**CS2263 Assignment 6**

Kohdy Nicholson

Modules

Link

**Description**  
The Link module contains the struct typedef for a link and also functions surrounding the creation, deleting and reporting of links.

**Code**

/\*

    LinkedList.h

    Description:

    Code file which contains functions for a Linked List

    Author:

    Kohdy Nicholson

    Date:

    2020-06-14

\*/

#include <stdio.h>

#include <stdlib.h>

#include "../Headers/Point2D.h"

#include "../Headers/Link.h"

/\* private functions \*/

// Function that mallocs a link

pLink mallocLink(){

    return (pLink)malloc(sizeof(Link));

}

/\* public functions \*/

// Creates an empty list

pLink createLink(pPoint2D point){

    pLink newLink = mallocLink();

    if(newLink != (pLink)NULL){

        newLink->point = point;

        newLink->next = (pLink)NULL;

    }

    return newLink;

}

void readLink(pLink link){

    printf("\nLink address: %p\n", link);

    printf("Link values:\n");

    printf("point\ttype: Point2D\tval:\n");

    point2DToString(link->point);

    printf("Next link address: %p\n", link->next);

}

void readLinks(pLink link){

    if(link != (pLink)NULL){

        readLinks(link->next);

        readLink(link);

    }

}

void destroyLink(pLink link){

    freePoint2D(link->point);

    free(link);

}

void destroyLinks(pLink link){

    if(link != (pLink)NULL){

        destroyLink(link->next);

        freePoint2D(link->point);

        destroyLink(link);

    }

}

Point2D

**Description**  
This module contains a typedef for a struct called Point2D. This is also one of the lowest level modules, having no dependencies on any other modules. The struct contains two variables of the double datatype, for holding x and y coordinates.

**Code**

/\*

    Point2D.c

    Description:

    Code which contains functions for a Point2D struct.

    Author:

    Kohdy Nicholson

    Date:

    2020-06-03

\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "../Headers/Point2D.h"

// Function that mallocs a single Point2D

pPoint2D mallocPoint2D(){

    return (pPoint2D)malloc(sizeof(Point2D));

}

// Function that mallocs a list of Point2D objects

pPoint2D\* mallocPoint2DList(int length){

    return (pPoint2D\*)malloc(sizeof(pPoint2D)\*length);

}

// Function that free's a single Point2D

void freePoint2D(pPoint2D point){

    free(point);

}

// Function that free's a list of Point2D structs

void freePoint2DList(pPoint2D\* points, int length){

    int i = 0;

    while(i < length){

        freePoint2D(points[i]);

        i++;

    }

    free(points);

}

// Function that sets the coordinates for a given point

void setPoint2D(pPoint2D point, double x, double y){

    point->x = x;

    point->y = y;

}

// Creates a point2d struct given x and y coordinates

pPoint2D createPoint2D(double x, double y){

    pPoint2D point = mallocPoint2D();

    if(point != (pPoint2D)NULL){

        setPoint2D(point, x, y);

    }

    return point;

}

// Deep copies a Point2D object

pPoint2D duplicatePoint2D(pPoint2D point){

    pPoint2D pt = createPoint2D(point->x, point->y);

    return pt;

}

// Gets the distance between 2 given Point2D objects using pythagorean theorem

double getDistancePoint2D(pPoint2D pt1, pPoint2D pt2){

    int xDiff = pt1->x - pt2->x;

    int yDiff = pt1->y - pt2->y;

    int squaredDist = (xDiff \* xDiff) + (yDiff \* yDiff);

    return sqrt(squaredDist);

}

// Gets x y coordinates from a file

pPoint2D fgetPoint2D(FILE\* file){

    double x, y;

    pPoint2D point;

    fscanf(file, "%lf %lf ", &x, &y);

    point = createPoint2D(x, y);

    return point;

}

// Print the point2d inner values to stdout

void point2DToString(pPoint2D point){

    printf("Point2D address: %p\n", point);

    printf("Point2D values:\n");

    printf("x\ttype: double\tval: %lf\n", point->x);

    printf("y\ttype: double\tval: %lf\n", point->y);

}

Stack

**Description**  
The uses links in order to represent a Stack data structure. Contains stack functions such as push, pop and peek.

