Alex McNurlin

CS121

Prog Assignment 3

03/09/16

Program Design

Description: The program starts by reading in a file given by the user. The file will contain a maze, with the first 2 lines being the dimensions of the maze. The program reads in the dimensions, declares a dynamic array with those dimensions, then reads in the rest of the maze. It loops through the maze until it finds the entrance and pushes that point into a queue.

The queue will be used to store points in the maze that will be used to find the goal. The queue uses a linked list class written for the previous assignment, and was re purposed to have an array of 2 ints as the value of each node.

Once the program has found the starting point, it dequeues the next point in the queue, takes the points above, below, to the right, and to the left of that given point. It checks if each of these points are a dot '.' or the goal 'G'. If the point is the goal, the program announces success and exits. If the point is a dot, then the program changes the point to a 'v', then enqueues the point. If the point is neither of the two, it does nothing. It then repeats the process by dequeuing the next point. If there is no point to dequeue, then the program announces that the maze cannot be solved and exits.

Classes:

Note: Only the classes that are used are listed. There are more classes that weren't used for this assignment.

* ll.h/ll.cpp – The linked list implementation
  + Ll::Ll() - Constructor function. Initializes empty list
  + Ll::~Ll() - Deconstructor
  + Ll::AddNode(int n[]) – Inserts a node with the value n at the beginning of the list
  + Ll::DeleteTail() - Deletes the last node of the list and returns it.
  + Ll::Size() - Returns the size of the list
* queue.h/queue.cpp – The queue implementation. It's a wrapper for the linked list class.
  + Queue::Enqueue( int n[] ) - Add n[] to the top of the queue.
  + Queue::Dequeue() - Removes and returns the bottom value in the stack
  + Queue::Size() - Returns the size of the queue.

Functions:

* Parse\_char(char\*\* maze, int x, int y, int x\_size, int y\_size, Queue &the\_queue);
  + Takes the maze, the coordinate in question, the maze size, and the queue as function arguments. Finds the character at the point, does error checking to verify it's within bounds, then queues the point if it's valid, exits with success if it's the goal, and does nothing if it's neither.

Programming Log:

Time log:

Mon, Mar 7 –

* 4:30 – 5:30pm
* 7:00 – 12:00 midnight

Tue, Mar 8

* 10:00 – 11:30 am

Total Time spent: 7.5 hours

Things encountered/learned

* Passing an array as a function argument requires declaring the array with the `static` keyword, so the array is not deleted when the function is finished.
* I spent a significant amount of time (almost 3 hours) trying to find a bug in the program. The cause of the bug was an improperly deleted pointer in my linked list. When the node that the pointer pointed to was deleted, the `head` of the list would still point to that address. As a result, the next pointer allocated via the `new` operator would refer to the same point in memory. This caused that new node to point to itself, creating an infinite loop when traversing the list. From this I learned the importance of being careful when deleting pointers. I also learned that newly allocated memory is often the most recently freed memory.

// Output

Script started on Tue 08 Mar 2016 12:01:48 PM PST

root@Laquisha:/home/alexmcnurlin/Google Drive/schoolStuff/cs121/hw3# ./hw3

Enter file to read: maze10.in

S##v#vv##v

vvvvvvvvvv

#vv#v#v#vv

.#v#vvv###

#vvvv#vvvv

.#v###v##v

.##vvvv##v

.#.#v##vvv

..vvv#..#v

##.v##.##G

The end is located at (9, 9)

root@Laquisha:/home/alexmcnurlin/Google Drive/schoolStuff/cs121/hw3# ./hw3

Enter file to read: maze20.in

Svvv#vvvv#####..#..#

v##vvv#vvvv#......#.

vvv##vvv#v#..#.###..

##v#vvv##vv#.#.####.

#vv##v#vvv#.....#...

vv####vvvvv#.##....#

vv###.##vv#.##..#...

#vvvv#vv#v###.#.....

#vvv###vvvv###.###..

vv#vvv#v#vvvv##...#.

v##vv#vv#v##vv###.#.

v#vvv#v#.#..#v...#.#

vv##v#vv#####.##..#.

v#vvvvvvv####..###..

vvvv####v#....#...#.

##v#v#.vvv##...###..

.#vvvvv#v#.#.......#

v#v##v.#.....#.###.#

vvvv#..##.###...##.#

Gvvv#.....#####....#

The end is located at (19, 0)

root@Laquisha:/home/alexmcnurlin/Google Drive/schoolStuff/cs121/hw3# ./hw3

Enter file to read: maze40.in

###vvvSvv##.....##....######........#...

#vvv####vvv###.##.##.#..##...######...#.

