Name: ____________________

- Instructions:
  - Show your work to receive partial credit.
  - Keep your eyes on your own paper and do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than the professor/proctor for ANY reason in ANY language in ANY manner.
  - This exam is closed notes, closed books, no calculator.
  - Turn all mobile devices off and put them away now. You cannot have them on your desk.
  - Write neatly and clearly indicate your answers. What I cannot read, I will assume to be incorrect.
  - Stop writing when told to do so at the end of the exam. I will take 5 points off your exam if I have to tell you multiple times.
  - Academic misconduct will not be tolerated. Suspected academic misconduct will be immediately referred to the Rollins Honor Council. Penalties for misconduct will be a zero on this exam, an F grade in the course, and/or other disciplinary action that may be applied by the Rollins Honor Council.
- Time: This exam has 8 questions on 9 pages including the title page. Please check to make sure all pages are included. You will have 75 minutes to complete this exam.

On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work. Also, I have read and understand the above policies for this exam.

Signature: ____________________

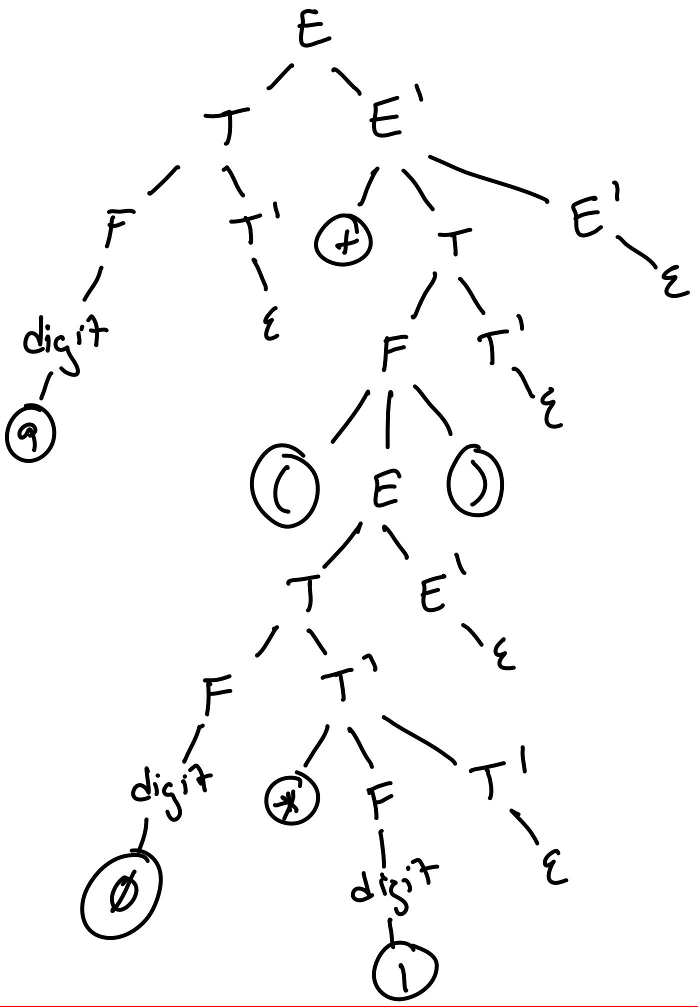
Question:	1	2	3	4	5	6	7	8	Total
Points:	10	10	10	6	8	8	10	8	70
Score:									

1. (10 points) Consider the following grammar:

```
E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | digit
digit -> 0 | 1 | 2 | ... | 9
```

Construct a parse tree for the sentence `9 + (0 * 1)`.

Solution:

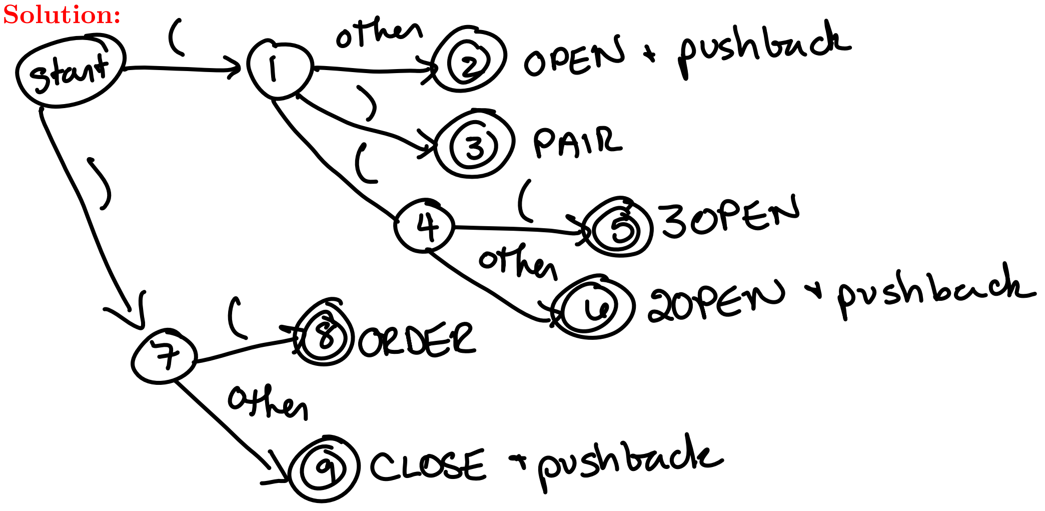


2. (10 points) Construct a transition diagram for a lexical analyzer that recognizes the following patterns (left hand column) and associated tokens (right hand column):

