In this assignment, you will improve your understanding of SMoL using the output from the Stacker. In every case, you will be shown a trace configuration and asked to perform some interpretive task. You are welcome to run the Stacker yourself if it will help you construct answers.

In this assignment, we will have two kinds of activities:

1. Given a configuration, *construct a program* that could have produced that trace. Ideally we want an *exact* match (except, of course, for the random addresses and line numbers), but get as close as you can.
2. Given a configuration, *determine the value of the program* from the trace.

For all programs, assume that the program contained a function called `pause` that had been defined in the program as follows:

```
(deffun (pause) 0)
```

(The following programs are intentionally written to only use additive arithmetic, so that the result of `pause` will not affect the result.) Imagine that the call to `pause` is about to finish at the point where the configuration was captured.

Configuration → Program

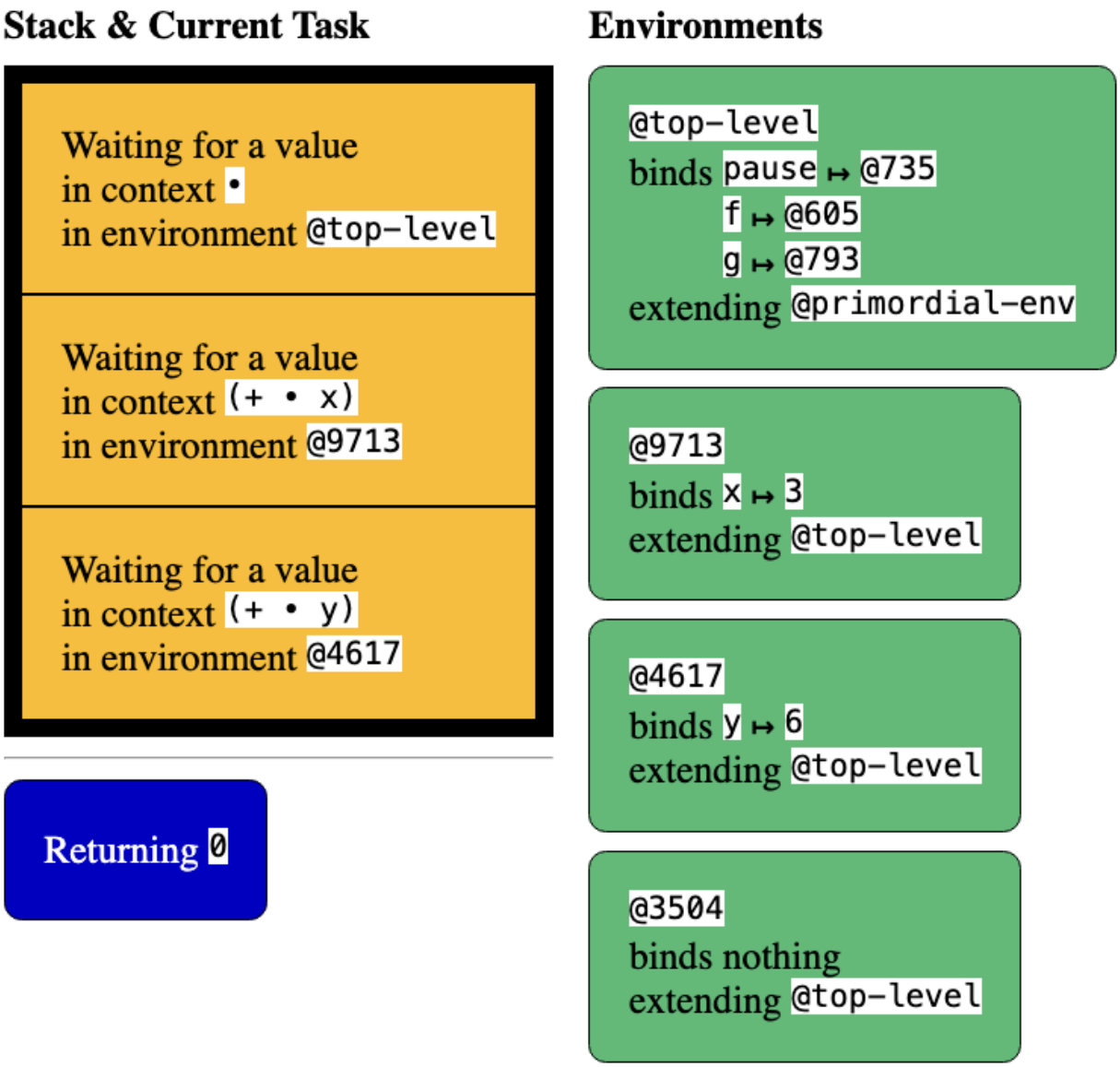
In this portion, we will show you two Stacker trace configurations. For each one, you should write a program that, when run, will result in this configuration.

In each problem, the "Returning `0`" and the bottommost environment are from `pause`. You should include `pause` and its definition in the programs that you construct.

As you might have guessed, infinitely many programs could have produced each configuration. Therefore, there is not only one "correct" answer. We do ask that you try to produce a reasonably minimal solution and avoid flights of fancy. If you really want to get clever, then also include a simple solution!