.#vvvv#vvvvvv##.......##....##....#.###.

#.#vvvvvv##vvvv###..#...#.##..##.#######

##vvvvv##vv#vv#...####.###.###....####..

vv#vv#v##v#.#vv##.#...#...#..##.#.####.#

v#vvv#vvvvv#.##.######...###.....#.....#

vvvv#v##v#vv#v##..#.####..##..##....####

v###vv#vvvv#vvv#.#.#.#..#.####.......###

vv###v#v##vv#vvv#.##.########....##.###.

#vvvvvv###v#vv##..##....##.#####..###.#.

.###v##vv#v#v##.#..#........#....###.#..

.###v##vvvvvvv#..#.###..######.###....#.

..##v#.#vvv##vv####v####..#########.#..#

##vvv###vvvvvvv###vv###.##.######..#.##.

..##vvv###v##vvv#vv##..#...########..##.

.###v##vv#vv##vv#v#..#..#...##.#..#.....

..##v#vv#.##.##vvvv##.##.#####.###.##..#

####vvvv##..###v#vvvv####..##......####.

vv#vvvv#vv#.#vvv##v##v##v##....###...#..

vvvv##vvvvv#.###.#vv#v#vv##.###...###.#.

v###vvv##v#..##..##vvvvvvvv#v#.#..#..##.

#.#.#vvv#v#####.#..#vv#vv#vvv#..#..#.#.#

.###.##vvvvvvv#...#vv#vv####vv####.####.

..###vvvv#vv##.#...###vv####vvv#vv###..#

.##..###vv#####.#.#...#vv##vvvvvvvv#..#.

.#.##vvvv#..##...#.###.#vvvvv#vvv#vv###.

..#.###vvv#...#......###v##v#v#vvv#v#...

.###.##v#vv#.##.##.##..###vvvv#vv#vvv###

###.##.#.#vv#..#..###.#..#vvv####.##vvvv

.####.#.##vvv##.##vvv#.#..##v##..###v##v

#.#..#..#.##vvv##vv##.#...###.###.#.##vv

...#..####..##vvvvvvv#.#...###.#....#vvv

.##....##.##vv#v###v###.###...####.#.vv#

...##...##vvv#vvv#vvvvv#vv##v.##...#..##

##......#.##vvv###vv#vvvv#vvv######.#...

.....##..#..#vv##vv#.##vvvv#v..#.....##.

.##..##.#....#########.#v###########....

.#.#.#....##...##.....##vvvv####.....##.

....###.###........######vvvG....#######

The end is located at (39, 28)

root@Laquisha:/home/alexmcnurlin/Google Drive/schoolStuff/cs121/hw3# ./hw3

Enter file to read: invalid\_maze.in

S##v#vv##v

vvvvvvvvvv

#vv#v#v#vv

.#v#vvv###

#vvvv#vvvv

v#v###v##v

v##vvvv##v

v#v#v##vvv

vvvvv#vv##

##vv##v##G

Maze cannot be solved!

root@Laquisha:/home/alexmcnurlin/Google Drive/schoolStuff/cs121/hw3# exit

exit

Script done on Tue 08 Mar 2016 12:02:22 PM PST

/////////////////////////////////////////////////////////////////////////////////  
// Alex McNurlin

// File: hw3.cpp

// CS121

// Programming assignment 3

// 3/9/16

#include "queue.h"

#include <iostream>

#include <fstream>

#include <cstdlib>

using namespace std;

int Parse\_char( char\*\* maze, int x, int y, int x\_size, int y\_size, Queue &the\_queue );

int main() {

char filename[256]; // 255 is the max length of of a filename in ext4 and fat32

// gets the file to read from

cout << "Enter file to read: ";

cin >> filename;

// Open file + error checking

ifstream infile;

infile.open(filename);

if (infile.fail()) {

cout << "Invalid file " << filename << endl << "Exiting..." << endl;

exit(1);

}

// Declace Queue, read in maze size.

Queue coordinates;

int x\_size, y\_size;

infile >> x\_size >> y\_size;

// Creating dynamically allocated array

char\*\* maze = new char\*[x\_size]; // Creates the first dimension of array

for ( int i=0; i<x\_size; i++ ) { // Creates second dimension of array

maze[i] = new char[y\_size];

for ( int j=0; j<y\_size; j++ ) { // Initialize array by reading infile

infile >> maze[i][j];

if ( maze[i][j] == 'S' ) { // Checks if a point is the starting point

int point[2] = {i,j};

coordinates.Enqueue( point );// Enqueues the starting point

}

}

}

// Now to actually solve the maze

while (1) {

// Checks if the queue is empty, which only happens when the maze

// cannot be solved

if ( !coordinates.Size() ) {

// Prints out the solved maze for good measure

for ( int i=0; i<x\_size; i++ ) {

for ( int j=0; j<x\_size; j++ ) {

cout << maze[i][j];

}

cout << endl;

}

cout << "Maze cannot be solved!" << endl;

exit(0);

}

// Dequeues the next point in the list

int \*point;

point = coordinates.Dequeue();

int x = point[0];

int y = point[1];

// Checks the points in all 4 directions.

