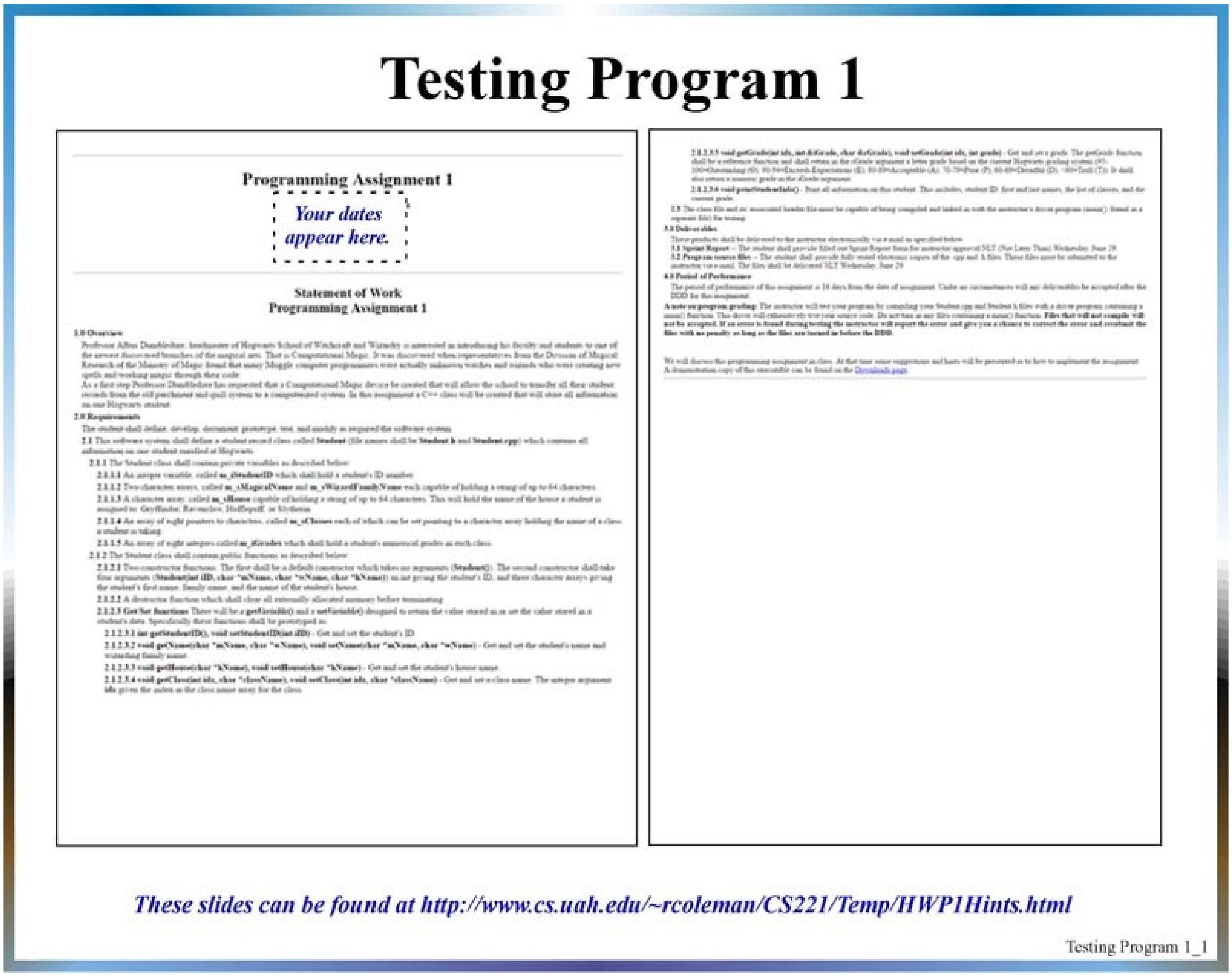
Bonus Slides

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 1



Testing Program 1

Programming Assignment 1

Your dates appear here.

Statement of Work
Programming Assignment 1

1.0 Overview

Professor Albus Dumbledore, headmaster of Hogwarts School of Witchcraft and Wizardry is interested in introducing his faculty and students to one of the newest discovered branches of the magical arts. That is Computational Magic. It was discovered when representatives from the Division of Magical Research of the Ministry of Magic found that many Muggle computer programmers were actually unknown witches and wizards who were creating new spells and working magic through their code.
As a first step Professor Dumbledore has requested that a Computational Magic device be created that will allow the school to transfer all their student records from the old parchment and quill system to a computerized system. In this assignment a C++ class will be created that will store all information on one Hogwarts student.

2.0 Requirements

The student shall define, develop, document, prototype, test, and modify as required the software system.

