PA8 – Boustrophedonic Lists

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: (•) Yes ( ) No

Name: Brian Bauman

Date: 06/05/2019

Submission Details

Final ***Changelist*** number: 362054

Verified build: (•) Yes ( ) No

Number Tests Passed: **4**

Required Configurations: Debug & Release

Discussion (What did you learn):

The value of not just trusting the existing unit tests... had to create some of my own to make sure this thing worked for bigger structures. Hopefully they were good enough. Also, the word Boustrophedonic.

Verify Builds

- Follow the Piazza procedure on submission
  - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
  - No – Generated files
    - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db, *.user
    - Anything that is generated by the compiler should not be included
  - No – Generated directories
    - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
  - *.sln, *.cpp, *.h
  - *.vcxproj, *.vcxproj.filters, CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
  - As soon as you get something working, submit to perforce
  - Have reasonable check-in comments
    - Points will be deducted if minimum is not reached

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
  - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
  - Warning level ALL …
  - NO Warnings or ERRORS
    - Your code should be squeaky clean.
  - Code needs to work "as-is".
    - No modifications to files or deleting files necessary to compile or run.
  - All your code must compile from perforce with no modifications.
    - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason…
  - It will not be graded and you get a 0

No Containers

- NO STL allowed {Vector, Lists, Sets, etc...}
  - No automatic containers or arrays
  - You need to do this the old fashion way - ***YOU EARNED IT***

Leave Project Settings

- Do NOT change the project or warning level
  - Any changing of level or suppression of warnings is an integrity issue

Simple C++

- No modern C++
  - No Lambdas, Autos, templates, etc…
  - No Boost
- NO Streams
  - Used fopen, fread, fwrite…
- No code in MACROS
  - Code needs to be in cpp files to see and debug it easy
- ***Exception:***
  - implicit problem needs templates

Leaking Memory

- If the program leaks memory
  - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
  - It is responsible for its deletion
- Any ***MEMORY*** dynamically allocated that isn't freed up is ***LEAKING***
  - Leaking is ***HORRIBLE***, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
  - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
  - All files must be active to get credit.
  - Better to lose points for unit tests than to disable and lose all points

No Adding files to this project

- This project will work "as-is" do not add files…
- Grading system will overwrite project settings and will ignore any student's added files and will returned program to the original state

UnitTestConfiguration file (if provided) needs to be set by user

- Grading will be on the UnitTestConfiguration settings
  - Please explicitly set which tests you want graded… no regrading if set incorrectly

Due Dates

- See Piazza for due date and time
- Submit program perforce in your student directory assignment supplied.
- Fill out your this ***Submission Report*** and commit to perforce
  - ***ONLY*** use Adobe Reader to fill out form, all others will be rejected.
  - Fill out the form and discussion for full credit.

Goals

- Programming Assessment
  - More practice with linked lists... (kind of)

Assignments

- Please ***VERIFY*** the correct builds for each project
- Write a single function: remove()
  - Signature and structure need exactly the same
  - Place function in file named (see supplied files):
    - Boustrophedonic.cpp:
      - Function
    - Boustrophedonic.h:
      - Declaration and structure

```
struct Node
{
    Node *pNorth;
    Node *pSouth;
    Node *pEast;
    Node *pWest;
}

void Remove( Node *&head, int row, int col );
```

- **→ ASSUMPTIONS ←**
  - Boustrophedonic list is complete and ***PRISTINE*** before you remove one node
  - Only ***ONE*** node will be removed in the function
  - The dimensions of the list are ***NOT*** given
  - Boustrophedonic list has an even number of columns
  - Boustrophedonic list has no restrictions on number of rows
  - On deletion, horizontal and vertical connections are preserved across the deleted node

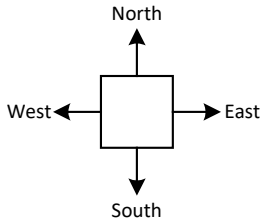
- Rules:
  - → **NOTE** ← **:**
    - Your list will be tested with different dimensions
    - Make sure you solve the problem in a generalized way without crashing
  - You cannot modify the Node.h in any way.
    - It's needed in the unit tests as is…
  - You can add helper functions to Boustrophedonic.h and Boustrophedonic.cpp
    - All your work should be done in Boustrophedonic.cpp
  - Make sure you are not leaking memory
    - You should delete the Node in your remove function
  - Check both Debug and Release mode

