Object Oriented Programming

Additional Handout

Call stacks, heaps and pointers

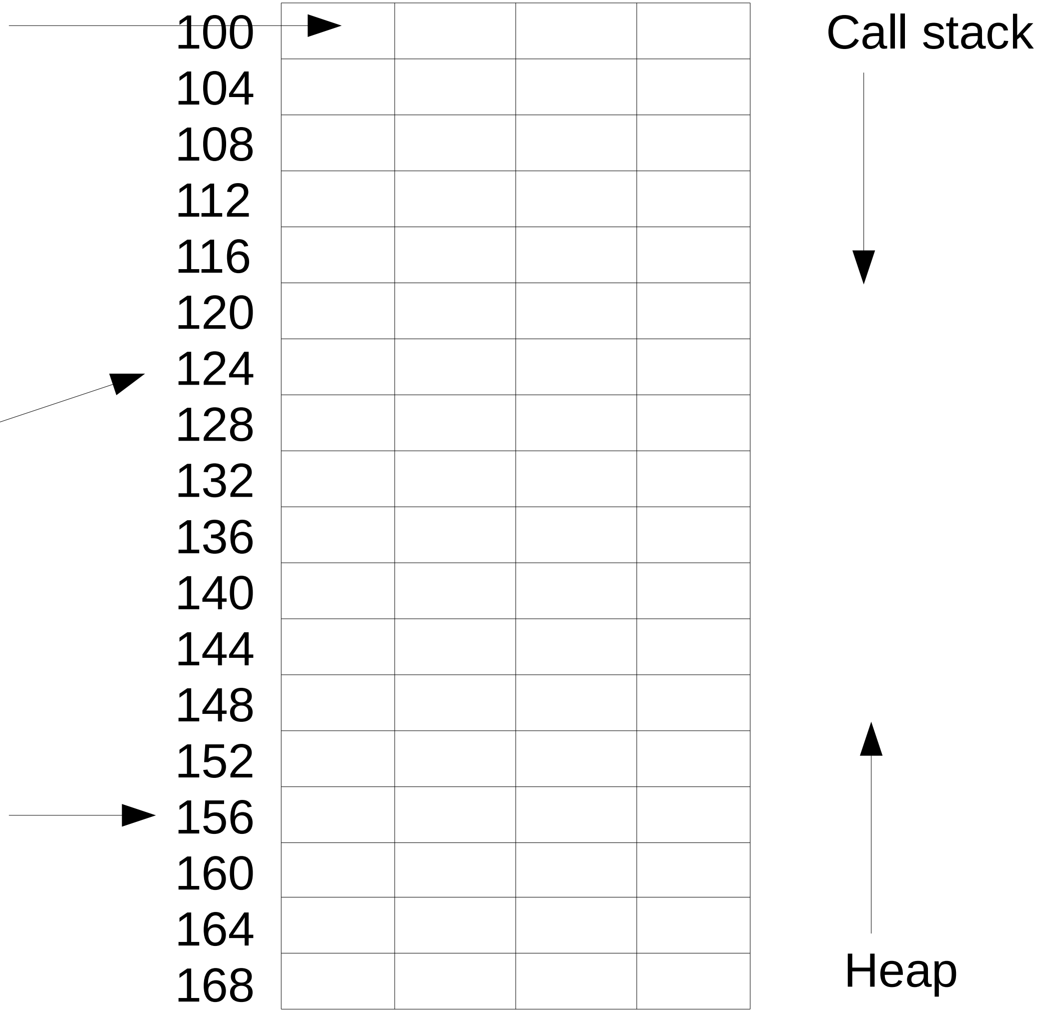
Andrew Rice
Nov 2018

Each cell is a ‘byte’

32-bit architecture
=> 4 bytes to a word

Address
(usually written
in hexadecimal)
e.g. 0x07C

Each row is a ‘word’



```
   void f(int x) {
       char c = 'a';
       long l = 1234;
       int i = 10;
   }

>> f(4);
```

100				
104				
108				
112				
116				
120				
124				
128				
132				
136				
140				
144				
148				
152				
156				
160				
164				
168				

This example is in C/C++

```
>> void f(int x) {
       char c = 'a';
       long l = 1234;
       int i = 10;
   }

   f(4);
```

100	4	0	0	0	x
104					c
108					l
112					
116					i
120					
124					
128					
132					
136					
140					
144					
148					
152					
156					
160					
164					
168					

```
   void f(int x) {
>>     char c = 'a';
       long l = 1234;
       int i = 10;
   }

   f(4);
```

100	4	0	0	0	x
104	97	.	.	.	c
108					l
112					
116					i
120					
124					
128					
132					
136					
140					
144					
148					
152					
156					
160					
164					
168					

```
   void f(int x) {
       char c = 'a';
>>     long l = 1234;
       int i = 10;
   }

   f(4);
```

100	4	0	0	0	x
104	97	.	.	.	c
108	210	4	0	0	l
112	0	0	0	0	
116					i
120					
124					
128					
132					
136					
140					
144					
148					
152					
156					
160					
164					
168					

1234 is bigger than one byte

1234 & 0xFF = 210
1234 >> 8 = 4

```
   void f(int x) {
       char c = 'a';
       long l = 1234;
>>     int i = 10;
   }

   f(4);
```

100	4	0	0	0	x
104	97	.	.	.	c
108	210	4	0	0	l
112	0	0	0	0	
116	10	0	0	0	i
120					
124					
128					
132					
136					
140					
144					
148					
152					
156					
160					
164					
168					

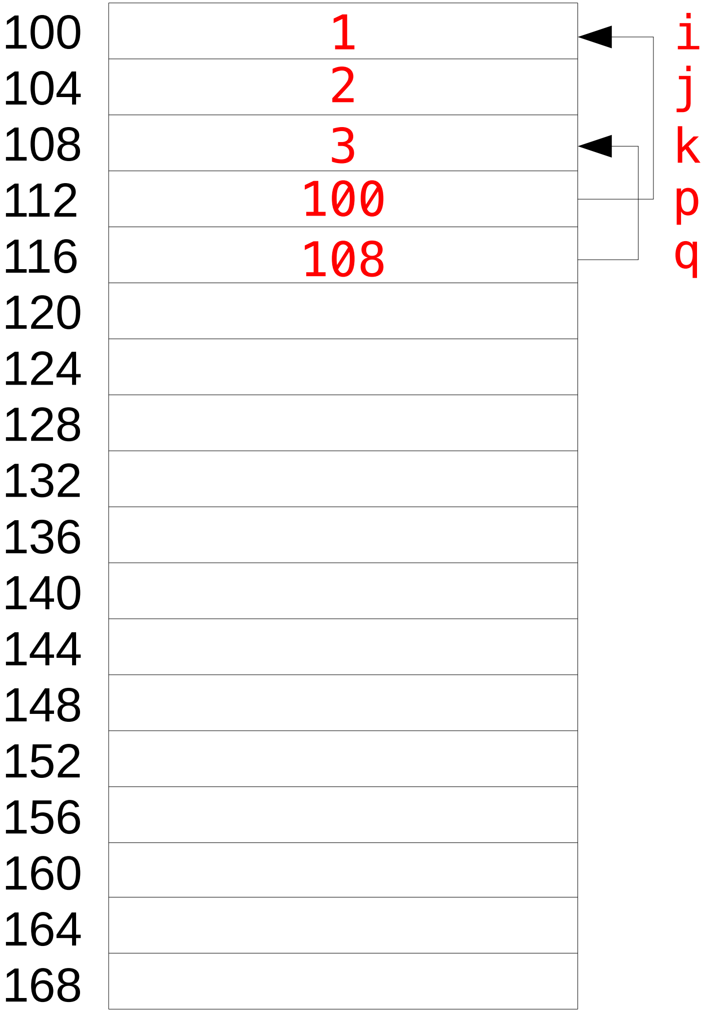
```
   void f(int x) {
       char c = 'a';
       long l = 1234;
>>     int i = 10;
   }

   f(4);
```

100	4	x
104	‘a’	c
108	1234	l
112		
116	10	i
120		
124		
128		
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```c
void f() {
        int i = 1;
        int j = 2;
        int k = 3;
        int* p = &i;
        int* q = &k;
}
```

* on a LHS means
‘its a pointer’

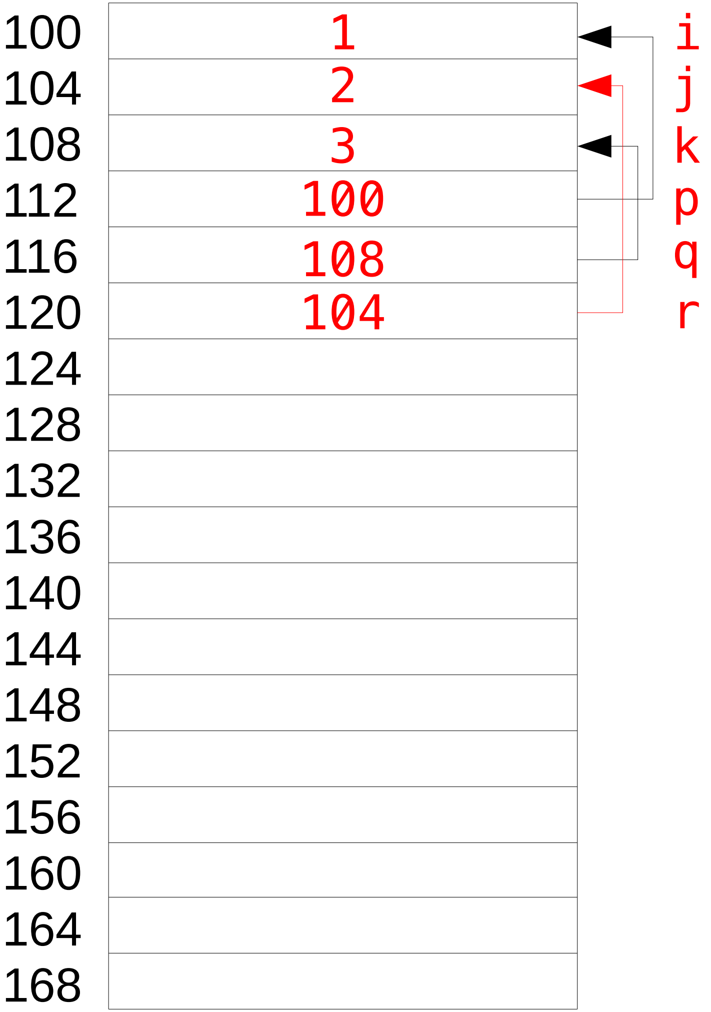
& on a RHS means
‘take the address of’



100	1	i
104	2	j
108	3	k
112	100	p
116	108	q
120		
124		
128		
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```
void f() {
        int i = 1;
        int j = 2;
        int k = 3;
        int* p = &i;
        int* q = &k;
        int* r = p + 1;
}
```

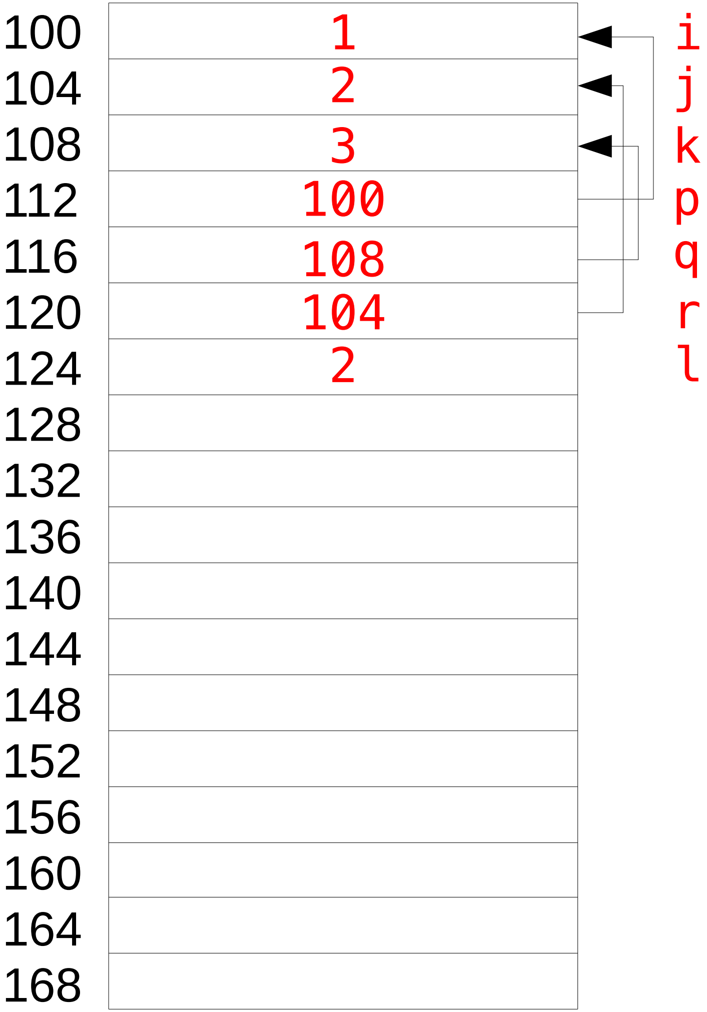
We can do arithmetic on pointers (based on the datatype size)



