CS 211 - 11/14/17

Tuesday, November 14, 2017 5:00 PM

Testing

Verify that the program or part of the program works.

Does the code perform X? (Yes or No)

Testing is separate from Debugging.

Testing : Is there a problem?

Debugging: What is the solution to the problem?

Each test needs the following steps:

Testing Step 1:

Determine what we want to test.

1a. Determine the feature (or part of code) to be tested

1b. Determine how to run the code to be tested

What input is needed?

What parameter values need to be set up?

1c. Determine what the desired out should be

Testing Step 2:

Initialize the code to prepare for the test

- Do I need to create a separate testing program?
- Do I need to get the data/state in a certain manner?

Testing Step 3

Run the test code

Testing Step 4

Determine if the test passed or failed

Development Style of Test Driven Development

1. Create a test that fails with the current code base.
2. Modify the Code until the test passes.
   2a. Verify that you haven't caused other test cases to fail.
3. Refactor the code to improve its execution/operation
4. Repeat step 1 until all requirements are met.

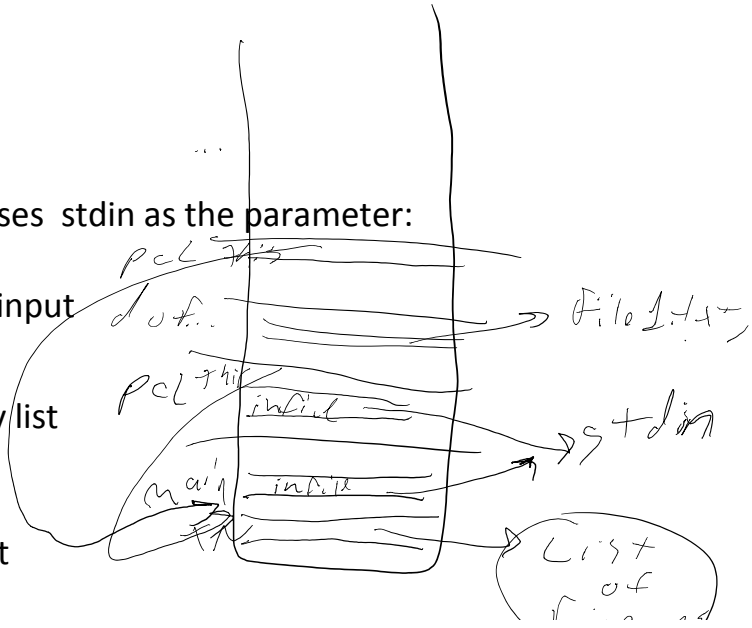
For the code review for weeks 12 and 13, you will be asked to write a simple test case.

From the Code Base for Project 6:

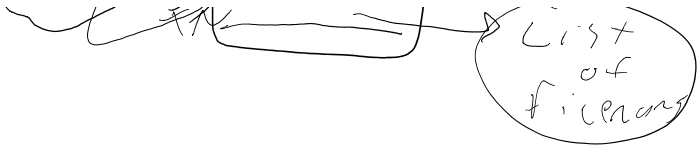
```
void doFile()
 {
  // get a filename from the input
  char* fname = strtok (NULL, " \n\t");
  if ( fname == NULL )
  {
   printf ("Filename expected\n");
   return;
  }

  printf ("Performing the File command with file: %s\n", fname);

  // next steps:  (if any step fails: print an error message and return )
  //  1. verify the file name is not currently in use
  //      1a.   Check if the filename is store in a list of CurrentFiles
  //                perhaps with and exists() or contains() method
  //       1b.   Add the filename to the list
  //  2. open the file using fopen creating a new instance of FILE*
  //  3. recursively call processCommandLoop() with this new instance of FILE* as the
parameter
  //  4. close the file when processCommandLoop() returns
  //      1c.   Remove the filename from the list
 }
};
```

Note the original call to processCommandLoop() uses stdin as the parameter:

```
// set up the variable inFile to read from standard input
  FILE* inFile = stdin;

  // set up the data needed for the airport adjcency list
  TravelNetwork airportData;

  // call the method that reads and parses the input
  airportData.processCommandLoop (inFile);
```



// call the method that reads and parses the input
airportData.processCommandLoop (inFile);



The project specifies that you are to write 4 classes

1. MyNode class for the linked list

2. MyList class that uses the Node class

```
class MyList
{
   private:
      MyNode* head;
  …
};
```

3. Airport class
   - List for that airport (perhaps inheritance might work here)
   - Boolean for the Visited info for DFS

4. Travel Network
   - Array of Airports
   - List of Filenames (this list may be a 5th class you need to write)