Creating a Boustrophedonic List

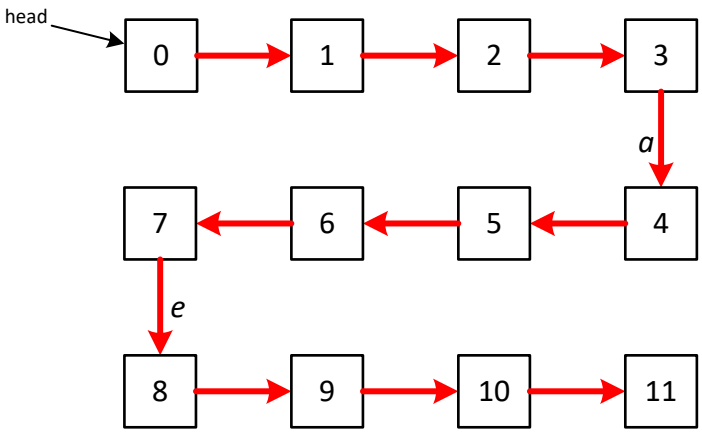
Boustrophedonic - Greek word:

- A method of writing shown in early Greek inscriptions, in which the lines run alternately from right to left and from left to right, as the furrows made in plowing a field, the plow passing alternately backward and forward.

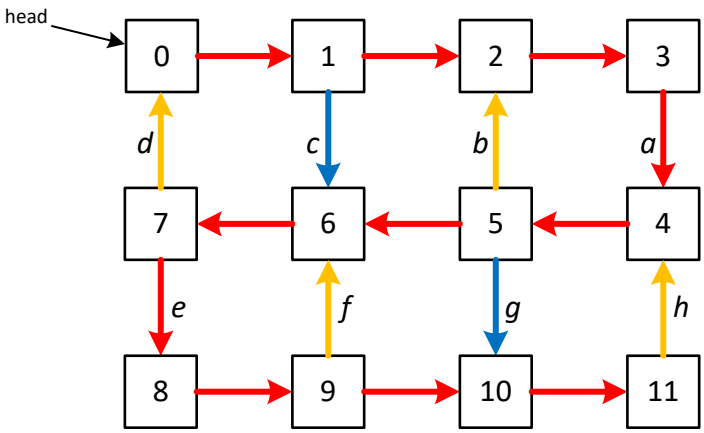
```
struct Node
{
    Node *pNorth;
    Node *pSouth;
    Node *pEast;
    Node *pWest;
}
```



- Start with the first node and continue in the Boustrophedonic path way, follow the RED LINE.
  - Start with the head, and sequentially go from 0 to 11 in the Boustrophedonic way

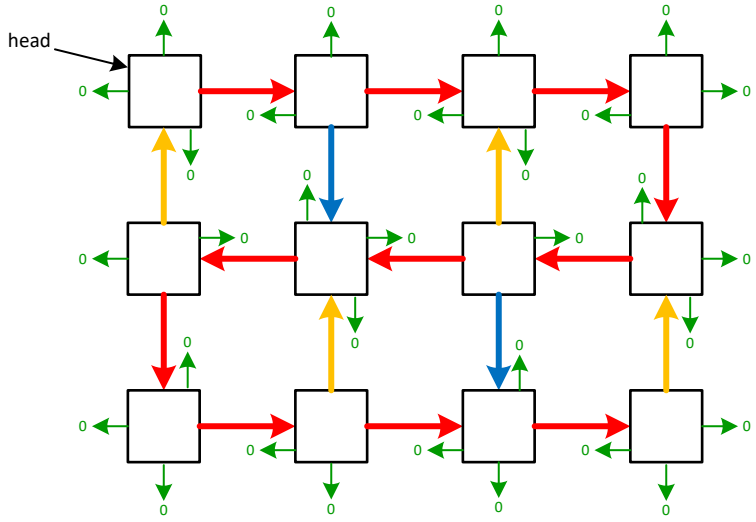


- Next we create the vertical connections.

