YEAH A2: Fun with Collections

CS106B Summer '21
Assignment 2
Jin-Hee Lee & Grant Bishko

Assignment 2 ~Logistics~

A2 is due on Wednesday, July 7th at 11:59pm PT

Grace period until Friday, July 9th at 11:59pm PT

As with all assignments this quarter, your work is ***individual*** (no sharing code!)

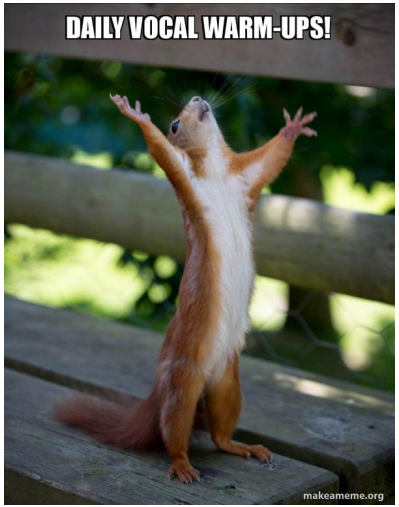
Assignment 2

1. Warmup
2. Maze
3. Search Engine
4. Beyond Algorithmic Analysis

Assignment 2

1. **Warmup**
2. Maze
3. Search Engine
4. Beyond Algorithmic Analysis

Warmups


DAILY VOCAL WARM-UPS!
makeameme.org


I NOTICED YOU DON'T WARM UP
I TOO LIKE TO LIVE DANGEROUSLY


WARM-UP?
HOW ABOUT
YES!

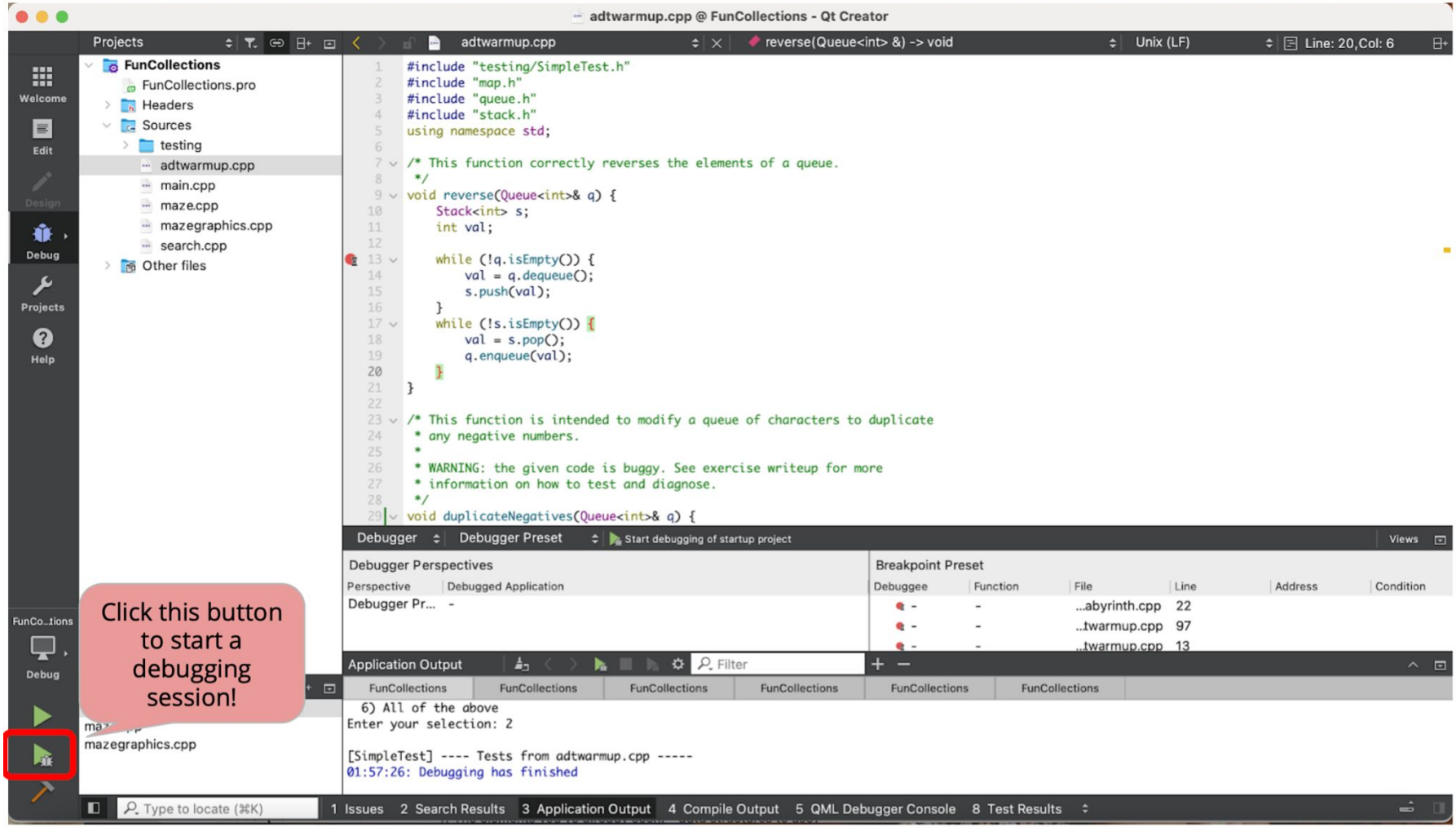
Warmups

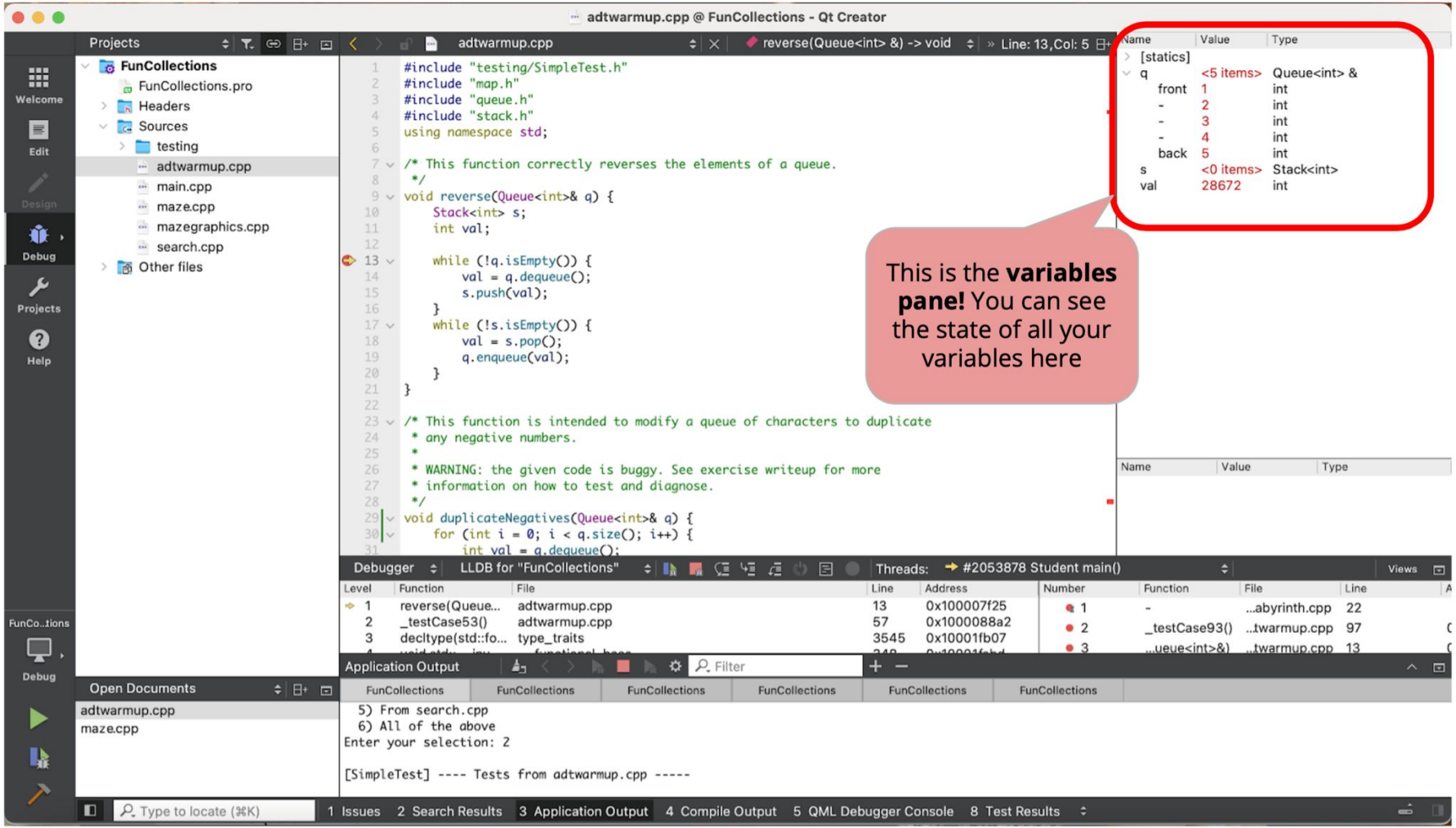
Welcome to the ***debugger***!!

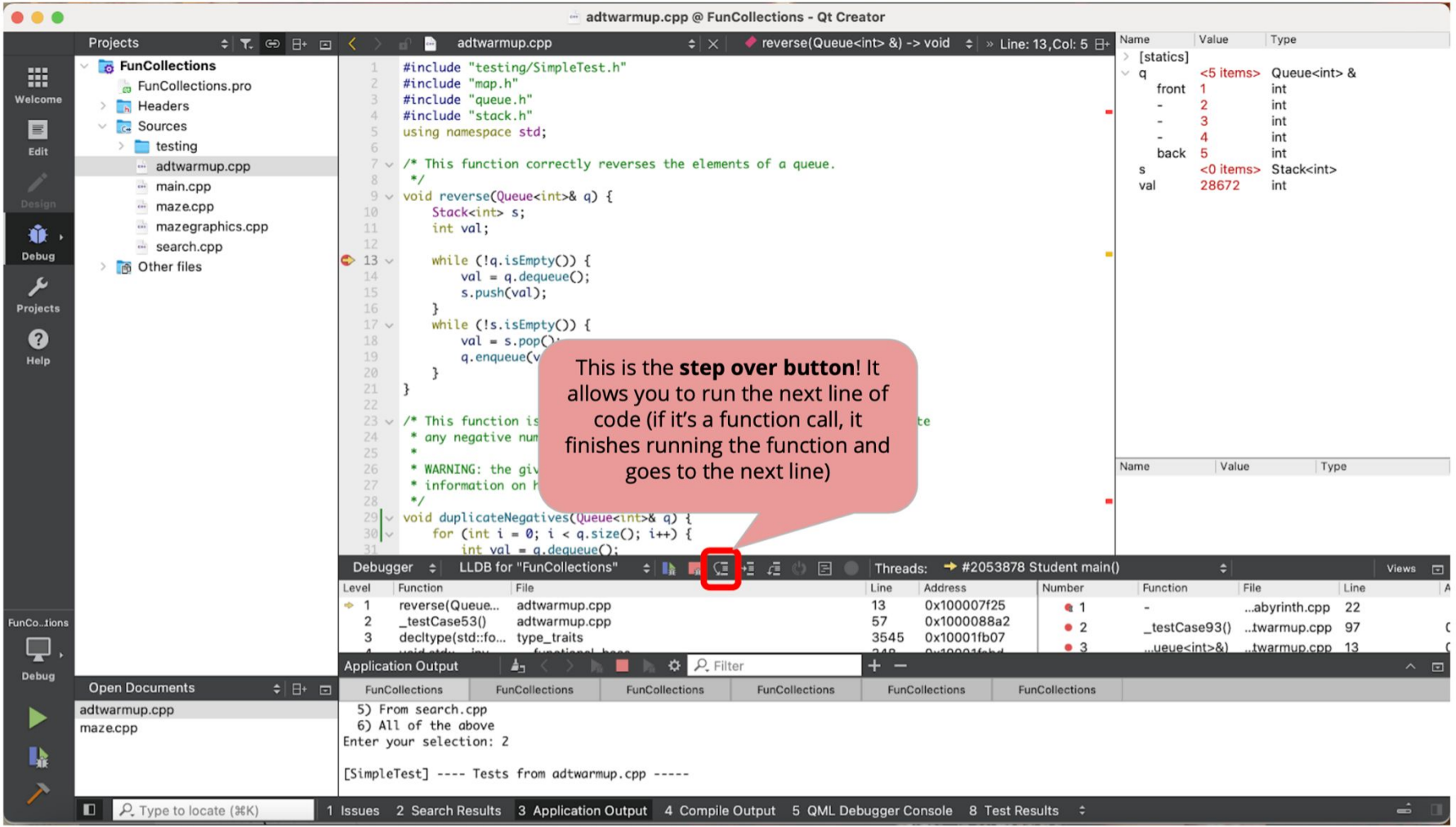
We hate bugs!