2.1 This software system shall define a student record class called **Student** (file names shall be **Student.h** and **Student.cpp**) which contains all information on one student enrolled at Hogwarts.

2.1.1 The Student class shall contain private variables as described below:

2.1.1.1 An integer variable, called **m_iStudentID** which shall hold a student's ID number.

2.1.1.2 Two character arrays, called **m_cMagicalName** and **m_cWizardFamilyName** each capable of holding a string of up to 64 characters.

2.1.1.3 A character array, called **m_cHouse** capable of holding a string of up to 64 characters. This will hold the name of the house a student is assigned to: Gryffindor, Ravenclaw, Hufflepuff, or Slytherin.

2.1.1.4 An array of eight pointers to characters, called **m_cClasses** each of which can be set pointing to a character array holding the name of a class a student is taking.

2.1.1.5 An array of eight integers called **m_iGrades** which shall hold a student's numerical grades in each class.

2.1.2 The Student class shall contain public functions as described below:

2.1.2.1 Two constructor functions. The first shall be a default constructor which takes no arguments (**Student()**). The second constructor shall take four arguments (**Student(int iID, char *mName, char *wName, char *hName)**) an int giving the student's ID, and three character arrays giving the student's first name, family name, and the name of the student's house.

2.1.2.2 A destructor function which shall clear all externally allocated memory before terminating.

2.1.2.3 Get/Set functions There will be a **getVariable()** and a **setVariable()** designed to return the value stored in or set the value stored in a student's data. Specifically these functions shall be prototyped as:

2.1.2.3.1 int getStudentID(), void setStudentID(int iID) - Get and set the student's ID.

2.1.2.3.2 void getName(char *mName, char *wName), void setName(char *mName, char *wName) - Get and set the student's name and wizarding family name.

2.1.2.3.3 void getHouse(char *hName), void setHouse(char *hName) - Get and set the student's house name.

2.1.2.3.4 void getClass(int idx, char *className), void setClass(int idx, char *className) - Get and set a class name. The integer argument **idx** gives the index in the class name array for the class.

2.1.2.3.5 void getGrade(int idx, int &iGrade, char &cGrade), void setGrade(int idx, int grade) - Get and set a grade. The getGrade function shall be a reference function and shall return in the cGrade argument a letter grade based on the current Hogwarts grading system (95-100=Outstanding (O), 90-94=Exceeds Expectations (E), 80-89=Acceptable (A), 70-79=Poor (P), 60-69=Dreadful (D), <60=Troll (T)). It shall also return a numeric grade in the iGrade argument.

2.1.2.3.6 void printStudentInfo() - Print all information on this student. This includes, student ID, first and last names, the list of classes, and the current grade.

2.5 The class file and its associated header file must be capable of being compiled and linked in with the instructor's driver program (main(), found in a separate file) for testing.

3.0 Deliverables

These products shall be delivered to the instructor electronically via e-mail as specified below:
3.1 Sprint Report – The student shall provide filled out Sprint Report form for instructor approval NLT (Not Later Than) Wednesday, June 29.
3.2 Program source files – The student shall provide fully tested electronic copies of the .cpp and .h files. These files must be submitted to the instructor via e-mail. The files shall be delivered NLT Wednesday, June 29.

