Pointer to student removed
//-------------------------------------------------------------------
Student *HogwartsSWW::RemoveStudent(char *house, int studentID)
{
        if(strcmp(house, "Gryffindor") == 0)
                return m_pGryffindor->RemoveStudent(studentID);
        else if(strcmp(house, "Ravenclaw") == 0)
                return m_pRavenclaw->RemoveStudent(studentID);
        else if(strcmp(house, "Hufflepuff") == 0)
                return m_pHufflepuff->RemoveStudent(studentID);
        else if(strcmp(house, "Slytherin") == 0)
                return m_pSlytherin->RemoveStudent(studentID);
        else
                return false;
}
```

Testing Program 2_6

Implementing and Testing Program 2

HogwartsSWW implementation.

```
//---------------------------------------------------------------
// FindStudent() - Find a student in a house
// Args: house - Name of the student's house
//       stu - pointer to a Student object
// Returns: Pointer to a clone of the student record
//---------------------------------------------------------------
Student *HogwartsSWW::FindStudent(char *house, int studentID)
{
        if(strcmp(house, "Gryffindor") == 0)
                return m_pGryffindor->FindStudent(studentID);
        else if(strcmp(house, "Ravenclaw") == 0)
                return m_pRavenclaw->FindStudent(studentID);
        else if(strcmp(house, "Hufflepuff") == 0)
                return m_pHufflepuff->FindStudent(studentID);
        else if(strcmp(house, "Slytherin") == 0)
                return m_pSlytherin->FindStudent(studentID);
        else
                return NULL;
}
//---------------------------------------------------------------
// FindStudent() - Find a student in a house
// Args: house - Name of the student's house
//       fname - Student's first name
//       lname - Student's last name
// Returns: Pointer to a clone of the student record
//---------------------------------------------------------------
Student *HogwartsSWW::FindStudent(char *house, char *fname, char *lname)
{
        if(strcmp(house, "Gryffindor") == 0)
                return m_pGryffindor->FindStudent(fname, lname);
        else if(strcmp(house, "Ravenclaw") == 0)
                return m_pRavenclaw->FindStudent(fname, lname);
        else if(strcmp(house, "Hufflepuff") == 0)
                return m_pHufflepuff->FindStudent(fname, lname);
        else if(strcmp(house, "Slytherin") == 0)
                return m_pSlytherin->FindStudent(fname, lname);
        else
                return NULL;
}
```

Testing Program 2_7

Implementing and Testing Program 2

HogwartsSWW implementation.

```
//-------------------------------------------------------------------------
// PrintHouses()
// Purpose: Print all information on all students in all houses
// Args: None
// Returns: void
//-------------------------------------------------------------------------
void HogwartsSWW::PrintHouses()
{
        cout << "Students at Hogwarts\n\n";
        m_pGryffindor->PrintHouseList();
        cout << "\nPress any key to see the next house listing...";
        getchar();
        cout << "\n\n";
        m_pRavenclaw->PrintHouseList();
        cout << "\nPress any key to see the next house listing...";
        getchar();
        cout << "\n\n";
        m_pHufflepuff->PrintHouseList();
        cout << "\nPress any key to see the next house listing...";
        getchar();
        cout << "\n\n";
        m_pSlytherin->PrintHouseList();
}
```

Testing Program 2_8

Implementing and Testing Program 2

Modifications to class Student from Program 1 for Program 2

Character Public Member Variables*

(1) Addition of a pointer to a Student called m_pNext

Character Public Member Functions*

(1) Addition of the function Student *Clone();

The Clone function must make a deep copy of itself. That means all instances of char arrays of class names must also be duplicated. You cannot just copy over the pointers.

```
Student *Student::Clone()
{
        Student *theClone = new Student();          // Create a new instance of Student

        theClone->setStudentID(m_iStudentID); // Copy the ID
        theClone->setName(m_sMagicalName, m_sWizardFamilyName); // Copy the student name
        // Do the same thing for the House Name
        // Duplicate all the classes and grades
        for(int i=0; i<8; i++)
        {
                // If the class name at this index is not NULL
                {
                        // Tell the new instance of Student to set its class name at index i to this class name
                        theClone->setClass(i, m_sClasses[i]);
                        // Tell the new instance of Student to set its class grade at index i to this class grade
                }
        }
        return theClone;
}
```

You cannot copy and paste this code as in into your Student class and have it work correctly. Note the comments for the additional code you must add.

*Spelling must be exactly as specified.

Testing Program 2_9

Implementing and Testing Changes to Student in Program 2

This only covers the final testing of the completed application.
It does not include the testing required at the end of Sprint 1.

Testing the Clone function:

Student *Clone()

