PA2 Report

Zhao Tang

1. Integers are tagged as “number” expressions, Booleans are tagged as “true” or “false” expressions and None are tagged as “none” expressions. There are three printing functions, print_num, print_bool and print_none. The first two takes in an i.32 argument while the last one doesn’t take in an argument. The type of function to call is decided in compile time based on the argument provided in the Chocopy code and therefore the appropriate call can be made in WASM.

```
1. x:int = 1+2
2. def fun(a:int)->int:
       print(a)
       return a
   fun(0)
3. def fun()->int:
       a:int = 0
       print(a)
       return a
     fun()
```

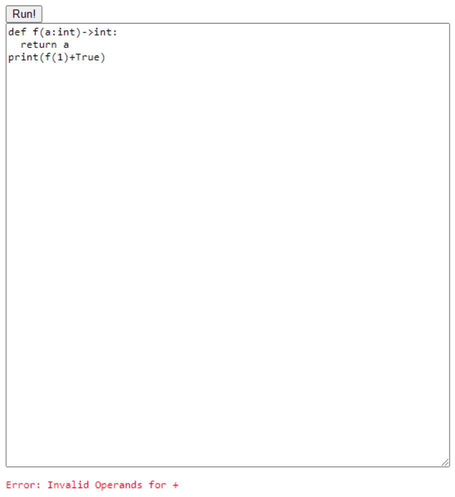
2. Global variables in type check pass are stored in the globals variable in tcProgram. Local variables in the compile pass are stored in a locals variable that’s passed in. Local variables in the type check pass are not really distinguished from global ones because that does not affect their type. Global variables are not stored in the compilation pass since the variables are already verified to exist we only check if it is local to decide on its reference scheme. Parameters only appear in code related to “define” tagged statements. During type check, it is merged into the variables argument. During compilation, the appropriate values are put on the stack in the correct order in order to be accessed as arguments.

```
while(1):
  pass
```

3. The page freezes. All interactions (including clicking on the button and interacting with the textbox) are not responsive.

4.

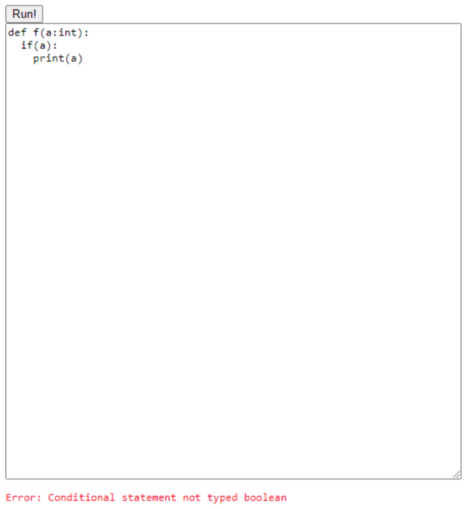
PA2 Report
Zhao Tang



```
def f(a:int)->int:
  return a
print(f(1)+True)
```

```
Error: Invalid Operands for +
```

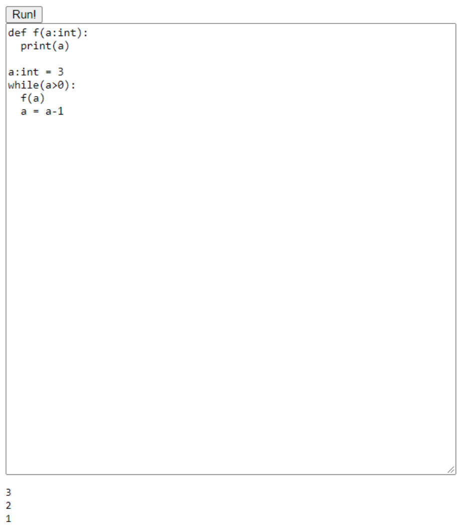
1)



```
def f(a:int):
  if(a):
    print(a)
```

```
Error: Conditional statement not typed boolean
```

2)

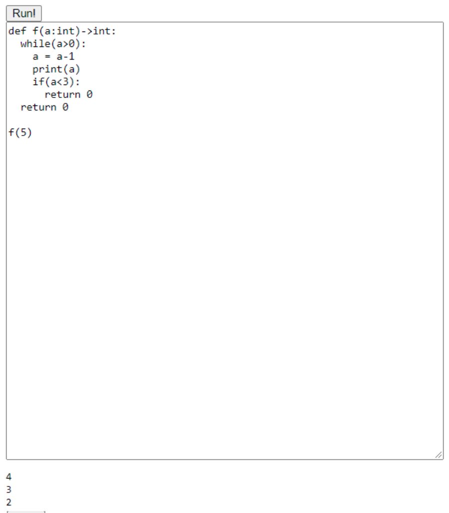


```
def f(a:int):
  print(a)

a:int = 3
while(a>0):
  f(a)
  a = a-1
```

```
3
2
1
```

3)