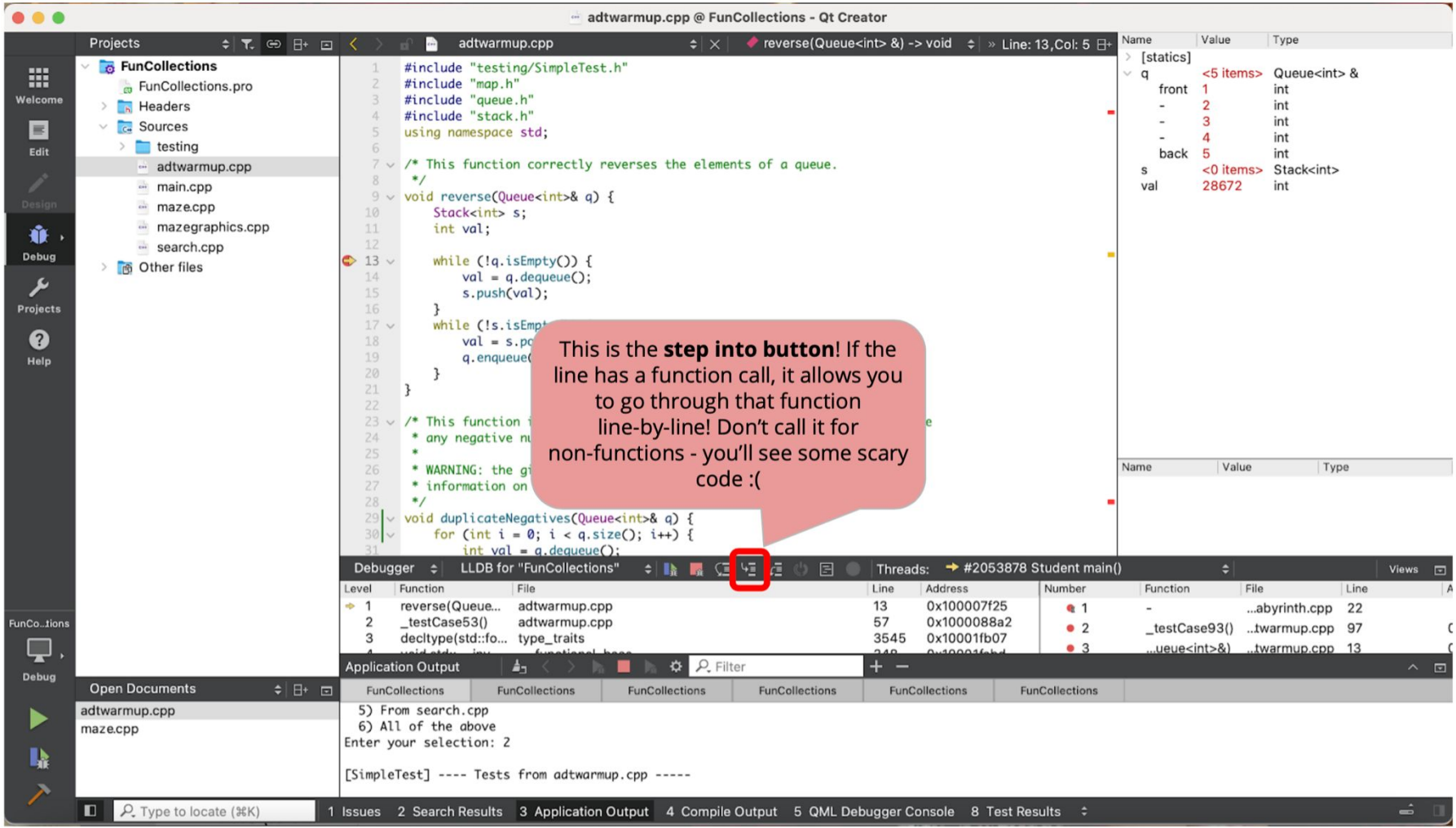
The ***QT Creator Debugger is a SUPER useful tool*** we can use to catch bugs and poke around our code

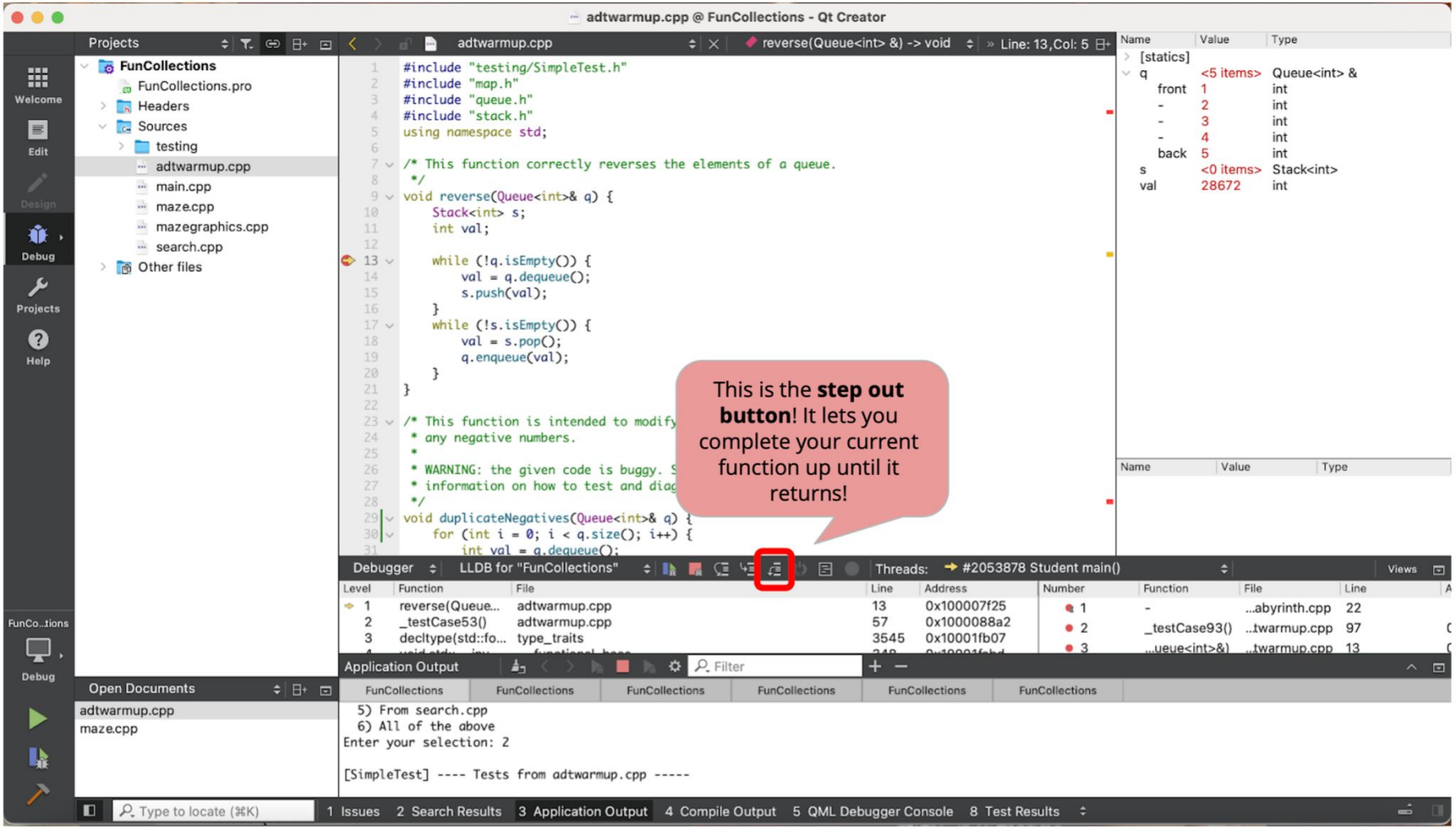
How do we use the debugger? Great question!