4.0 Period of Performance

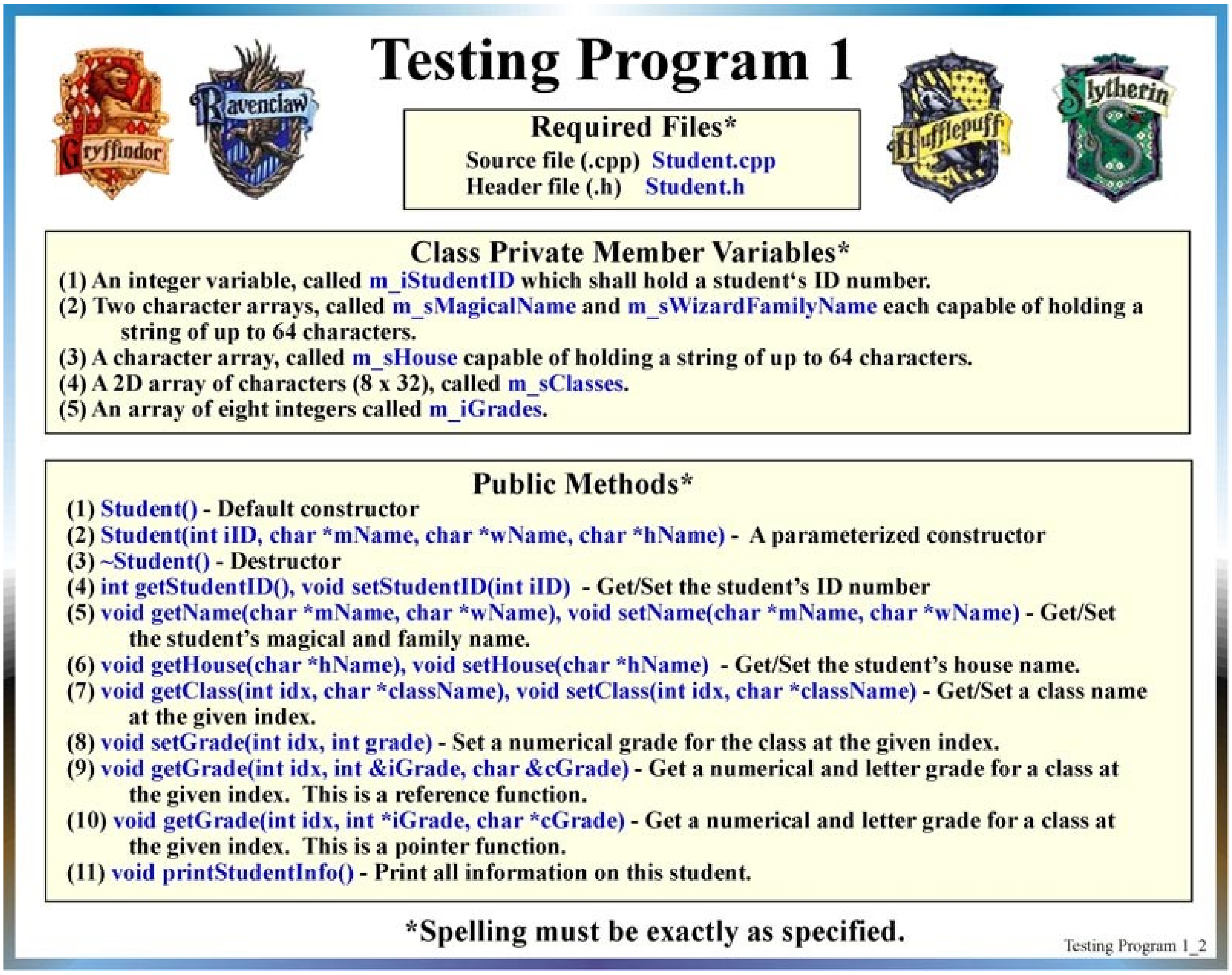
The period of performance of this assignment is 16 days from the date of assignment. Under no circumstances will any deliverables be accepted after the DDD for this assignment.

A note on program grading: The instructor will test your program by compiling your Student.cpp and Student.h files with a driver program containing a main() function. This driver will exhaustively test your source code. Do not turn in any files containing a main() function. **Files that will not compile will not be accepted. If an error is found during testing the instructor will report the error and give you a chance to correct the error and resubmit the files with no penalty as long as the files are turned in before the DDD.**

We will discuss this programming assignment in class. At that time some suggestions and hints will be presented as to how to implement the assignment. A demonstration copy of this executable can be found on the Downloads page.

These slides can be found at http://www.cs.uah.edu/~rcoleman/CS221/Temp/HWP1Hints.html

