Homework 4 Answers

November 11, 2023

Problem 1 (1 point each) is Problem 1 in Discussion Questions in 4.26. of *Problem Solving with Algorithms and Data Structures using Python*. Put the answer in a text file named `hw3_last_first/p1.txt`. Convert the following values to binary using "divide by 2." Show the stack of remainders.

a) 17

ANSWER

$$
\begin{aligned}
17/2 &= 8r1 \text{ push 1, stack: [1]} \\
8/2 &= 4r0 \text{ push 0, stack: [1, 0]} \\
4/2 &= 4r0 \text{ push 0, stack: [1, 0, 0]} \\
2/2 &= 2r0 \text{ push 0, stack: [1, 0, 0, 0]} \\
1/2 &= 0r1 \text{ push 1, stack: [1, 0, 0, 0, 1]}
\end{aligned}
$$

Now pop the stack until empty to obtain the binary number: 10001.
It is not required that the answer be presented exact as above.
Any notion of the stack [1, 0, 0, 0, 1] is adequate.

b) 45

ANSWER

$$
\begin{aligned}
45/2 &= 22r1 \text{ push 1, stack: [1]} \\
22/2 &= 11r0 \text{ push 0, stack: [1, 0]} \\
11/2 &= 5r1 \text{ push 1, stack: [1, 0, 1]} \\
5/2 &= 2r1 \text{ push 1, stack: [1, 0, 1, 1]} \\
2/2 &= 1r0 \text{ push 0, stack: [1, 0, 1, 1, 0]} \\
1/2 &= 1r1 \text{ push 1, stack: [1, 0, 1, 1, 0, 1]}
\end{aligned}
$$

Now pop the stack until empty to obtain the binary number: 101101

c) 96

ANSWER

$$
\begin{aligned}
96/2 &= 48r0 \text{ push 0, stack: } [0] \\
48/2 &= 24r0 \text{ push 0, stack: } [0, 0] \\
24/2 &= 12r0 \text{ push 0, stack: } [0, 0, 0] \\
12/2 &= 6r0 \text{ push 0, stack: } [0, 0, 0, 0] \\
6/2 &= 3r0 \text{ push 0, stack: } [0, 0, 0, 0, 0] \\
3/2 &= 1r1 \text{ push 1, stack: } [0, 0, 0, 0, 0, 1] \\
1/2 &= 0r1 \text{ push 1, stack: } [0, 0, 0, 0, 0, 1, 1]
\end{aligned}
$$

Now pop the stack util empty to obtain the binary number 1100000.

Problem 2 (2 points)

a) Create a `LinkedList` class. It must pass the `hw3/p2/test_linked_list.py`. The class MUST not use any Python built-in or standard library collection class, i.e., do not wrap a list or deque.

ANSWER
I included an implementation of `linked_list.py` in the repository. The answer provided by the student must pass `test_linked_list.py`.

b) Copy the Stack implementation found in the repository

`https://git.cs.olemiss.edu/harrison/csci-356`

in `lecture13and14/stack.py` into your homework directory `hw3_last_first/p2`, rename the class `LinkedListStack` and modify it so that it is implemented using your `LinkedList`. It must pass the unit tests in the repository in the directory `hw3/test_linked_stack.py`. It MUST use your LinkedList. the new Stack class MUST NOT use any Python built-in or standard library collection class, i.e., the `LinkedListStack` class MUST not wrap a `list` or `deque`.

ANSWER
I included an implementation of `linked_list_stack.py` in the repository in `hw3/`. The answer provided by the student must pass `test_linked_list.py`.

Problem 3 (1 point each) Problem 3 in Discussion Questions in 4.26 of *Problem Solving with Algorithms and Data Structures using Python*.

Convert the infix expressions to postfix (use full parentheses).

a) `(A+B)*(C+D)*(E+F)`

ANSWER
The book suggests the following procedure:

1. Create an empty stack called opstack for keeping operators. Create an empty list for output.

2. Convert the input infix string to a list by using the string method split.

3. Scan the token list from left to right.

4. If the token is an operand, append it to the end of the output list.

5. If the token is a left parenthesis, push it on the opstack.

6. If the token is a right parenthesis, pop the opstack until the corresponding left parenthesis is removed. Append each operator to the end of the output list.

7. If the token is an operator, *, /, +, or -, push it on the opstack. However, first remove any operators already on the opstack that have higher or equal precedence and append them to the output list.

When the input expression has been completely processed, check the opstack. Any operators still on the stack can be removed and appended to the end of the output list.

We would start with `(A+B)*(C+D)*(E+F)`.

```
(A+B)*(C+D)*(E+F)  opstack = [], output=""
A+B)*(C+D)*(E+F)   opstack = ['('], output=""
+B)*(C+D)*(E+F)    opstack = ['('], output="A"
B)*(C+D)*(E+F)     opstack = ['(', '+'], output="A"
)*(C+D)*(E+F)      opstack = ['(', '+'], output="AB"
*(C+D)*(E+F)       opstack = [], output="AB+"
(C+D)*(E+F)        opstack = ['*'], output="AB+"
C+D)*(E+F)         opstack = ['*', '('], output="AB+"
+D)*(E+F)          opstack = ['*', '('], output="AB+C"

D)*(E+F)           opstack = ['*', '(', '+'], output="AB+C"
)*(E+F)            opstack = ['*', '(', '+'], output="AB+CD"
*(E+F)             opstack = ['*'], output="AB+CD+"
(E+F)              opstack = ['*'], output="AB+CD+*"
E+F)               opstack = ['*', '('], output="AB+CD+*"
+F)                opstack = ['*', '('], output="AB+CD+*E"
F)                 opstack = ['*', '(', '+'], output="AB+CD+*E"
)                  opstack = ['*', '(', '+'], output="AB+CD+*EF"
                   opstack = [], output="AB+CD+*EF+*"
```

So the postfix generated by the algorithm presented in the book is given by

`AB+CD+*EF+*`

b) `A+((B+C)*(D+E))`

ANSWER
Using the analogous procedure given for the conversion for **??**, we generate the following postfix:
`ABC+DE+*+`

c) `A*B*C*D+E+F`

ANSWER
The procedure in the book would provide the following result:
`AB*C*D*E+F+`
However, the following is an equivalent postfix notation for the given mathematical expression:
`ABCD***EF++`