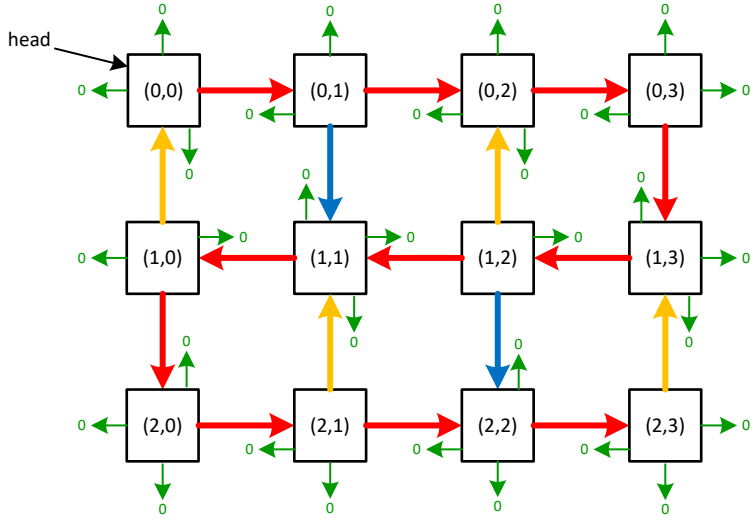


- Create the vertical connections between row 0 {0,1,2,4} and row 1 {7,6,5,4}.
  - Find arrow *a* between 3 and 4.
    - That arrow goes down, from 3 to 4.
  - Alternate direction of *a*, (*a* goes down in RED)
    - *b* goes up YELLOW,
    - *c* goes down BLUE,
    - *d* goes up YELLOW.
- Similar for the connections between row 1 {7,6,5,4} and row 2 {8,9,10,11}
  - Alternate direction of *e*, (*e* goes down in RED)
    - *f* goes up YELLOW,
    - *g* goes down BLUE,
    - *h* goes up YELLOW.

- Drawing the list with all the links displayed.
  - Each node actually has 4 pointers, most are nulls.
    - North, South, East, West
  - Here the list is drawn with GREEN links showing the explicit null pointers.

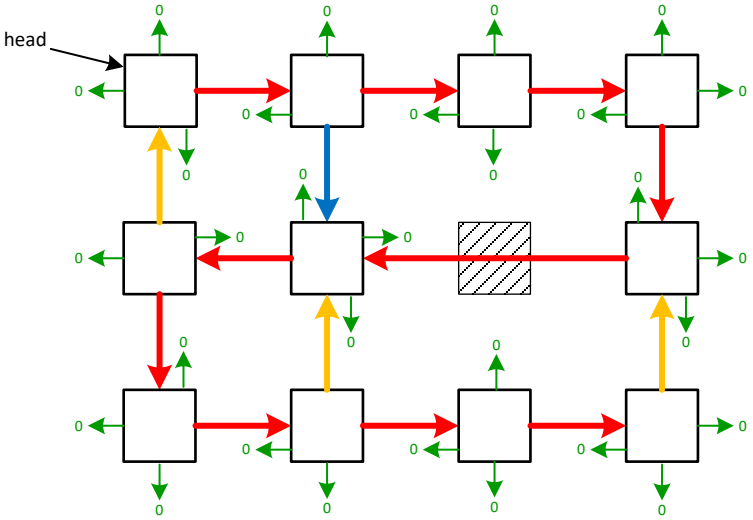


- Rows and Columns
  - Rows and columns are not provided, nor are they added in the Node structure.
  - You need to calculate these as you walk the list.
  - The lists will vary in dimensions, Row x Column (row,col)
  - There will be an even number of columns.