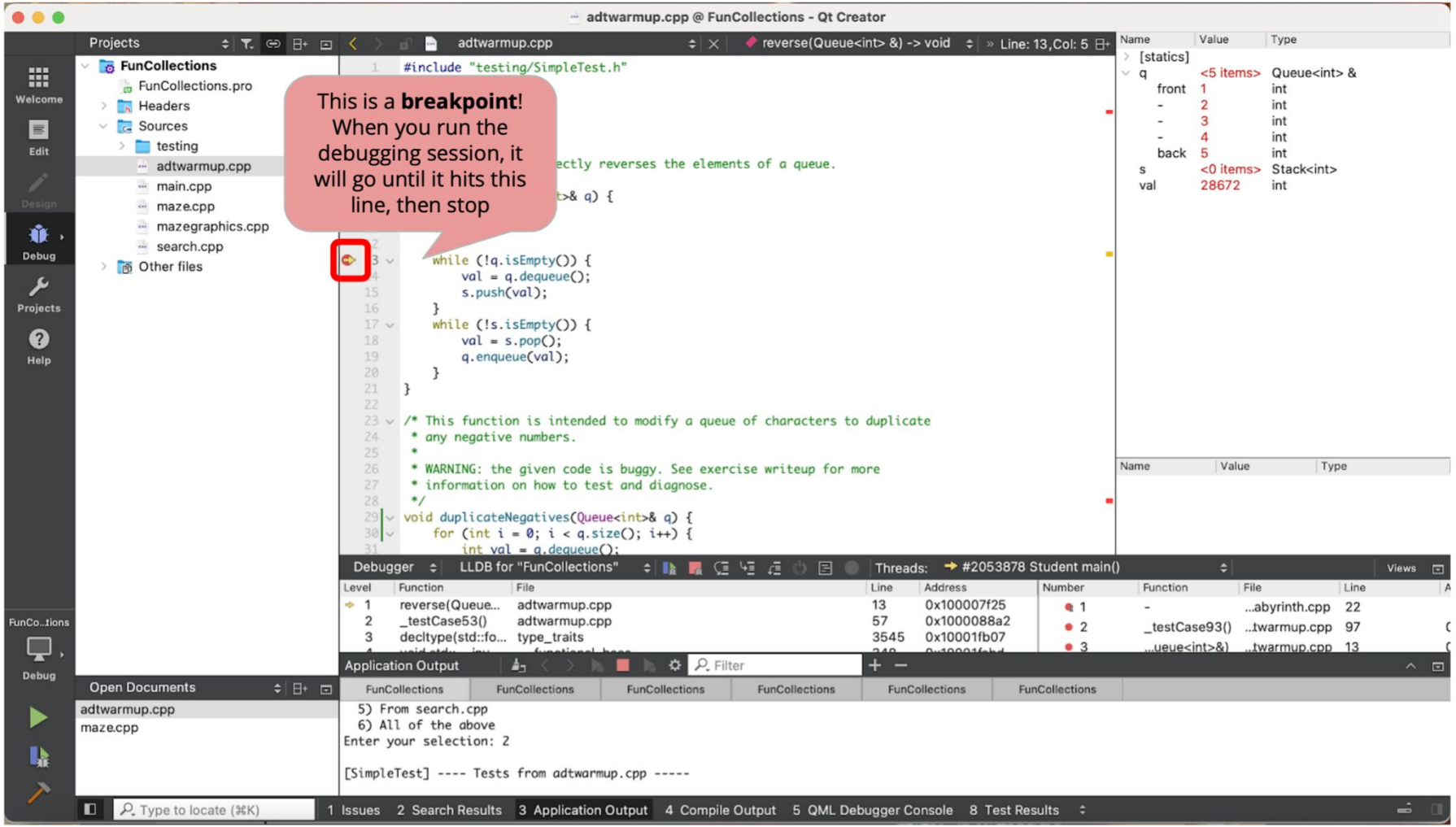

adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
adtwarmup.cpp
reverse(Queue<int> &) -> void
Unix (LF)
Line: 20,Col: 6
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(val);
}
}
/* This function is intended to modify a queue of characters to duplicate
* any negative numbers.
*
* WARNING: the given code is buggy. See exercise writeup for more
* information on how to test and diagnose.
*/
void duplicateNegatives(Queue<int>& q) {
Debugger
Debugger Preset
Start debugging of startup project
Views
Debugger Perspectives
Perspective
Debugged Application
Debugger Pr...
-
Breakpoint Preset
Debuggee
Function
File
Line
Address
Condition
...abyrinth.cpp 22
...twarmup.cpp 97
...twarmup.cpp 13
Application Output
Filter
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
01:57:26: Debugging has finished
Click this button to start a debugging session!
mazegraphics.cpp
FunCo...tions
Debug
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results

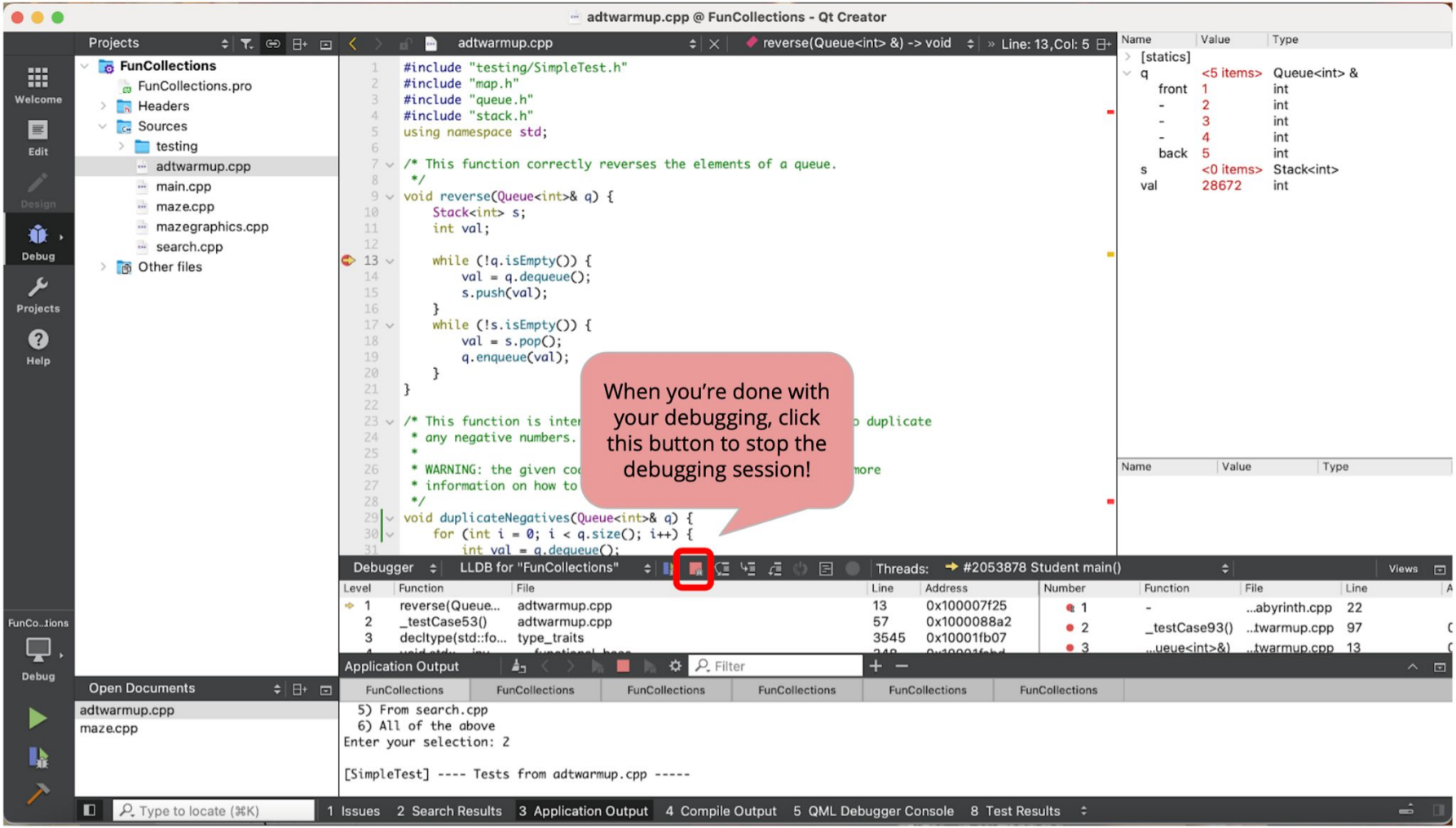

adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
adtwarmup.cpp
reverse(Queue<int> &) -> void
Line: 13,Col: 5
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(val);
}
}
/* This function is intended to modify a queue of characters to duplicate
* any negative numbers.
*
* WARNING: the given code is buggy. See exercise writeup for more
* information on how to test and diagnose.
*/
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
int val = q.dequeue();
Name Value Type
[statics]
q <5 items> Queue<int> &
front 1 int
- 2 int
- 3 int
- 4 int
back 5 int
s <0 items> Stack<int>
val 28672 int
This is the variables pane! You can see the state of all your variables here
Debugger
LLDB for "FunCollections"
Threads: #2053878 Student main()
Views
Level Function File Line Address
1 reverse(Queue... adtwarmup.cpp 13 0x100007f25
2 _testCase53() adtwarmup.cpp 57 0x1000088a2
3 decltype(std::fo... type_traits 3545 0x10001fb07
Number Function File Line
1 - ...abyrinth.cpp 22
2 _testCase93() ...twarmup.cpp 97
3 ...ueue<int>&) ...twarmup.cpp 13
Application Output
Filter
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Open Documents
adtwarmup.cpp
maze.cpp
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results


adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
adtwarmup.cpp
reverse(Queue<int> &) -> void
Line: 13,Col: 5
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(v
}
}
/* This function is
* any negative num
*
* WARNING: the giv
* information on H
*/
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
int val = q.dequeue();
This is the step over button! It allows you to run the next line of code (if it's a function call, it finishes running the function and goes to the next line)
Name
Value
Type
[statics]
q
<5 items>
Queue<int> &
front
1
int
-
2
int
-
3
int
-
4
int
back
5
int
s
<0 items>
Stack<int>
val
28672
int
Debugger
LLDB for "FunCollections"
Threads:
#2053878 Student main()
Views
Level
Function
File
Line
Address
1
reverse(Queue...
adtwarmup.cpp
13
0x100007f25
2
_testCase53()
adtwarmup.cpp
57
0x1000088a2
3
decltype(std::fo...
type_traits
3545
0x10001fb07
Number
Function
File
Line
1
-
...abyrinth.cpp
22
2
_testCase93()
...twarmup.cpp
97
3
...ueue<int>&)
...twarmup.cpp
13
Application Output
Filter
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Open Documents
adtwarmup.cpp
maze.cpp
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results


adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
adtwarmup.cpp
reverse(Queue<int> &) -> void
Line: 13,Col: 5
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpt
val = s.po
q.enqueue(
}
}
/* This function
* any negative nu
*
* WARNING: the gi
* information on
*/
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
int val = q.dequeue();
This is the step into button! If the line has a function call, it allows you to go through that function line-by-line! Don't call it for non-functions - you'll see some scary code :(
Name
Value
Type
[statics]
q
<5 items>
Queue<int> &
front
1
int
-
2
int
-
3
int
-
4
int
back
5
int
s
<0 items>
Stack<int>
val
28672
int
Debugger
LLDB for "FunCollections"
Threads:
#2053878 Student main()
Views
Level
Function
File
Line
Address
1
reverse(Queue...
adtwarmup.cpp
13
0x100007f25
2
_testCase53()
adtwarmup.cpp
57
0x1000088a2
3
decltype(std::fo...
type_traits
3545
0x10001fb07
Number
Function
File
Line
1
-
...abyrinth.cpp
22
2
_testCase93()
...twarmup.cpp
97
3
...ueue<int>&)
...twarmup.cpp
13
Application Output
Filter
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Open Documents
adtwarmup.cpp
maze.cpp
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results


adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
adtwarmup.cpp
reverse(Queue<int> &) -> void
Line: 13,Col: 5
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(val);
}
}
/* This function is intended to modify
* any negative numbers.
*
* WARNING: the given code is buggy.
* information on how to test and diag
*/
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
int val = q.dequeue();
Name Value Type
[statics]
q <5 items> Queue<int> &
front 1 int
- 2 int
- 3 int
- 4 int
back 5 int
s <0 items> Stack<int>
val 28672 int
This is the step out button! It lets you complete your current function up until it returns!
Debugger
LLDB for "FunCollections"
Threads: #2053878 Student main()
Views
Level Function File Line Address
1 reverse(Queue... adtwarmup.cpp 13 0x100007f25
2 _testCase53() adtwarmup.cpp 57 0x1000088a2
3 decltype(std::fo... type_traits 3545 0x10001fb07
Number Function File Line
1 - ...abyrinth.cpp 22
2 _testCase93() ...twarmup.cpp 97
3 ...ueue<int>&) ...twarmup.cpp 13
Application Output
Filter
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Open Documents
adtwarmup.cpp
maze.cpp
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results