Testing Program 1_1



Testing Program 1

Required Files*

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

Class Private Member Variables*

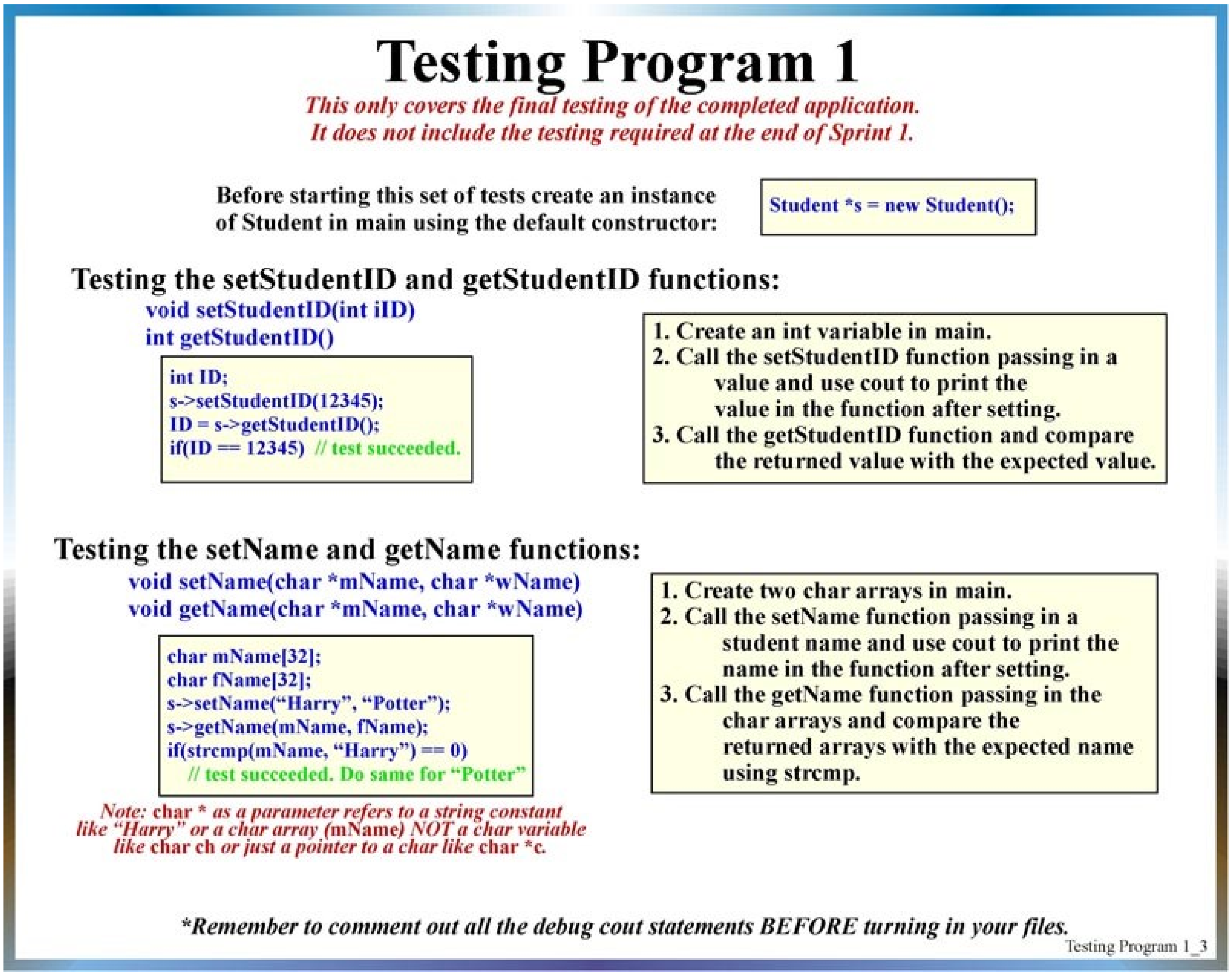
(1) An integer variable, called m_iStudentID which shall hold a student's ID number.
(2) Two character arrays, called m_sMagicalName and m_sWizardFamilyName each capable of holding a string of up to 64 characters.
(3) A character array, called m_sHouse capable of holding a string of up to 64 characters.
(4) A 2D array of characters (8 x 32), called m_sClasses.
(5) An array of eight integers called m_iGrades.

Public Methods*

(1) Student() - Default constructor
(2) Student(int iID, char *mName, char *wName, char *hName) - A parameterized constructor
(3) ~Student() - Destructor
(4) int getStudentID(), void setStudentID(int iID) - Get/Set the student's ID number
(5) void getName(char *mName, char *wName), void setName(char *mName, char *wName) - Get/Set the student's magical and family name.
(6) void getHouse(char *hName), void setHouse(char *hName) - Get/Set the student's house name.
(7) void getClass(int idx, char *className), void setClass(int idx, char *className) - Get/Set a class name at the given index.
(8) void setGrade(int idx, int grade) - Set a numerical grade for the class at the given index.
(9) void getGrade(int idx, int &iGrade, char &cGrade) - Get a numerical and letter grade for a class at the given index. This is a reference function.
(10) void getGrade(int idx, int *iGrade, char *cGrade) - Get a numerical and letter grade for a class at the given index. This is a pointer function.
(11) void printStudentInfo() - Print all information on this student.

***Spelling must be exactly as specified.**

Testing Program 1_2



Testing Program 1

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

Before starting this set of tests create an instance of Student in main using the default constructor:

```
Student *s = new Student();
```

Testing the setStudentID and getStudentID functions:

```
void setStudentID(int iID)
int getStudentID()
```

```
int ID;
s->setStudentID(12345);
ID = s->getStudentID();
if(ID == 12345)  // test succeeded.
```

1. Create an int variable in main.
2. Call the setStudentID function passing in a value and use cout to print the value in the function after setting.
3. Call the getStudentID function and compare the returned value with the expected value.