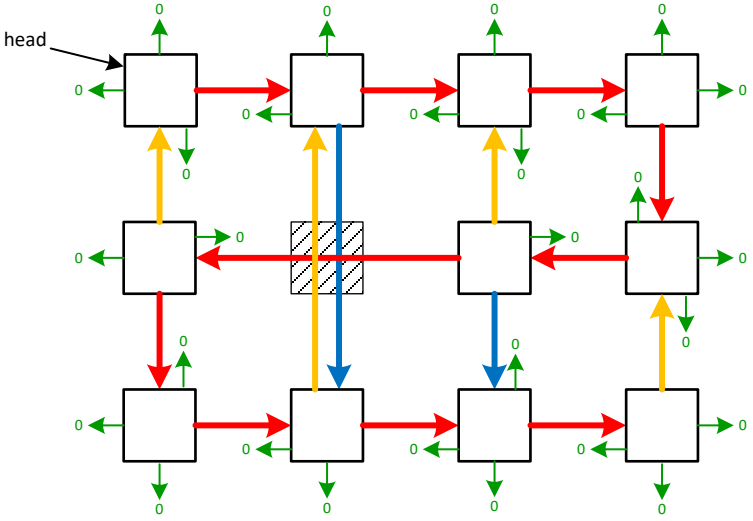


- Deletion
  - Remove only one node at a time
  - Keep horizontal connections, through the deleted node
    - Fix adjacent nodes connections
  - Keep vertical connections, through the delete node
    - Fix adjacent nodes connections
  - Many examples to follow

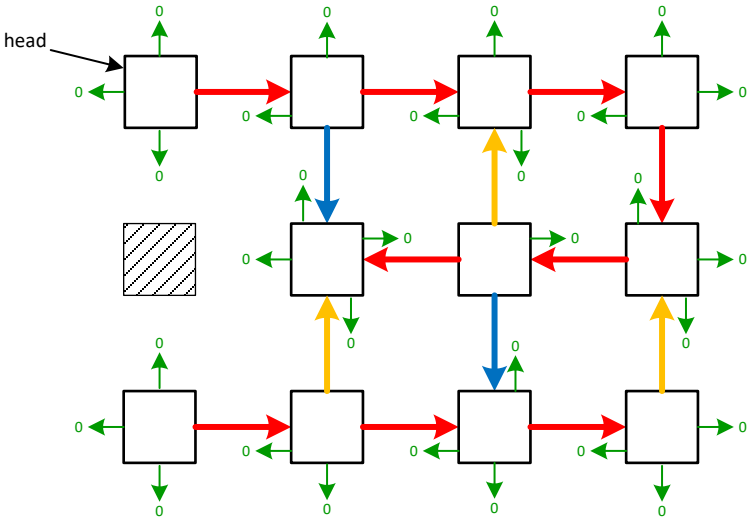
Delete Node (1,2)



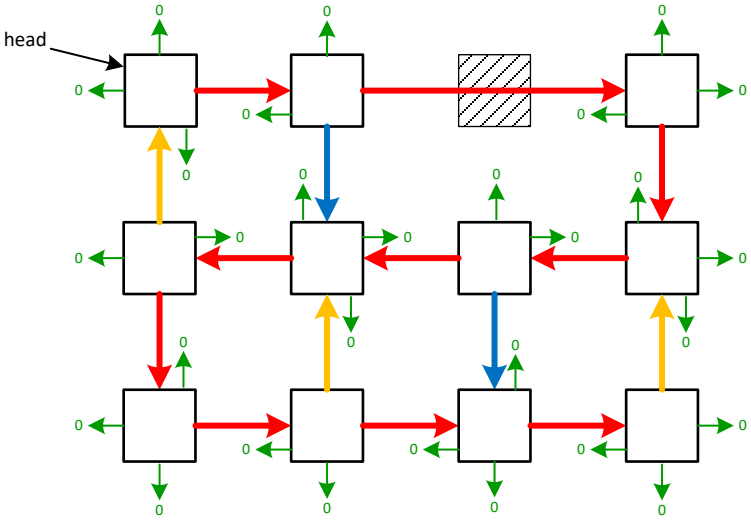
Delete Node (1,1)