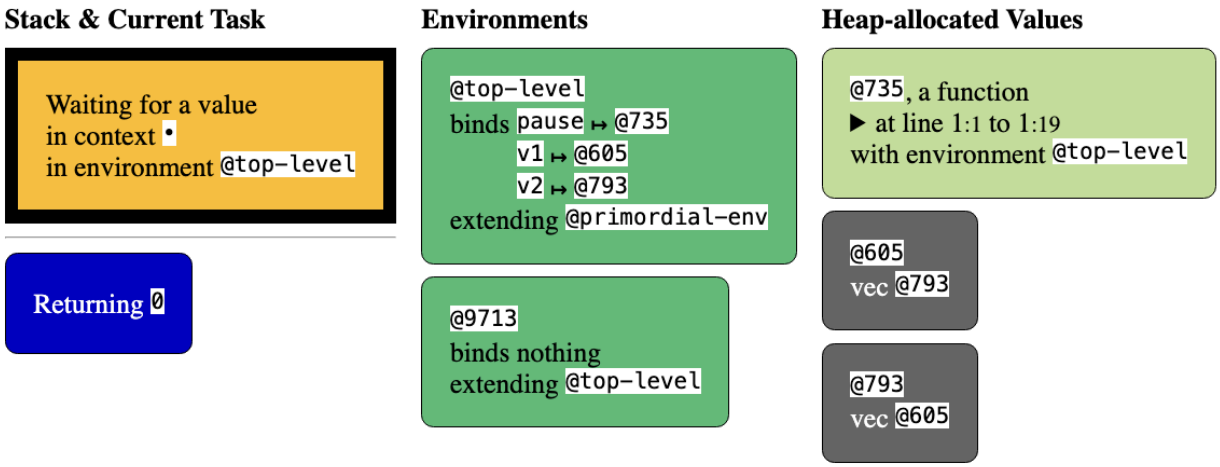
(There are two problems, one each on the two successive pages.)

1.



(We hide the "**Heap-allocated Values**" column from the screenshot)

2.


Stack & Current Task
Waiting for a value
in context •
in environment @top-level
Returning 0
Environments
@top-level
binds pause ↦ @735
v1 ↦ @605
v2 ↦ @793
extending @primordial-env
@9713
binds nothing
extending @top-level
Heap-allocated Values
@735, a function
▶ at line 1:1 to 1:19
with environment @top-level
@605
vec @793
@793
vec @605

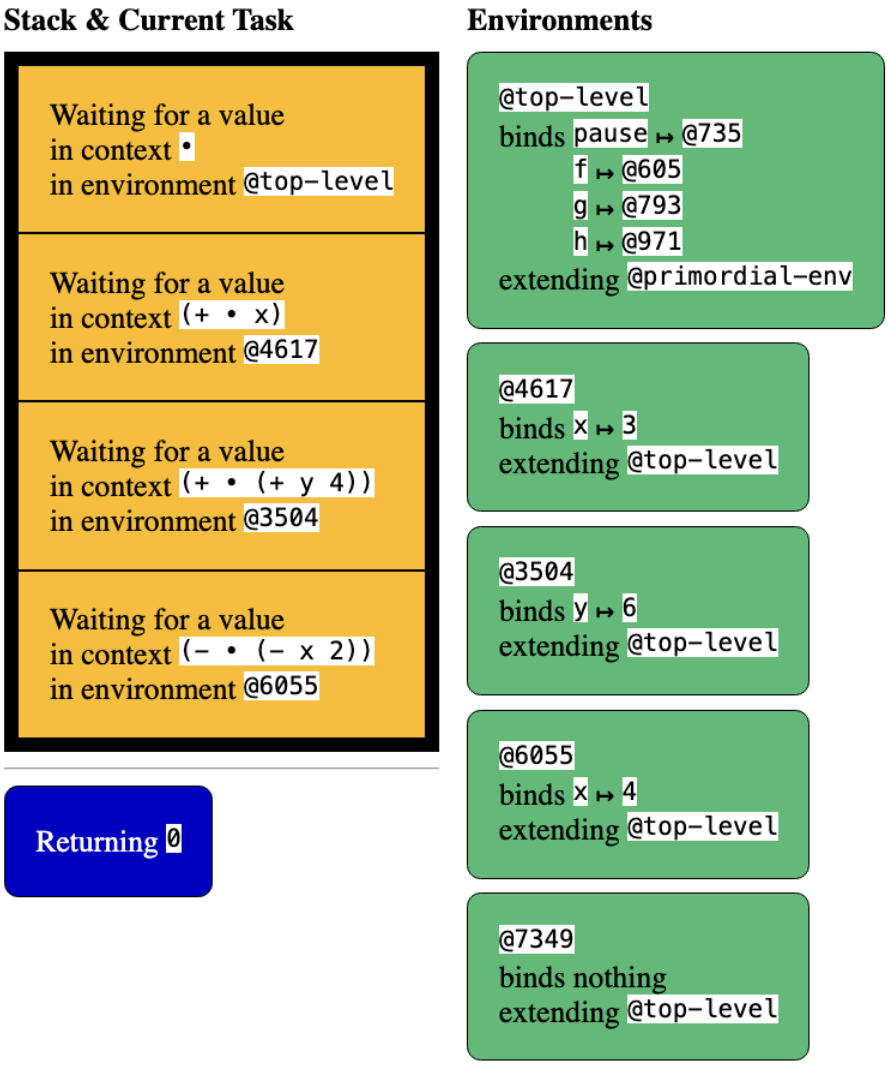
Configuration → Value

In this part, we will show you two configurations and ask you to determine what value the program will produce. Recall that `pause` is designed to not impact the answer.