adtwarmup.cpp @ FunCollections - Qt Creator
This is a **breakpoint**! When you run the debugging session, it will go until it hits this line, then stop
Welcome
Edit
Design
Debug
Projects
Help
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Open Documents
adtwarmup.cpp
maze.cpp
#include "testing/SimpleTest.h"
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(val);
/* This function is intended to modify a queue of characters to duplicate
* any negative numbers.
* WARNING: the given code is buggy. See exercise writeup for more
* information on how to test and diagnose.
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
Name Value Type
[statics]
q <5 items> Queue<int> &
front 1 int
2 int
3 int
4 int
back 5 int
s <0 items> Stack<int>
val 28672 int
Debugger
LLDB for "FunCollections"
Threads: #2053878 Student main()
Views
Level Function File Line Address
1 reverse(Queue... adtwarmup.cpp 13 0x100007f25
2 _testCase53() adtwarmup.cpp 57 0x1000088a2
3 decltype(std::fo... type_traits 3545 0x10001fb07
Number Function File Line
1 - ...abyrinth.cpp 22
2 _testCase93() ...twarmup.cpp 97
3 ...ueue<int>&) ...twarmup.cpp 13
Application Output
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results


adtwarmup.cpp @ FunCollections - Qt Creator
Projects
FunCollections
FunCollections.pro
Headers
Sources
testing
adtwarmup.cpp
main.cpp
maze.cpp
mazegraphics.cpp
search.cpp
Other files
Welcome
Edit
Design
Debug
Projects
Help
Open Documents
adtwarmup.cpp
maze.cpp
adtwarmup.cpp
reverse(Queue<int> &) -> void
Line: 13,Col: 5
#include "testing/SimpleTest.h"
#include "map.h"
#include "queue.h"
#include "stack.h"
using namespace std;
/* This function correctly reverses the elements of a queue.
*/
void reverse(Queue<int>& q) {
Stack<int> s;
int val;
while (!q.isEmpty()) {
val = q.dequeue();
s.push(val);
}
while (!s.isEmpty()) {
val = s.pop();
q.enqueue(val);
}
}
/* This function is inter
* any negative numbers.
*
* WARNING: the given co
* information on how to
*/
void duplicateNegatives(Queue<int>& q) {
for (int i = 0; i < q.size(); i++) {
int val = q.dequeue();
o duplicate
more
When you're done with your debugging, click this button to stop the debugging session!
Name
Value
Type
[statics]
q
<5 items>
Queue<int> &
front
1
int
-
2
int
-
3
int
-
4
int
back
5
int
s
<0 items>
Stack<int>
val
28672
int
Debugger
LLDB for "FunCollections"
Threads:
#2053878 Student main()
Views
Level
Function
File
Line
Address
1
reverse(Queue...
adtwarmup.cpp
13
0x100007f25
2
_testCase53()
adtwarmup.cpp
57
0x1000088a2
3
decltype(std::fo...
type_traits
3545
0x10001fb07
Number
Function
File
Line
1
-
...abyrinth.cpp
22
2
_testCase93()
...twarmup.cpp
97
3
...ueue<int>&)
...twarmup.cpp
13
Application Output
Filter
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
FunCollections
5) From search.cpp
6) All of the above
Enter your selection: 2
[SimpleTest] ---- Tests from adtwarmup.cpp -----
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML Debugger Console
8 Test Results

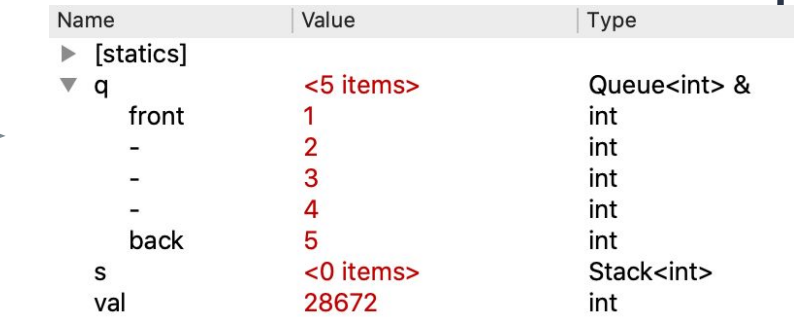
Warmups

1) View ADTs in debugger
2) Test duplicateNegatives
3) Debug duplicateNegatives
4) Recognize a common ADT error in **`removeMatchPair`**

Responses for the warmups will all be written as short answers in **`short_answer.txt`**

1) View ADTs in Debugger

1. Set a breakpoint on the beginning of the first while loop
2. Run code in debug mode
3. Look at the variable pane
4. “Step Over” the code until the stack s is <1 items>



Name	Value	Type
▶ [statics]		
▼ q	<5 items>	Queue<int> &
front	1	int
-	2	int
-	3	int
-	4	int
back	5	int
s	<0 items>	Stack<int>
val	28672	int

Q1. The display of the Stack in the debugger uses the labels **`top`** and **`bottom`** to mark the two ends of the stack. How are the contents labeled when the Stack contains only one element?

2) Test DuplicateNegatives

duplicateNegatives intends to:

- Modify a Queue<int> to duplicate each negative number
  - {3, -5, 10} → {3, -5, -5, 10}
  - But it's buggy!

Q2. For which type of inputs does the function produce a correct result?

Q3. For which type of inputs does the function go into an infinite loop?

3) Debug DuplicateNegatives

Now that you have fully tested the function, it is time to debug it and fix the code!

- Should only take 1-2 lines of code to fix the problem
- Make sure you understand part 2 before doing this warmup!!

Q4. What is the bug within **`duplicateNegatives`** that causes it to get stuck in an infinite loop?

4) Recognize common ADT Error in removeMatchPair

The function removeMatchPair is intended to modify a Map<string, string> to remove any pair where the key is equal to the value.

But, this error comes up when running some of the provided tests!

```
Test failed due to the program triggering an ErrorException.

This means that the test did not fail because of a call
to EXPECT() or EXPECT_ERROR() failing, but rather because
some code explicitly called the error() function.
```

4) Recognize common ADT Error in removeMatchPair

From Eunji and Jason:

“This means that the test did not fail because of a call to EXPECT() or EXPECT_ERROR() failing”: Your code did not return a wrong answer! We use EXPECT_EQUAL(yourReturnVal, actualAnswer) in our test cases to show that we want yourReturnVal to equal actualAnswer; if EXPECT_EQUAL fails, it means yourReturnVal does not equal actualAnswer so there’s a mistake in your code. That’s not what happened here!

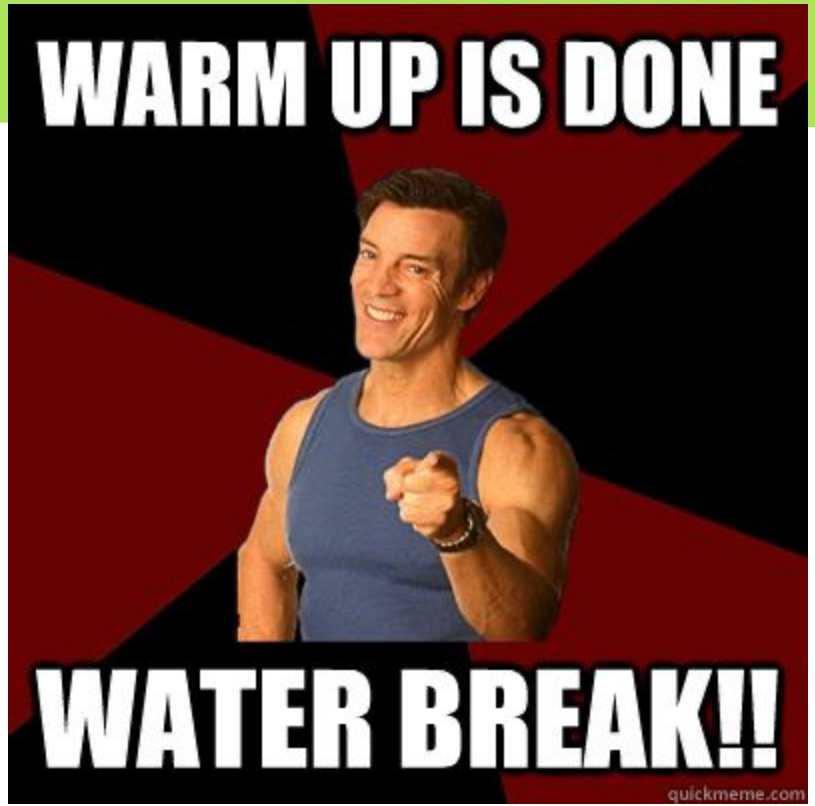
So what happened? **“some code explicitly called the error() function.”** means your code crashed somewhere! This happens with illegal operations: index was out of bounds, attempt to read a non-existent file, or modify a collection while iterating over it, etc.

4) Recognize common ADT Error in removeMatchPair

Q5. What is the value of the variables (as reported in the debugger variable pane) right before the program crashes? What happens if you try to step over the line where the program crashes?

1) First time: set breakpoint in test case, **step into** removeMatchPairs(), then **step over** until you get to the crash. Note the line number of the crash
2) Second time: set breakpoint at the **line of the crash**, then examine state of variables to answer the question

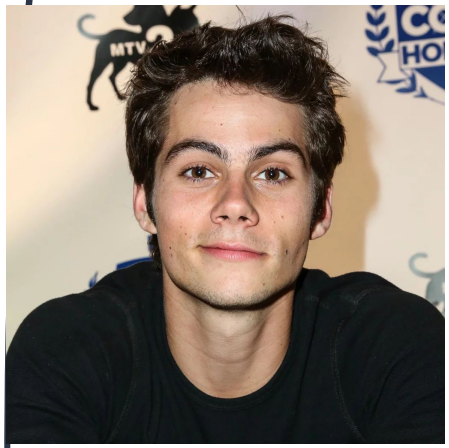
```
93  PROVIDED_TEST("removeMatchPair, remove one") {
94      Map<string, string> m = {{"Thomas", "Tom"}, {"Jan", "Jan"}, {"Margaret", "Meg"}};
95      Map<string, string> expected = {{"Thomas", "Tom"},  {"Margaret", "Meg"}};
96
97      removeMatchPairs(m);
98      EXPECT_EQUAL(m, expected);
99  }
```


WARM UP IS DONE
WATER BREAK!!
quickmeme.com

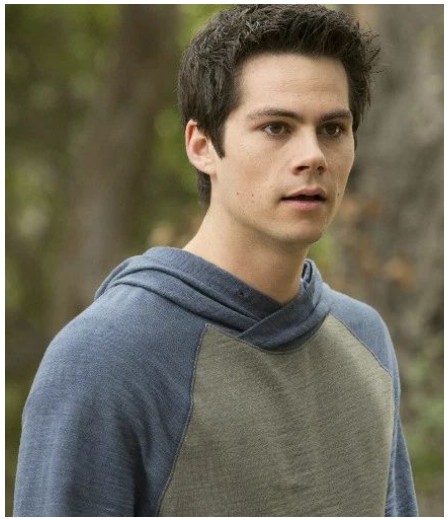
Assignment 2

1. Warmup
2. **Maze**
3. Search Engine
4. Beyond Algorithmic Analysis

Maze -- quick appreciation for Dylan O'Brien



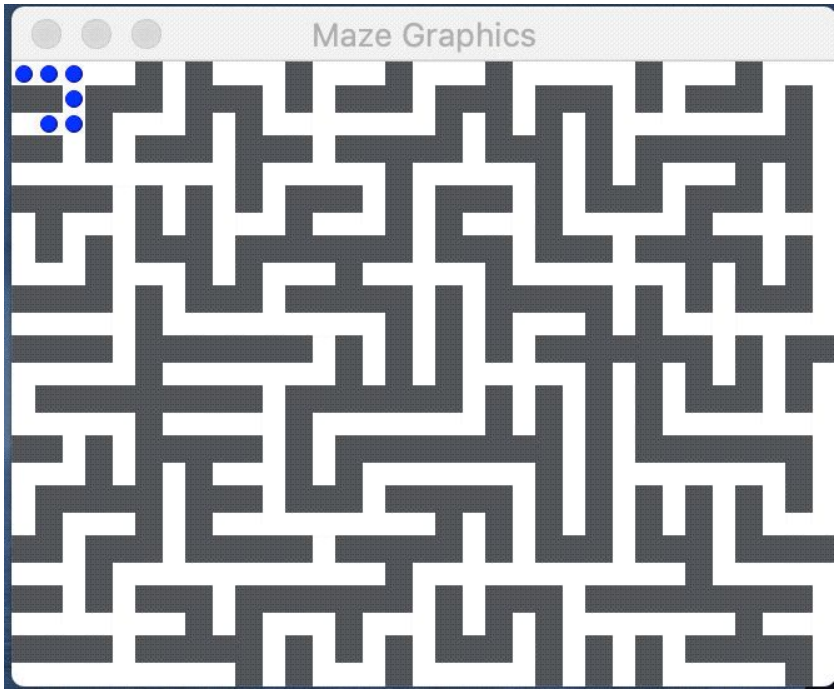
Famous for running from mazes. We will not run!