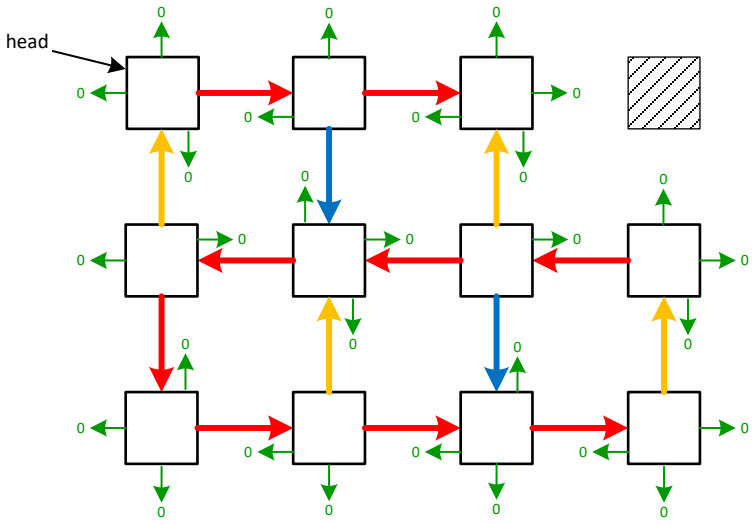
Delete Node (1,0)



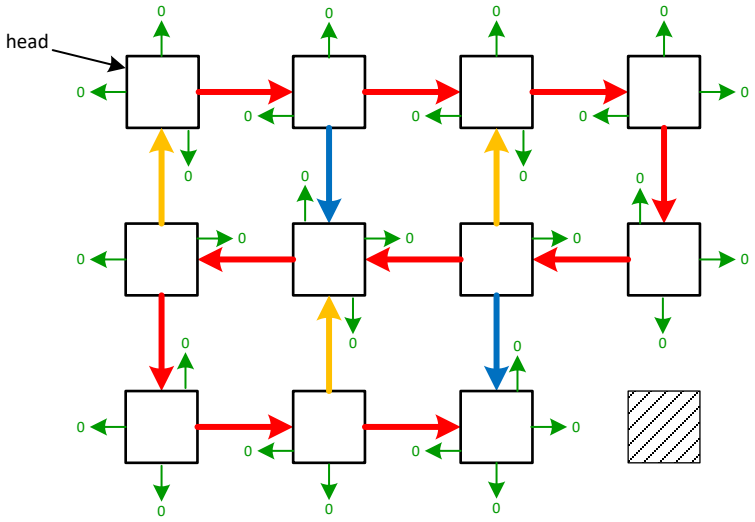
Delete Node (0,2)



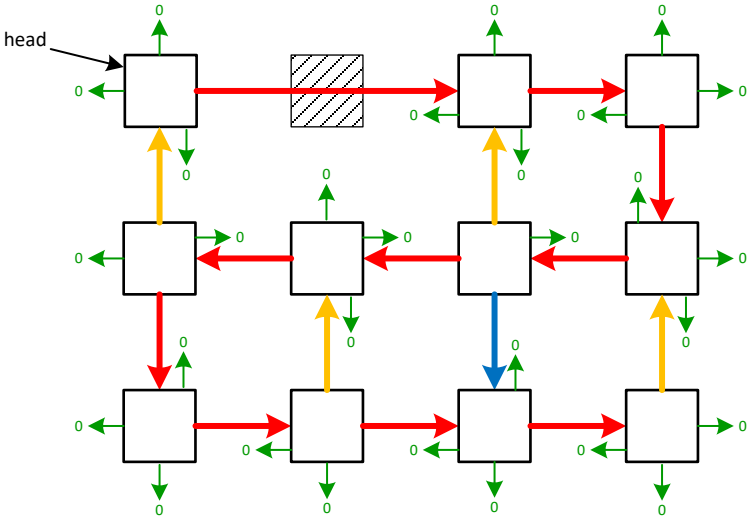
Delete Node (0,3)