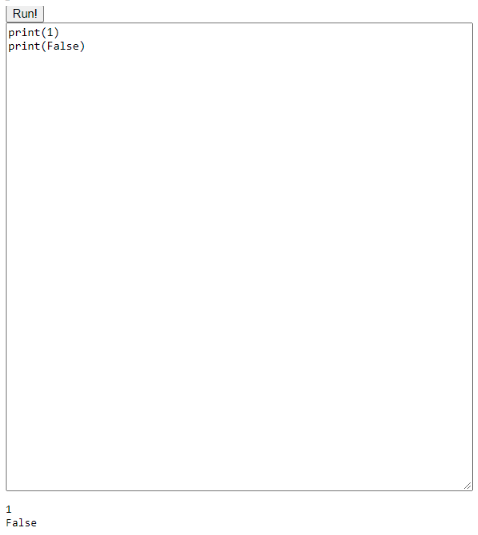


```
def f(a:int)->int:
  while(a>0):
    a = a-1
    print(a)
    if(a<3):
      return 0
  return 0

f(5)
```

```
4
3
2
```

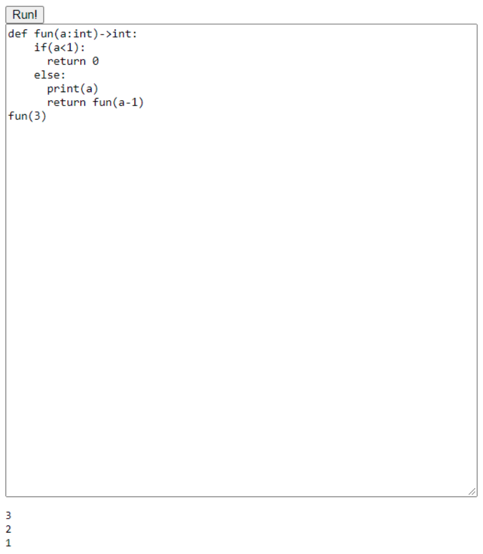
4)



```
print(1)
print(False)
```

```
1
False
```

5)

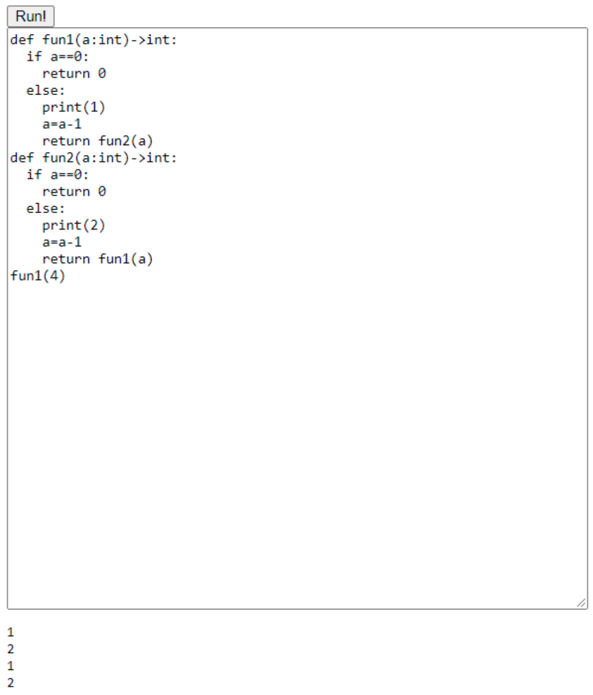


```
def fun(a:int)->int:
    if(a<1):
      return 0
    else:
      print(a)
      return fun(a-1)
fun(3)
```

```
3
2
1
```

6)

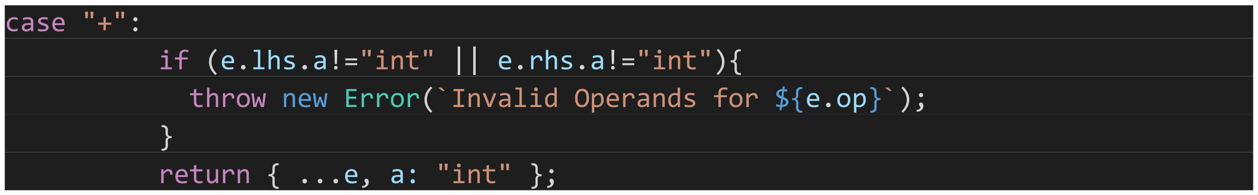
PA2 Report
Zhao Tang



```
def fun1(a:int)->int:
  if a==0:
    return 0
  else:
    print(1)
    a=a-1
    return fun2(a)
def fun2(a:int)->int:
  if a==0:
    return 0
  else:
    print(2)
    a=a-1
    return fun1(a)
fun1(4)
```

```
1
2
1
2
```

7)



```
case "+":
          if (e.lhs.a!="int" || e.rhs.a!="int"){
            throw new Error(`Invalid Operands for ${e.op}`);
          }
          return { ...e, a: "int" };
```

5. In example (1) the operands of "+" op were not typed int. Since WASM uses i32 representation for both Boolean and integer, not type checking here would cause an undefined behavior without any warning of such.
6. I did not implement the "is" binop. This is because I find it irrelevant given we are not tapping into memory locations and object references. The is statement here would simply always return False except in statements that look like "a is a". It cannot do anything useful since we cannot compare memory addresses at this point.