Testing the setName and getName functions:

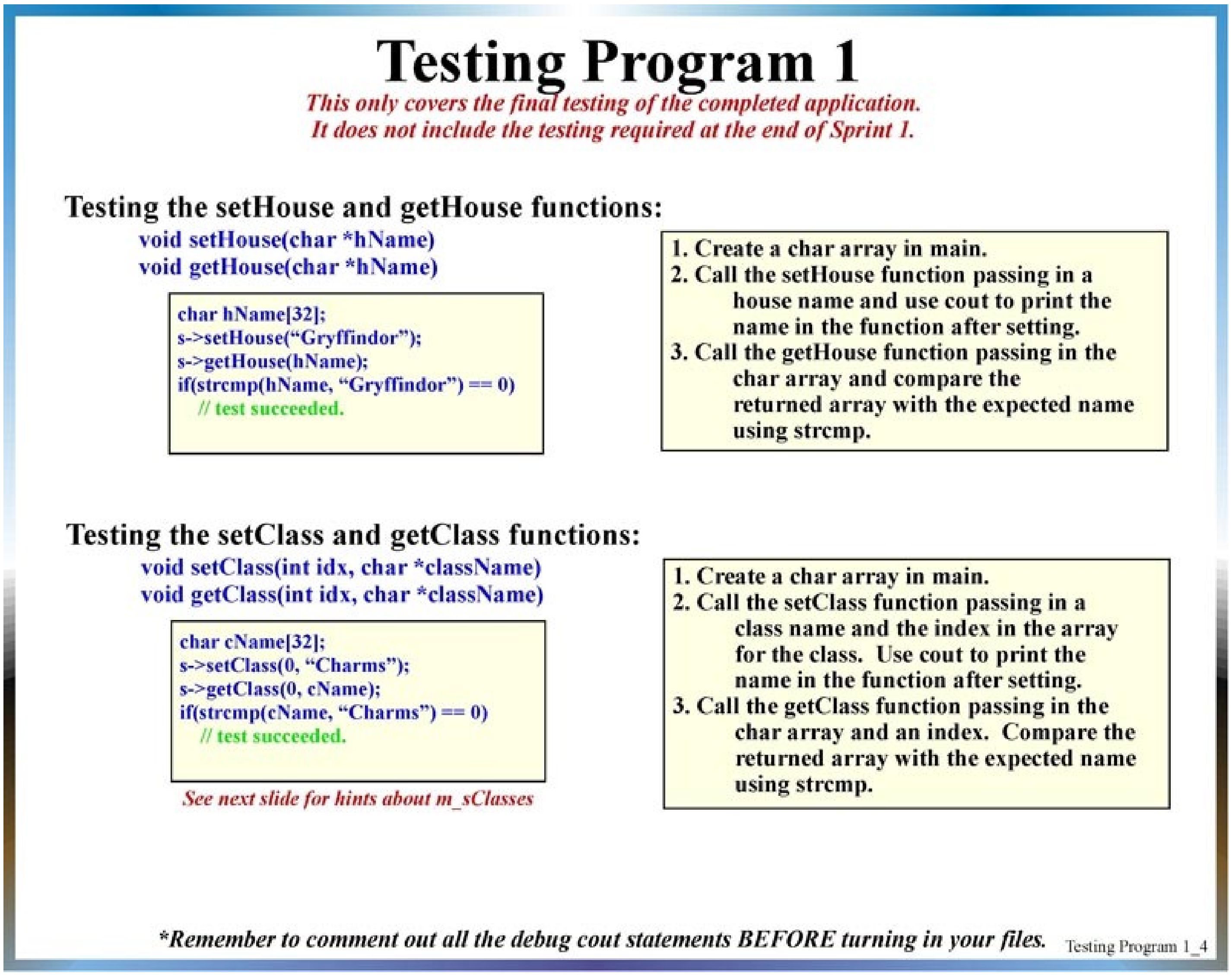
```
void setName(char *mName, char *wName)
void getName(char *mName, char *wName)
```

```
char mName[32];
char fName[32];
s->setName("Harry", "Potter");
s->getName(mName, fName);
if(strcmp(mName, "Harry") == 0)
   // test succeeded. Do same for "Potter"
```

Note: char * *as a parameter refers to a string constant like "Harry" or a char array* (mName) *NOT a char variable like* char ch *or just a pointer to a char like* char *c.

1. Create two char arrays in main.
2. Call the setName function passing in a student name and use cout to print the name in the function after setting.
3. Call the getName function passing in the char arrays and compare the returned arrays with the expected name using strcmp.

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



Testing Program 1

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 setHouse and getHouse functions:

```
void setHouse(char *hName)
void getHouse(char *hName)
```

```
char hName[32];
s->setHouse("Gryffindor");
s->getHouse(hName);
if(strcmp(hName, "Gryffindor") == 0)
   // test succeeded.
```

1. Create a char array in main.
2. Call the setHouse function passing in a house name and use cout to print the name in the function after setting.
3. Call the getHouse function passing in the char array and compare the returned array with the expected name using strcmp.