100	1	i
104	2	j
108	3	k
112	100	p
116	108	q
120	104	r
124		
128		
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```c
void f() {
        int i = 1;
        int j = 2;
        int k = 3;
        int* p = &i;
        int* q = &k;
        int* r = p + 1;
        int l = *r;
}
```

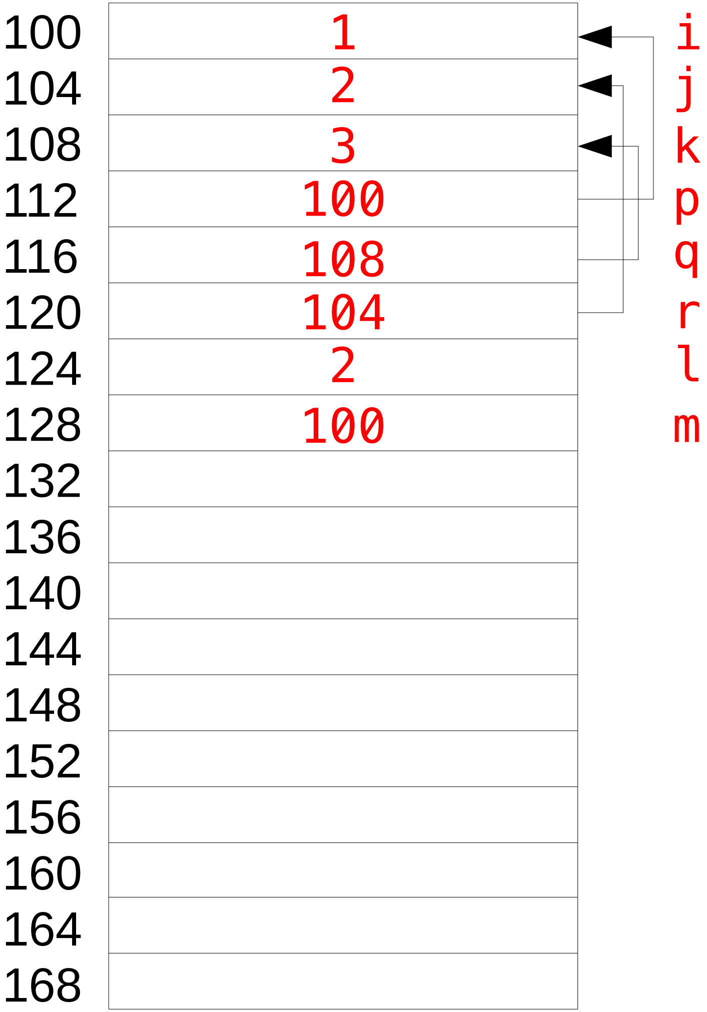
* on the RHS means ‘dereference’ i.e. follow the pointer.



100	1	i
104	2	j
108	3	k
112	100	p
116	108	q
120	104	r
124	2	l
128		
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```c
void f() {
    int i = 1;
    int j = 2;
    int k = 3;
    int* p = &i;
    int* q = &k;
    int* r = p + 1;
    int l = *r;
    int m = *(q + 1);
}
```

Nothing will stop you making mistakes!

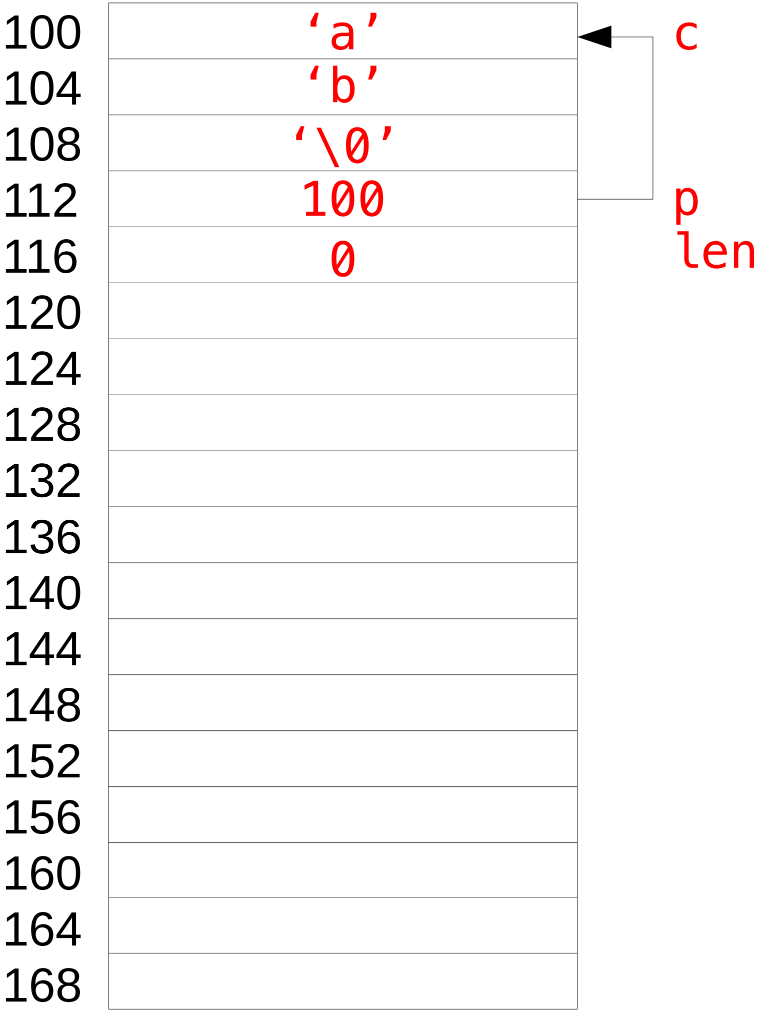


100	1	i
104	2	j
108	3	k
112	100	p
116	108	q
120	104	r
124	2	l
128	100	m
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```
int len() {
    char[] c =
        {'a','b','\0'};
    char* p = c;
    int len = 0;
    while(*p++) len++;
    return len;
}
```

One use of pointers is to iterate over an array

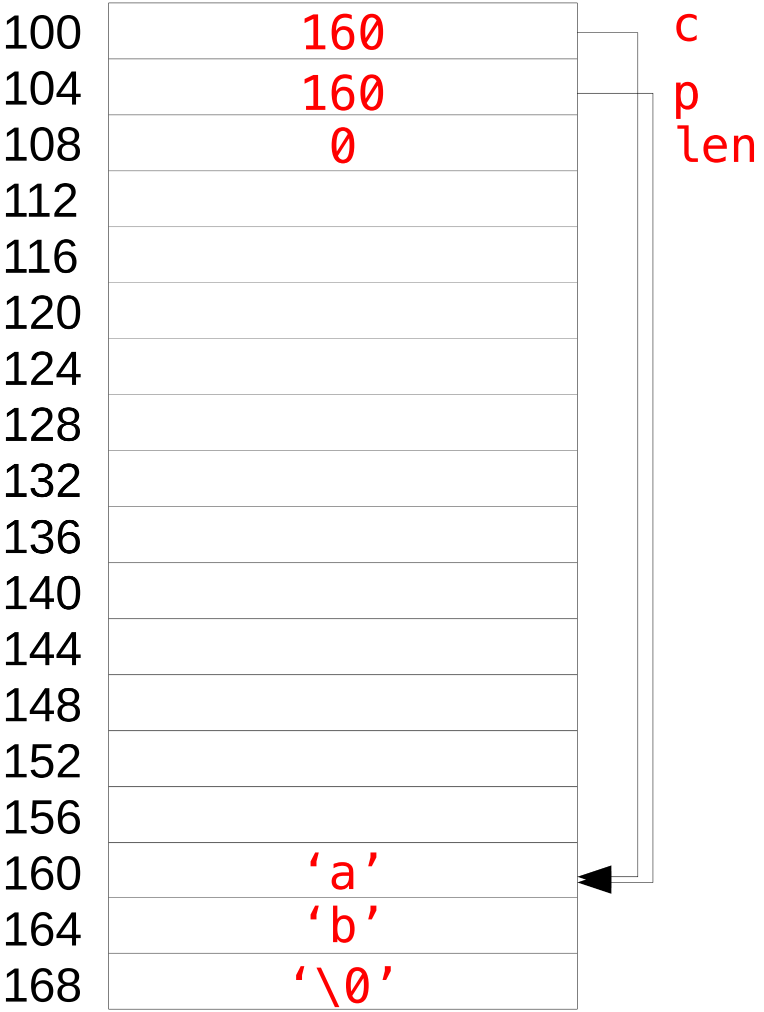
Evaluate to the character pointed to by p and then post-increment p



100	‘a’	c
104	‘b’	
108	‘\0’	
112	100	p
116	0	len
120		
124		
128		
132		
136		
140		
144		
148		
152		
156		
160		
164		
168		

```
int len() {
    char[] c = new[]
        {'a','b','\0'};
    char* p = c;
    int len = 0;
    while(*p++) len++;
    delete[] c;
    return len;
}
```

In C++ you can choose whether you want your array on the stack or the heap



100	160	c
104	160	p
108	0	len
112		
116		
120		
124		
128		
132		
136		
140		
144		
148		
152		
156		
160	‘a’	
164	‘b’	
168	‘\0’	

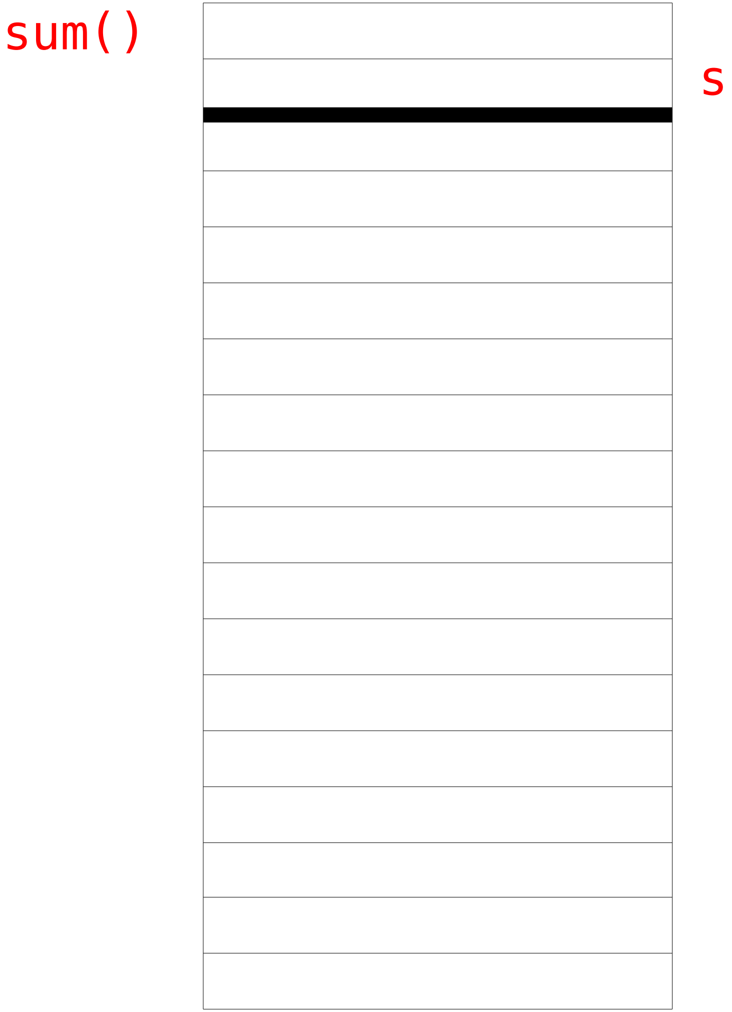
Items on the stack exist only for the duration of your function call

Items on the heap exist until they are deleted

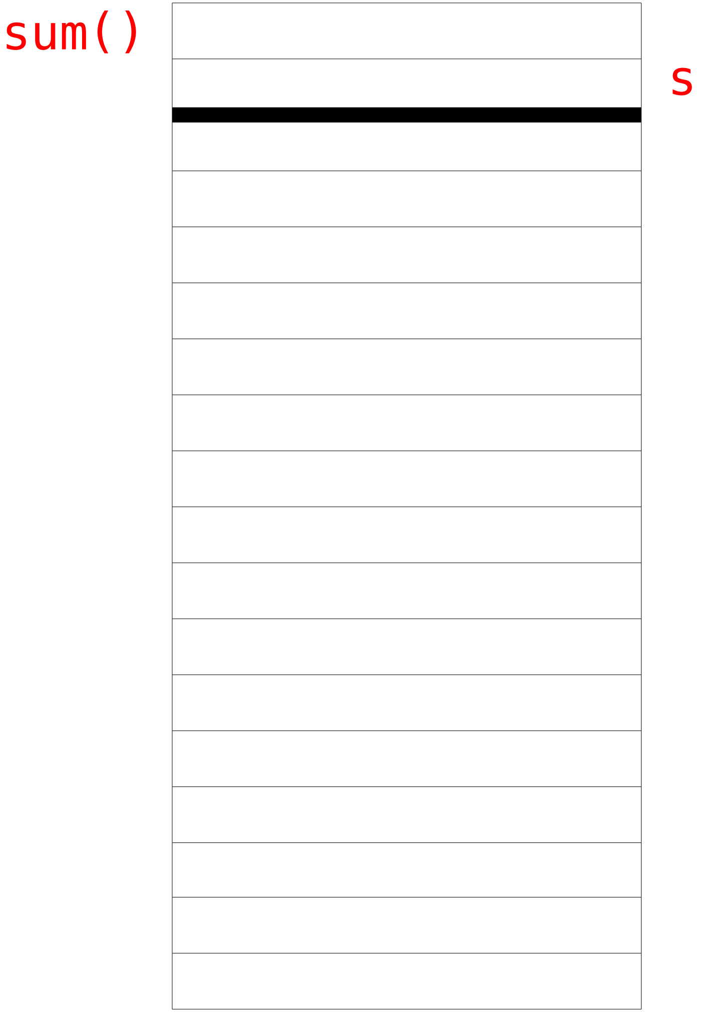
```
>> static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