Maze -- we will solve it!

rip.

Maze -- we will be writing a program to solve one!



Stanford Collections

1. Grids
2. Stacks
3. Queues

Stanford Collections

1. Grids
   a. `grid.inBounds(row, col)`: Returns true if the specified row and column position is inside the bounds of the grid
   b. `grid[row][col] or grid.get(row, col)`: returns value in at the specified row and col
   c. You will be dealing with `Grid<bool>` and some collection of `GridLocations`
2. Stacks
3. Queues

Stanford Collections

1. Grids
2. Stacks
   a. `s.push(val)` : pushes val onto the stack (adds it at the top)
   b. `s.pop()` : removes the topmost (most recently added) value from stack and **returns it**
   c. `s.peek()` : returns the topmost (most recently added) value from stack but **does not remove it**
3. Queues

Stanford Collections

1. Grids
2. Stacks
3. Queues
   a. `q.enqueue(val)` : adds `val` to back of the queue
   b. `q.dequeue()` : removes and returns the first item (top) in the queue

Grid representation of a maze

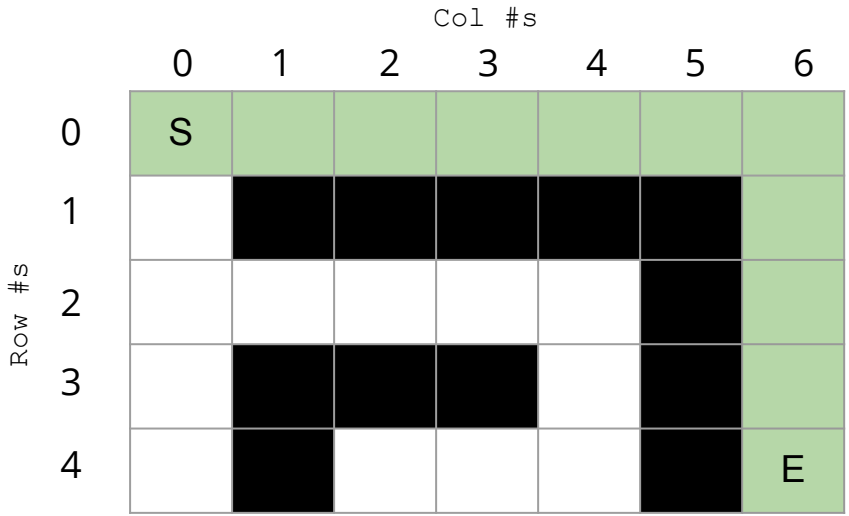
- Maze is represented by a Grid<bool>
  - true → open corridor
  - false → wall
- GridLocation is what it sounds like, a location on a grid:
  - I am still confused???
  - This isn't a *type* I've seen before....?

GridLocation -- How to initialize, compare, change

```
// Declare a new GridLocation
GridLocation chosen; // default initialized to 0,0
chosen.row = 3;      // assign row of chosen
chosen.col = 4;      // assign col of chosen

// Initialize row,col of exit as part of its declaration
GridLocation exit = { maze.numRows()-1, maze.numCols()-1 }; // last row, last col

// You can use a GridLocation to index into a Grid (maze is a Grid)
if (maze[chosen]) // chosen was set to {3, 4} so this accesses maze[3][4]
...

// You can directly compare two GridLocations
if (chosen == exit)
...

// You can also access a GridLocation's row,col separately
if (chosen.row == 0 && chosen.col == 0)
...
```

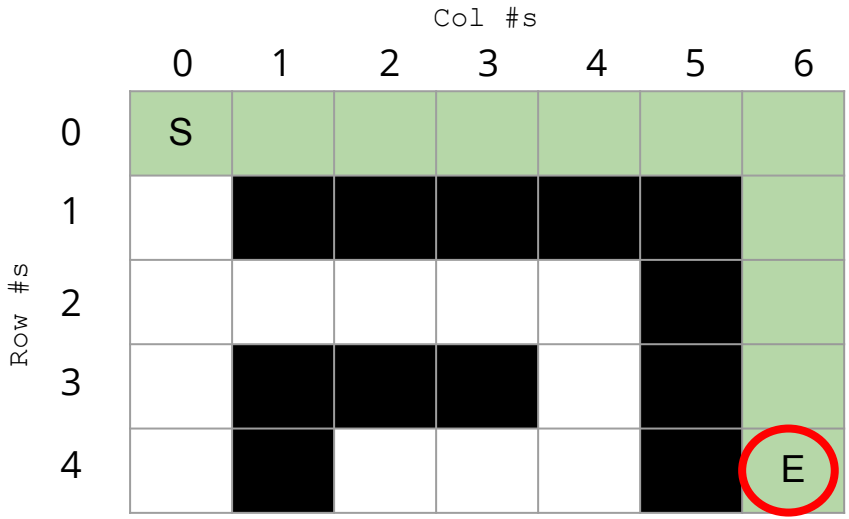
A Stack of GridLocations is a Path

```
Stack<GridLocation> myPath = {r0c0, r0c1, r0c2, r0c3, r0c4, r0c5, r0c6, r1c6, r2c6, r3c6, r4c6}
```



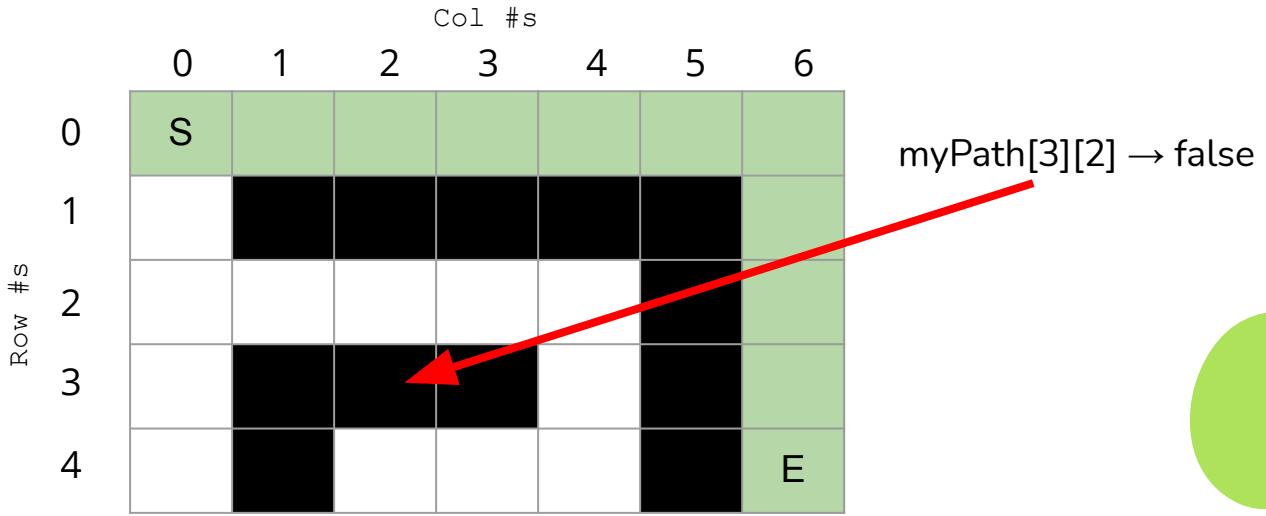
A Stack of GridLocations is a Path

```
Stack<GridLocation> myPath = {r0c0, r0c1, r0c2, r0c3, r0c4, r0c5, r0c6, r1c6, r2c6, r3c6, r4c6}
```



A Stack of GridLocations is a Path

```
Stack<GridLocation> myPath = {r0c0, r0c1, r0c2, r0c3, r0c4, r0c5, r0c6, r1c6, r2c6, r3c6, r4c6}
```



Maze -- three functions!

1) `generateValidMoves()`
2) `checkSolution()`
3) `solveMaze()`

GENERATEVALIDMOVES -- WHAT ARE VALID MOVES?

```
Set<GridLocation> generateValidMoves(Grid<bool>& maze, GridLocation cur)
```

- Write a function that takes in a given maze, and a “cur”, and returns a **Set<gridLocation>** of valid gridLocations that “cur” can move to
- What makes a GridLocation valid?
  - Either directly north, south, east, or west (N, S, E, W) of **cur**
  - Only one "step" away from **cur** in the grid
  - Is open corridor, not a wall (true, not false)
  - Not out-of-bounds of the provided maze (hint: use grid.inBounds!)
- Write your own student tests!! You need 3-4 to make sure your function works!

Maze -- three functions!

1) `generateValidMoves()`
2) **`checkSolution()`**
3) `solveMaze()`

Check a completed path to see if it works!

```
void checkSolution(Grid<bool>& maze, Stack<GridLocation> path)
```

A given path is a valid solution if:

1) **The path must start at the entry (upper left corner) of the maze**
   a) HINT: the entry is the LAST element of the path -- how can you check that location? Remember path.pop() gives you the EXIT… is there a way to check the entry without popping off the *entire* stack?? Think about *when* in your code you want to check that the path starts at {0, 0}!!!

Check a completed path to see if it works!

```
void checkSolution(Grid<bool>& maze, Stack<GridLocation> path)
```

A given path is a valid solution if:

1) **The path must start at the entry (upper left corner) of the maze**
2) **The path must end at the exit (lower right corner) of the maze**
    a) Same idea here, how can you check where the path ends?

Check a completed path to see if it works!

```
void checkSolution(Grid<bool>& maze, Stack<GridLocation> path)
```

A given path is a valid solution if:

1) **The path must start at the entry (upper left corner) of the maze**
2) **The path must end at the exit (lower right corner) of the maze**
3) **Each location in the path is a valid move away from the previous location**
    a) HINT: call the helper function you already wrote to help confirm a move is valid, rather than re-implement its logic!
    b) HINT: you need to keep track of 2 GridLocations here…

Check a completed path to see if it works!

```
void checkSolution(Grid<bool>& maze, Stack<GridLocation> path)
```

A given path is a valid solution if:

1) **The path must start at the entry (upper left corner) of the maze**
2) **The path must end at the exit (lower right corner) of the maze**
3) **Each location in the path is a valid move away from the previous location**
4) **The path contains no loops, i.e. a location appears at most once in the path**
    a) HINT: determine a way to keep track of all the GridLocations you have already seen/been to along your path!