Testing the setClass and getClass functions:

```
void setClass(int idx, char *className)
void getClass(int idx, char *className)
```

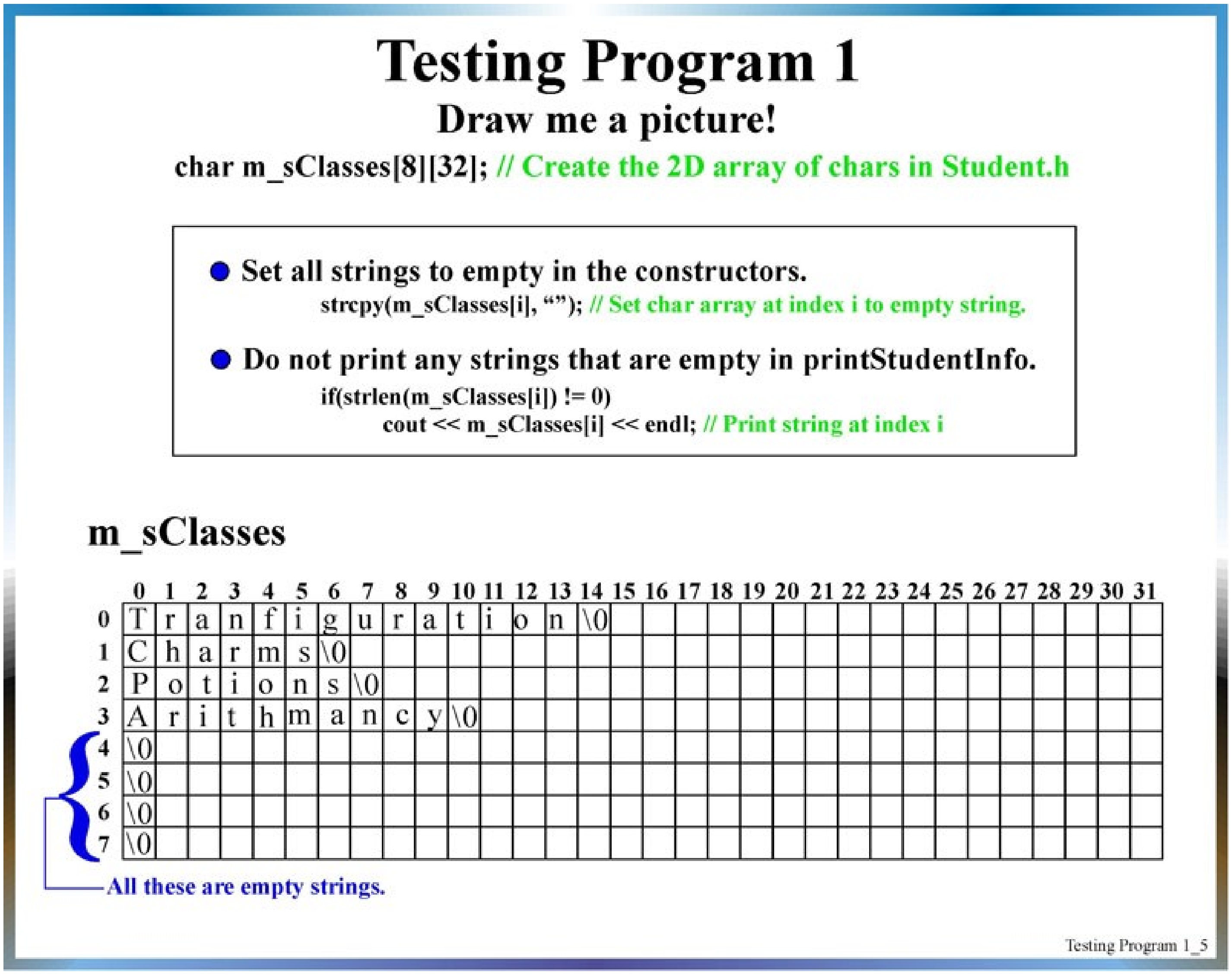
```
char cName[32];
s->setClass(0, "Charms");
s->getClass(0, cName);
if(strcmp(cName, "Charms") == 0)
   // test succeeded.
```

See next slide for hints about m_sClasses

1. Create a char array in main.
2. Call the setClass function passing in a class name and the index in the array for the class. Use cout to print the name in the function after setting.
3. Call the getClass function passing in the char array and an index. Compare the returned array with the expected name using strcmp.

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

Testing Program 1_4



Testing Program 1

Draw me a picture!

```
char m_sClasses[8][32]; // Create the 2D array of chars in Student.h
```

- Set all strings to empty in the constructors.

```
        strcpy(m_sClasses[i], “”); // Set char array at index i to empty string.
```

- Do not print any strings that are empty in printStudentInfo.

