CMPUT 175

Lab 4 Exercises

For this lab, finish Exercise 1 first, then Exercise 2, then Exercise 3, then Exercise 4. Exercise 5 is not for marks can be done for extra practice.

Exercise 1

Write a program that uses the Stack implementation given to you in stack.py to reverse a String. The program should ask the user for a String, and should then display the reversed String. The Stack class must be used to reverse the String, one character at a time.

Exercise 1 User Input Example

Enter a String to reverse: Hello, how are you?
The reversed String is: ?uoy era woh ,olleH

Exercise 2

Complete the queue.py class provided. A queue is an abstract data type which keeps elements in order. Items can only be added to the back of the queue, and can only be removed from the front of the queue. A queue is a FIFO data structure (first in, first out). Note that any variables you create in the Queue class must be private variables.

Included in queue.py is a main() function for testing your queue implementation.

Exercise 2 Methods #1

The following methods need to be implemented in queue.py:

- __init__()
  - Creates a new empty Queue.
- queue(item)
  - Adds a new item to the back of the queue, and returns nothing.
- dequeue()
  - Removes and returns the front-most item in the queue. Returns nothing if the queue is empty.
- peek()
  - Returns the front-most item in the queue, and DOES NOT change the queue.

Exercise 2 Methods #2

- is_empty()
  - Returns True if the queue is empty, and False otherwise.
- size()
  - Returns the number of items in the queue.
- clear()
  - Removes all items from the queue, and sets the size to 0.
- __str__()
  - Returns a string representation of the queue. This method can simply call the repr() function on whatever data structure you use to store your queue.

Exercise 3

Write a program that represents a grocery store lineup, using the Queue class you implemented in Exercise 2. The program will ask the user if they want to add a person to the lineup, serve the next person in the line, or quit the program.

The program should never allow more than 3 people in the lineup at once. If the user tries to add another person to the lineup when there are already 3 people, the program should print an error message and continue.

If the user tries to serve the next person when there is no one in the lineup, the program should inform the user that the lineup is empty.

When a person is served, the program should print the name of that person.

Exercise 3 User Input Example

Add, Serve, or Exit: Serve
The lineup is already empty.
Add, Serve, or Exit: Add
Enter the name of the person to add: Bob
Add, Serve, or Exit: Add
Enter the name of the person to add: Anna
Add, Serve, or Exit: Add
Enter the name of the person to add: Sarah
Add, Serve, or Exit: Add
You cannot add more people, the lineup is full!
Add, Serve, or Exit: Serve
Bob has been served.
Add, Serve, or Exit: Serve
Anna has been served.
Add, Serve, or Exit: Exit

Exercise 4

Write a program that uses the Stack implementation given to you in stack.py to check for matching parentheses (brackets) in a Python file. There are three types of parentheses your program must check for: (), {}, and []. Your program should also count how many pairs of each type of parentheses occur.

Your program should start by asking the user for the name of a file. The program must then read the file, checking if there is a matching closing parenthesis for each opening parenthesis. Finally, it will print out whether or not there was a matching closing and opening parenthesis for each type, and how many pairs of parentheses occurred.

You do not need to do any exception handling for this exercise.

Exercise 4 Example Files

test1.py:

```
print("}")
number_list = [1, 2, 3, 4, 5]
print(number_list[2])
dict = {}
```

test2.py:

```
word_list = ["Hello", "World"]
person_dict = {"name": "Bob", "age": 34}
print(word_list[0])
print(word_list[1])
person_dict["job"] = "Firefighter"
print(person_dict)
```

Exercise 4 User Input Example #1

Enter the file name: test1.py
() pairs: 2
{} pairs: 1
[] pairs: 2
All () matched.
Not all {} matched.
All [] matched.

Exercise 4 User Input Example #2

Enter the file name: test2.py
() pairs: 3
{} pairs: 1
[] pairs: 4
All () matched.
All {} matched.
All [] matched.