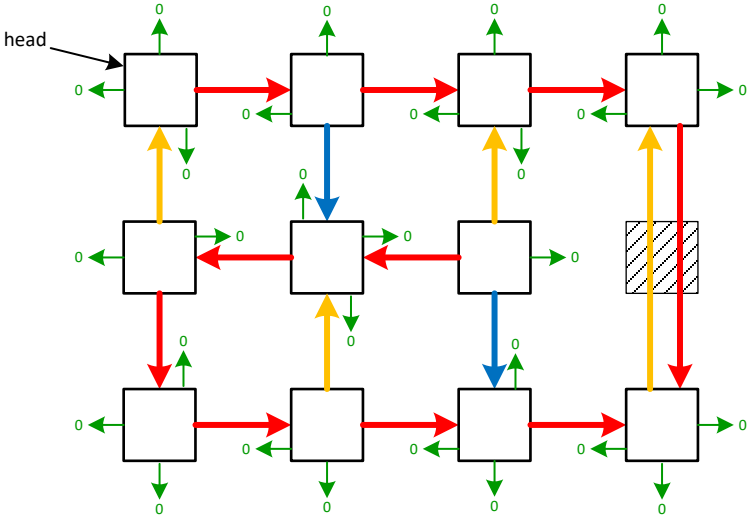
Delete Node (2,3)



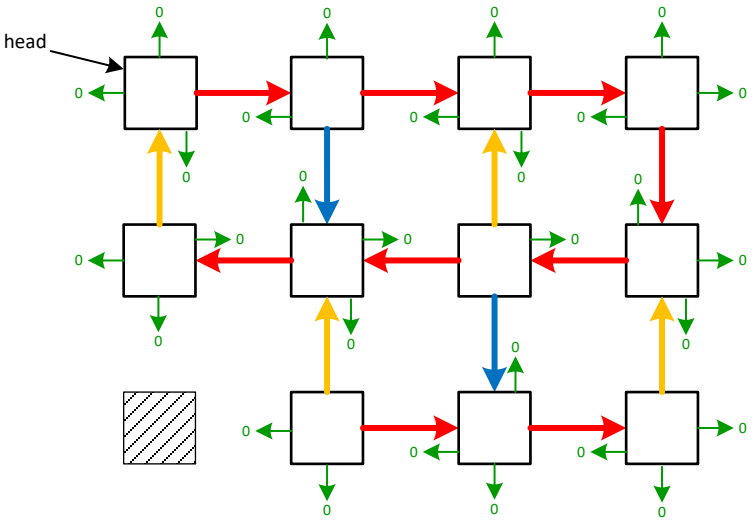
Delete Node (0,1)



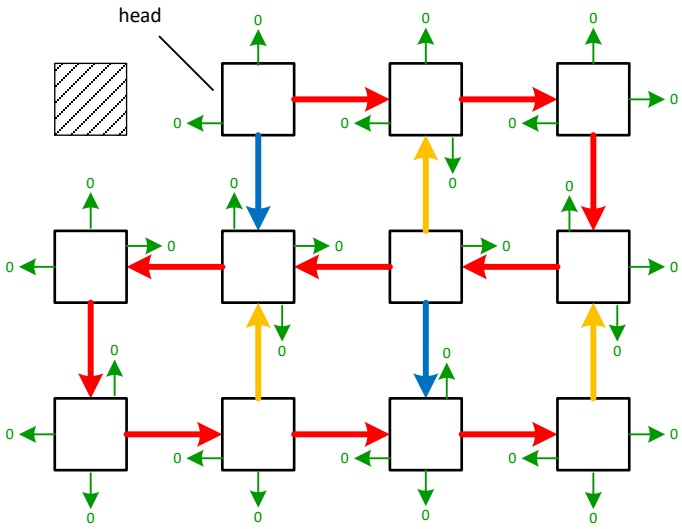
Delete Node (1,3)



Delete Node (2,0)



Delete Node (0,0)



Validation

Simple checklist to make sure that everything is submitted correctly

- Is the project compiling and running without any errors or warnings?
- Does the project run ***ALL*** the unit tests execute without crashing?
- Is the submission report filled in and submitted to perforce?
- Follow the verification process for perforce
    - Is all the code there and compiles “as-is”?
    - No extra files
- Is the project leaking memory?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
    - Iteration is easy and it helps.
    - Perforce is good at it.
- Do diagrams…