This example is in Java

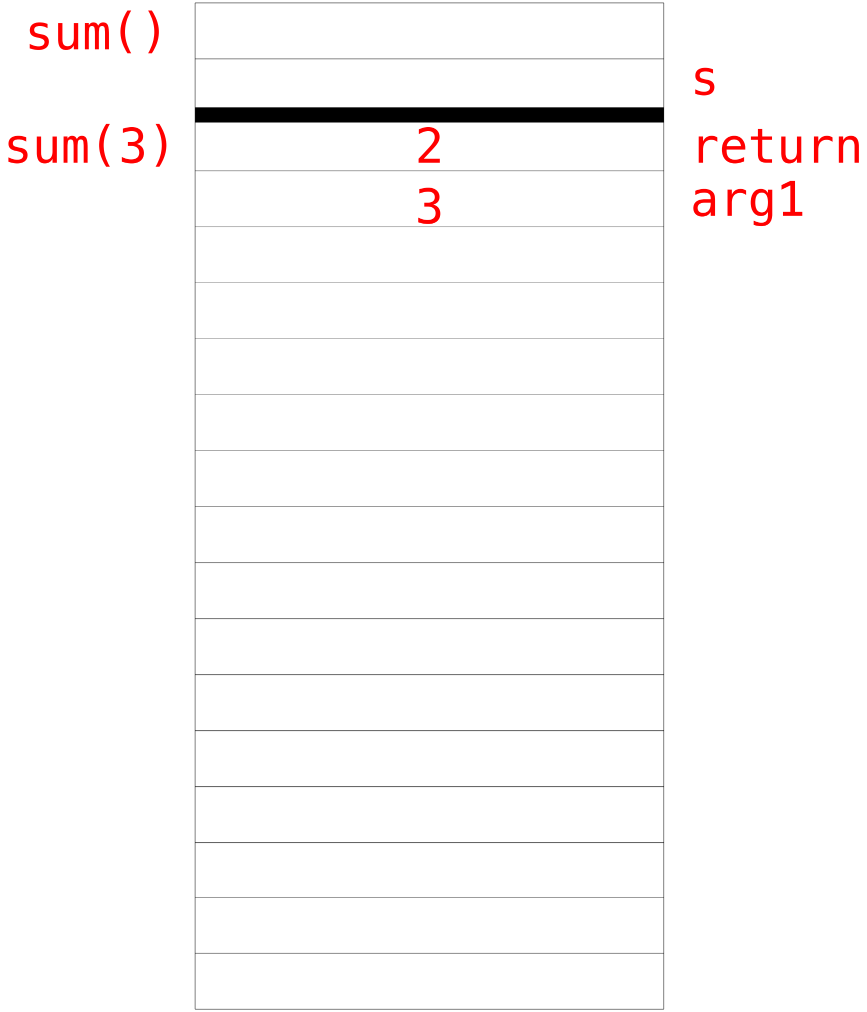
```
>> static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



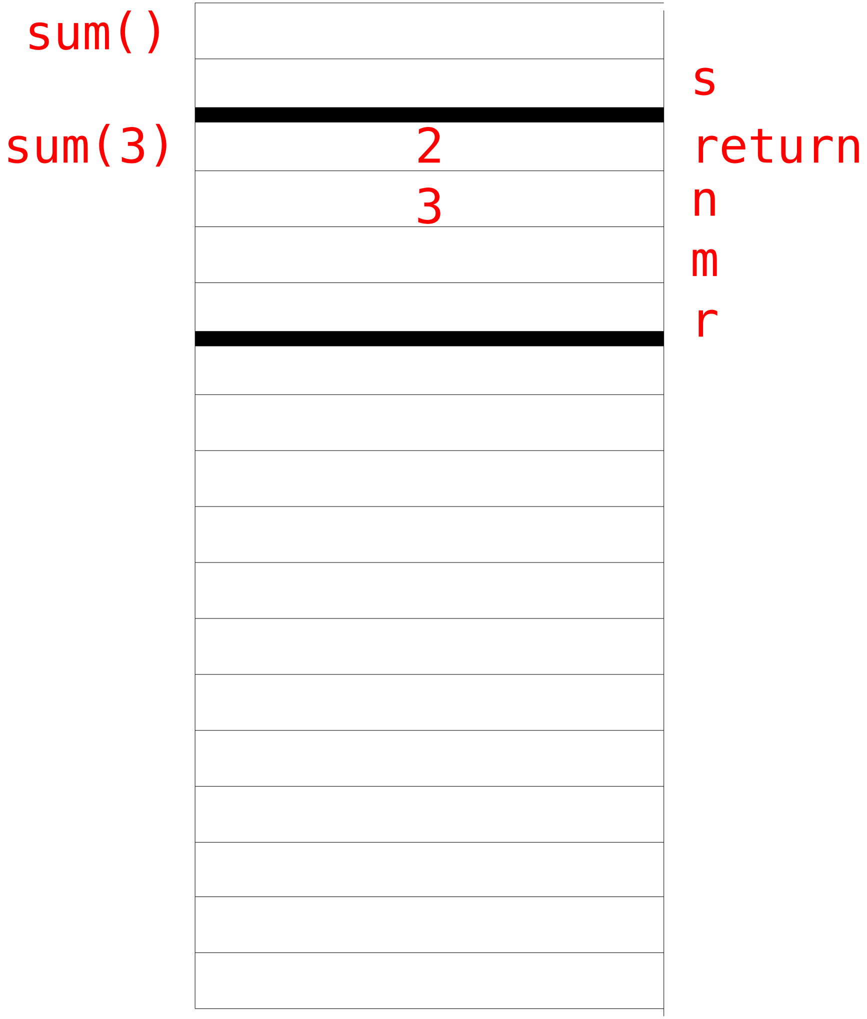
```
   static int sum() {
>>   int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



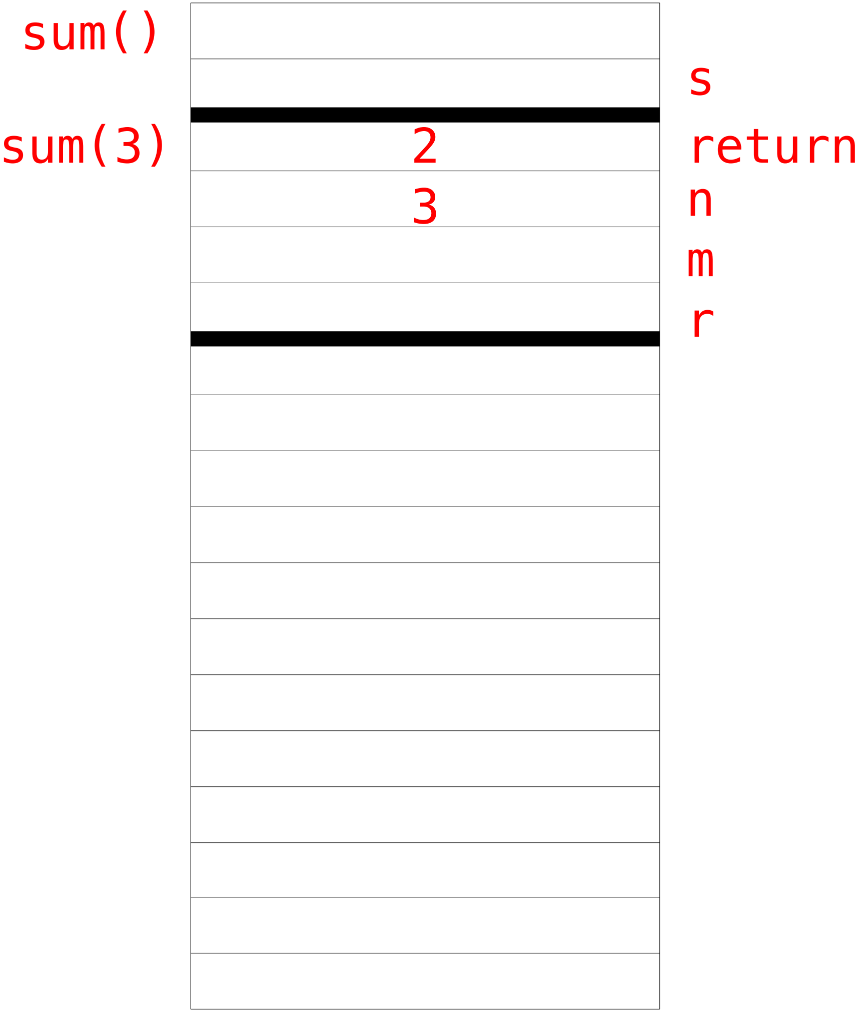
```
   static int sum() {
>>   int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



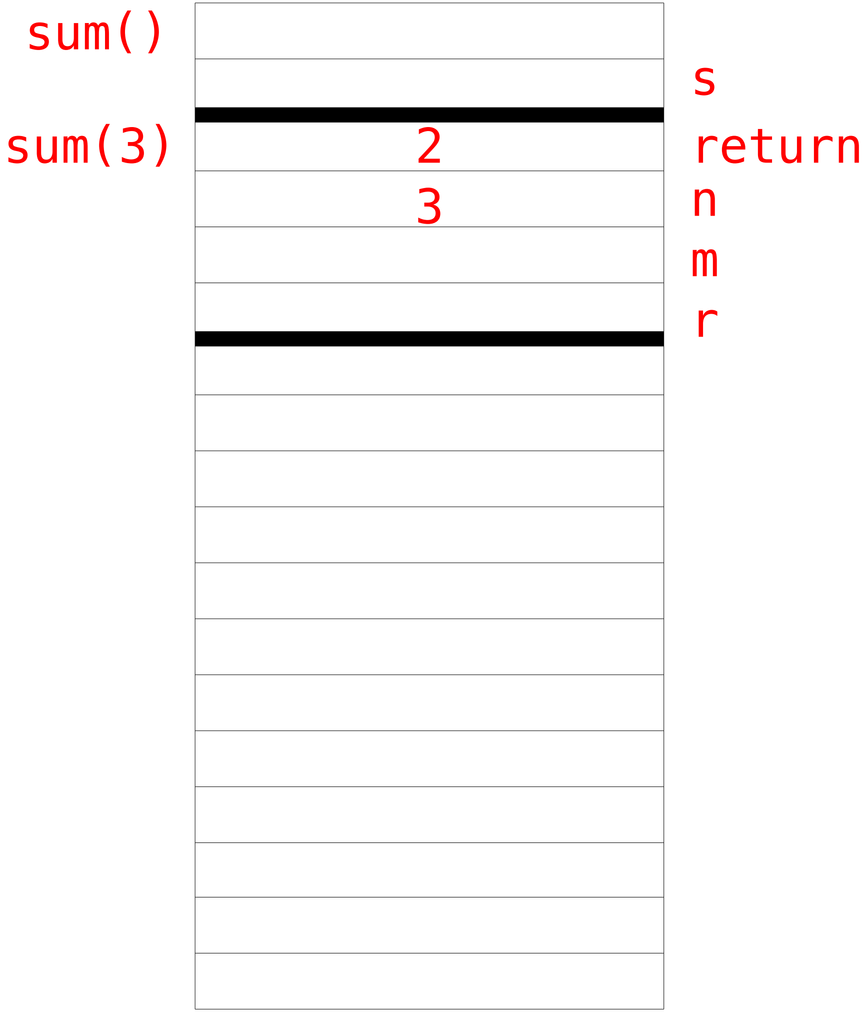
```
   static int sum() {
     int s = sum(3);
     return s;
   }

