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 Assingment 3

Implementing and Testing Program 3

Programming Assignment 3

Your dates appear here.

Statement of Work
Programming Assignment 3

These slides can be found at http://www.cs.uah.edu/~rcoleman/CS221/Temp/HWP3Hints.html

Testing Program 3_1

Implementing and Testing Program 3

Required Files*

Source file (.cpp) Library.cpp
Header file (.h) Library.h
Header file (.h) Book.h

Library Private Member Variable*

(1) Pointer to an instance of Book called m_pRoot

Possessions Class Methods*

Public Functions

(1) Library() - Default constructor. Sets m_pRoot to NULL.
(2) ~Library() - Destructor. Calls destroyTree passing in m_pRoot to delete all instances of Book in the tree.
(3) buildLibrary(char *fileName) - Read the data file and build the tree. Function provided by the instructor.
(3) bool addBook(Book *newBook) - Add a Book to the binary tree of Books.
(4) Book *removeBook(int bookNum) - Find an item in the tree, remove and return it.
(5) Book *getBookByNumber(int bookNum) - Locate a Book in the tree searching by its' ID number and return a pointer to it or NULL if not found.
(6) Book *getBookByTitle(char *title) - Call the private getBookByTitle and return what it returns.
(7) void printLibrary() - Function to initiate printing of all books in the tree.

Private Functions

(1) Book *getBookByTitle(char *title, Book *rt) - Traverse the tree and locate a Book in the tree searching by its' title and return a pointer to it or NULL if not found.
(2) void printOne(Book *book) - Print all information in a single instance of Book.
(3) void printAll(Book *rt) - Traverse the tree, in order, and print all information on all Books.
(8) void destroyTree(Book *rt) - Function called by the destructor to recursively destroy each book in the tree.
(9) bool getNextLine(ifstream &inFile, char *line, int lineLen) - Read a line from the data file. Provided by the instructor.

*Spelling must be exactly as specified.

Testing Program 3_2

Implementing and Testing Program 3

Hogwarts Library Binary Tree

This is the tree (book numbers shown) built by buildTree() using the provided data file HWBookDB.txt

The provided data file HWBookDB.txt

```
# Hogwarts Library Books for Programming Assignment 3
#
# The books listed below are all those mentioned in Harry Potter books
#   by J.K.Rowling.  Many of the authors were also created by Ms Rowling
#   others were made up by Dr. Coleman.  Any resemblance to real or
#   imagined people is purely a coincidence.
#
# Note: the books are listed by their book number in an order that will
#       ensure that a binary tree using the book number as the key will
#       be as close to balanced as possible.
#
12500
The Standard Book of Spells
Miranda Goshawk
#
12425
A History of Magic
Bathilda Bagshot
#
12625
Magical Theory
Adalbert Waffling
#
12385
A Beginners' Guide to Transfiguration
Emeric Switch
#
12465
One Thousand Magical Herbs and Fungi
Phyllida Spore
#
12545
Magical Drafts and Potions
Arsenius Jigger
#
12705
Fantastic Beasts and Where to Find Them
Newt Scamander
#
12365
The Dark Forces: A Guide to Self-Protection
Quentin Trimble
#
12405
Curses and Countercurses
Vindictus Viridian
#
12445
Great Wizards of the Twentieth Century
Horatio Twiblee
#
12485
Quidditch Through the Ages
Kennilworthy Whisp
#
12525
Dragon Breeding for Pleasure and Profit
Draco Wyrm
#
12605
Break with a Banshee
Gilderoy Lockhart
#
12685
Gadding with Ghouls
Gilderoy Lockhart
#
12785
Travels with Trolls
Gilderoy Lockhart
#
12355
Voyages with Vampires
Gilderoy Lockhart
#
12375
Wanderings with WereWolves
Gilderoy Lockhart
#
12395
Year with the Yeti
Gilderoy Lockhart
#
12415
Magical Me
Gilderoy Lockhart
#
12435
Most Potent Potions
Chelone Glabra
#
12455
Prefects Who Gained Power
Maria Leach
#
12475
Flying with the Canons
Jason Calibre
#
12495
The Monster Book of Monsters
Krecher Krowley
#
12515
The Invisible Book of Invisibility
Veronica Void
#
12535
Unfogging the Future
Cassandra Vablatsky
#
12555
Broken Balls: When Fortunes Turn Foul
Cassandra Vablatsky
#
12615
Death Omens: What to Do When You Know the Worst Is Coming
Cassandra Vablatsky
#
12665
Hogwarts: A History
Armando Dippet
#
12695
The Life and Lies of Albus Dumbledore
Rita Skeeter
#
12775
The Tales of Beedle the Bard
Gladys Patron
#
12795
Armando Dippet: Master or Moron
Rita Skeeter
```

