ANSWERS

A3.2 Abstract data types and stacks

Data structures · A level · OCR H446 1.4.2, AQA 7517 4.2.1.4, Eduqas A500QS 1.1 · about 20 min

What this lesson is about

ADTs, static and dynamic structures, and a stack with a top pointer: an undo stack for the robot's moves.

- Abstract data types
- Static and dynamic structures
- The stack
- Tracing a stack
- Undoing moves
- Where stacks are used

Questions

6 marks in all

1. Which rule describes a stack? [1 mark]

- ☐ **A** Last in, first out
- ☐ **B** First in, first out
- ☐ **C** Highest priority first
- ☐ **D** Smallest key first

> **Answer:** A. Items are pushed and popped at the same end, the top, so the last item pushed is the first popped.

2. Put the steps for pushing an item onto a static stack in order, where top holds the index of the top item. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `Add 1 to the top pointer`

___ `Store the item at stack[top]`

___ `If it is full, report stack overflow and stop`

___ `Check whether the stack is full`

> **Answer:**
>
> ```
> Check whether the stack is full
> If it is full, report stack overflow and stop
> Add 1 to the top pointer
> Store the item at stack[top]
> ```
>
> The full check comes first. The pointer moves up before the item is stored in the slot it now points to.

3. What does this program print? [1 mark]

```
stack = []
for item in ["A", "B", "C"]:
    stack.append(item)
stack.pop()
stack.append("D")
print(stack.pop(), stack.pop(), len(stack))
```

> **Answer:**
>
> ```
> D B 1
> ```
>
> C is popped, D is pushed, then D and B come off the top, leaving only A.

4. A static stack of size 5 has top = -1 when it is empty. After 4 pushes and then 2 pops, what is the value of top? [1 mark]

> **Answer:** 1. Four pushes take top from -1 to 3; two pops take it back to 1.

5. Which is an advantage of a dynamic data structure over a static one? [1 mark]

- ☐ **A** It uses only as much memory as the data needs, and can grow while the program runs
- ☐ **B** Any element can be reached directly by its index
- ☐ **C** It needs no extra memory for pointers
- ☐ **D** Its maximum size is known before the program runs

> **Answer:** A. A dynamic structure takes memory from the heap as needed. Direct access and a known size are advantages of static structures.

6. What is the name for the error of popping from an empty stack? [1 mark]

> **Answer:** underflow. Popping an empty stack is underflow; pushing onto a full one is overflow.

The task: undo back home

Build a static stack the way exam pseudocode does: an array `stack` of `SIZE` elements (here 4) and a `top` pointer that starts at -1, both at the top level of the program. Do not use the list's own `append` or `pop`. - Write `push(item)`: if the stack is full, print `overflow` and return `False`; otherwise add the item and return `True`. - Write `pop()`: if the stack is empty, print `underflow` and return `None`; otherwise remove the top item and return it. Both change `top`, so each needs `global top` as its first line. Then: 1. For each `(move, cm)` in `route`, push the tuple, and make the move with `do(move, cm)` only if the push worked. The fifth move will not fit, so it is never made. 2. Undo: while the stack is not empty, pop a move, print `undo <move> <cm>` (for example `undo left 15`), and make the opposite move with `do(OPPOSITE[move], cm)`. 3. Call `pop()` once more, so the empty stack prints `underflow`. The robot should finish within 6 cm of where it started.

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

OPPOSITE = {"forward": "backward", "backward": "forward", "left": "right", "right": "left"}

def do(move, cm):
    """move is "forward", "backward", "left" or "right"; cm is how far."""
    if move == "forward":
        forward(50, distance=cm)
    elif move == "backward":
        backward(50, distance=cm)
    elif move == "left":
        left(50, distance=cm)
    elif move == "right":
        right(50, distance=cm)

SIZE = 4
stack = [None] * SIZE
top = -1

def push(item):
    global top

def pop():
    global top

route = [("forward", 30), ("right", 25), ("forward", 20), ("left", 15), ("forward", 10)]
```

The hint students can ask for: Push a move before you make it, and only make it if the push worked. To undo, keep popping until the stack is empty, doing the opposite of each move. One last pop after that shows the underflow check working.

A solution

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

OPPOSITE = {"forward": "backward", "backward": "forward", "left": "right", "right": "left"}

def do(move, cm):
    if move == "forward":
```

```
        forward(50, distance=cm)
    elif move == "backward":
        backward(50, distance=cm)
    elif move == "left":
        left(50, distance=cm)
    elif move == "right":
        right(50, distance=cm)

SIZE = 4
stack = [None] * SIZE
top = -1

def push(item):
    global top
    if top == SIZE - 1:
        print("overflow")
        return False
    top = top + 1
    stack[top] = item
    return True

def pop():
    global top
    if top == -1:
        print("underflow")
        return None
    item = stack[top]
    top = top - 1
    return item

route = [("forward", 30), ("right", 25), ("forward", 20), ("left", 15), ("forward", 10)]
for move, cm in route:
    if push((move, cm)):
        do(move, cm)

while top != -1:
    move, cm = pop()
    print("undo", move, cm)
    do(OPPOSITE[move], cm)
pop()
```

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