Data structures and Algorithms LAB – BSDSF21
(Morning and Afternoon)

Lab 09 – 14-03-2023

Task 1: *Algorithm and demonstrations of converting Infix to postfix expression is explained in provided document* ***"Infix to Postfix transformation.jpg"*** *and* ***"Infix to Postfix Examples.pdf"***.

You have to achieve following functional capabilities:

- function **convertToPostfix** that converts the infix expression to postfix notation, where parameter and return types are string arrays contains the expressions. You are needed to write a function to **display** and **input** these expressions.
  string * convertToPostfix(const string * infix)

To implement this function, you might need to implement following additional functions for help.

- function **isOperator** that determines whether char is an operator
  bool isOperator(string operand)

- function **precedence** that determines returns the precedence of operator (Hint: set precedence of +,- to 1 and *,/ to 2, etc).
  int precedence(string operator)

Task 2: *Understand and correct the code provided in file* ***maze.py*** *which solves a maze problem*.

Task 3: *In this programming exercise, you'll design a CPU scheduler simulator. This simulator would use a* ***list*** *(as* ***queue)*** *data structure. The scheduler should take in* ***processes*** *from a file named* ***readylist.txt*** *and add them to a queue. The format of the file would be (an example is given, have a look at it):*

```
<process_name, execution_time>
```

This file could have processes in any order. After reading in, your program should empty the file **readylist.txt**

- Once all the processes are read from the **readylist.txt** file into a **list**, your program would execute each of the processes in **highest-execution-time-first** (the process having a highest execution time will be run first) order.

- But before executing any process, the program should print the **sorted** contents of the list, and as an example the contents of the ready queue could be:

  sms.exe, 23
  mms.exe, 13
  explorer.exe, 9
  devmgmt.exe, 4

Lab 09 – 14-03-2023

.....
.....

- Now it should start executing the processes in the list. For each process, execution just means that process stays at the head of the list until time equal to its **execution time** has passed. The process is deleted from the list after that. At this moment your program should print that such and such process has finished execution.

  For Example

  sms.exe, <Execution Time>, <Waiting Time>, <Turnaround Time>

  e.g. if a process arrives at time 0, has an execution time of 4 and terminates at time 12 the output should be:
  sms.exe, 4, 8, 12

- You must remember that this is a simulator. This means that you'll have to define your own timer, one which increments in unit steps (hint: use loop).

- After the passage of every **15** time units, your program should pause until the user presses **<enter>** and then check the contents of the **readylist.txt** file to check if any new processes are added to it (you have to add them appropriately and save the file after verification of validation of data). It should continue the same way after reading the new processes in and print the contents of the list and so on.

- At the end your program should output the average waiting time and the average turnaround time for all the jobs.

  Waiting Time for one Job = Execution Start Time – Arrival Time
  Turnaround Time for one Job = Execution Ending Time – Arrival Time

Note: Jobs that are in the file at the start of the program are assumed to have arrival time 0. Jobs that appear in the file at time 10 have arrival time 10 and so on.

Your program should terminate when there are no more processes in the queue and the **readylist.txt** file is empty.

The credit of task 3 goes to Sir Umair Babar and Mam Umme Ammara (taken from their LABs once, with permission).

********* The end *********