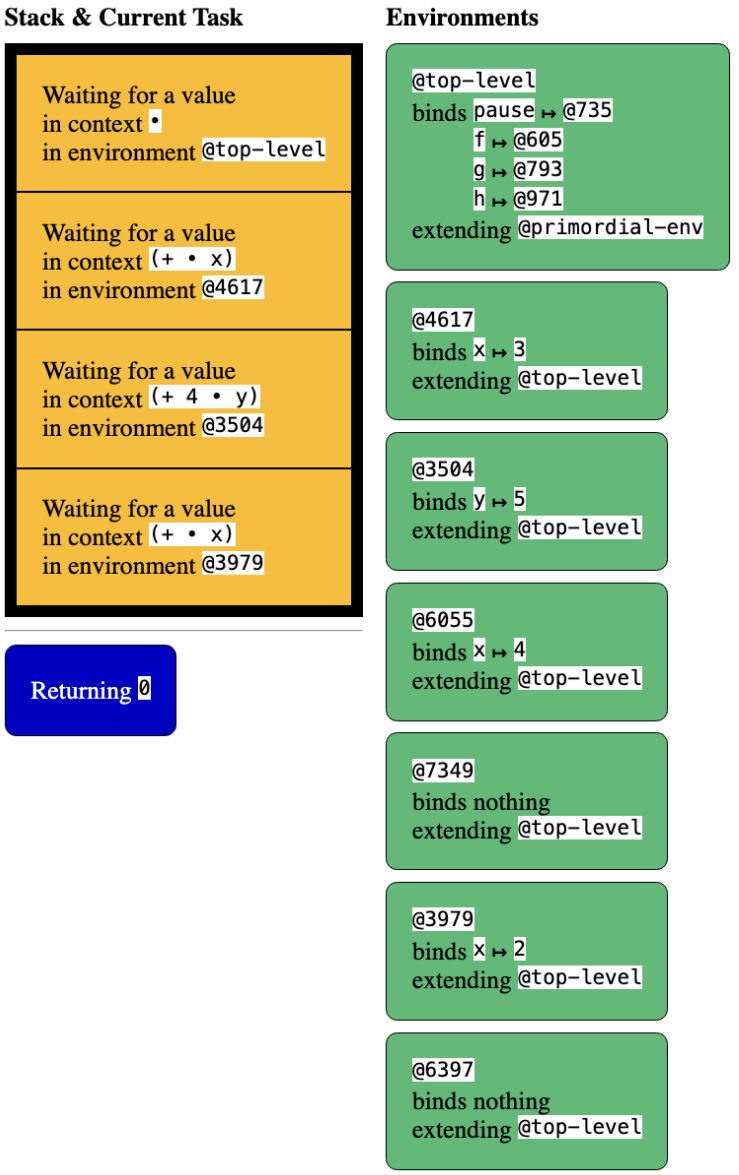
In your response, as your school math teachers used to say, “show your work”. Don’t just give us the answer (a number) but give us a sense of how you arrived at it. You don’t have to be very verbose, just enough to confirm that you understand the mapping from configurations to program results.

(There are two problems, one each on the two successive pages.)

3.


Stack & Current Task
Environments
Waiting for a value
in context •
in environment @top-level
Waiting for a value
in context (+ • x)
in environment @4617
Waiting for a value
in context (+ • (+ y 4))
in environment @3504
Waiting for a value
in context (- • (- x 2))
in environment @6055
Returning 0
@top-level
binds pause ↦ @735
f ↦ @605
g ↦ @793
h ↦ @971
extending @primordial-env
@4617
binds x ↦ 3
extending @top-level
@3504
binds y ↦ 6
extending @top-level
@6055
binds x ↦ 4
extending @top-level
@7349
binds nothing
extending @top-level

4.


Stack & Current Task
Environments
Waiting for a value
in context •
in environment @top-level
Waiting for a value
in context (+ • x)
in environment @4617
Waiting for a value
in context (+ 4 • y)
in environment @3504
Waiting for a value
in context (+ • x)
in environment @3979
Returning 0
@top-level
binds pause ↦ @735
f ↦ @605
g ↦ @793
h ↦ @971
extending @primordial-env
@4617
binds x ↦ 3
extending @top-level
@3504
binds y ↦ 5
extending @top-level
@6055
binds x ↦ 4
extending @top-level
@7349
binds nothing
extending @top-level
@3979
binds x ↦ 2
extending @top-level
@6397
binds nothing
extending @top-level