>> static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



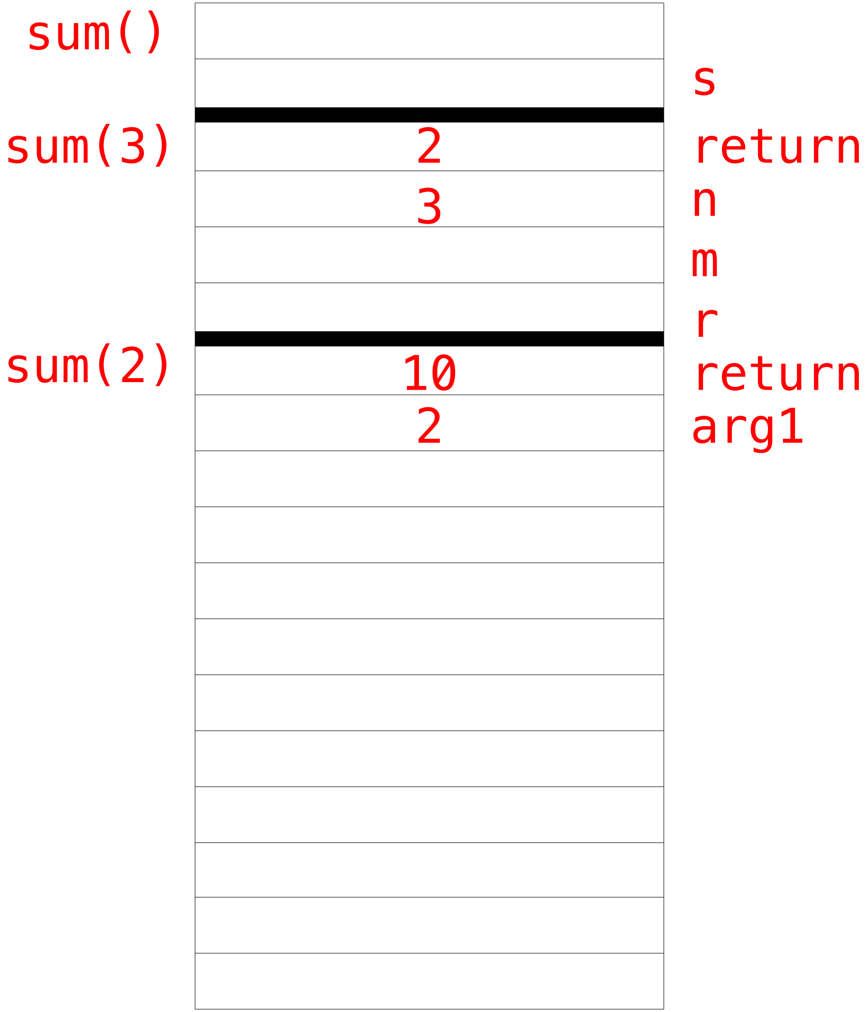
```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
>>   if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



```
static int sum() {
  int s = sum(3);
  return s;
}

static int sum(int n) {
  if (n == 0) {
    return 0;
  }
>>  int m = sum(n - 1);
  int r = m + n;
  return r;
}
```

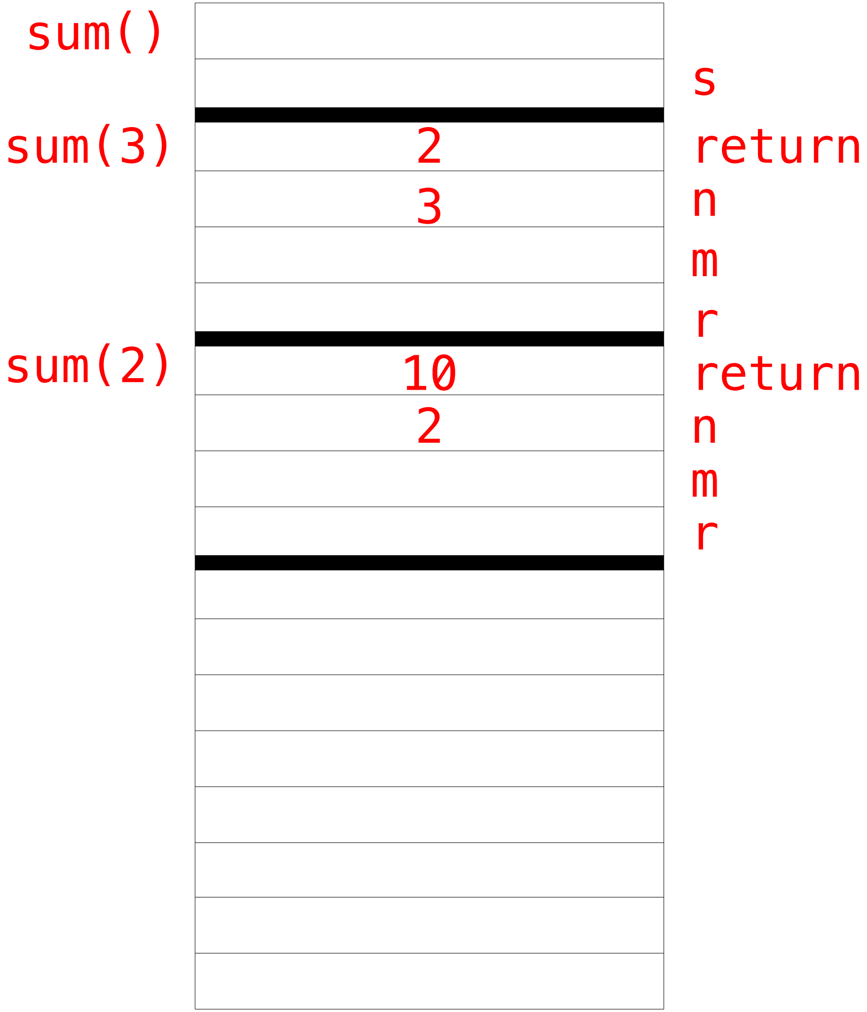


```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	arg1

```
   static int sum() {
     int s = sum(3);
     return s;
   }

>> static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r

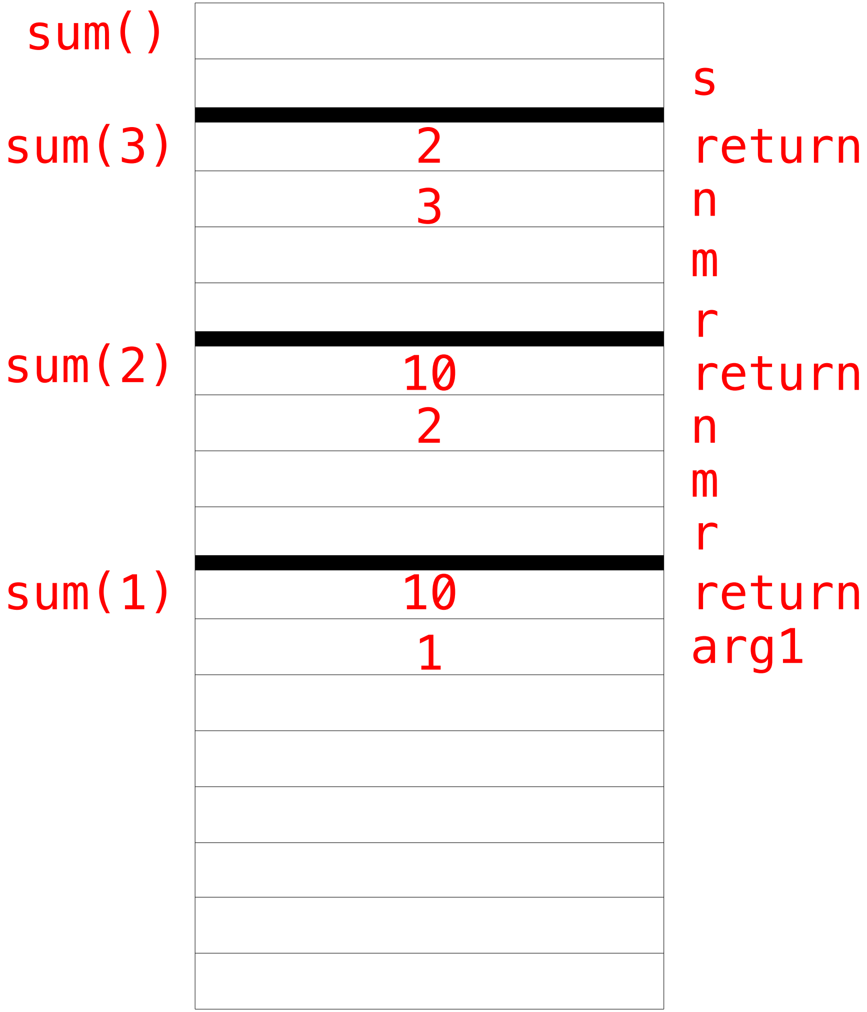
```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
>>   if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r

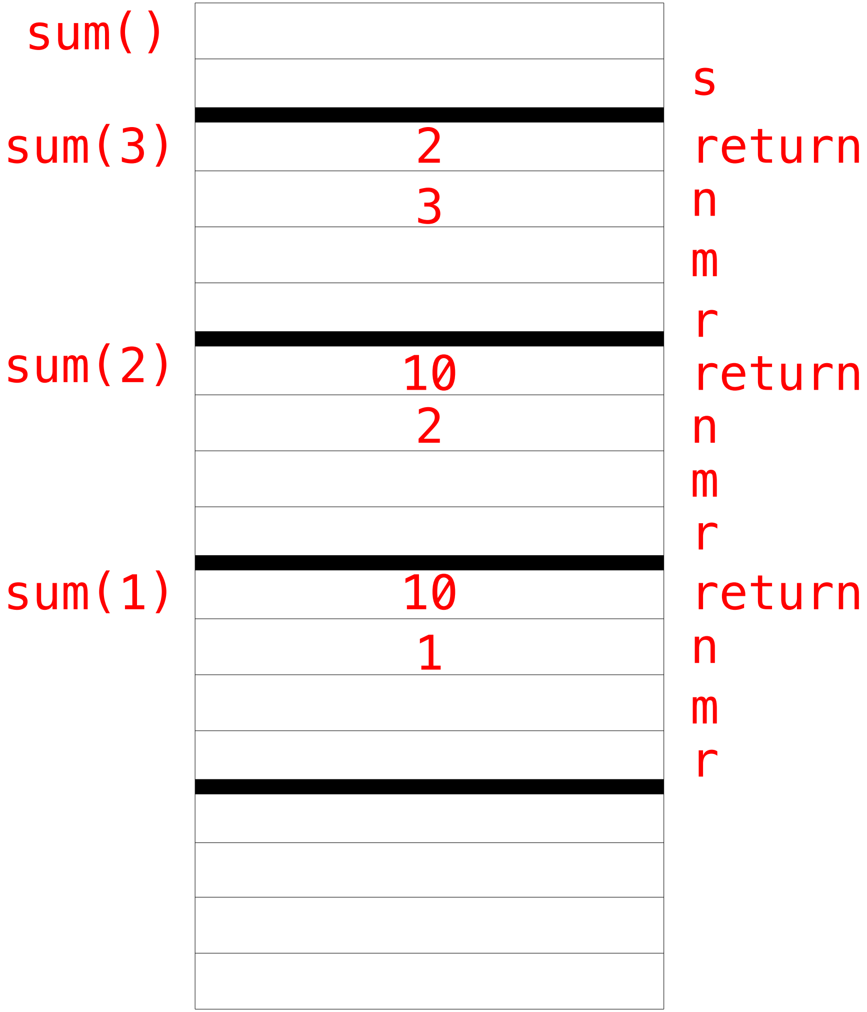
```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



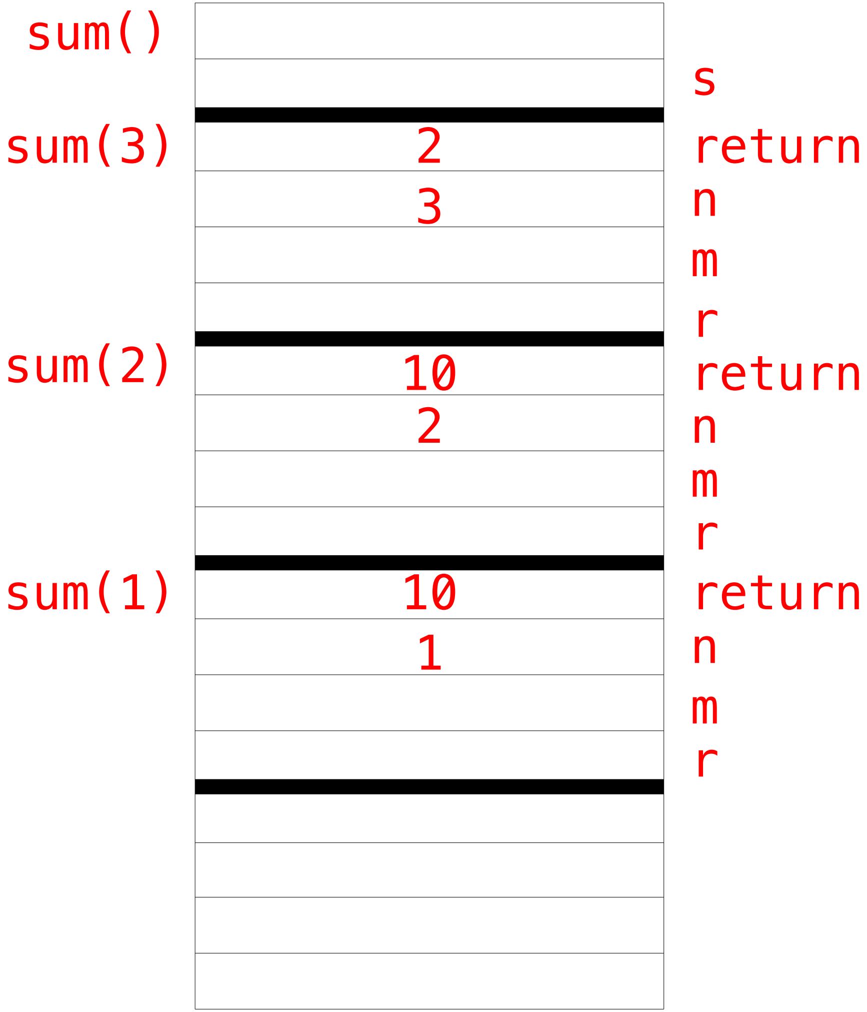
```
   static int sum() {
     int s = sum(3);
     return s;
   }