Parse\_char( maze, x-1, y, x\_size, y\_size, coordinates) ;

Parse\_char( maze, x, y-1, x\_size, y\_size, coordinates) ;

Parse\_char( maze, x+1, y, x\_size, y\_size, coordinates) ;

Parse\_char( maze, x, y+1, x\_size, y\_size, coordinates) ;

}

infile.close();

} // End main()

// Function declaration on next page

int Parse\_char( char\*\* maze, int x, int y, int x\_size, int y\_size, Queue &the\_queue ) {

// Checks if the points are within the bounds of the maze

if ( !( x<0 || x>=x\_size || y<0 || y>=y\_size ) ) {

switch ( maze[x][y] ) {

// If there's a path, mark the point as visited, and queue the point

case '.':

int temp[2];

temp[0] = x;

temp[1] = y;

maze[x][y] = 'v';

the\_queue.Enqueue( temp );

break;

// If the point is the goal, exit with success

case 'G':

// Prints out the solved maze for good measure

for ( int i=0; i<x\_size; i++ ) {

for ( int j=0; j<x\_size; j++ ) {

cout << maze[i][j];

}

cout << endl;

}

cout << "The end is located at (" << x << ", " << y << ")" << endl;

exit(0);

}

}

}

/////////////////////////////////////////////////////////////////////////////////

// Alex McNurlin

// File: queue.h

// Date: 3/9/16

#include "ll.h"

#ifndef QUEUE\_H

#define QUEUE\_H

// A simple Queue implementation, that acts as a wrapper for Ll.h

class Queue {

public:

Queue() {};

~Queue() {};

void Enqueue( int point[2] );

int\* Dequeue();

int Size();

private:

Ll list;

};

#endif

/////////////////////////////////////////////////////////////////////////////////

// Alex McNurlin

// File: queue.cpp

// Date: 3/9/16

using namespace std;

#include "queue.h"

// Add a point to the beginning of the list

void Queue::Enqueue( int point[2] ) {

list.AddNode(point);

}

// Delete from the end of the list

int\* Queue::Dequeue() {

return list.DeleteTail();

}

// Get the size

int Queue::Size() {

return list.Size();

}

/////////////////////////////////////////////////////////////////////////////////

// Alex McNurlin

// File: ll.h

// Date: 3/9/16

#ifndef LL\_H

#define LL\_H

#include <string>

// A class implementation of a linked list.

class Ll {

public:

Ll();

~Ll();

void AddNode(int n[]);

int\* DeleteTail();

int Size();

private:

struct node {

node \*next;

int value[2];

};

node \*head; // Points to the head of the list.

};

#endif

/////////////////////////////////////////////////////////////////////////////////

// Alex McNurlin

// File: ll.cpp

// Date: 3/9/16

#include "ll.h"

#include <iostream>

using namespace std;

Ll::Ll() {

head = NULL;

}

Ll::~Ll() {

node \*p = head;

node \*q = p;

head = head->next;

while (q != NULL) {

p = q;

q = p->next;

delete p;

}

}

// Adding a single node to the head of the list.

void Ll::AddNode(int n[]) {

node \*p;

p = new node();

p->value[0] = n[0];

p->value[1] = n[1];

p->next = head;

head = p;

}

int\* Ll::DeleteTail() {

if (head->next == NULL) {

static int return\_value[2];

return\_value[0] = head->value[0];

return\_value[1] = head->value[1];

delete head;

head = NULL;

return return\_value;

} else if ( head == NULL ) {

static int return\_value[2];

return\_value[0] = 1000;

return\_value[1] = 1000;

return return\_value;

} else {

node \*q, \*p;

q = head;

p = head->next;

while (p->next != NULL) {

q = p;

p = p->next;

}

static int return\_value[2];

return\_value[0] = p->value[0];

return\_value[1] = p->value[1];

delete p;

q->next = NULL;

return return\_value;

}

}

// Returns the size of the list. 0 if empty.

int Ll::Size() {

int i = 0;

node \*p;

p = head;

while (p != NULL) {

i += 1;

p = p->next;

}

return i;

}