It's a void function... What do I actually do?

checkSolution will do nothing if the path that it is given works and is a good solution!

HOWEVER, if you run into a problem (one of the 4 criteria failed in the previous slides)

- Call an error which suggests that the path is faulty in some way.

```
error("Here is my message about what has gone wrong");
```

Again, write your own tests to check this!

Short Answers again!

Q6. So far in this class, we've passed our data structures by reference. Why do you think **`checkSolution`** passes **`path`** by value instead of by reference?

Q7. After you have written your tests, describe your testing strategy to determine that your **`checkSolution`** works as intended.

Maze -- three functions!

1) `generateValidMoves()`
2) `checkSolution()`
3) **`solveMaze()`**

solveMaze()

We’ve generated valid moves, we’ve checked completed paths to see if they complete the maze...

Now it’s time to **solve the maze!**

solveMaze()

```
Stack<GridLocation> solveMaze(Grid<bool>& maze)
```

- You will be writing a function that takes in a maze, and returns a path (stack of GridLocations) to solve that maze!
- You will be using a **Breadth-First-Search (BFS)** to do so

We will be keeping track of many potential paths, until we find the solution to our maze. Using a queue of paths:

Queue<Stack<GridLocation>>

solveMaze() pseudocode explained

1. Create a queue of paths. A path is a stack of grid locations.
2. Create a length-one path containing just the entry location. Enqueue that path.
    - In our mazes, the entry is always the upper-left corner and exit in the lower-right.

Stack<GridLocation> myPath → has only *one* element in it, and that element is {0, 0}

enqueue myPath onto our full queue of potential solutions

solveMaze() pseudocode explained

1. Create a queue of paths. A path is a stack of grid locations.
2. Create a length-one path containing just the entry location. Enqueue that path.
    - In our mazes, the entry is always the upper-left corner and exit in the lower-right.
3. While there are still more paths to explore:

solveMaze() pseudocode explained

1. Create a queue of paths. A path is a stack of grid locations.
2. Create a length-one path containing just the entry location. Enqueue that path.
    - In our mazes, the entry is always the upper-left corner and exit in the lower-right.
3. While there are still more paths to explore:
    - Dequeue path from queue.
    - If this path ends at exit, this path is the solution!
    - If path does not end at exit:
        - For each viable neighbor from path end, make copy of path, extend by adding neighbor and enqueue it. → New potential solution!
        - A viable neighbor is a location that is a valid move and that has not yet been visited.

Keep track of the locations you visit!

Add graphics!

```
MazeGraphics::drawGrid(Grid<bool>& grid)
MazeGraphics::highlightPath(Stack<GridLocation> path, string color)
```

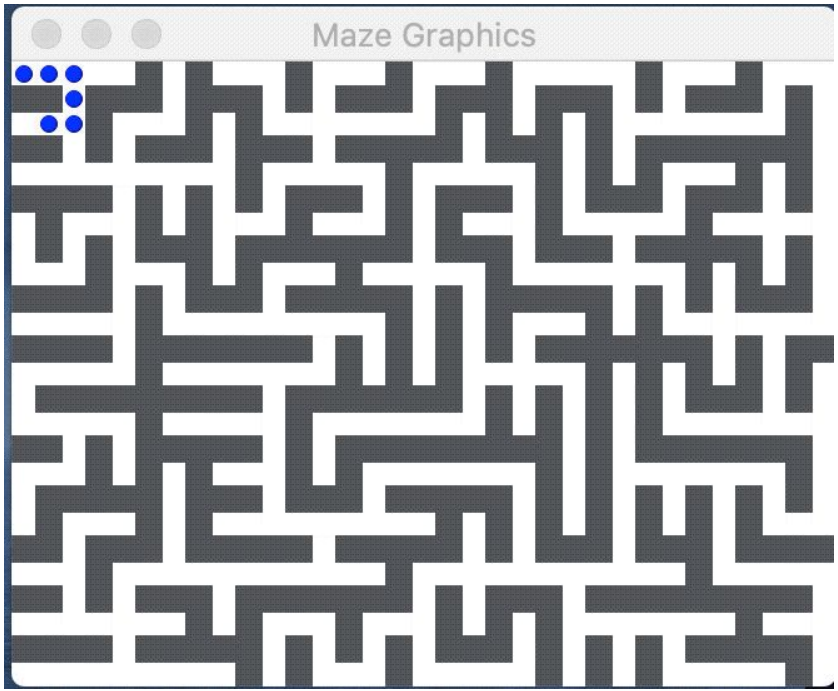
You need to call highlightPath()! Dw about drawGrid() we do that for you!

- Hint: it's only one line of code!
- Call it every time you update the path (aka whenever you dequeue a new candidate path)

Revisiting this graphic

Live-time representation of a BFS!

SO COOL.



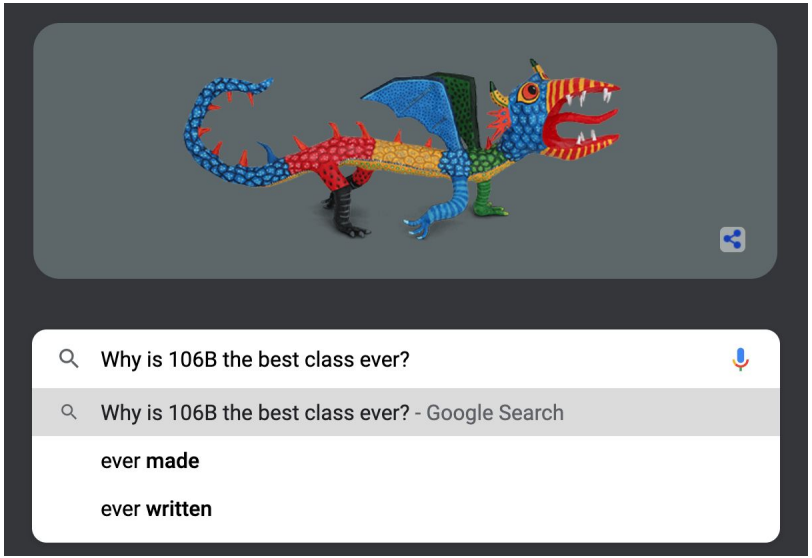
Questions about Maze?

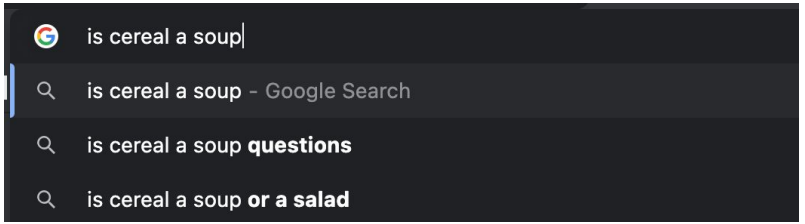
But we DID.
And we did it with code.
Boom.

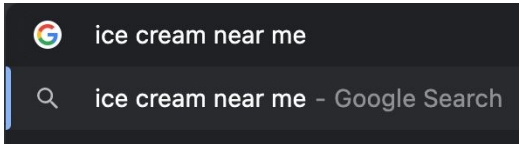
Assignment 2

1. Warmup
2. Maze
3. **Search Engine**
4. Beyond Algorithmic Analysis

Search Engine







Our Version of a Search Engine

- Each web page has a URL ("Uniform Resource Locator")
  - The URL is the page's unique ID
  - We use a string to contain the body text of the page

Our Tasks...

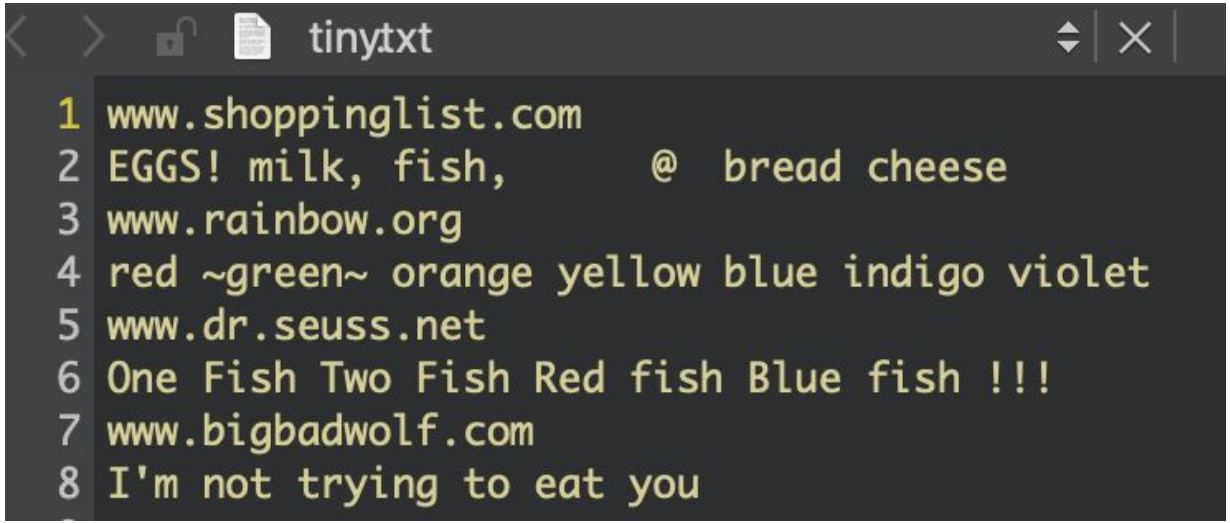
- Process the body text and populate the data structure
- Search for pages that match a search query
- Allow the user to enter many search queries and retrieve the matching web pages

Understanding a Database File

Lines are grouped into pairs:

- First line is the page URL
- Second line is that page's body text represented as a single string

Example File: `tiny.txt`



Searching Efficiently

While that example file was tiny, we will be searching through much bigger files.

Searching through each file on the web would take eons!

So, we will be using an **inverted index** to search.

Inverted Index

Creates a mapping from *content* to *locations*.

Example: a science textbook's index

- You want to read about the mitochondria.
- The index tells you that you can find this word on pages 71, 120.

Inverted Index

Creates a **mapping** from *content* to *locations*.

Example: a science textbook's index

- You want to read about the mitochondria.
- The index tells you that you can find this word on pages 71, 120.

`char` the Charmander, back at it again with the notes on efficiency:

While processing the entire document or set of documents might take a while, we have to do so to create the inverted index.

Once it's been created, searching for words becomes a breeze! AKA - well worth it.

Decomposition Time!

Part 1: `cleanToken()`

Our job is to write `string cleanToken(string token)`

- token: string of characters from the body text
- *return value*: a cleaned version

Part 1: `cleanToken()`

- Remove all punctuation from the beginning and end of a token, but not from inside a token.
  - "!wowwee!" → "wowwee"
  - "wow!wee" → "wow!wee"

a. If a character `c` fulfills `ispunct(c)`, it qualifies as punctuation.
b. Paying attention to the beginning and the end of the token

Part 1: `cleanToken()`

- If the token does not contain at least one letter, return the empty string.
  - "!@#..." → ""

a. At least 1 character `c` in `token` should fulfill `isalpha(c)` for us to treat it as a valid word.

Part 1: `cleanToken()`

- Convert the token to lowercase before returning.
  - "WAH" → "wah"

a. Check strlib.h to see which function converts a string to lowercase!
b. Watch out: make sure you're never trying to index into the empty string.