Testing Program 3_3

Implementing and Testing Program 3

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 addBook function:

bool addBook(Book *newBook)

Testing the print functions:

void printLibrary()
void printOne(Book *book)
void printAll(Book *rt)

```
Library *lib = new Library();
lib->buildLibrary("HWBookDB.txt");
lib->printLibrary();
// Check that all books are listed and in order by
//   book number
```

1. Create an instance of Library in main.
2. Using the provided buildLibrary function and the provided data file build the Library.
3. After adding all the books call the public function printLibrary and check that all books have been added and that they are listed in order by book number.

You cannot directly test the private functions printAll and printOne, but you can indirectly test them by call the public printLibrary function which in turn calls the private functions.

Remember it is your responsibility to test your program. It is not the instructor's responsibility.

**Remember to comment out all the debug cout statements BEFORE turning in your files.* Testing Program 3_4

Implementing and Testing Program 3

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 getBookByNumber function:

Book *getBookByNumber(int bookNum)

```
Book *bk;
bk = lib->getBookByNumber(12485);
if((bk != NULL) && (strcmp(bk->Title,
        "Quidditch Through the Ages") == 0))
{
        cout << "Search successful" << endl;
// Repeat for other books and a book number not
//   in the library.
```

1. Create a pointer to struct Book in main.
2. Call getBookByNumber passing in numbers of books in the Library. Check to be sure the returned pointer is not NULL and that it points to the correct book. (Repeat for the root, leaf, and interior node.)
3. Call the getBookByNumber function passing in a book number known to not be in the Library.

Testing the getBookByTitle functions:

Book *getBookByTitle(char *title)
Book *getBookByTitle(char *title, Book *rt)

```
Book *bk;
bk = lib->getBookByTitle("Quidditch Through the Ages");
if((bk != NULL) && (strcmp(bk->Title,
        "Quidditch Through the Ages") == 0))
{
        cout << "Search successful" << endl;
// Repeat for other books and a book title not
//   in the library.
```

1. Create a pointer to struct Book in main.
2. Call getBookByTitle passing in titles of books in the Library. Check to be sure the returned pointer is not NULL and that it points to the correct book. (Repeat for the root, leaf, and interior node.)
3. Call the getBookByTitle function passing in a book number known to not be in the Library.

You cannot directly test the private version of getBookByTitle, but you can indirectly test it by call the public getBookByTitle function which in turn calls the private function.

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

Testing Program 3_5

Testing Program 3

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 removeBook function:

Book *removeBook(int bookNum)

1. Create a different data file using a subset of the books in the provided file. See next slides for an example.
2. Call removeBook passing in the number of a book known to not be in the Library then check for NULL being returned.
2. Call removeBook passing in the number of a book known to be in the Library.
3. Check the pointer to the instance of Book returned that it is not NULL and it is the Book to be removed
4. Do this for several books making sure to check for deleting
   (1) Node with no children, (2) node with one child on the left, (3) node with one child on the right, (4) node with two children. Test each of the above when the node to be deleted is the root and when it is another node in the tree.

