Procedure/Subroutine Calls For HLL

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

Calling Functions/Methods

The Java (abc.java) and the Python (abc.py) programs perform the same actions

- Method/Function “a” calls “b”, “b” then calls “c”, “c” returns, then “b” returns, then “a” returns
  - a is called from main
  - b is called from a
  - c is called from b
  - c returns to b
  - b returns to a
  - a returns to main.

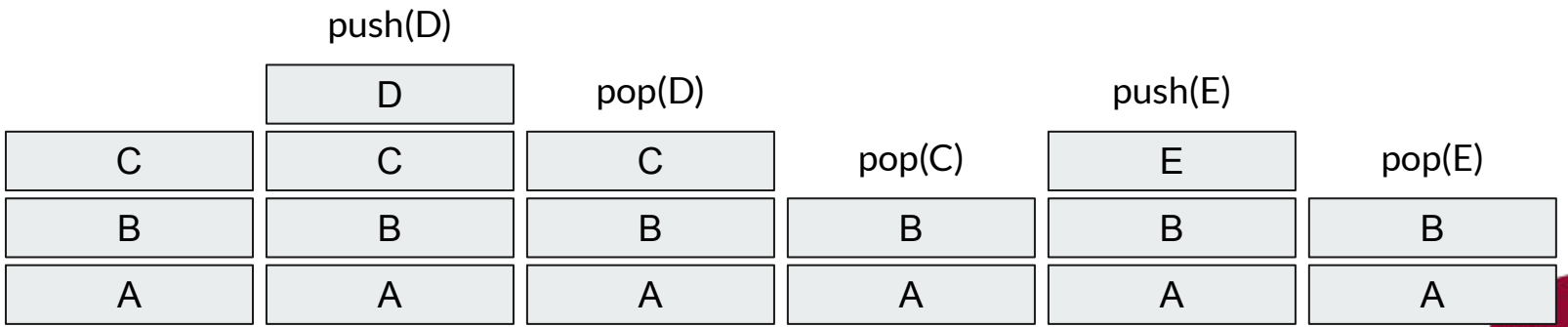
```
if __name__ == “__main__”:
        a()
```

Calling Functions/Methods

- From the perspective of a:
  - main is the caller
  - a is the callee
- From the perspective of b:
  - a is the caller
  - b is the callee
- The callee returns to the caller
- Each time a function/method is called, the language must store the next statement's address (PC) to execute, so the function/method returns to the point after it was called

Stacks

- Data structure where the last item pushed, is the first item popped
- A stack is sometimes called a LIFO, last in, first out
- Stack of dishes



Stacking abc

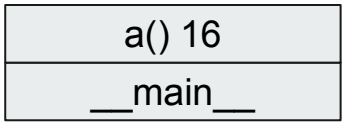
```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

$ python abc.py

__main__

Stacking abc

```
1 def a() :
2     print('start a')
3     b()
4     print('end a')
5
6 def b() :
7     print('start b')
8     c()
9     print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```



Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

$ python abc.py

a() 16
__main__

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
```

a() 16
__main__

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
```

b() 4
a() 16
__main__

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

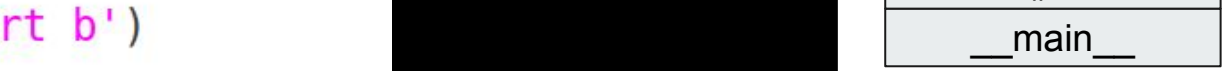
```
$ python abc.py
start a
```

b() 4
a() 16
__main__

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
```



MEMORIAL UNIVERSITY
www.mun.ca

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

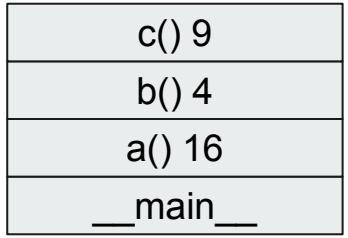
```
$ python abc.py
start a
start b
```

c() 9
b() 4
a() 16
__main__

Stacking abc

```
 1 def a() :
 2     print('start a')
 3     b()
 4     print('end a')
 5
 6 def b() :
 7     print('start b')
 8     c()
 9     print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

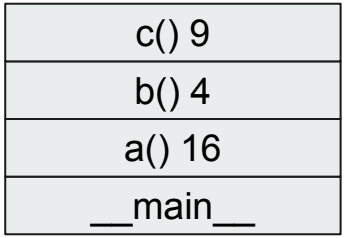
```
$ python abc.py
start a
start b
```



Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
```

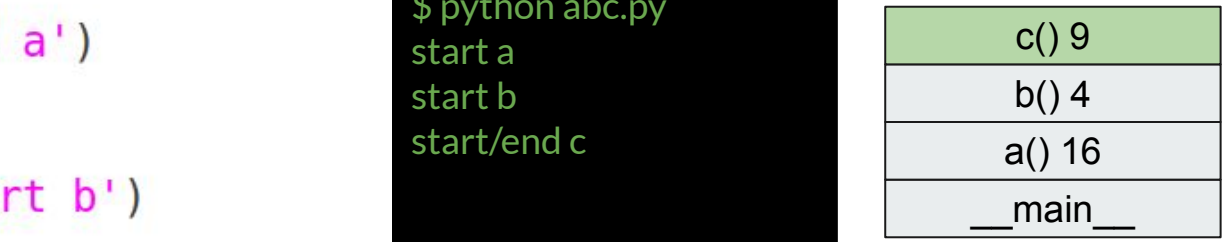


MEMORIAL UNIVERSITY
www.mun.ca

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
```