**Code**

/\*

    Stack.c

    Description:

    Code file which contains function signatures for a stack

    Author:

    Kohdy Nicholson

    Date:

    2020-06-14

\*/

#include <stdio.h>

#include <stdlib.h>

#include "../Headers/Point2D.h"

#include "../Headers/Link.h"

#include "../Headers/Stack.h"

// Function that mallocs a stack

pStack mallocStack(){

    return (pStack)malloc(sizeof(Stack));

}

// Function that free's a stack

void freeStack(pStack stack){

    free(stack);

}

// Creates an empty stack

pStack createStack(){

    pStack newStack = mallocStack();

    if(newStack != (pStack)NULL){

        newStack->top = (pLink)NULL;

        newStack->size = 0;

    }

    return newStack;

}

// Displays the contents of the stack

void readStack(pStack stack){

    if(stack->size > 0){

        readLinks(stack->top);

    }else{

        printf("Stack is empty, nothing to read.\n");

    }

}

// Displays the top of the stack.

void peekStack(pStack stack){

    if(stack->size > 0){

        readLink(stack->top);

    }else{

        printf("Stack is empty, nothing to peek.\n");

    }

}

// Pops and returns the next item off the stack list

pPoint2D pop(pStack stack){

    pPoint2D item = (pPoint2D)NULL;

    pLink top;

    if(stack->size > 0){

        item = duplicatePoint2D(stack->top->point);

        top = stack->top;

        stack->top = stack->top->next;

        stack->size--;

        destroyLink(top);

    }

    return item;

}

// Pushing a point onto the stack list

void push(pStack stack, pLink link){

    link->next = stack->top;

    stack->top = link;

    stack->size++;

}

// function that destroys a stack

void destroyStack(pStack stack){

    destroyLinks(stack->top);

    freeStack(stack);

}

Queue

**Description**  
The Queue module is similar to the stack module in the sense it uses links to represent a queue. It contains queue functions such as enqueue, and dequeue.

**Code**

/\*

    Queue.c

    Description:

    Code file which contains function signatures for a queue

    Author:

    Kohdy Nicholson

    Date:

    2020-06-14

\*/

#include <stdio.h>

#include <stdlib.h>

#include "../Headers/Point2D.h"

#include "../Headers/Link.h"

#include "../Headers/Queue.h"

// Function that mallocs a queue

pQueue mallocQueue(){

    return (pQueue)malloc(sizeof(Queue));

}

// Function that free's a queue

void freeQueue(pQueue queue){

    free(queue);

}

// Creates an empty queue

pQueue createQueue(){

    pQueue newQueue = mallocQueue();

    if(newQueue != (pQueue)NULL){

        newQueue->front = (pLink)NULL;

        newQueue->back = (pLink)NULL;

        newQueue->size = 0;

    }

    return newQueue;

}

// Displays the contents of the queue

void readQueue(pQueue queue){

    if(queue->size > 0){

        readLinks(queue->front);

    }else{

        printf("Queue is empty, nothing to read.\n");

    }

}

// Displays the top of the queue.

void peekQueue(pQueue queue){

    if(queue->size > 0){

        readLink(queue->front);

    }else{

        printf("Queue is empty, nothing to peek.\n");

    }

}

// Pops and returns the next item off the queue list

pPoint2D dequeue(pQueue queue){

    pPoint2D item = (pPoint2D)NULL;

    pLink front;

    if(queue->size > 0){

        item = duplicatePoint2D(queue->front->point);

        front = queue->front;

        queue->front = queue->front->next;

        queue->size--;

        destroyLink(front);

    }

    return item;

}

// Pushing a point onto the queue list

void enqueue(pQueue queue, pLink link){

    if(queue->size == 0){

        queue->front = link;

        queue->back = link;

    }else{

        queue->back->next = link;

        queue->back = link;

    }

    queue->size++;

}

// function that destroys a queue

void destroyQueue(pQueue queue){

    destroyLinks(queue->front);

    freeQueue(queue);

}

playStack

**Description**  
Main program for testing the stack functionality.

**Code**

// playStack.c

#include <stdio.h>

#include <stdlib.h>

#include "Headers/Stack.h"

#include "Headers/Link.h"

#include "Headers/Point2D.h"

#define PUSH 1

#define POP 0

#define LIST 2

#define PEEK 3

int main(int argc, char\* argv[])

{

  int iChoice;

  int iNRead;

  double x, y;

  pPoint2D point;

  pLink link;

  pStack stack = createStack();

  /\* Processing loop \*/

  printf("Choice (1=add, 0=remove, 2=list, 3=peek): ");

  iNRead = scanf("%d", &iChoice);

  while(iNRead == 1)

  {

    switch(iChoice)

    {

      case PUSH:

        printf("Point value to add: "); // Obviously you need to read the x and y values

        // Read the element, add it to the stack

        scanf("%lf %lf", &x, &y);

        point = createPoint2D(x, y);

        if(point == (pPoint2D)NULL){

          printf("Failed to allocate memory. Exiting..");

          return EXIT\_FAILURE;

        }

        link = createLink(point);

        if(link == (pLink)NULL){

          printf("Failed to allocate memory. Exiting..");

          return EXIT\_FAILURE;

        }

        push(stack, link);

      break;

      case POP:

        // Pop the Point2D and print it out.

        point = pop(stack);

        if(point != (pPoint2D)NULL){

          point2DToString(point);

          freePoint2D(point);

        }else{

          printf("Stack is empty, nothing to pop.\n");

        }

      break;

      case LIST:

        // Print out the stack elements

        readStack(stack);

      break;

      case PEEK:

        // Print out the next value to be popped.

