Nicolas Ayala
nmayala@ucsc.edu

Pre-lab 1:

1)

```
Char *input = getinput()
bool adjacency_matrix[26][26];

int line = 0;

while(line<LinesIn(input)){
        int vertex_one = input[line*3] - 65;
        Int vertex_two = input[line*3+1] - 65;
        adjacency_matrix[vertex_one][vertex_two] = true;
}
```

2)

AB, BC, CF, FZ, BD, DE

3)

The worst case would be

Pre-Lab 2:

```
Stack *stack_create(uint32_t size){
        struct Stack *just_made;
        just_made->items = (int)malloc(size * 4);
        just_made->capacity = size;
        just_made->top = 0;
        return just_made;
}

void stack_delete(Stack *s){
        free(s->items);
        free(s);
}

bool stack_empty(Stack *s){
        return s->top == 0;
}

uint32_t stack_size(Stack *s){
        return s->top;
}
```

```
bool stack_push ( Stack *s, uint32_t item ){
        if(s->top  <  s->capacity){
                s->items[top] = item;
                top++;
                return true;
        }else{                                          // create a new stack, twice the size
                struct Stack *double_size;
                double_size = stack_create(s->capacity * 2);
                for(int i = 0;i<s->capacity;i++){
                        double_size[i] = s->items[i];
                }
                double_size[s->capacity]  = item;
                double_size->top = s->capacity + 1;
                s = double_size;
                return true;
        }
        return false;
}

bool stack_pop ( Stack *s, uint32_t * item ){
        if(s->top < 0){
                top--;
                *item = s->items[top];
                return true;
        }
        return false;
}

void stack_print(Stack *s){
        for(int i = 0; i < s->top; i++){
                print(s->top[i] + “\n”);
        }
}
```

Design of program

Program will have a stack struct, whose implementation can be seen in Pre-Lab 2)

Program will have a function called fill_matrix which takes as input a string (pointer to a char), a 2d array of booleans (boolean pointer), and a boolean called directed.

The implementation can be found in Pre Lab 1), with the addition of

```
if(not directed){
        adjacency_matrix[vertex_two][vertex_one] = true;
}
```

Added under the line

```
adjacency_matrix[vertex_one][vertex_two] = true;
```

Program will have a main function, that uses getopt() to take the following options:

-i <input>
-u
-d
-m
i specifies the file <input> containing the graph,
u means that the graph is undirected,
d means that the graph is directed,
m means that the adjacency matrix will be printed.

The flags -u and -d can’t both be present, the rest are independent.

An adjacency matrix will be initialized, and then passed to fill_matrix

Now that we have a filled matrix, make a stack of capacity 26,
and pop A to the stack (A means 0, B means 1...,
Make a bool array called dead_end[26]
Now we will traverse the labyrinth, by checking all the nodes connected to the node we are on(top of stack) and taking the first path. When we take a path we unmark it from the matrix, when a node has no outgoing paths, it is a dead end

```
while(stack->items[stack->top] != Z){
    if(!dead_end[ stack->items[stack->top] ]){
        dead_end[ stack->items[stack->top] ] = true;
        for(int i = 0; i < 26; i++){
            if(adjacency_matrix[ stack->items[stack->top] ][i]){
                stack->push(i);
                adjacency_matrix[ stack->items[stack->top] ][i] = 0;
                dead_end[ stack->items[stack->top] ] = false;
                break;
            }
        }
    }else{
        stack->pop();
    }
}
```

And finally,we print the stack, and if the -m flag was true, we print the adjacency matrix

Change in design

Maze is traversed in this manner instead:
A function called depth_first_search which as input a matrix, a stack, and a node
A bool array called Visited is normalized to false

The following preparation happens before the first call to the function:

```
    Matrix is filled
    ‘A’ is pushed to stack
    Visited[0] is true
```

```
depth_first_search(){
    if(node is exit){
        print(“found path”)
        Print stack
    }
    for( each node outgoing from current node that has yet to be visited)
        Push node
        Mark node as visited
        depth_first_search(matrix, stack, this node)
        Mark node as unvisited
        Pop node
    }
}
```