CS522 - Programming Language Semantic

Class Project

Implement small-step semantics for IMP in Racket

Syntax

```
#lang brag

prog    : stmt

id_list : ID
        | ID "," id_list

stmt    : block
        | "print" "(" aexp ")" ";"
        | "let" id_list "in" stmt
        | "int" id_list ";"
        | ID "=" aexp ";"
        | stmt stmt
        | "if" "(" bexp ")" block "else" block
        | "while" "(" bexp ")" block

block   : "{" "}"
        | "{" stmt "}"

aexp    : INT
        | ID
        | aexp "+" aexp
        | aexp "/" aexp
        | "++" ID

bexp    : BOOL
        | aexp "<=" aexp
        | "!" bexp
        | bexp "&&" bexp

```

Implementation

To be able to handle local variables properly, we need to have two data structures: (1) a memory map (*mem*) from each variable name to its value (2) a location set (*loc*) to store variables, which can be accessed in the current program scope. When the program execution exits a block or scope, we just restore the current location to the location being defined before entering the block.

Files

The directory "private-racket" contains multiple test files, whose file name starts with "imp-test". The semantics is defined in "imp-expander-small-step.rkt". "demo.mov" is a video demonstrating how we run all the tests and showing the results.

Features

- local variables
- printing for output

- variable increment

Running examples

Here we explain the results of the following examples:

```
ping128$ racket imp-test-block.rkt
rStart executing
Finish executing
Output:

Memory:
((x . 2) (y . 2))
Location:
#<set: y x>
```

This program tests whether the block inside an if statement is working properly. It also checks the evaluation of boolean in the if condition.

```
ping128$ racket imp-test-div-by-zero.rkt
Start executing
Finish executing
Output:
/: division by zero
  context...:
   /Users/ping128/Documents/UIUC/CS522/Project/private-racket/imp-expander-small-step.rkt:134:7
   /Users/ping128/Documents/UIUC/CS522/Project/private-racket/imp-expander-small-step.rkt:90:7
   /Users/ping128/Documents/UIUC/CS522/Project/private-racket/imp-expander-small-step.rkt:34:2: prog
   "/Users/ping128/Documents/UIUC/CS522/Project/private-racket/imp-test-div-by-zero.rkt": [running body]
   temp37_0
   for-loop
   run-module-instance!125
   perform-require!78
```

This program tests the division by zero.

```
ping128$ racket imp-test-empty-block.rkt
Start executing
Finish executing
Output:

Memory:
((x . 0))
Location:
#<set: x>
```

This program tests whether an empty block including a nested empty block is handled properly and the program can finish executing.

```
ping128$ racket imp-test-local-variables.rkt
Start executing
Finish executing
Output:
1
2
3
4
3
5

Memory:
((x . 3) (y . 4) (a . 5))
Location:
#<set: a y x>
```

This program checks whether we can update variable y inside the block. It also contains duplicate declaration of variable x inside the block. We design our language to just ignore any additional declaration, so the value of x is still 3 after the program exits the block.

```
ping128$ racket imp-test-local-variables-segfault.rkt
Start executing
Finish executing
Output:
0
Undefined variable: y
```

Variable y should only be accessible within the block it is declared. Thus, accessing variable y outside the block results in "Undefined variable" or "segmentation fault".

```
ping128$ racket imp-test-loop.rkt
Start executing
Finish executing
Output:
6

Memory:
((x . 4) (s . 6))
Location:
#<set: s x>
```

This program checks whether a while loop is working.

```
ping128$ racket imp-test-nested-loop.rkt
Start executing
Finish executing
Output:
16

Memory:
((i . 4) (j . 4) (s . 16))
Location:
#<set: j s i>
```

This program checks whether a nested while loop is working.

```
ping128$ racket imp-test-plus.rkt
Start executing
Finish executing
Output:

Memory:
((x . 2))
Location:
#<set: x>
```

This program contains a simple addition operation and an assignment of variable x.

```
ping128$ racket imp-test-plusplus.rkt
Start executing
Finish executing
Output:

Memory:
((x . 1) (y . 1))
Location:
#<set: y x>
```

This program tests whether operator "++" is working properly.