ANSWERS

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

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]

> **Answer:** 5. The end value is included, so i takes the values 0, 1, 2, 3 and 4.

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))
```

> **Answer:**
>
> ```
> 8
> ```
>
> It adds the digits: 3 + 0 + 5, with the calls returning from the base case back up.

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
```

Answer:

```
0 2 4
3 3 4
4 4 4
```

mid is 2 (9), so low becomes 3; mid is (3 + 4) // 2 = 3 (14), so low becomes 4; mid is 4 and 20 is found.

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

Answer: A. BRZ branches only if the accumulator is 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

Answer: A. Each call waits for the one it made, so the base case returns first, as the call stack unwinds.

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
```

Answer:

```
2
1
0
```

It outputs the count, stops when the accumulator is 0, and otherwise subtracts one: 2, 1, 0.

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]
```

The hint students can ask for: Print the row as soon as mid has been worked out, before comparing, so every comparison has a row. Then decide which half the target must be in, and move low or high past mid. The function returns the index, or -1, and the main program prints the result.

A solution

```
# 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]

def binary_search(items, target):
    low = 0
    high = len(items) - 1
    while low <= high:
        mid = (low + high) // 2
        print(f"low={low} mid={mid} high={high} item={items[mid]}")
        if items[mid] == target:
            return mid
        elif items[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

for target in [34, 13]:
    print("searching for", target)
    where = binary_search(readings, target)
    if where == -1:
        print(target, "not found")
    else:
        print("found", target, "at", where)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.