>> static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
>>   if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

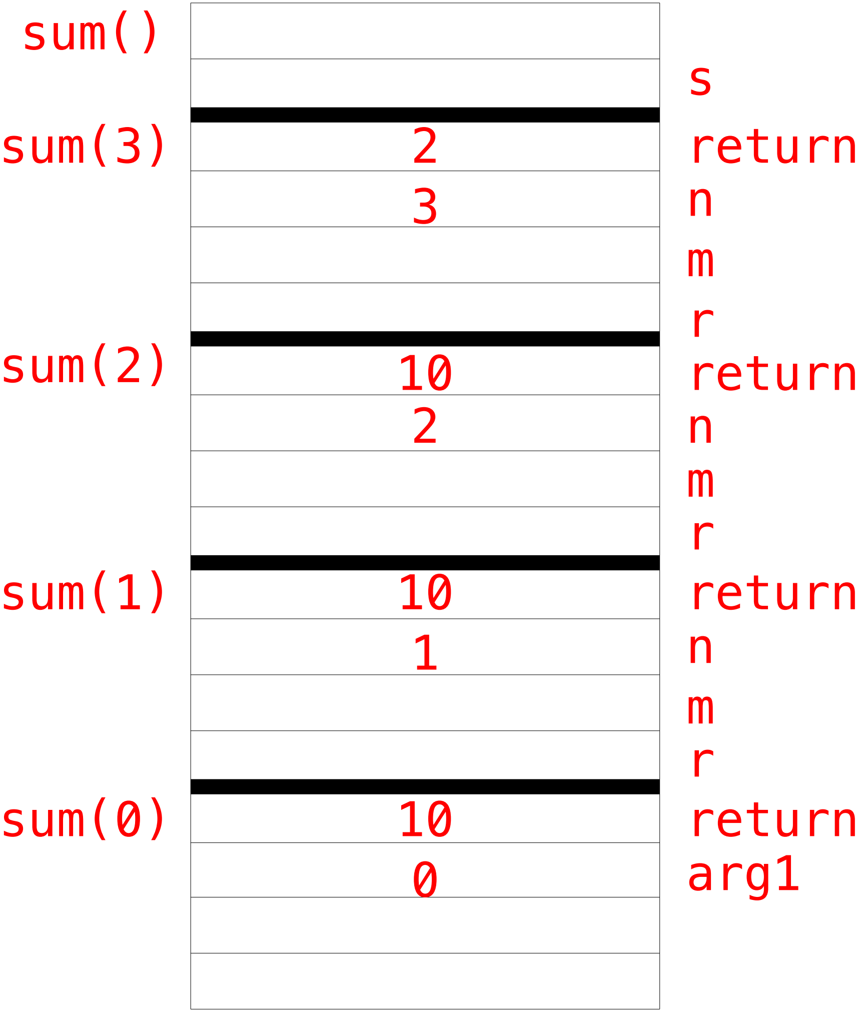


```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

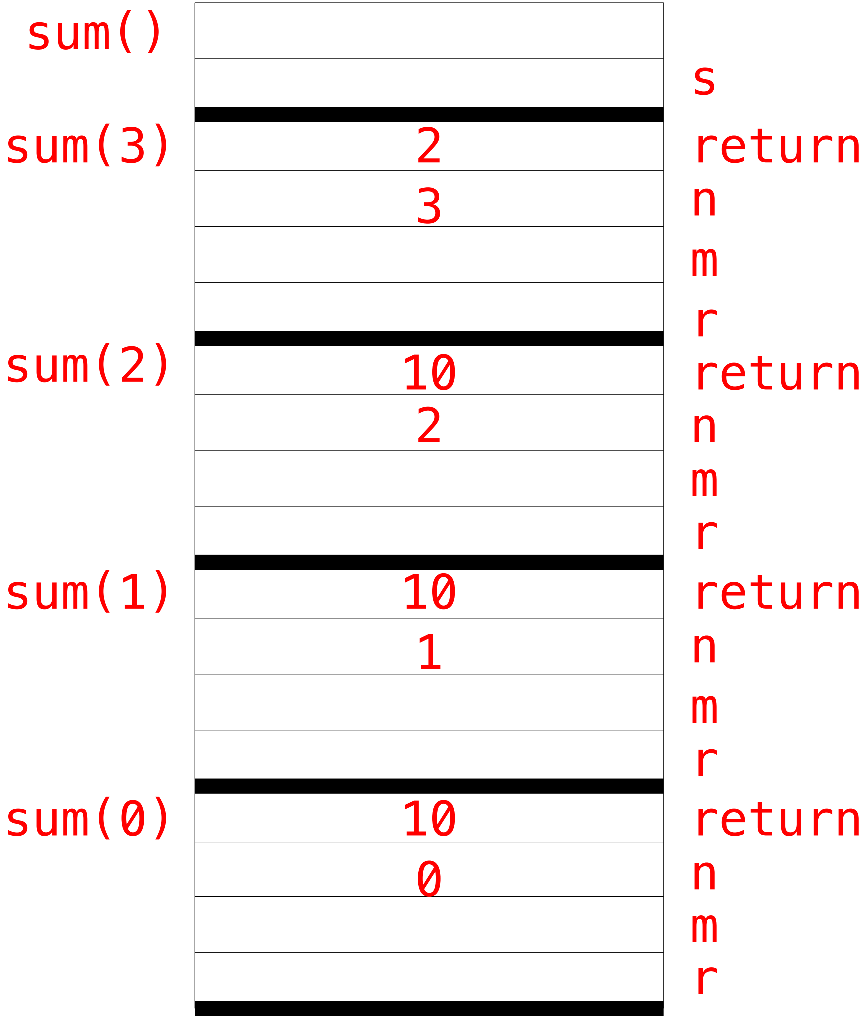


sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	arg1

```
   static int sum() {
     int s = sum(3);
     return s;
   }

>> static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	n
		m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
>>   if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

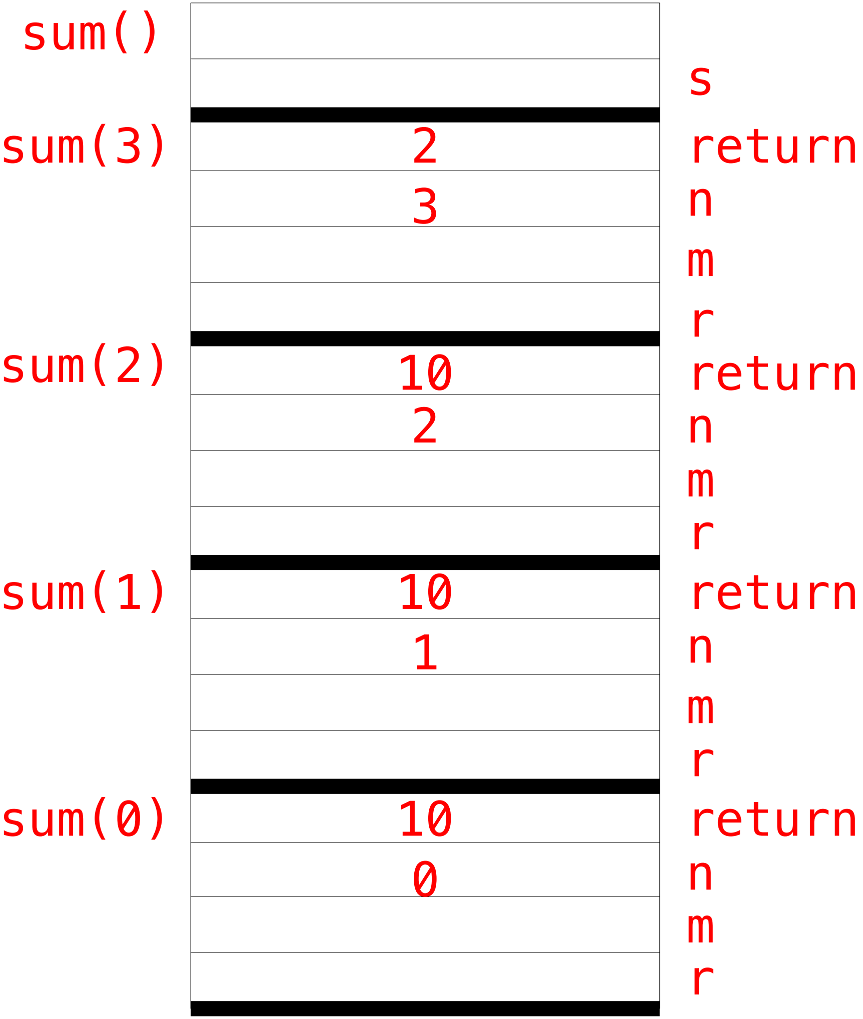


sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	n
		m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
>>     return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	n
		m
		r

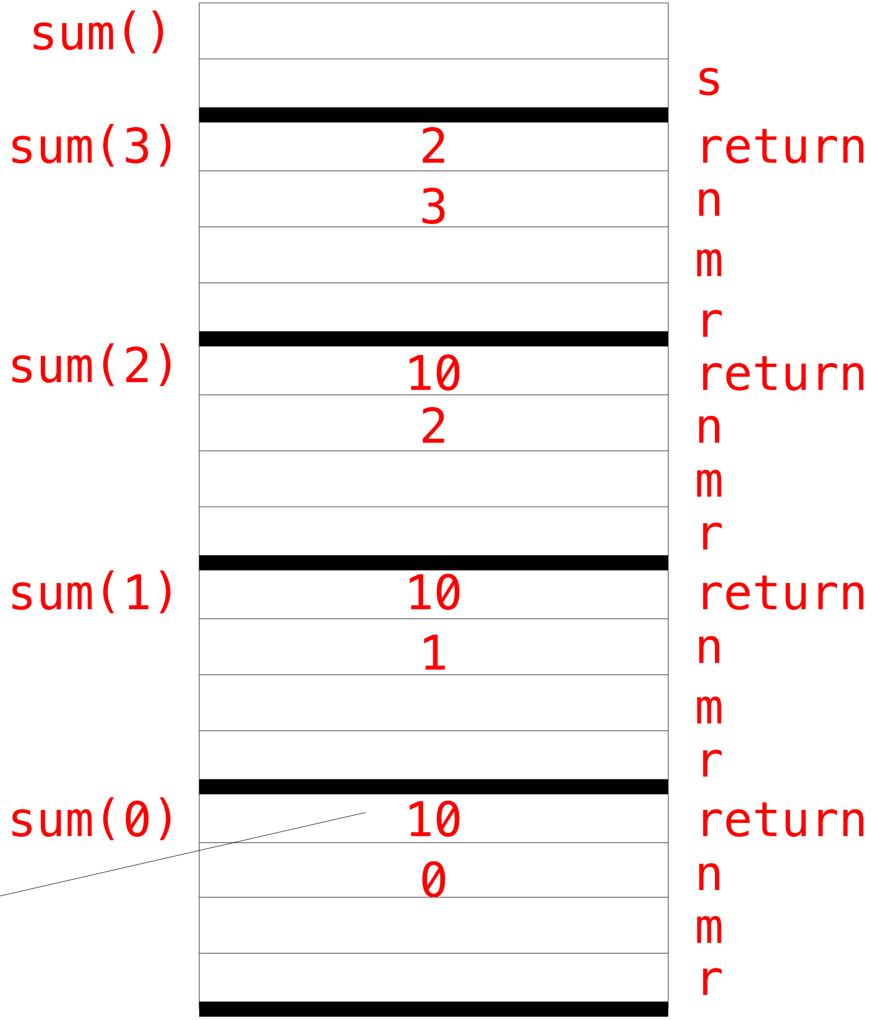
```
static int sum() {
  int s = sum(3);
  return s;
}

static int sum(int n) {
  if (n == 0) {
>>  return 0;
  }
  int m = sum(n - 1);
  int r = m + n;
  return r;
}
```



sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	n
		m
		r

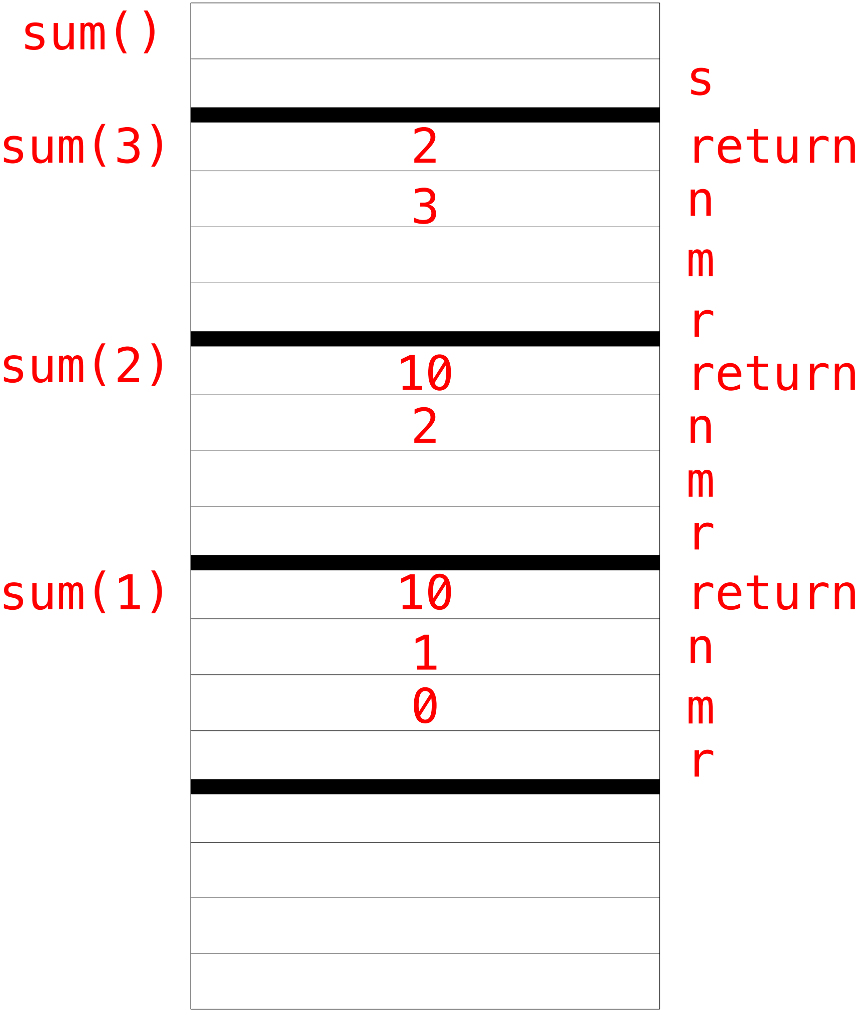
```
   1  static int sum() {
   2    int s = sum(3);
   3    return s;
   4  }
   5
   6  static int sum(int n) {
   7    if (n == 0) {
>> 8      return 0;
   9    }
  10    int m = sum(n - 1);
  11    int r = m + n;
  12    return r;
  13  }
```

Return the value 0 and
then execute instruction 10



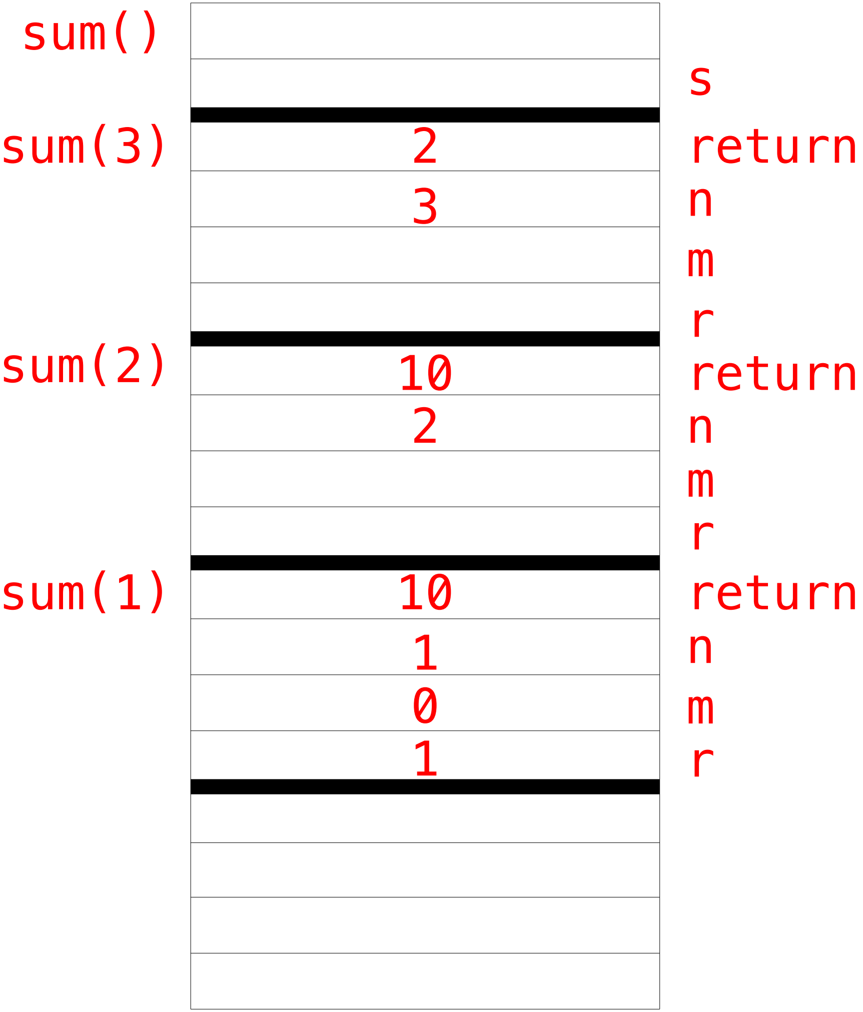
sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
		m
		r
sum(0)	10	return
	0	n
		m
		r

```java
static int sum() {
  int s = sum(3);
  return s;
}

static int sum(int n) {
  if (n == 0) {
    return 0;
  }
>>  int m = sum(n - 1);
  int r = m + n;
  return r;
}
```



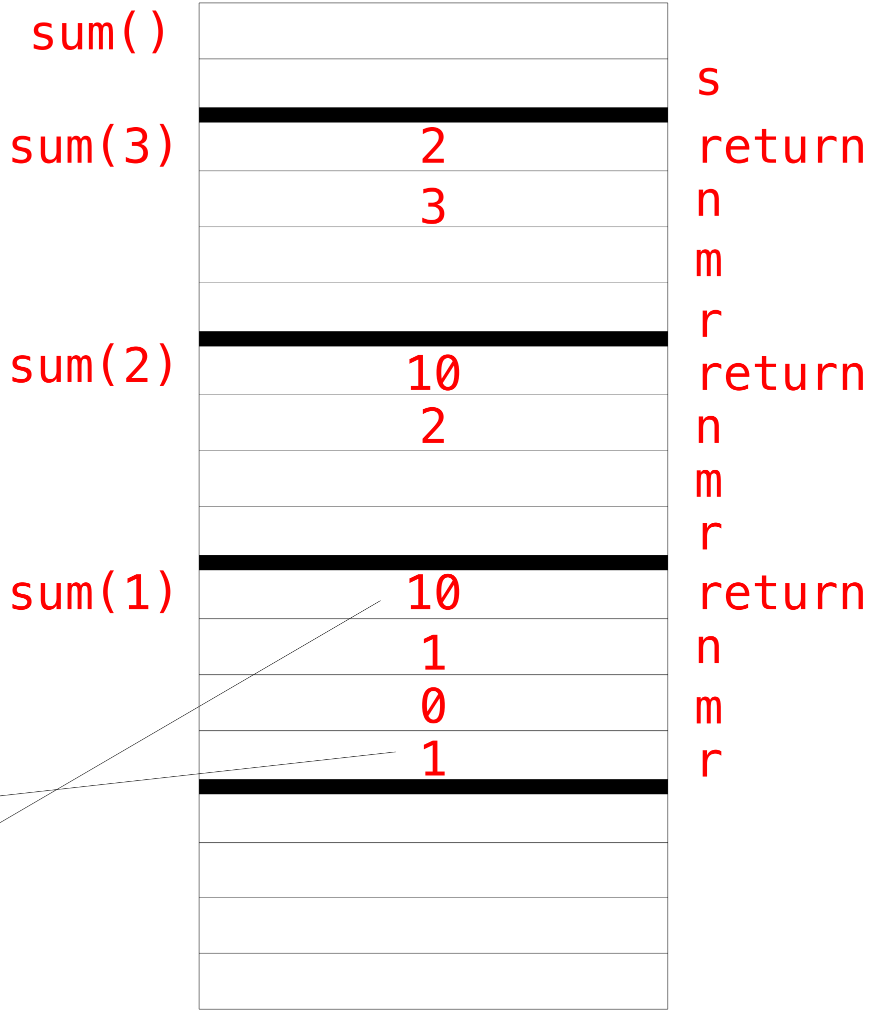
sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
	0	m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
>>   int r = m + n;
     return r;
   }
```



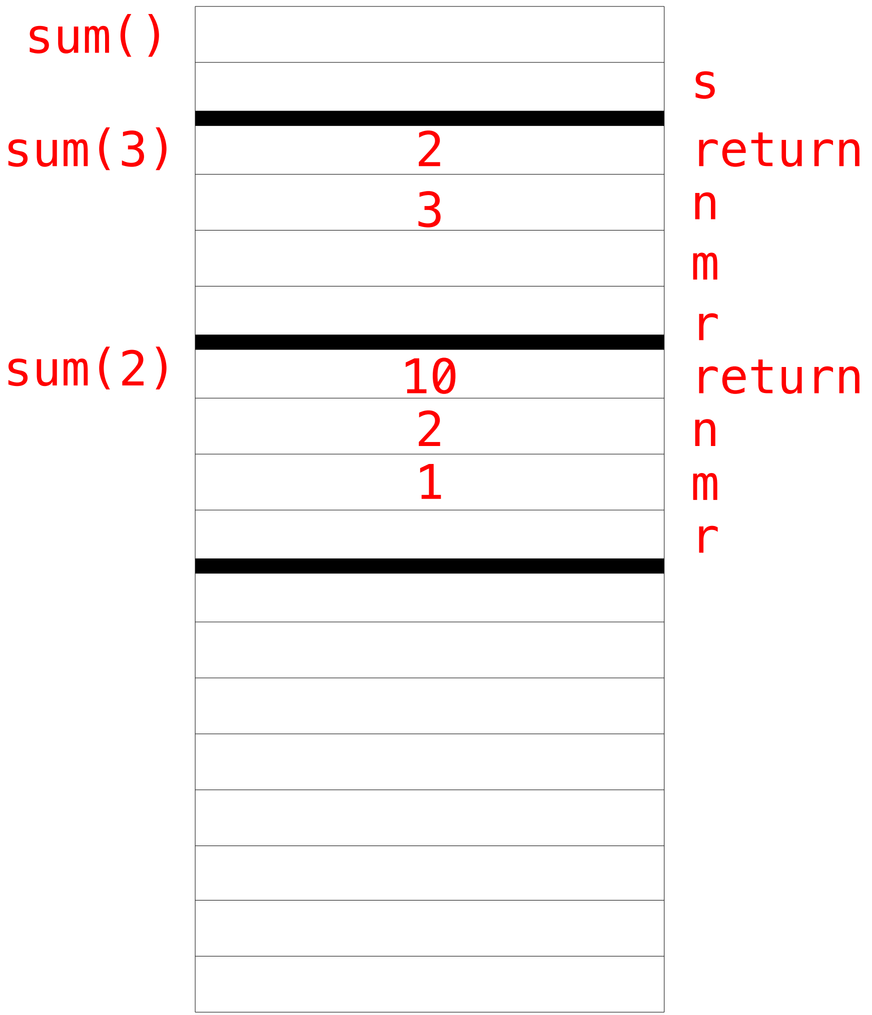
sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
		m
		r
sum(1)	10	return
	1	n
	0	m
	1	r