```
Student *stu1 = new Student();
// Set all variables include names and grades
//   for at least 3 classes.
Student *stu2; // Create a pointer to a Student
stu2 = stu1->Clone();  // Make the duplicate
delete stu1;     // Delete this one
// Call all the get functions in stu2 to see if you
//    get back the correct information.
```

1. Create an instance of Student in main and set all the variables including at least 3 classes.
2. Create an pointer to an instance of Student and set it to what is returned by calling the Clone function in the first instance of Student.
3. Call each of the get functions and compare the returned values to the expected values.

Implementing and Testing HogwartsSWW in Program 2

You do not have to test HogwartsSWW. It is being provided by the instructor who has already thoroughly tested it and confirmed that it works correctly.

**Remember to comment out all the debug cout statements BEFORE turning in your files.*

Testing Program 2_10

Implementing and Testing House in Program 2

This only covers the final testing of the completed application.
It does not include the testing required at the end of Sprint 1.

Testing the get/set house name functions:

void SetHouseName(char *name)
char *GetHouseName()

```
House *h = new House();
h->SetHouseName("Gryffindor");
char tempName[64];
strcpy(tempName, h->GetHouseName());
if(strcmp(tempName, "Gryffindor") == 0)
        cout << "Test was successful." << endl;
```

1. Create an instance of House.
2. Call the SetHouseName function passing in a name for the house.
3. In the SetHouseName function cout the private m_sHouseName after setting it.
4. Call the GetHouseName and compare the name returned with the expected name.

Testing the PrintHouseList function:

void PrintHouseList()

1. After adding all the students to the list as described in the next test call the function. Compare the results printed to what was expected.

**Remember to comment out all the debug cout statements BEFORE turning in your files.*

Testing Program 2_11

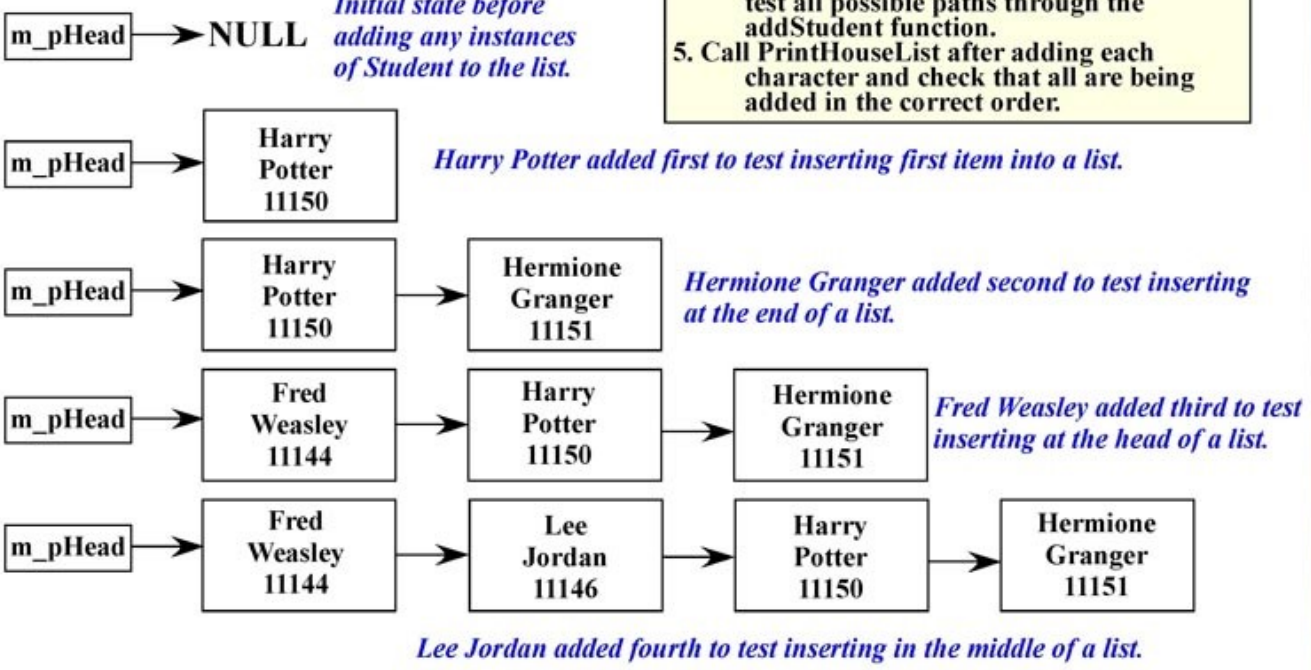
Implementing and Testing House in Program 2

Sample list used in the instructor's test program.

1. Create an instance of House in main.
2. Create and set values in an instance of Student This includes adding a number of classes to the student
3. Call the addStudent function to add the student to the list.
4. Repeat steps 2 and 3 to add a total of 4 instances of Student to the list. These must be added in the correct order to test all possible paths through the addStudent function.
5. Call PrintHouseList after adding each character and check that all are being added in the correct order.

Testing the AddStudent function:

bool AddStudent(Student *stu)



Testing Program 2_12

Implementing and Testing House in Program 2

Do each of these tests using the list created when testing AddStudent.

Testing the FindStudent by ID function:

Student *FindStudent(int studentID)