Test Checklist for removeBook

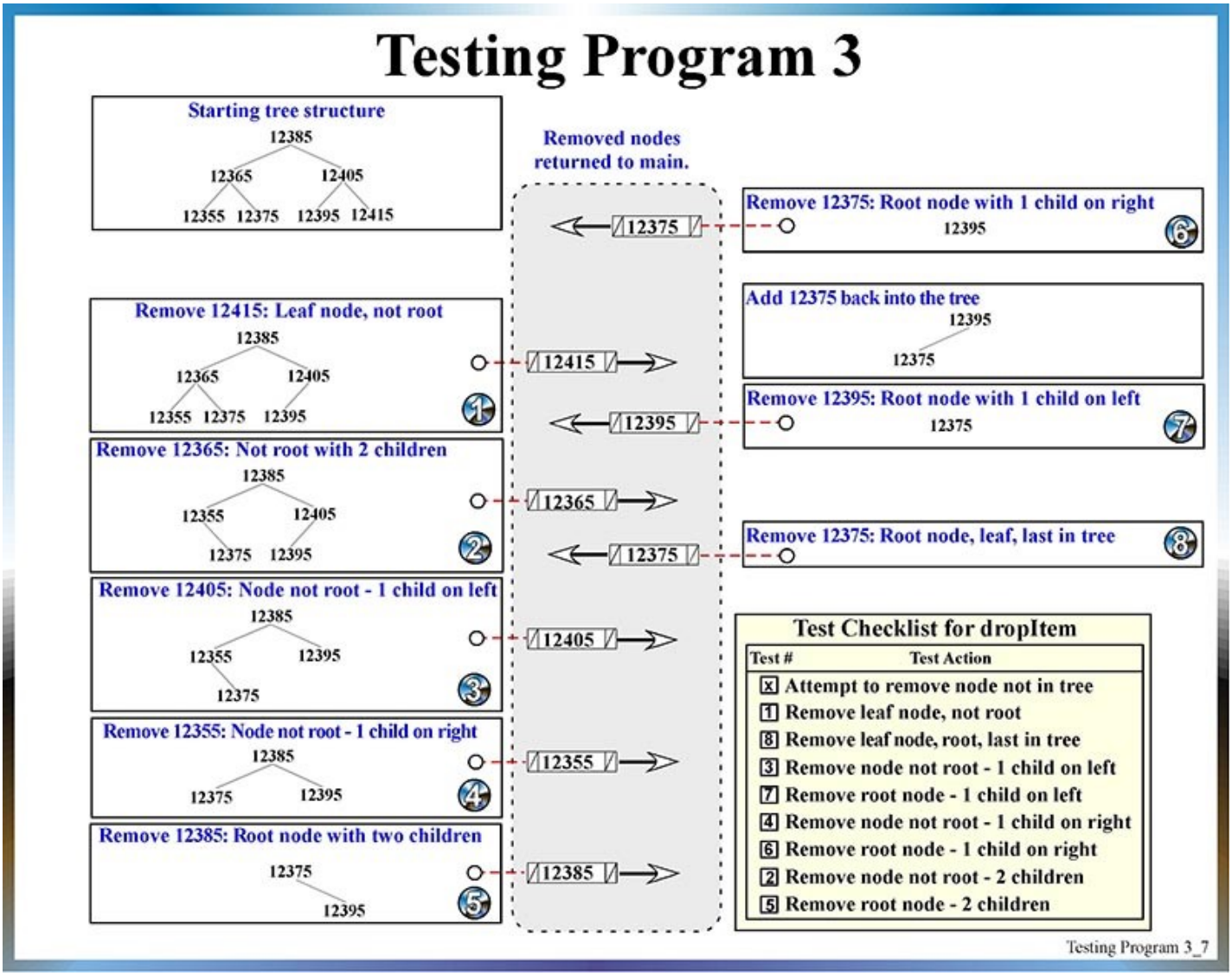
- ☐ Attempt to remove node not in tree
- ☐ Remove leaf node, not root
- ☐ Remove leaf node, root, last in tree
- ☐ Remove node not root - 1 child on left
- ☐ Remove root node - 1 child on left
- ☐ Remove node not root - 1 child on right
- ☐ Remove root node - 1 child on right
- ☐ Remove node not root - 2 children
- ☐ Remove root node - 2 children

Sample data file for testing the removeBook Function

```
#
12385
A Beginners' Guide to Transfiguration
Emeric Switch
#
12365
The Dark Forces: A Guide to Self-Protection
Quentin Trimble
#
12405
Curses and Countercurses
Vindictus Viridian
#
12355
Voyages with Vampires
Gilderoy Lockhart
#
12375
Wanderings with WereWolves
Gilderoy Lockhart
#
12395
Year with the Yeti
Gilderoy Lockhart
#
12415
Magical Me
Gilderoy Lockhart
```

See next slide for details of this testing.

