WORKSHEET

A15.3 Trace tables and hand-tracing

Exam preparation · A level · OCR H446 2.2.1, AQA 7517 4.4.1.2, Eduqas A500QS 1.8 · about 45 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

Tracing loops, recursion and Little Man Computer programs without slips, and a binary search that prints its own trace table.

- The rules that stop slips
- Tracing a search
- Tracing recursion
- Tracing assembly language
- Arguing that it works

Questions

6 marks in all

1. OCR's exam reference language has the loop for i = 0 to 4. How many times does the body run? [1 mark]

2. What does this recursive function print? [1 mark]

```
def total(n):
    if n == 0:
        return 0
    return n % 10 + total(n // 10)
print(total(503))
```

3. What does this binary search trace print? [1 mark]

```
items = [2, 5, 9, 14, 20]
low, high = 0, len(items) - 1
while low <= high:
    mid = (low + high) // 2
    print(low, mid, high)
    if items[mid] == 20:
        break
    elif items[mid] < 20:
        low = mid + 1
    else:
        high = mid - 1
```

4. In an LMC program, BRZ end is reached with 3 in the accumulator. What happens? [1 mark]

- ☐ **A** No branch: the next instruction in sequence runs
- ☐ **B** The program jumps to end
- ☐ **C** The program halts
- ☐ **D** The accumulator is set to zero

5. When tracing a recursive function, in what order are the return values filled in? [1 mark]

- ☐ **A** From the deepest call back up to the first
- ☐ **B** From the first call down to the deepest
- ☐ **C** In any order
- ☐ **D** All at the same time

6. What does this LMC-style countdown print? [1 mark]

```
acc = 2
count = acc
while True:
    acc = count
    print(acc)
    if acc == 0:
        break
    acc = acc - 1
    count = acc
```

The task: trace the search

Make a binary search print its own trace table. Write `binary_search(items, target)`. `items` is a list of whole numbers in ascending order and `target` is a whole number. Use `low` starting at 0 and `high` starting at the last index, and loop `while low <= high`. Each time round, set `mid = (low + high) // 2`, then immediately print `low=<low> mid=<mid> high=<high> item=<items[mid]>`. If `items[mid]` equals `target`, return `mid`; if it is less than `target`, set `low = mid + 1`; otherwise set `high = mid - 1`. If the loop ends, return `-1`. Then, for each target in `[34, 13]` in order: print `searching for <target>`, call the function, and print `found <target> at <index>` or `<target> not found`. The robot stays still.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

readings = [3, 8, 12, 17, 21, 26, 30, 34, 41, 45]
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a15-3-trace-tables/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Trace the search by hand for the target 45 before you run it, then check your table against the output.
2. Change `mid` to `(low + high + 1) // 2`. Does the search still work? Trace 13 again to find out.
3. Trace the LMC program with the input 0. How many values does it output?