```
Student *s;
s = h->FindStudent(11144);
if((s!=NULL) && (s->getStudentID() == 11144))
        cout << "Test succeeded" << endl;
// Repeat for test of Student at end of list, in the
//   middle of the list, and not in the list.
```

1. Create a pointer to Student.
2. Call FindStudent passing in the ID of Fred Weasley (first in the list).
3. Check that the pointer is not NULL and that it is pointing to Fred Weasley.
4. Repeat with a check for Hermione Granger (last in the list).
5. Repeat with a check for Harry Potter (in the middle of the list).
6. Repeat with a check for ID 99999 (one known to not be in the list). Check for pointer now set to NULL.

Testing the FindStudent by name:

Student *FindStudent(char *mName, char *fName)

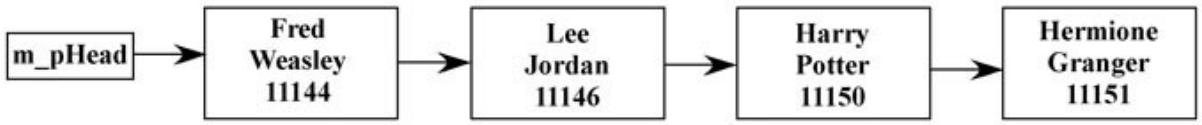
```
Student *s;
char mName[32], wName[32];
s = h->FindStudent("Fred", "Weasley");
if(s!=NULL)
{
    s->getName(mName, wName);
    if((strcmp(mName, "Fred") == 0) &&
       ((strcmp(wName, "Weasley") == 0))
            cout << "Test succeeded" << endl;
}
// Repeat for test of Student at end of list, in the
//   middle of the list, and not in the list.
```

1. Create a pointer to Student.
2. Call FindStudent passing in the name Fred Weasley (first in the list).
3. Check that the pointer is not NULL and that it is pointing to Fred Weasley.
4. Repeat with a check for Hermione Granger (last in the list).
5. Repeat with a check for Harry Potter (in the middle of the list).
6. Repeat with a check for John Doe (one known to not be in the list). Check for pointer now set to NULL.

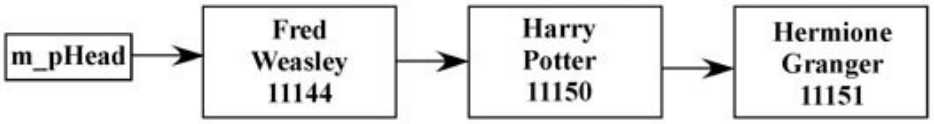
Testing Program 2_13

Implementing and Testing House in Program 2

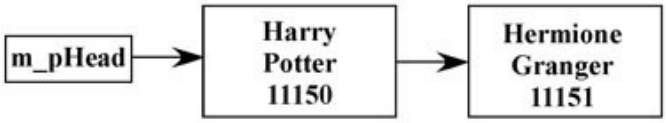
Sample list used in the instructor's test program.



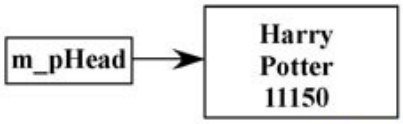
Initial state before removing any Student from the list.



Lee Jordan deleted first to test deleting from the middle of a list.



Fred Weasley deleted second to test deleting from the head of a list.



Hermoine Granger deleted third to test deleting from the end of a list.

Harry Potter deleted last to test deleting last student in the list.

Testing the RemoveStudent function:

Student *RemoveStudent(int studentID)

```
Student *s;
s = h->RemoveStudent(11144);
if((s!=NULL) && (s->getStudentID() == 11144))
{
        cout << "Test succeeded" << endl;
        delete s;  // Delete removed student
// Repeat for test of Student at end of list, in the
//   middle of the list, and not in the list.
```

1. Create a pointer to Student in main.
2. Set the pointer to what is returned by calling RemoveStudent with a name known to not be in the list. Check that NULL was returned.
3. Call the RemoveStudent function to remove a Student in the middle of the list. Check that the pointer returned is to the correct Student.
4. Repeat step 3 to remove a Student at the beginning and end of the list.
5. Keep removing Students till a call can be made to remove the last Student in the list.

Testing Program 2_14