Functions Revisited

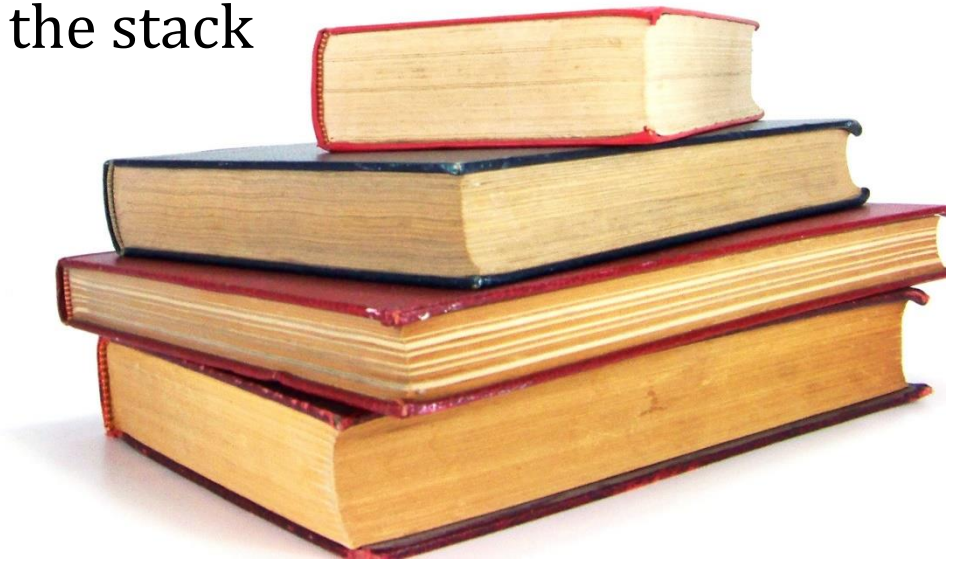
Function Internals

Activation Record

- When a function is invoked, an “activation record” is created
- Activation records hold:
  - Location of the invocation
  - Function being invoked
  - Copies of the argument values for this activation
  - Space for automatic variable values
  - Space for a return value
- After the function returns, the activation record is deleted

Activation Records are kept in a stack

- "Current" function on top of stack
- When a function is invoked,
  - add its activation record to the top of the stack
- After a function returns,
  - remove it's activation record from the top of the stack



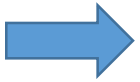
Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
OS	main		a=	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

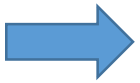


Inv	Fn	args	vars	Ret
3	addem	x=3,y=4		

Inv	Fn	args	vars	Ret
OS	main		a=	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```



Inv	Fn	args	vars	Ret
3	addem	x=3,y=4		7

Inv	Fn	args	vars	Ret
OS	main		a=	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
OS	main		a=7	