Stacking abc

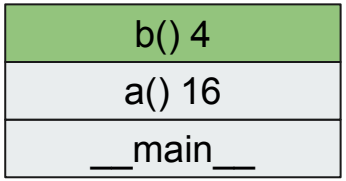
```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
end b
```

Stacking abc

```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
end b
```



MEMORIAL UNIVERSITY
www.mun.ca

Stacking abc

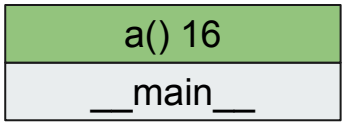
```
1  def a() :
2      print('start a')
3      b()
4      print('end a')
5
6  def b() :
7      print('start b')
8      c()
9      print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
end b
end a
```

Stacking abc

```
1 def a() :
2     print('start a')
3     b()
4     print('end a')
5
6 def b() :
7     print('start b')
8     c()
9     print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
end b
end a
```



Stacking abc

```
1 def a() :
2     print('start a')
3     b()
4     print('end a')
5
6 def b() :
7     print('start b')
8     c()
9     print('end b')
10
11 def c() :
12     print('start/end c')
13
14 if __name__ == "__main__":
15     a()
```

```
$ python abc.py
start a
start b
start/end c
end b
end a
```

__main__

Stack Traces

- In programming languages, a stack is used to store the address where a subroutine must return to
- When an exception occurs in Python or Java, then a stack trace is displayed
- Notice that Java and Python print the stack trace in reverse order but always following the stack tracing

Stack Traces

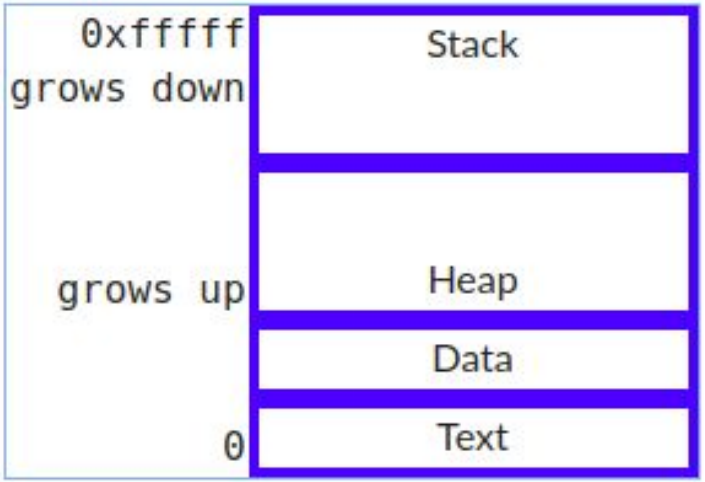
```
$ python abc_d0.py
start a
start b
Traceback (most recent call last):
  File "abc_d0.py", line 15, in <module>
    a()
  File "abc_d0.py", line 3, in a
    b()
  File "abc_d0.py", line 8, in b
    c()
  File "abc_d0.py", line 12, in c
    return 1 // 0;
ZeroDivisionError: integer division or modulo by zero
```

Stack Traces

```
$ java abc_d0
start a
start b
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at abc_d0.c(abc_d0.java:17)
    at abc_d0.b(abc_d0.java:12)
    at abc_d0.a(abc_d0.java:6)
    at abc_d0.main(abc_d0.java:22)
```

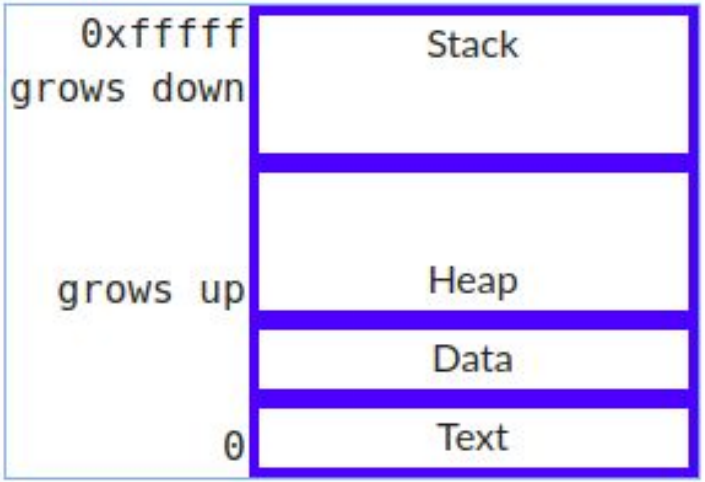
Memory Layout

- In many programming languages, the PC is saved on a stack (LIFO)
- A stack stores the return information in a Last In First Out order
- The last function called, is the first function to return
- The stack also usually stores the local static declared variables that are deleted once the function return
    - allocated and freed automatically



Memory Layout

- Assuming a total memory of 0x100000, a common memory layout for a program is in this image
- The stack starts off at the highest address and grows down as you call functions
- Java and Python objects live on the heap, dynamically created objects
- In Java, an object is create by new
- This area is garbage collected for Java and Python
- Heap usually stores dynamic memory
- Once you have allocated memory on the heap, you are responsible for using **free()**



Next steps

- abc in c example in the notes
- ARM subroutine calls

Questions?