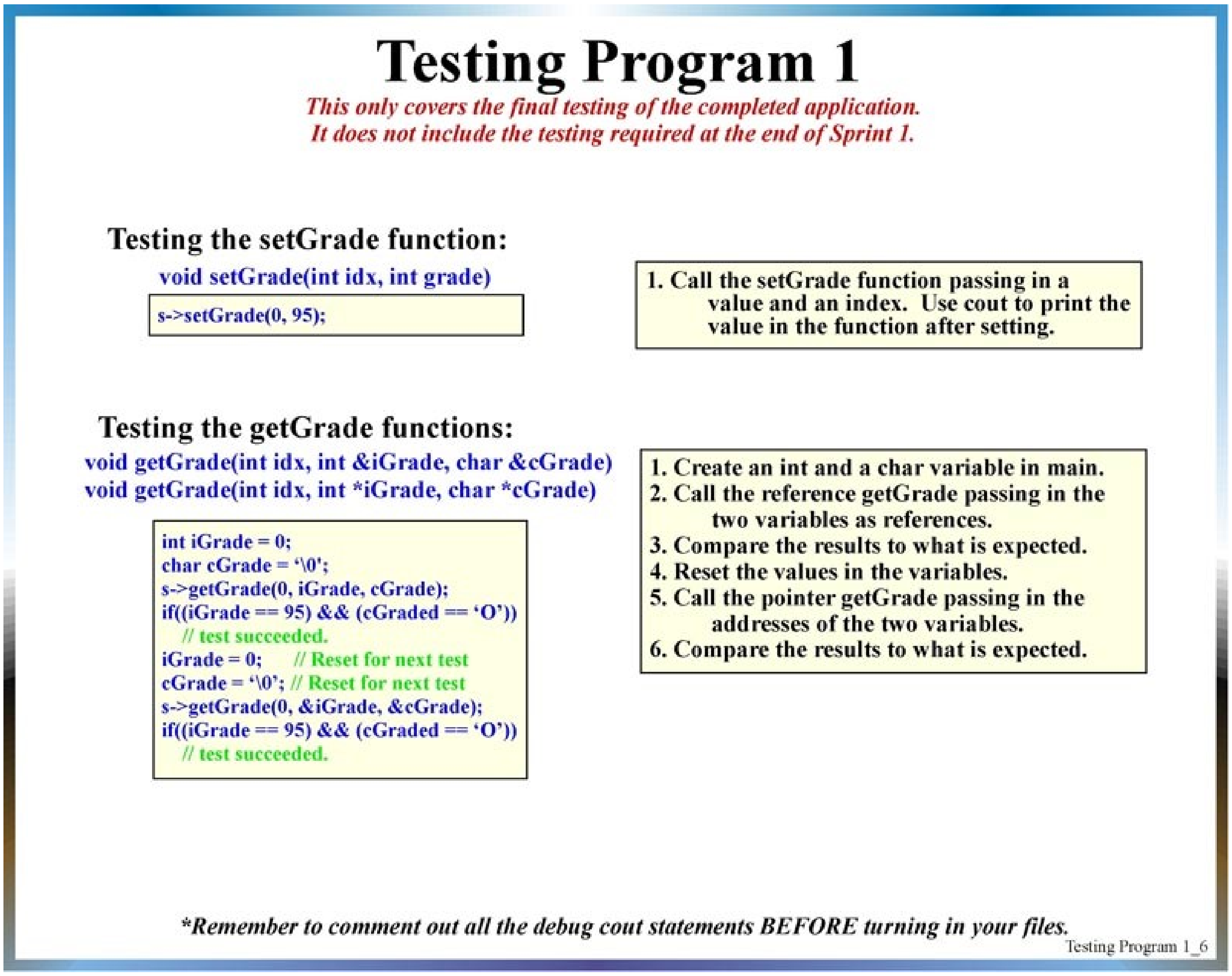
```
        if(strlen(m_sClasses[i]) != 0)
              cout << m_sClasses[i] << endl; // Print string at index i
```

m_sClasses

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	T	r	a	n	f	i	g	u	r	a	t	i	o	n	\0																	
1	C	h	a	r	m	s	\0																									
2	P	o	t	i	o	n	s	\0																								
3	A	r	i	t	h	m	a	n	c	y	\0																					
4	\0																															
5	\0																															
6	\0																															
7	\0																															

All these are empty strings.

Testing Program 1_5



Testing Program 1

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 setGrade function:

void setGrade(int idx, int grade)

```
s->setGrade(0, 95);
```

1. Call the setGrade function passing in a value and an index. Use cout to print the value in the function after setting.