```
    1  static int sum() {
    2    int s = sum(3);
    3    return s;
    4  }
    5
    6  static int sum(int n) {
    7    if (n == 0) {
    8      return 0;
    9    }
   10    int m = sum(n - 1);
   11    int r = m + n;
>> 12    return r;
   13  }
```



Return the value 1 and
then execute instruction 10

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

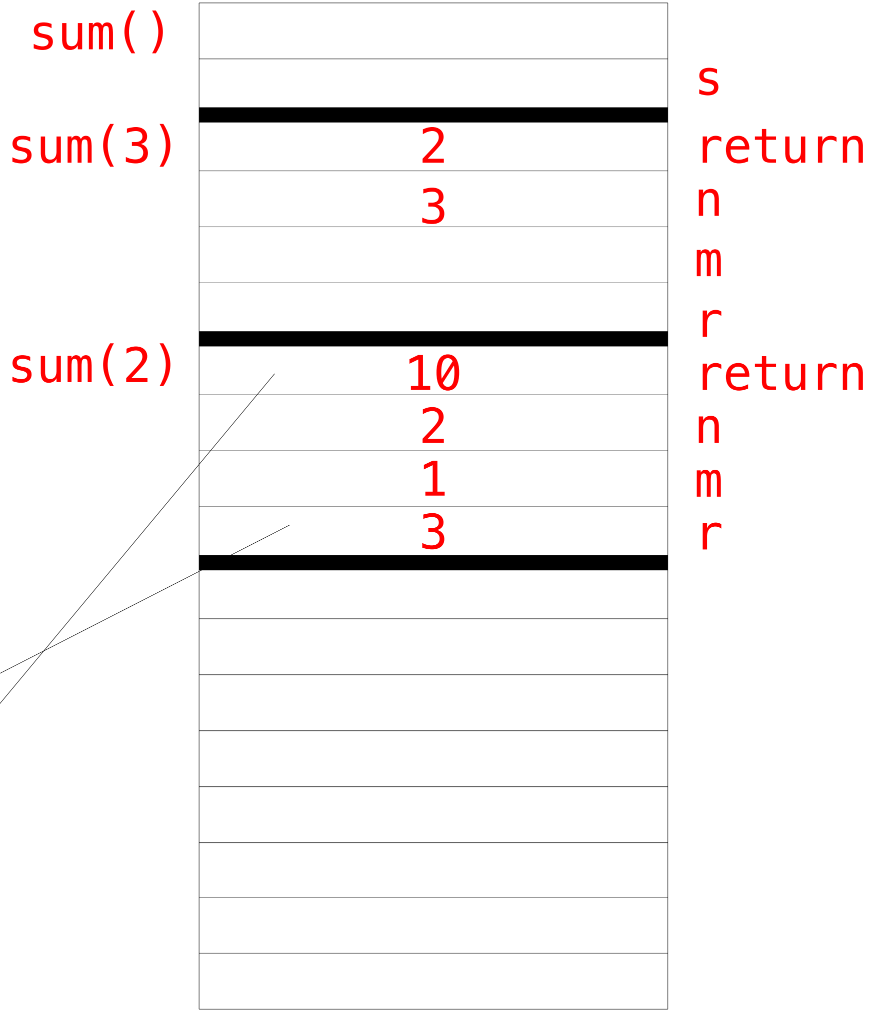


sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
	1	m
		r

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
>>   int r = m + n;
     return r;
   }
```

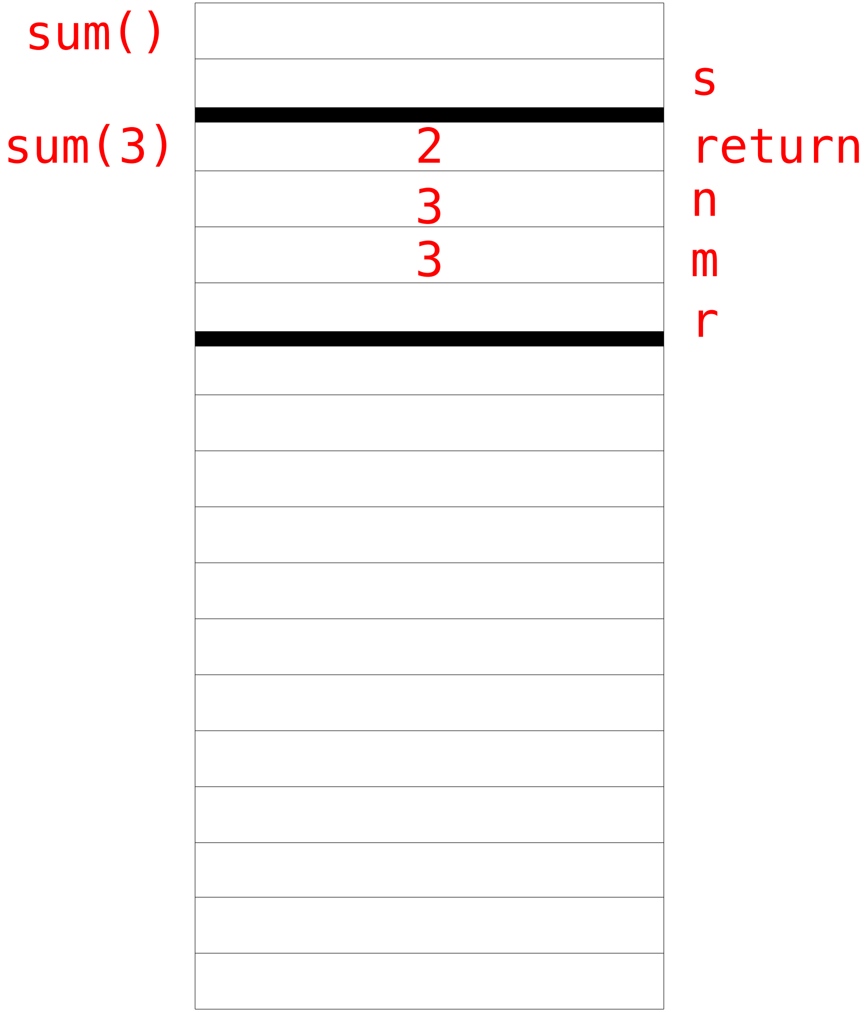
sum()		
		s
sum(3)	2	return
	3	n
		m
		r
sum(2)	10	return
	2	n
	1	m
	3	r

```
   1  static int sum() {
   2    int s = sum(3);
   3    return s;
   4  }
   5
   6  static int sum(int n) {
   7    if (n == 0) {
   8      return 0;
   9    }
  10    int m = sum(n - 1);
  11    int r = m + n;
>> 12    return r;
  13  }
```

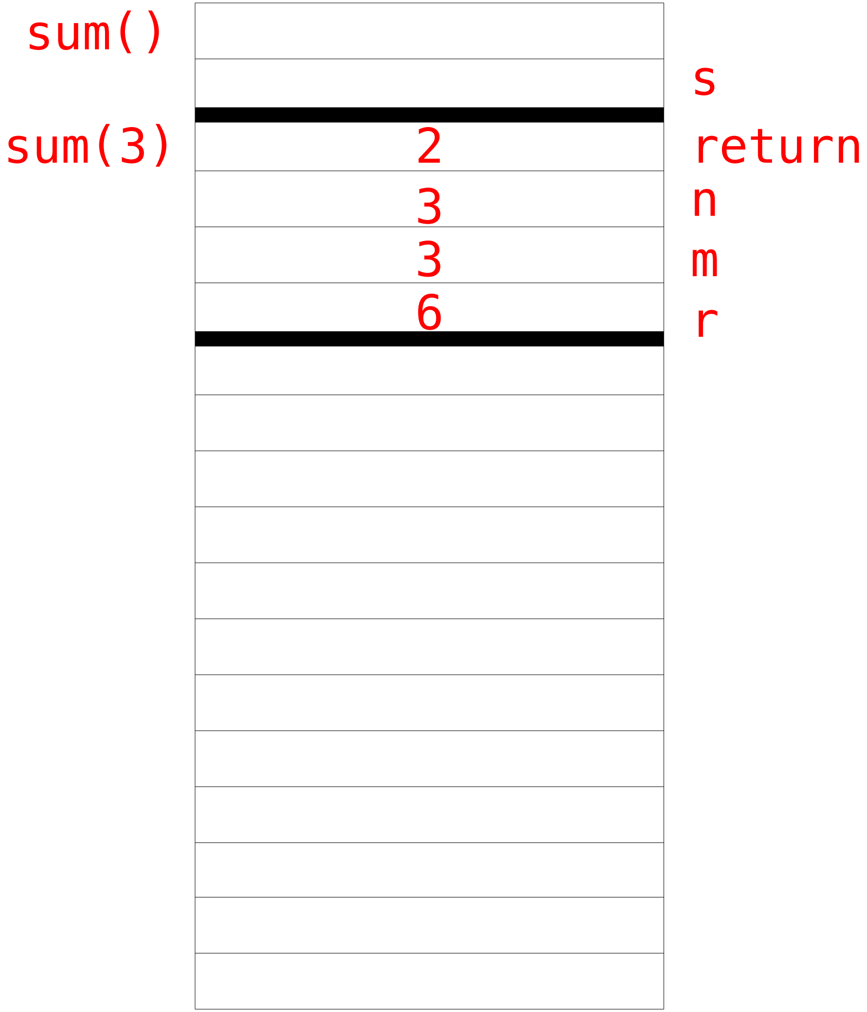


Return the value 3 and
then execute instruction 10

```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
>>   int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```

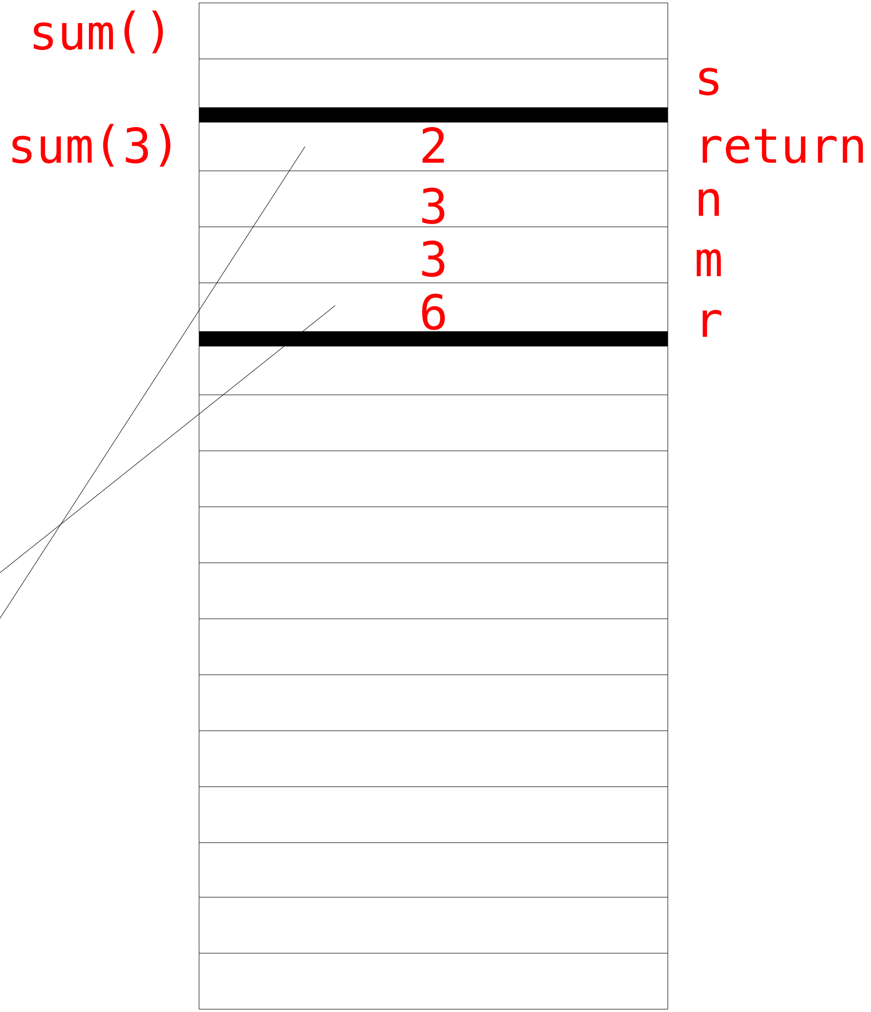