Exercise 5 – Not For Marks

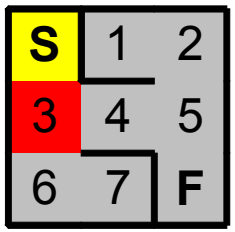
Exercise 5 is an extra exercise and is not for marks.

Write a program that uses the Stack implementation given to you in stack.py to check if there is a solution to a maze. A simple maze implementation is given to you in maze.py. To solve a maze using the Stack, use the following algorithm:

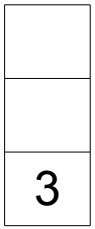
1. Add the start square to the stack.
2. Repeat the following as long as the stack is not empty:
   a) Pop a square off the stack (the current square)
   b) If the current square is the finish square, the solution has been found
   c) Otherwise, get the list of squares which can be moved to from the current square, and add them to the stack

Exercise 5 – Algorithm Example

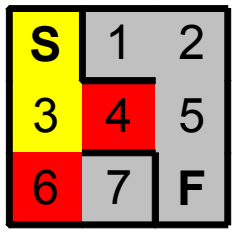
Step 1:



Stack:

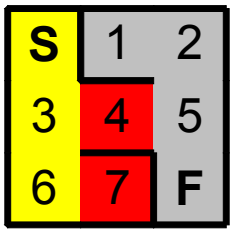


Step 2:



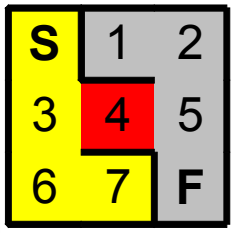
Stack:

Step 3:

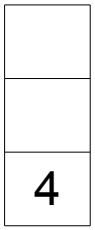


Stack:

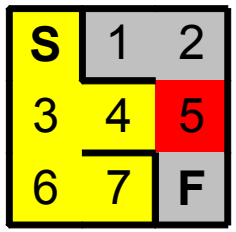
Step 4:



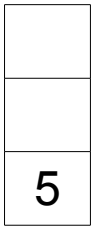
Stack:



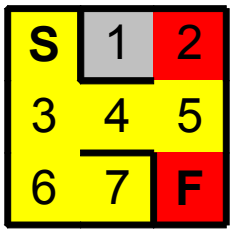
Step 5:



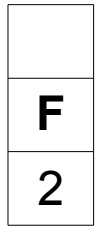
Stack:



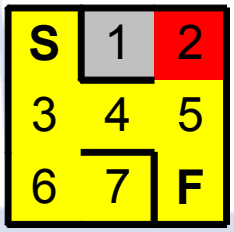
Step 6:



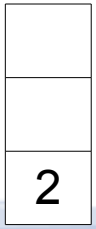
Stack:



Step 7:



Stack:



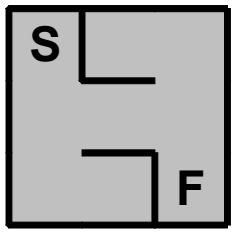
Solution found!

Exercise 5 – Maze Description

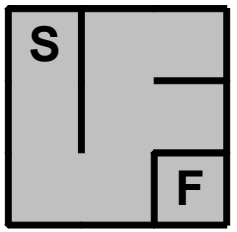
The maze.py file has two classes: Maze and MazeSquare. You can create a new maze by calling the constructor: *Maze(file_name)*, where *file_name* is the name of a file containing a maze. You can get the start square of the maze by calling *maze.get_start_square()*. This method returns a MazeSquare. To check if a square is the finish square, call *maze.is_finish_square(square*), where *square* is a MazeSquare. The MazeSquare class represents a single square in the Maze. Calling *maze_square.get_legal_moves()* returns a list of MazeSquares that can be moved to.

Exercise 5 – Maze Files

Two mazes files are included for this exercise: solvable_maze.txt and unsolvable_maze.txt. A graphic representation of each maze is given below:



solvable_maze.txt



unsolvable_maze.txt