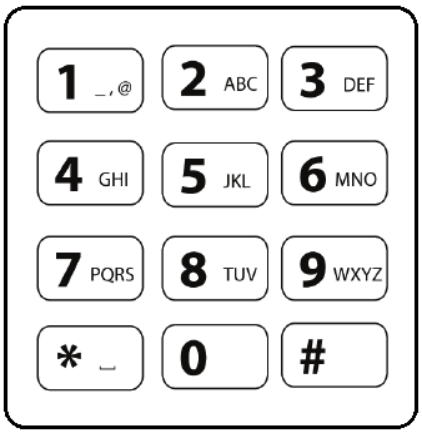
Data Structures

Fall 2022 FAST-NU, Lahore

Assignment 2 – Recursion+STACK

Question 1. Given a sequence of numbers from a mobile keypad, write a recursive function to print all possible combinations of words formed from the letters written on the corresponding keys. Marks 50



Input format:

- User can input any number in the range 2-9 inclusive
- User can enter minimum of 2 numbers and maximum of 8 numbers

A sample run of this program is given below:

```
Input: 3  4  8

//Output

DGT   DGU   DGV   DHT   DHU   DHV   DIT   DIU   DIV
EGT   EGU   EGV   EHT   EHU   EHV   EIT   EIU   EIV
FGT   FGU   FGV   FHT   FHU   FHV   FIT   FIU   FIV
```

Question 2. Provide a recursive function that takes a pointer to the middle of an infinite doubly linked list along with an integer key and searches the list for the given key. The list grows infinitely in both directions. Your algorithm should be able to find the key if it is present in the list, otherwise it should continue the search infinitely. Marks 20

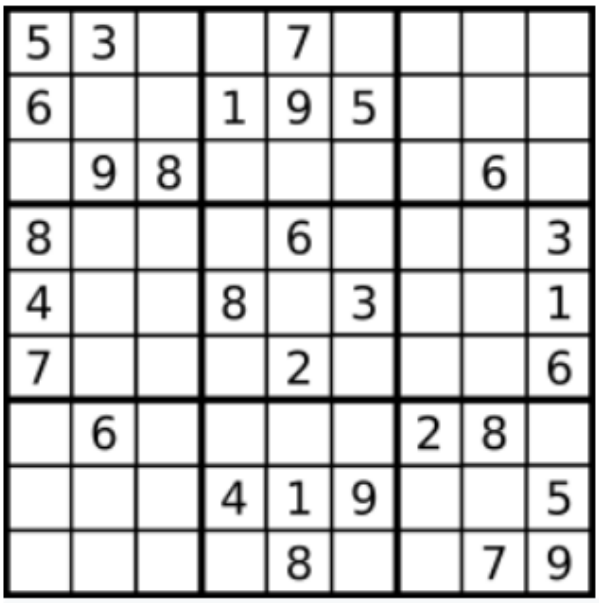
Question 3. Solve a puzzle: marks 50

USE BACTRACKING.
Rules to follow:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.

Example 1:




```
Input: board = [["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],
[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],
["4",".",".","8",".","3",".",".","1"],
["7",".",".",".","2",".",".",".","6"],
[".","6",".",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],
[".",".",".",".","8",".",".","7","9"]]
```

Output: [["5","3","4","6","7","8","9","1","2"],
["6","7","2","1","9","5","3","4","8"],
["1","9","8","3","4","2","5","6","7"],
["8","5","9","7","6","1","4","2","3"],
["4","2","6","8","5","3","7","9","1"],
["7","1","3","9","2","4","8","5","6"],
["9","6","1","5","3","7","2","8","4"],
["2","8","7","4","1","9","6","3","5"],
["3","4","5","2","8","6","1","7","9"]]

Explanation: The input board is shown above and the only valid solution is shown below:

Explanation: The input board is shown above and the only valid solutic is shown below:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Constraints:

- `board.length == 9`
- `board[i].length == 9`
- `board[i][j]` is a digit or `'.'`.
- It is **guaranteed** that the input board has only one solution.

Question 4. **"A stack of queues"** **marks100**

In this problem, you will use the built-in STL string, vector, stack and queue. You may add these to your program by using the lines:

```
#include <string>

#include <vector>

#include <stack>
#include <queue>

using namespace std;
```

(a) Create an object of the STL stack of type character. Use this stack to detect whether a string of

0's and 1's entered by the user is such a string that it is a repetition of a certain number of zeroes following by an equal number of ones. For example, the following strings are all of this type:

01

001101

00110100011101

The following are
invalid: 001 0 10
001100011110

(b) The power of templates is that you can create an object with any type of data you wish. For example, you can make a vector of integers, a stack of floats, a queue of characters etc. You can even make a vector of queues of integers, using just one line of STL code:

```
vector< queue<int> > sqn; //a stack of queues of numbers
```

Where might you need to use such a data structure? Imagine, you are making network 'receiver' node that manages multiple recourses of a certain type; these could be printers, transmission links, files, and so on. Then the node should be able to receive requests for each resource; make these requests wait in queues

(a separate queue for requests for a specific resource) and serve these request in FIFO order.

Create a class Receiver with the following definition:

```
class Receiver{
  vector<queue<int>>
  sqn;

public:

 void addRequestforResource(int rid, int reqno);


//adds a new quest with number reqno to the
resource sqn[rid] //rid is between 0 and 4, as we
have 5 resources

void serviceRequestatResource(int rid);

//services the request (dequeues) at front of sqn[rid]

void printQueues()

//prints all queues line by line, numbers separated by spaces
}
```

Now write a loop that **simulates** this process of request reception and servicing and shows how the queues grow longer and shorter.

First ask the user to provide 5 ‘servicing rates’ for the 5 resources. For example, if the user provides 250 for resource 1, this means that resource 1 services its request after every 250 milliseconds, i.e. after every 250 milliseconds it removes a request from the front of its queue sqn[1]. Based on these servicing rates you should remove a request from the respective queue, shortening it.

Meanwhile, you should add a new request with a random reqid to a random resource id (between 0 and 5) after every 500 milliseconds (this addition rate is fixed).

Use the printQueues method to show the status of all queues after every 250 milliseconds. Do this by clearing the screen each time and printing the queues at the top left of the screen.

Important Notes:

- You can make a loop make one iteration per 1 millisecond by adding the line Sleep(1) at the end of the loop. This will add a delay of 1 millisecond after

each iteration. This means that after 250 iterations of the loop, exactly 250 ms would have passed.

- For the first 100 iterations do not wait for 30 seconds and only add requests without servicing any (no dequeues).

---*** END ***