Pattern	Token Name
`(`	`OPEN`
`)`	`CLOSE`
`)(`	`ORDER`
`()`	`PAIR`
`((`	`2OPEN`
`(((`	`3OPEN`

Label each arc with the symbol that triggers its transition and each accepting state with the token name (from the right hand column) it recognizes. Note the places where a character must be pushed back on the input stream.

Solution:



Common mistakes:
* not indicating which states/transitions require a pushback. * having a final state with a transition OUT of it into another final state.

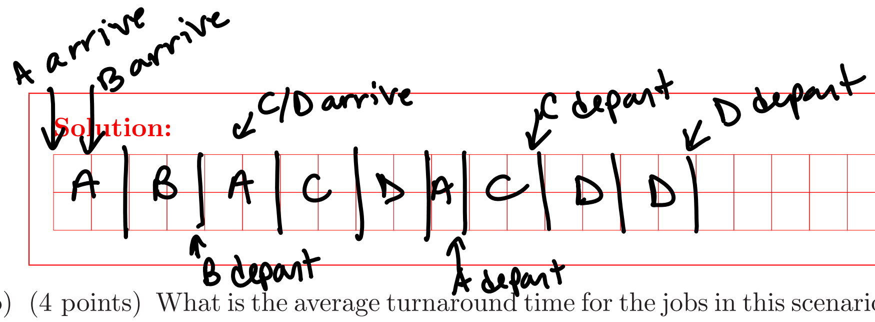
3. Suppose that a system uses a round-robin scheduling discipline. The system is initially empty, then the following sequence of jobs arrives

- Job A arrives at time 0 and has a service requirement of 5
- Job B arrives at time 1 and has a service requirement of 2
- Job C arrives at time 5 and has a service requirement of 4
- Job D arrives at time 5 and has a service requirement of 6

The system keeps the jobs in alphabetical order. When it's time to make a scheduling decision, the scheduler always chooses the next available job in the alphabetical listing. The quanta size is 2.

(a) (6 points) Draw a scheduling diagram on the grid showing the order in which the jobs execute. Indicate the arrival time and departure time of each job. Let 1 grid square be 1 unit of time.

Solution:



(b) (4 points) What is the average turnaround time for the jobs in this scenario? Show your calculations for partial credit.

Solution:

$Process$	$T_arrival$	T_comp	T_turn
A	0	11	11
B	1	4	3
C	5	13	8
D	5	17	12

Average turnaround time =

$\frac{11+3+8+12}{4}$ =

$\frac{34}{4}$ =

8.5

4. When a system call occurs, the hardware will redirect execution to special trap handler code in the OS. The OS must set up a trap table to inform the hardware of the location of all interrupt-handling routines.

   (a) (3 points) Explain when and how the trap table is initialized and who is responsible for the initialization.

   Solution: OSTEP Ch. 6, bottom of page 4 and top of page 5.

   Kernel is responsible for setting up the trap table at boot time. This table stores the locations of code for privileged operations. So if a user process requests a specific (privileged) operation, the code to run is NOT supplied by the process (this would be a huge security issue). Instead, the code can be looked up in the trap table, and the starting address of the code to be executed will be stored and can then be executed.

   Common mistake: confusing trap table with trap handler. Several people descibed the trap handler process (part b) instead of the trap table.

   (b) (3 points) When a user process executes a system call, it uses a a special trap machine language instruction (`syscall` on the x86-64 architecture). What does this instruction do?

   Solution: OSTEP, Ch. 6, pgs 3-5 (chart on pg. 5 particularly releveant

   The user process requests a `syscall` with a specific number to identify which syscall it needs. It does this via a **`trap`** instruction. The hardware elevates the mode from user mode to kernel mode, and the OS will execute the correct trap handler (via the lookup procedure described in part a). Eventually, when the handler has finished, the OS issues an instruction, **`return-from-trap`**. This causes the hardware to return the mode to user mode, and the user process can continue via the LDE protocol.

   For full credit, you had to explain the entire process. A common error was talking about how the trap handler started, but not talking about how it completed/finished and returned to user mode. Another common error was not being clear about how the user process, OS, and HW interacted to handle the different parts of the process.