Part 2: `gatherTokens()`

Not that kind :(

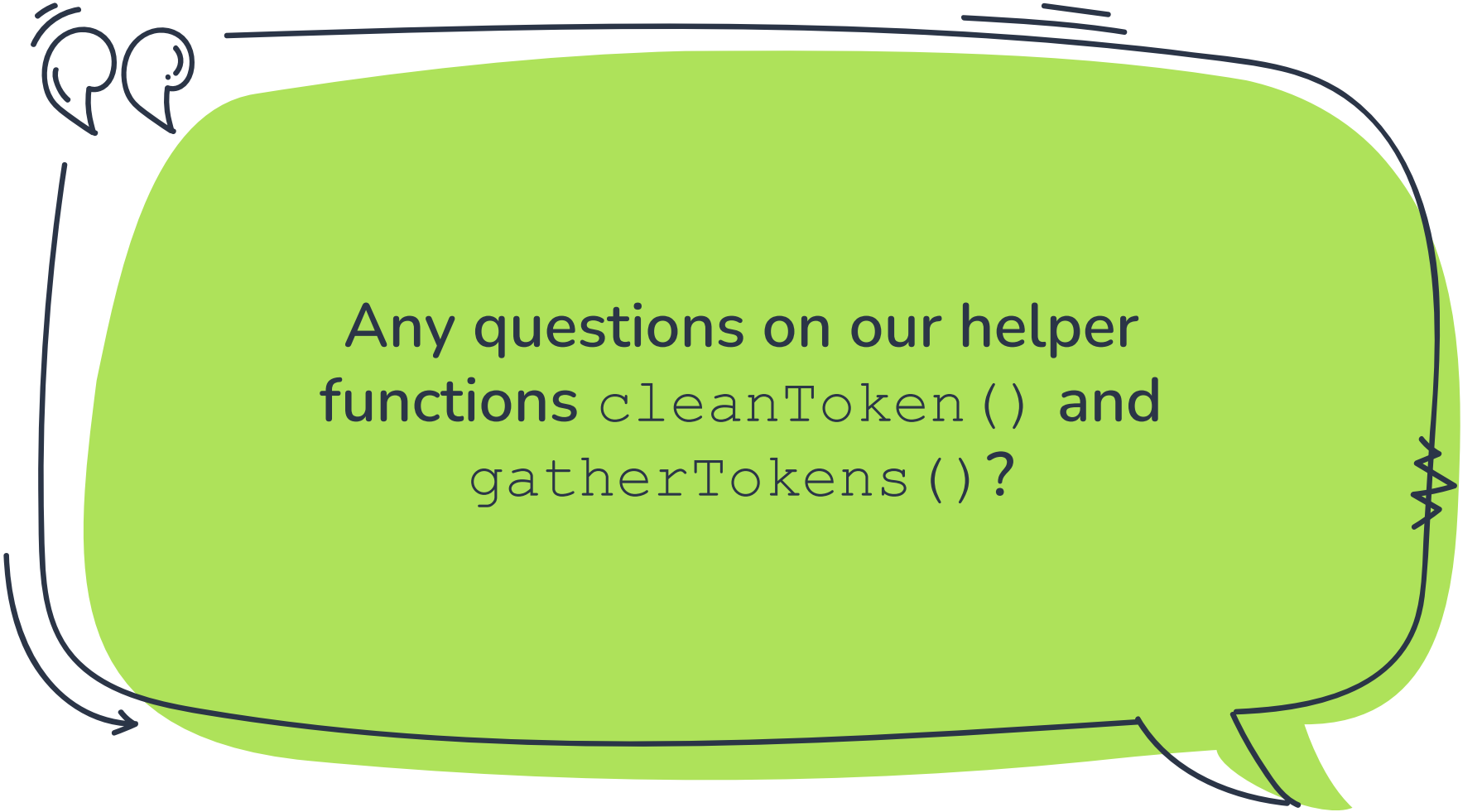
Part 2: `gatherTokens()`

Our job is to write `Set<string> gatherTokens(string bodytext)`

- bodyText: body text from a single web page
- *return value*: all of the unique, cleaned tokens that appear in the body text

Part 2: `gatherTokens()`

1. Tokenize the body text by whitespace
    a. Use stringSplit()
2. Clean each token
    a. Store each cleaned token in a set
        i. Review: why is a set the ideal data structure to use here?
3. Return the set you've made!

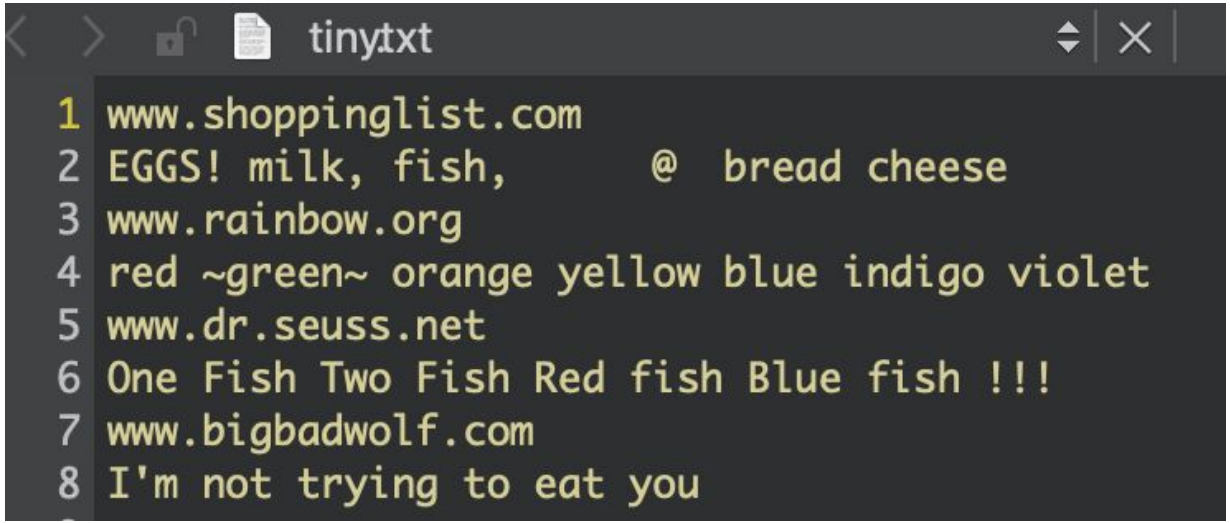

Any questions on our helper functions cleanToken() and gatherTokens()?

Part 3: `buildIndex()`

```
int buildIndex(string dbfile, Map<string, Set<string>>& index)
```

- dbfile: name of database file
- index: the map to be populated
  - Key: token
  - Value: the webpages on which this token is found
  - Review: why is index passed by reference?
- *return value*: number of documents processed

Remember...



These lines come in 2s!

Part 3: `buildIndex()`

1. Read in contents of dbfile
   a. Check the provided `readMazeFile` in maze.cpp to see how to open a file and read the contents into a vector!
   b. You can reuse this code for `buildIndex()`

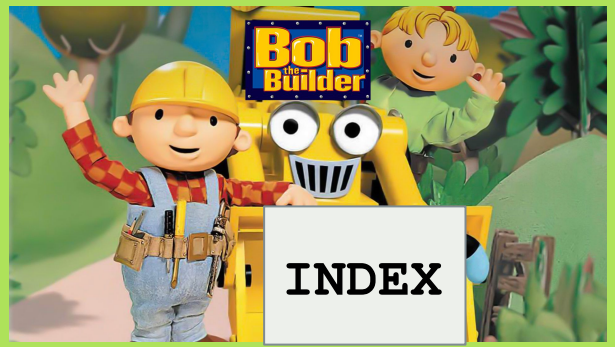
Part 3: `buildIndex()`

1. Read in contents of dbfile
2. Loop through the contents
    a. Hint: think about how to best loop to look at lines 2 at a time!

Part 3: `buildIndex()`

1. Read in contents of dbfile
2. Loop through the contents
3. After reading the line with the body text, gather all unique tokens from that line
    a. For each token, update the `index` map to show that the token can be found on the page's URL

Any questions on `buildIndex()`?



Part 4: `findQueryMatches()`

```
Set<string> findQueryMatches(Map<string, Set<string>>& index, string query)
```

- index: the inverted index map from tokens to URLs
- query: what you'll be searching for
- *return value*: a set of all URLs on which that query is found

Part 4: `findQueryMatches()`

```
Set<string> findQueryMatches(Map<string, Set<string>>& index, string query)
```

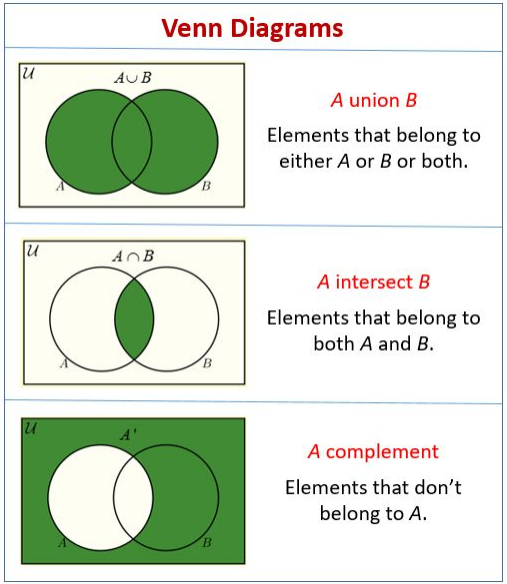
- index: the inverted index map from tokens to URLs
- **query**: what you'll be searching for
- *return value*: a set of all URLs on which that query is found

Query

A query can be

a. A single search term
b. A compound sequence of multiple terms

Compound Queries



Multiple search terms are unioned by default.

- a valid match can be from any search term

If a search term starts with the modifier **+**, then the matches for this term are intersected with existing matches.

- a valid match must be a match for both terms

If a search term starts with the modifier **-**, then the matches for this term are removed from existing matches.