Testing Program 3_6


Testing Program 3
Starting tree structure
12385
12365 12405
12355 12375 12395 12415
Removed nodes returned to main.
Remove 12415: Leaf node, not root
12385
12365 12405
12355 12375 12395
1
Remove 12365: Not root with 2 children
12385
12355 12405
12375 12395
2
Remove 12405: Node not root - 1 child on left
12385
12355 12395
12375
3
Remove 12355: Node not root - 1 child on right
12385
12375 12395
4
Remove 12385: Root node with two children
12375
12395
5
12375
12415
12395
12365
12375
12355
12385
Remove 12375: Root node with 1 child on right
12395
6
Add 12375 back into the tree
12395
12375
Remove 12395: Root node with 1 child on left
12375
7
Remove 12375: Root node, leaf, last in tree
8
Test Checklist for dropItem
Test # Test Action
X Attempt to remove node not in tree
1 Remove leaf node, not root
8 Remove leaf node, root, last in tree
3 Remove node not root - 1 child on left
7 Remove root node - 1 child on left
4 Remove node not root - 1 child on right
6 Remove root node - 1 child on right
2 Remove node not root - 2 children
5 Remove root node - 2 children
Testing Program 3_7

Implementing and Testing Program 3

Library.cpp - Build Library Function

```
//-----------------------------------------------------------
// Function: buildLibrary()
// Purpose: Build the library from a data file
// Args: fileName - character array holding the name of the
//       datafile defining the library.
// Returns; True if library was successsfully built.
//-----------------------------------------------------------
bool Library::buildLibrary(char *fileName)
{
        ifstream  inFile;
        Book *bk;
        char line[128];

        inFile.open(fileName, ifstream::in);
        if(!inFile.is_open())
        {
        // inFile.is_open() returns false if the file could not be found or
        //    if for some other reason the open failed.
        cout << "Unable to open file " << fileName << ". \nProgram terminating...\n";
        return false;
        }

        while (getNextLine(inFile, line, 127)) // While the next line is readable
        {
                // Create a new Book and set it's pointers to NULL
                bk = new Book();
                bk->left = bk->right = NULL;

                // Set the book number
                bk->bookNumber = atoi(line);

                // Read the book title
                getNextLine(inFile, line, 127);
                strcpy(bk->Title, line);

                // Read the author
                getNextLine(inFile, line, 127);
                strcpy(bk->Author, line);

                // Add this book to the tree
                addBook(bk);
        }
        return true;

}
```

This function will add Books to the Library by calling the addBook function which implements the insert into a binary tree algorithm

Note: This is also testing the addBook function.

This function is provided by the Instructor

Testing Program 3_8

Implementing and Testing Program 3

Library.cpp - Get Next Line Function

```
//-------------------------------------------------------------------------------------------
// Function: getNextLine()
// Purpose: Read a line from the data file skipping blank lines
//          and comment lines beginning with #
// Args: inFile - reference argument to an open ifstream object
//         to read from.
//       line - character array into which the data line is read
//       lineLen - maximum number of characters which can be read
//         into the array line
// Returns: True if a successful read was done.  If False is
//       returned then the array line will be zero length.
//-------------------------------------------------------------------------------------------
bool Library::getNextLine(ifstream &inFile, char *line, int lineLen)
{
   int    done = false;
   while(!done)
   {
      inFile.getline(line, lineLen);
      if(inFile.good())    // If a line was successfully read
      {
         if(strlen(line) == 0)  // Skip any blank lines
            continue;
         else if(line[0] == '#')  // Skip any comment lines
            continue;
         else done = true;    // Got a valid data line so return with this line
      }
      else
      {
         strcpy(line, "");
         return false;    // Flag end of file with null string and return false
      }
   } // end while
   return true; // Flag successful read
}
```

This function is provided by the Instructor

Testing Program 3_9