5. Consider the following context-free grammar:

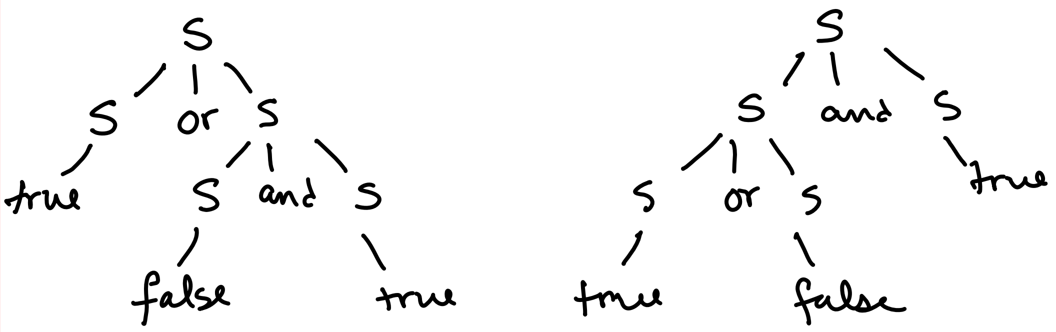
```
S -> S and S | S or S | (S) | true | false
```

(a) (2 points) List the **terminal(s)**: and, or, (,), true, false.

(b) (2 points) List the **non-terminal(s)**: S.

(c) (4 points) A grammar can be ambiguous or unambiguous. The above grammar is ambiguous. Provide an example (and associated parse trees) which prove this fact.

Solution:
Ambiguous grammars mean that 1 statement can be produced multiple ways (different applications of the production rules). Consider the statement `true or false and true` and the two following parse trees:



Common errors: listing production rules for non-terminals.

6. (8 points) Summarize how variables and assignment statements are implemented in the interpreter program.

Solution: This problem comes from project 1.5.

Variables and their associated values are stored in a symbol table. This is traditionally a hash table/hash map, but I accepted any generic term about “memory” or “table” or anything along those lines.

When an assignment statement is encountered, it must be in the form of `NAME EQUALS EXPR` (where EXPR can be a combination of other operators, variable names, and value tokens).

The interpreter looks up the `NAME` token in the hashtable. It can then update the value with the value of `EXPR` once it is evaluated by updating the value associated with `NAME`.

If a variable is used as part of an expr (for example `y` in the statement `x = y + 5`), the interpreter can just look up the value in the symbol table and return the value to be further used in the evaluation of the left hand side.

Common mistakes:
* explaining the overall process of going from source code to low level code, but not addressing the actual question of variables/assignments.
* talking non-specifically about something like “parsing a steam of tokens” without addressing the variables portion specifically.

7. We described two metrics for assessing the quality of scheduling disciplines: turnaround time and response time.

   (a) (4 points) Define the two terms.

   Solution: Turnaround time is the time between the arrival of a job and its completition. Mathematically: $T_{turnaround} = T_{completion} - T_{arrival}$

   Response time is the time between the arrival of a job and its first service (when it starts running). Mathematically: $T_{response} = T_{service} - T_{arrival}$

   See OSTEP, Ch. 7, pgs 2 (turnaround) and 6 (response)

   (b) (6 points) Explain how the multi-level feedback queue implements scheduling policies that attempt to ensure good performance on both metrics.

   Solution: See OSTEP, Ch. 8

   Boiled down:
   1. MLFQ starts all jobs at the highest priority, and all jobs in highest queue run in RR. Thus each job will be serviced relatively quickly (based on the quanta size) and this will keep the response time metric low. 2. Turnaround time is more variable, but running in RR will keep high(er) priority threads from starving. Periodically the low priority threads will be boosted to ensure them some running time as well.

   Common mistakes:
   * saying that a MLFQ is sized-based. It's not. The way it's implemented has the effect that shorter jobs finish earlier and thus have an excellent turnaround time metric, but the algorithm isn't based on size in the way other scheduling algorithms are.
   * explaining lots of detail about the policies of a MLFQ (and how it's implemented, etc, etc), but never actually stating how those policies effect either response or turnaround time. You got partial credit for listing the policies, but full credit required explicitly discussing how they related to response and turnaround time.

8. Scheduling algorithms can be classified along two axes: preemptive versus non-preemptive and size-based versus non-size-based.

   (a) (4 points) Explain the the terms *preemptive* and *size-based* in the context of scheduling algorithms.

   Solution:
   See OSTEP, Ch. 7, pgs 5 (preemptive) - scheduler can interrupt a job before it's done; not a requirement that job runs to completion

   Size based: scheduler takes into account the size (more formally, the service requirement) of a job (or portion of a job) when determining what to do next.

   Common mistake: stating that preemptive algos had something to do with size or priority. They may not. But a preemptive algo is one which can stop one job from running and start another one (i.e., force a context switch).

   (b) (4 points) Fill in the table below, placing the five scheduling algorithms we discussed (FIFO, SJF, STCF, RR, and MLFQ) into the appropriate quadrant.

   Solution:

	size-based	non-size-based
preemptive	STCF (Ch 7, p7)	RR (Ch 7, p7-8) MLFQ (Ch 8)
non-preemptive	SJF (Ch 7, p4)	FIFO (Ch 7, p3)

   Common mistake: Putting MLFQ into the preemptive, size-based quadrant. MLFQ is not explicitly sized based. It is PRIORITY based. For example: a job starts in the highest priority queue regardless of its size. The EFFECTS of the MLFQ policies mean that short jobs are handled quickly, but the algo. itself isn't explicitly sized based.