- a valid match for term B must NOT be a valid match for term A

Query

A query can be

a. A single search term
b. A compound sequence of multiple terms

- Search terms should be separated into tokens using `stringSplit()`

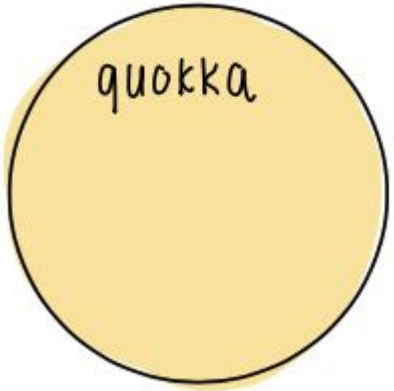
Let's look at the examples on the spec:

Example Queries

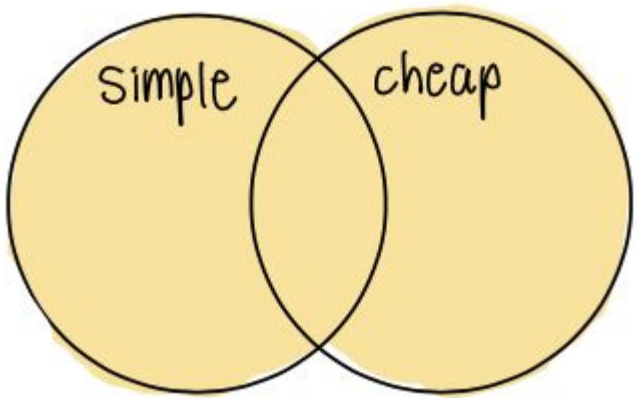
- `quokka`
  - matches all pages containing the term "quokka"
- `simple cheap`
  - means `simple OR cheap`
  - matches pages that contain either "simple" or "cheap" or both
- `tasty +healthy`
  - means `tasty AND healthy`
  - matches pages that contain both "tasty" and "healthy"
- `tasty -mushrooms`
  - means `tasty WITHOUT mushrooms`
  - matches pages that contain "tasty" but do not contain "mushrooms"
- `tasty -mushrooms simple +cheap`
  - means `tasty WITHOUT mushrooms OR simple AND cheap`
  - matches pages that match ((("tasty" without "mushrooms") or "simple") and "cheap")

Note: if there are multiple operators, they should simply be processed from left to right, just like any other query

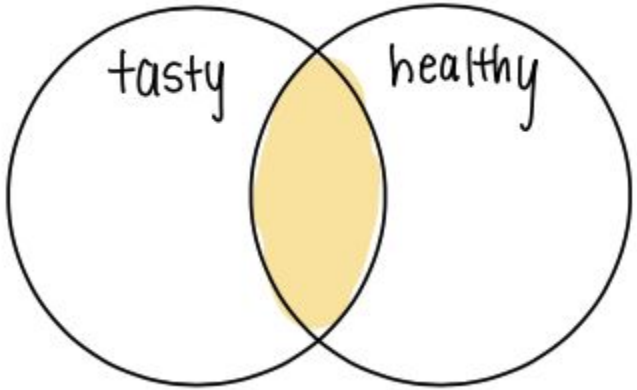
- **quokka**
  - matches all pages containing the term "quokka"



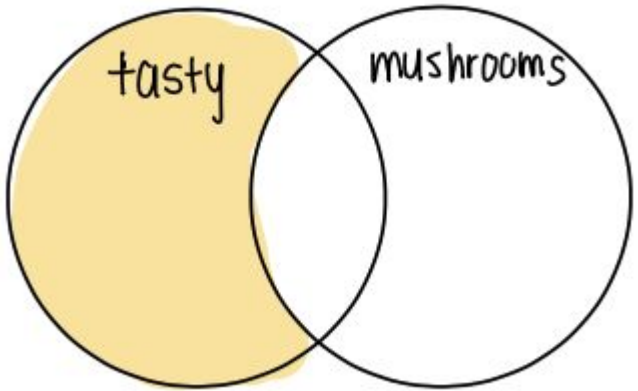
- **`simple cheap`**
  - means **`simple OR cheap`**
  - matches pages that contain either "simple" or "cheap" or both



- **`tasty +healthy`**
  - means **`tasty AND healthy`**
  - matches pages that contain both "tasty" and "healthy"

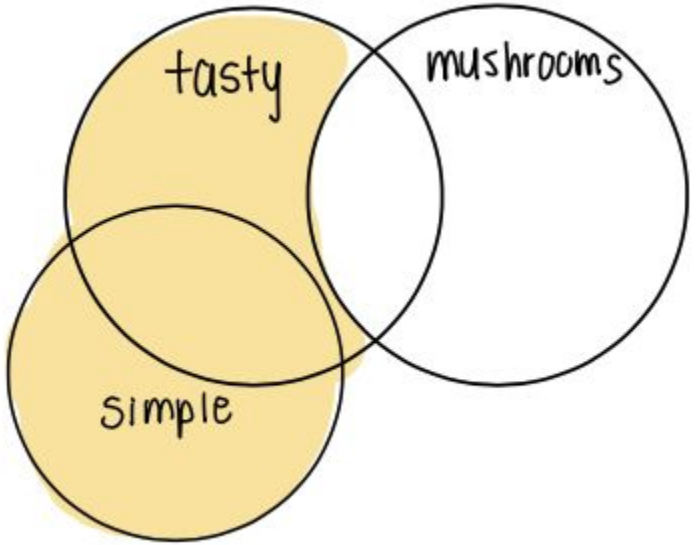


- `tasty -mushrooms`
  - means **`tasty WITHOUT mushrooms`**
  - matches pages that contain "tasty" but do not contain "mushrooms"

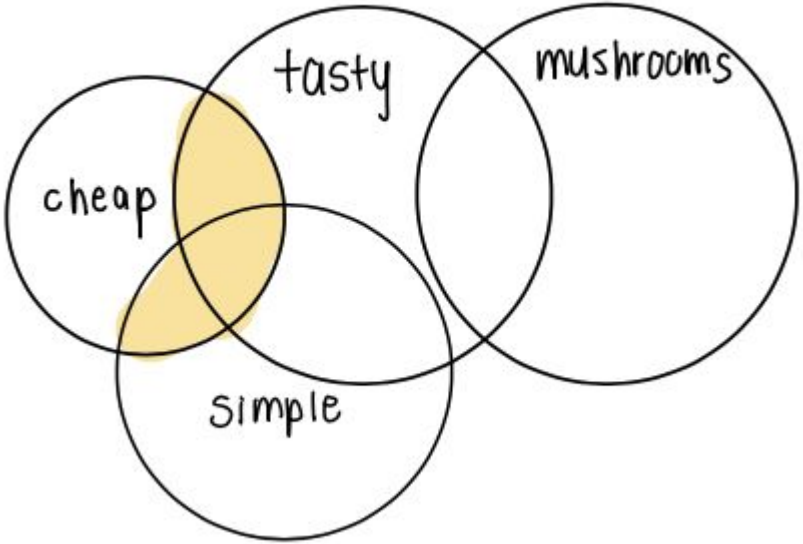


- **`tasty -mushrooms simple +cheap`**
  - means **`tasty WITHOUT mushrooms OR simple AND cheap`**
  - matches pages that match ((("tasty" without "mushrooms") or "simple") and "cheap")

(("tasty" without "mushrooms") or "simple")



(((“tasty” without “mushrooms”) or “simple”) and “cheap”)



Note on Compounds

Don't get too bogged down by the complex compound queries!

Hint: regardless of the length of the query sentence, you should be able to simplify the search for a term to 3 cases.

Part 4: `findQueryMatches()`

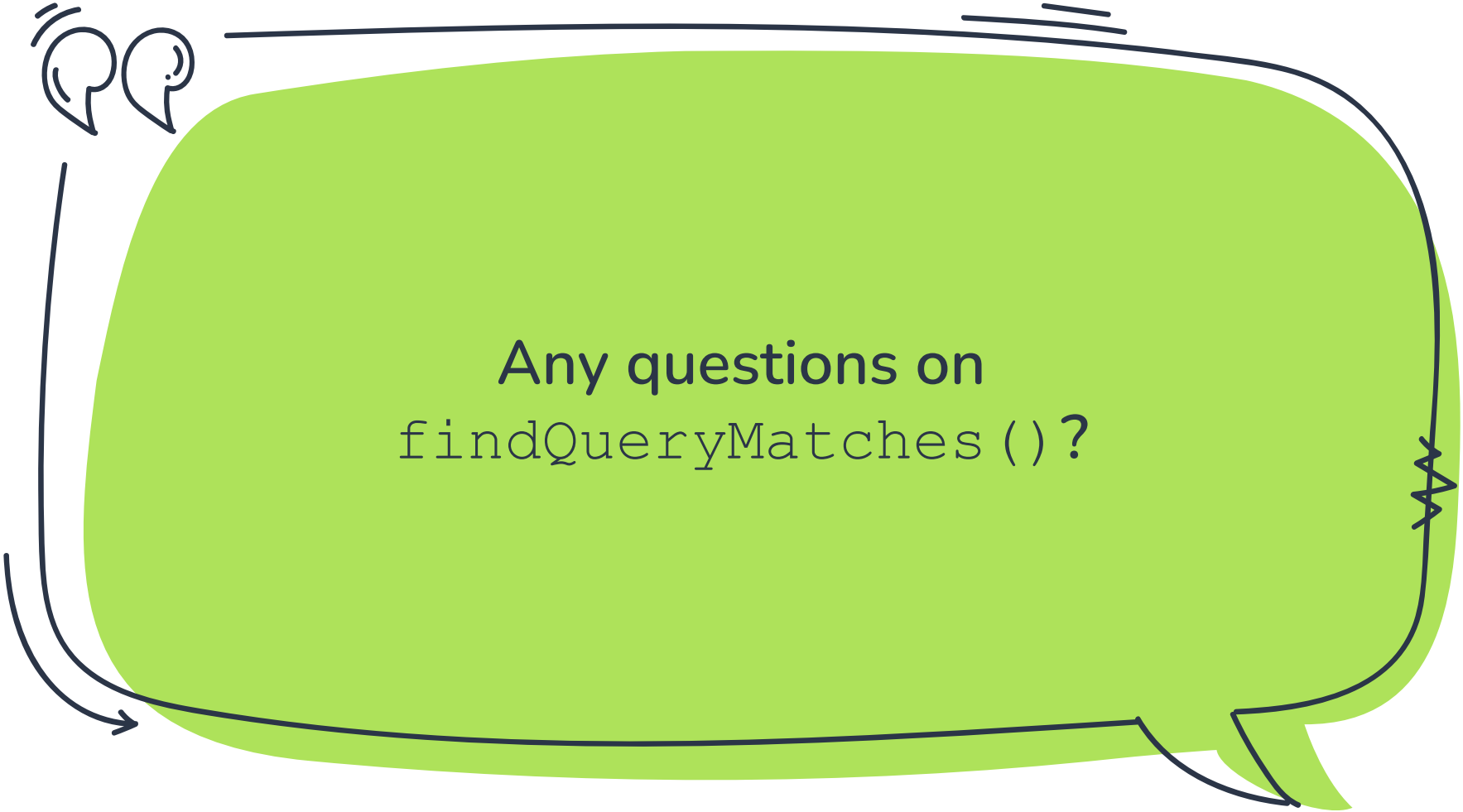
Processing a search term before searching for it:

- Clean your token (the search term)
- Convert to lowercase

Part 4: `findQueryMatches()`

Assumptions we can safely make about the query:

- The query sentence is non-empty and will contain at least 1 search term
- If a search term has a modifier it will be the 1st character in that search term
  - A modifier will not appear on its own as a search term
- The first search term in the query sentence will *never* have a modifier
- No search term will clean to the empty string


Any questions on
findQueryMatches()?

Part 5: `searchEngine()`

```
void searchEngine(string dbfile)
```

- dbfile: the name of the database file being used for the search

Part 5: `searchEngine()`

1. Build the inverted index
2. Print how many web pages were processed to build that index and how many total distinct words were found
3. Enter a loop that asks the user for a query
4. For each query entered, find and print the matching web pages' URLs
5. Understand that the empty string indicates the end of the program

Part 5: `searchEngine()`

- You've done the hard work of creating robust helper functions -- put them to use here!
- Prompting the user
  - Should happen forever, unless the "" input indicates that it should break
- Input, you say?
  - Sounds like our practice using getLine() will come in handy...

Any questions on Search Engine?

You've basically just made Google, I guess.

Haha jk! Unless...

Assignment 2

1. Warmup
2. Maze
3. Search Engine
4. **Beyond Algorithmic Analysis**

Beyond Algorithmic Analysis...

Ethics are a fundamental component of CS education.

We want to foster our ability to think critically about the social impacts of computer programs using the concepts we've discussed in class.

- Like Big-O analysis!

Beyond Algorithmic Analysis...

We'll leave the thinking here to you.

Really, we just want to see thoughtful responses that show you've engaged with the questions and that you are keeping ethics at the forefront of your mind.

Final questions?

Go forth and have fun with collections!

We hope you feel **`char`**ged up!