Testing the getGrade functions:

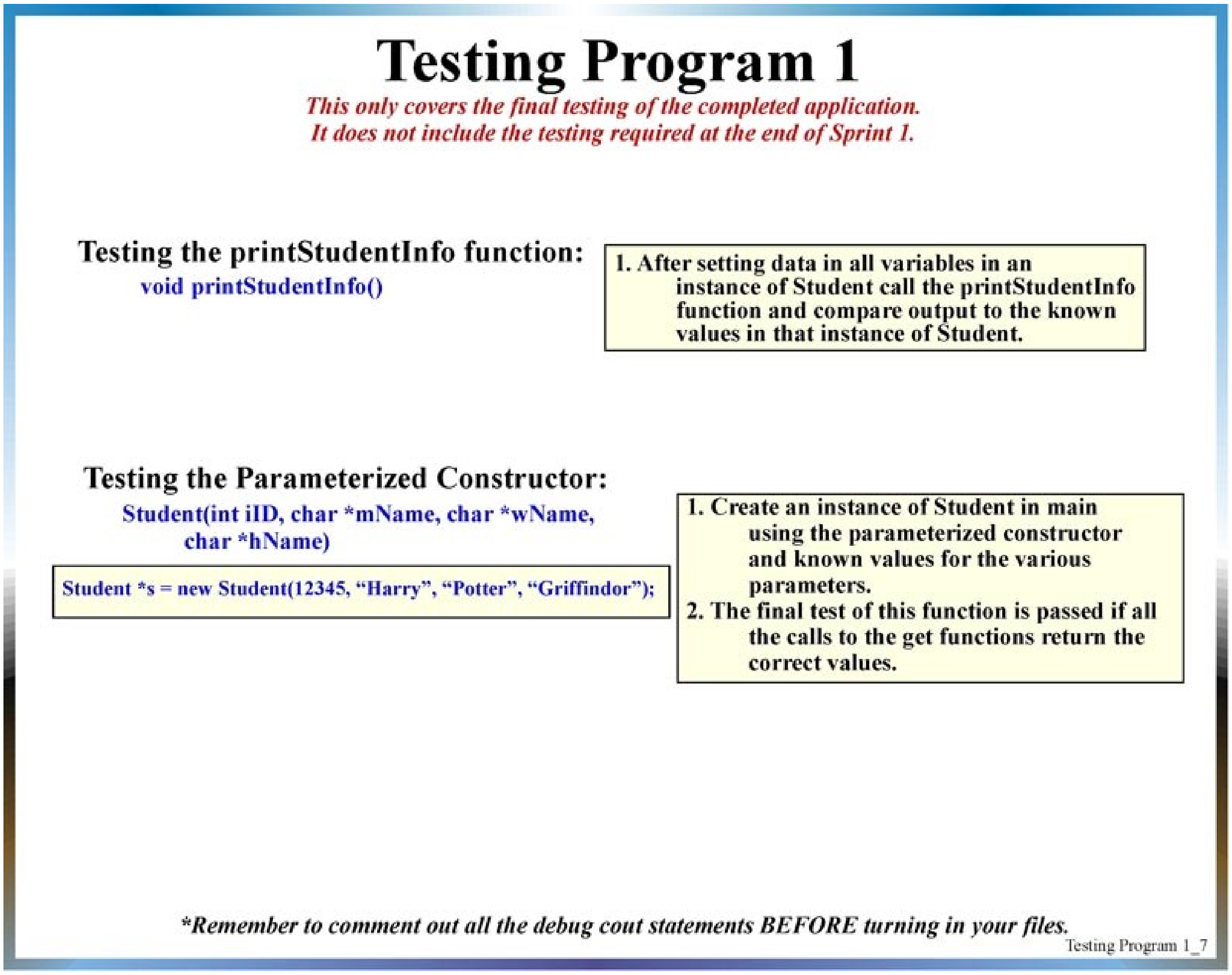
void getGrade(int idx, int &iGrade, char &cGrade)
void getGrade(int idx, int *iGrade, char *cGrade)

```
int iGrade = 0;
char cGrade = '\0';
s->getGrade(0, iGrade, cGrade);
if((iGrade == 95) && (cGraded == 'O'))
   // test succeeded.
iGrade = 0;     // Reset for next test
cGrade = '\0'; // Reset for next test
s->getGrade(0, &iGrade, &cGrade);
if((iGrade == 95) && (cGraded == 'O'))
   // test succeeded.
```

1. Create an int and a char variable in main.
2. Call the reference getGrade passing in the two variables as references.
3. Compare the results to what is expected.
4. Reset the values in the variables.
5. Call the pointer getGrade passing in the addresses of the two variables.
6. Compare the results to what is expected.

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

Testing Program 1_6



Testing Program 1

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 printStudentInfo function:

void printStudentInfo()

1. After setting data in all variables in an instance of Student call the printStudentInfo function and compare output to the known values in that instance of Student.

Testing the Parameterized Constructor:

```
Student(int iID, char *mName, char *wName,
        char *hName)
```

```
Student *s = new Student(12345, “Harry”, “Potter”, “Griffindor”);
```

1. Create an instance of Student in main using the parameterized constructor and known values for the various parameters.
2. The final test of this function is passed if all the calls to the get functions return the correct values.

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

Testing Program 1_7