        peekStack(stack);

      break;

      default:

        return 0;

    }

    printf("\nChoice (1=add, 0=remove, 2=list, 3=peek): ");

    iNRead = scanf("%d", &iChoice);

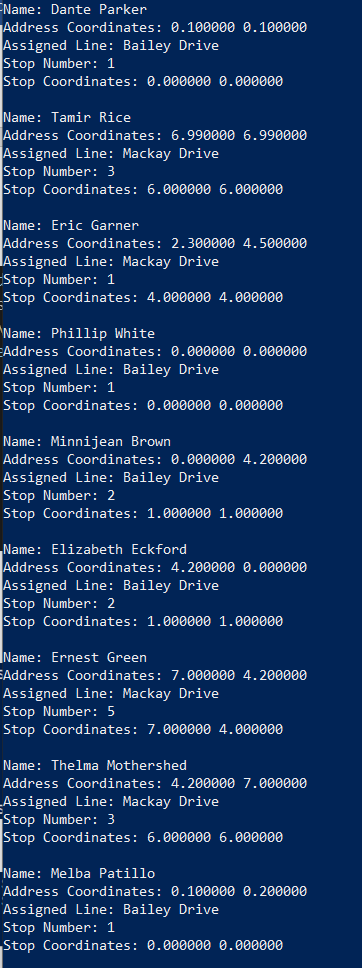
  }

  destroyStack(stack);

  return EXIT\_SUCCESS;

}

**Output** (Partial)

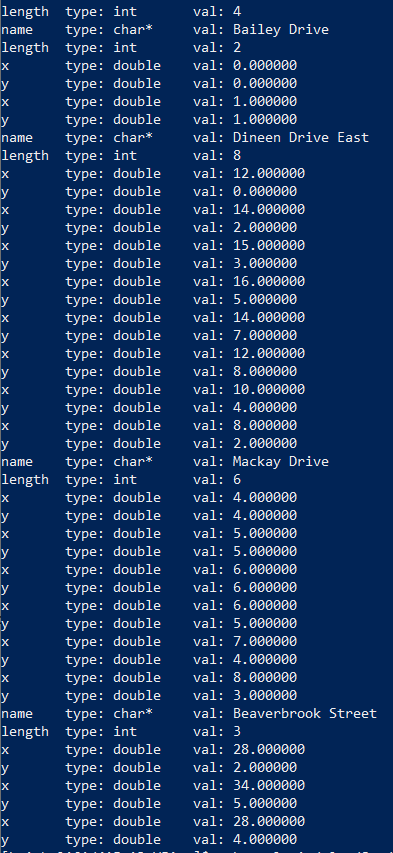


Test Drivers

loadRoutesTest

**Description**  
This simply loads in bus lines from a file given to the command line. I then display all of the data contained in the list of bus lines. I will include the full output in the logs folder.

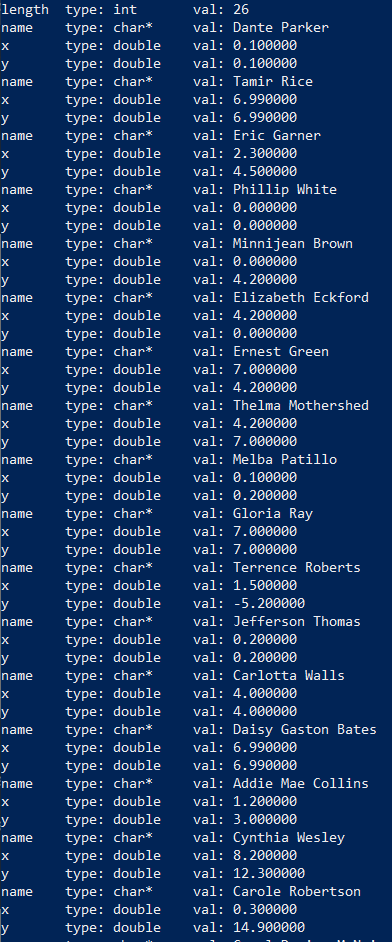
**Output**



loadStudentsTest

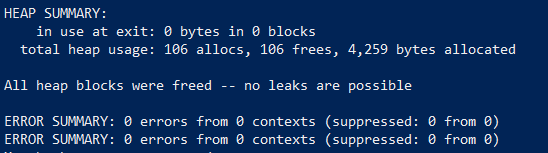
**Description**  
Loads student data from stdin. Creates a list of students are displays their information to the console. I will include the full output in the logs folder.

**Output** (Partial)

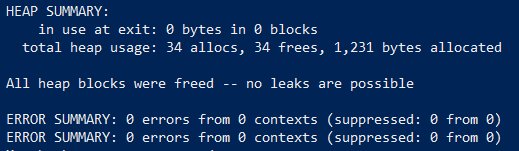


Valgrind

**Output loadStudentsTest:**



**Output loadRoutesTest:**



**Output RoutePlanner:**

