Lecture 26:

- Quiz 6
- Exam 2 overview
- Call and Return

Announcements:

- HW-4 out (hard deadline Mon)
- HW-5 out
- Exam 2 next Wed

Exam 2 Overview

Basics:

- Closed notes
- Worth 50 points
- 4 multipart questions

Possible Topics: *Everything fair game since ASTs Construction*

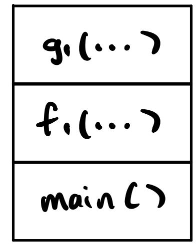
- AST creation and traversal (e.g., similar to quiz)
- Concepts in semantic analysis (symbol table, types of errors, typing rules)
- Writing specific checks via visitor setup
- Programming Lang VM concepts (instructions, bytecode, frames, call stack, operand stack)
- MyPL VM instructions (e.g., similar to quiz)

To study:

- Quiz yourself from the lecture notes
- Redo lecture note check-ins
- Re-answer the quizzes from scratch

Example: f(...) $\overset{calls}{\Longrightarrow}$ g(...) $\overset{calls}{\Longrightarrow}$ g(...)

Frame call stack prior to 2nd g(...) call:



(1) Get stack frame info for (2nd call to) g(...)

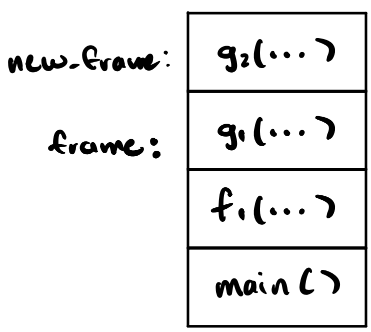
```
fun_name = instr.operand
```

(2) Instantiate a new frame for 2nd g(...) call

```
new_frame_template = self.frame_templates[fun_name]
new_frame = VMFrame(new_frame_template)
```

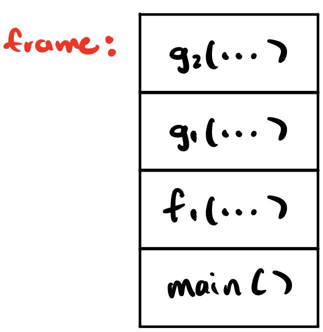
And push it onto the frame call stack:

```
self.call_stack.append(new_frame)
```



(3) Copy arg_count arguments into new_frame operand stack

```
arg = frame.operand_stack.pop()
new_frame.operand_stack.append(arg)
```

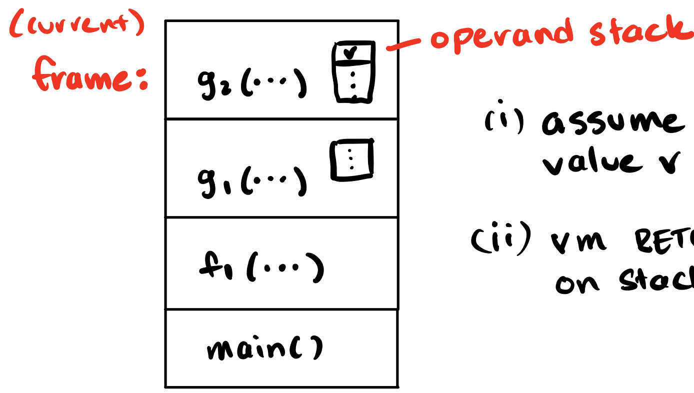


(4) Set current frame in VM to new_frame

```
frame = new_frame
```

Example: Returning from f(...) ⇒ g(...) ⇒ g(...)

Frame stack before return



(i) assume g₂ pushed its return value v

(ii) vm RET() requires a return value on stack even if just nullptr

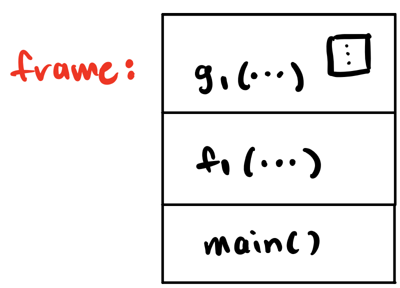
(1) Grab return value

return_val = frame.operand_stack.pop()

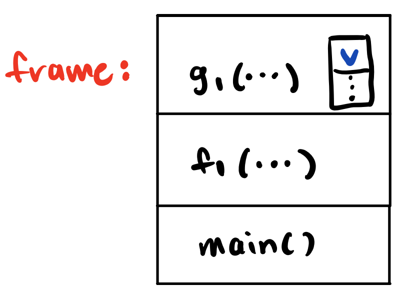
(2) Pop frame

self.call_stack.pop()

frame = self.call_stack[-1]



(3) If frame exists, push return value



frame.operand_stack.append(return_val)

(d) Heap Details (in `mypl_vm.py`)

- heap is separated into struct free space and array free space
- each heap object is assigned a unique object id on allocation

The "struct" heap

- Each struct object represented as a field-value dictionary
- Instructions to set and get fields via the object id

For example: **`ALLOCS()`** (struct allocation instruction) ...

```
oid = self.next_obj_id
self.next_obj_id += 1
self.struct_heap[oid] = {}
frame.operand_stack.append(oid)
```

The "array" heap

- Each array object represented as a Python list
- Assigned a fixed number of slots on allocation (containing None)
- Instructions to set and get items by index

For example: **`ALLOCA()`** (array allocation instruction) ... pop default val, pop size

```
oid = self.next_obj_id
self.next_obj_id += 1
array_length = frame.operand_stack.pop()
# ... check for valid array length value ...
self.array_heap[oid] = [None for _ in range(array_length)]
frame.operand_stack.append(oid)
```

Summary – Things to Know

1. Structure of MyPL VM: Frame Templates, Frames, Variables, Operand Stack, Instructions, Heaps, VM

2. What the MyPL instructions are and how they generally work

3. How to write simple programs using the instructions

4. How to implement each instruction within the VM

5. How function calls work

6. How the heap works generally wrt object ids and storage