```
   static int sum() {
     int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
>>   int r = m + n;
     return r;
   }
```



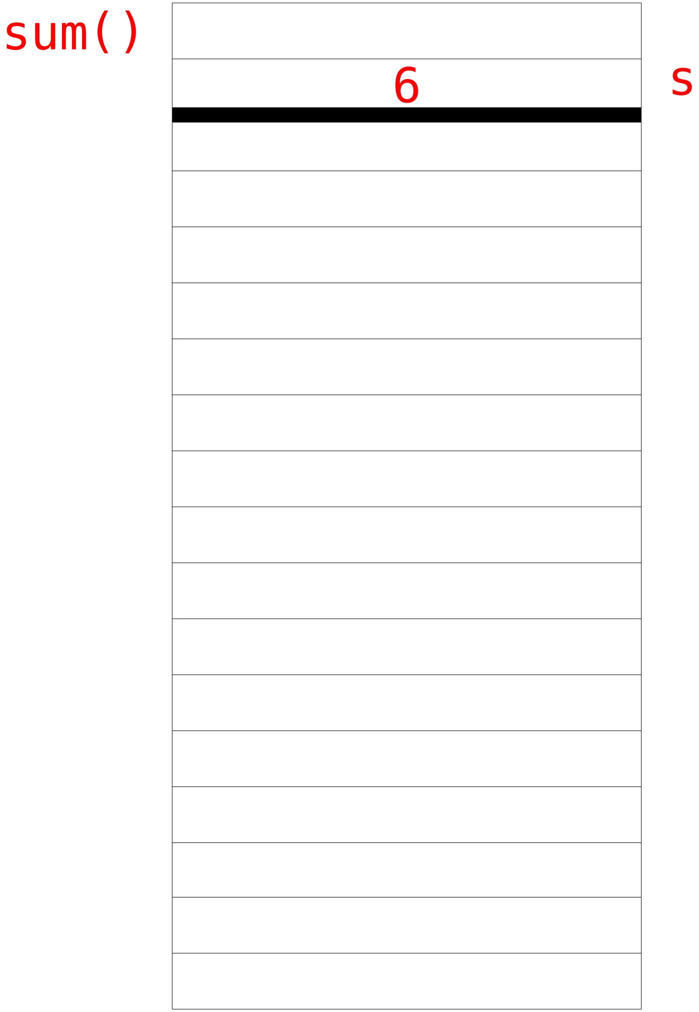
sum()		
		s
sum(3)	2	return
	3	n
	3	m
	6	r

```
    1  static int sum() {
    2    int s = sum(3);
    3    return s;
    4  }
    5
    6  static int sum(int n) {
    7    if (n == 0) {
    8      return 0;
    9    }
   10    int m = sum(n - 1);
   11    int r = m + n;
>> 12    return r;
   13  }
```

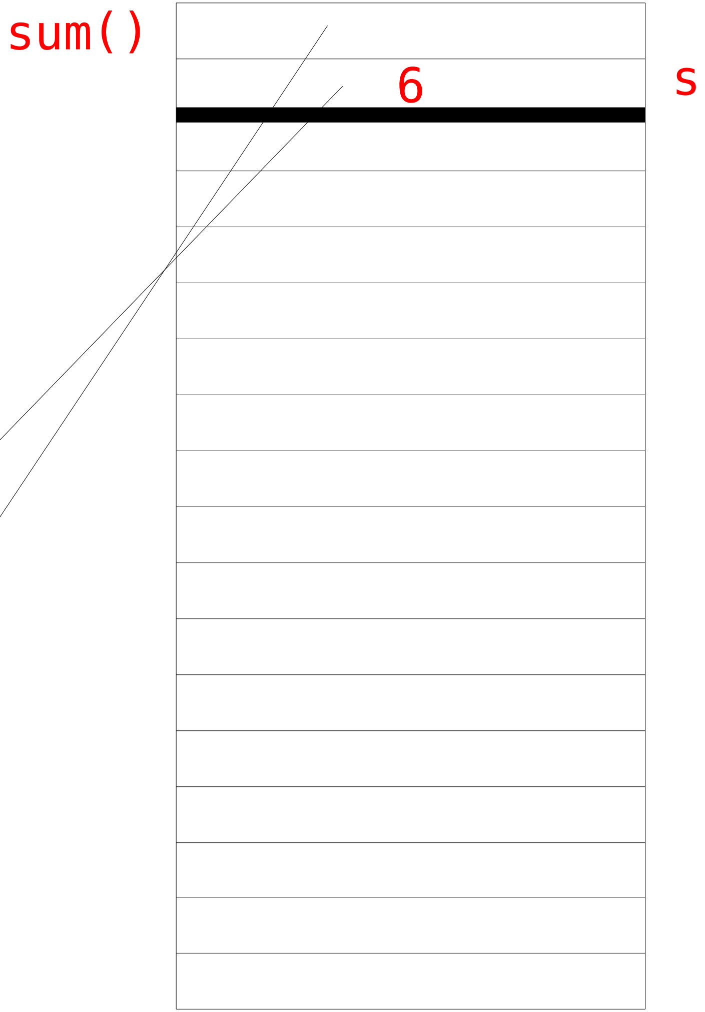


Return the value 6 and
then execute instruction 2

```
   static int sum() {
>>   int s = sum(3);
     return s;
   }

   static int sum(int n) {
     if (n == 0) {
       return 0;
     }
     int m = sum(n - 1);
     int r = m + n;
     return r;
   }
```



```
    1  static int sum() {
    2    int s = sum(3);
>>  3    return s;
    4  }
    5
    6  static int sum(int n) {
    7    if (n == 0) {
    8      return 0;
    9    }
   10    int m = sum(n - 1);
   11    int r = m + n;
   12    return r;
   13  }
```

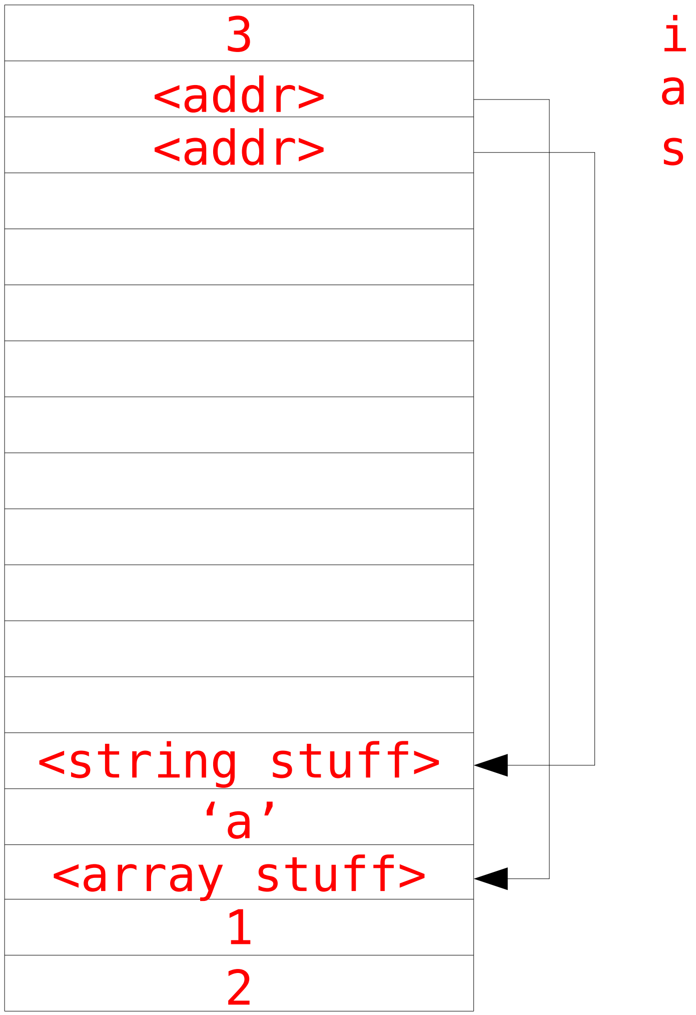


Return the value 6 and then execute whatever called us

In Java primitive types go on the stack

Everything else goes on the heap

```java
static void test() {
    int i = 3;
    int[] a = new int[] {1,2};
    String s = "a";
}
```



3

<addr>

<addr>

<string stuff>

‘a’

<array stuff>

1

2

Java delete’s for us automatically. This is called Garbage Collection

This example is in Java

```java
static void test() {
    int i = 3;
    int[] a = new int[] {1,2};
    String s = "a";
}
```

‘a’ and ‘s’ are references. These are like pointers but you can’t do arithmetic on them.

When you say s.toUpperCase() you are ‘dereferencing’ s and calling the method toUpperCase on it.

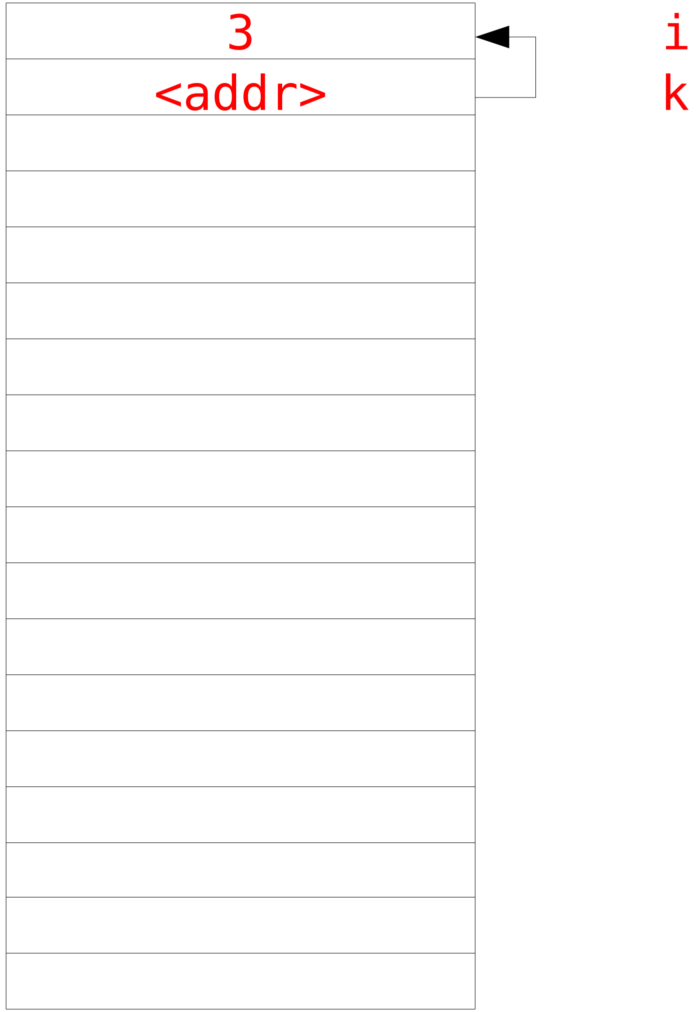
References in C++ are a completely different concept!

```
    static void test() {
>>      int i = 3;
        int* k = &i;
        int& j = i;
    }
```

3	i

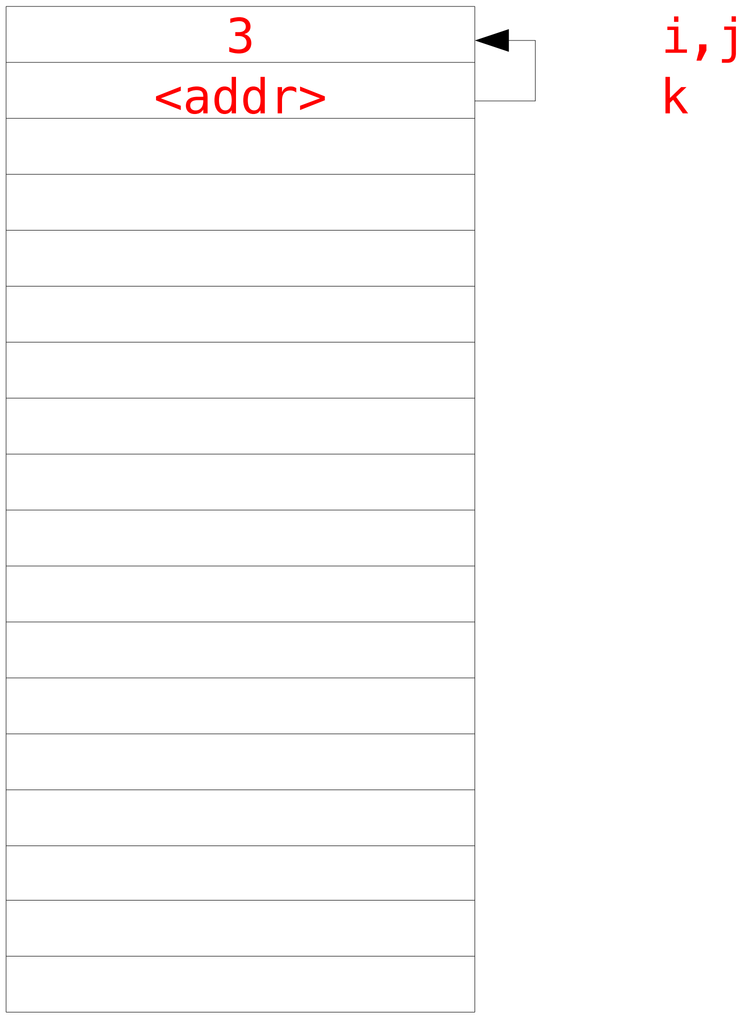
This example is in C++

```
   static void test() {
       int i = 3;
>>     int∗ k = &i;
       int& j = i;
   }
```



```
    static void test() {
        int i = 3;
        int* k = &i;
>>      int& j = i;
    }
```

& on the LHS means ‘reference’



Recap for Java

- Primitive types on the stack
- Everything else on the heap
- References are values on the stack that ‘point’ to somewhere on the heap
- References are like pointers but you can’t do artithmetic on them
- Java references are not much like C++ references