Example Call Stack

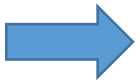
```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
4	addem	x=7,y=4		

Inv	Fn	args	vars	Ret
OS	main		a=7	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```



Inv	Fn	args	vars	Ret
4	addem	x=7,y=4		11

Inv	Fn	args	vars	Ret
OS	main		a=7	

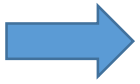
Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
OS	main		a=11	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```



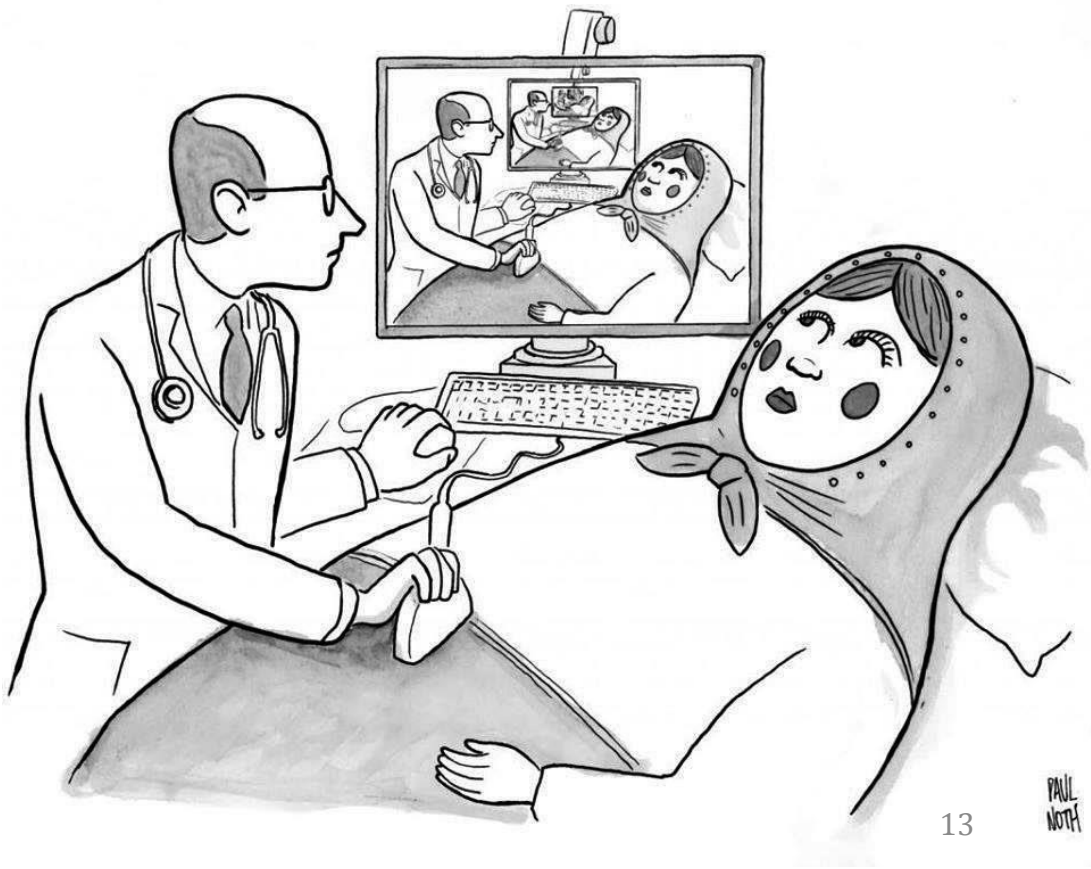
Inv	Fn	args	vars	Ret
OS	main		a=11	0

Notes on Call Stacks

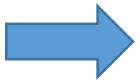
- Arguments are passed by value
  - Can't update callers value!
- GDB "where" command prints call stack
- Automatic local variables useful
  - Don't need to worry about caller's environment!
  - Don't need to worry about previous invocations
  - Enables "recursion"

Recursive Function

- A “recursive” function is a function which calls itself
- For example, “factorial”
  - fact(1)=1
  - fact(n)=n x fact(n-1) for n>1
- For example:
  - fact(2) = 2 x fact(1) = 2 x 1 = 2
  - fact(3) = 3 x fact(2) = 3 x 2 = 6
  - fact(4) = 4 x fact(3) = 4 x 6 = 24
  - …



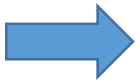
Factorials in C



```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C



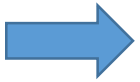
```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



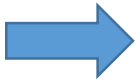
Inv	Fn	args	vars	Ret
8	fact	3		

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



Inv	Fn	args	vars	Ret
8	fact	2		

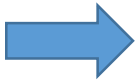
Inv	Fn	args	vars	Ret
8	fact	3		

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



Inv	Fn	args	vars	Ret
8	fact	1		

Inv	Fn	args	vars	Ret
8	fact	2		

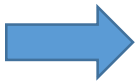
Inv	Fn	args	vars	Ret
8	fact	3		

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



Inv	Fn	args	vars	Ret
8	fact	1		1

Inv	Fn	args	vars	Ret
8	fact	2		

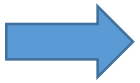
Inv	Fn	args	vars	Ret
8	fact	3		

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



Inv	Fn	args	vars	Ret
8	fact	2		2

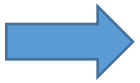
Inv	Fn	args	vars	Ret
8	fact	3		

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



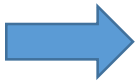
Inv	Fn	args	vars	Ret
8	fact	3		6

Inv	Fn	args	vars	Ret
3	fact	4		

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```



Inv	Fn	args	vars	Ret
3	fact	4		24

Inv	Fn	args	vars	Ret
OS	main		a	

Factorials in C

```
1. int fact(int x);
2. int main() {
3.     int a = fact(4);
4.     return 0;
5. }
6. int fact(int x) {
7.     if (x==1) return 1;
8.     return x*fact(x-1);
9. }
```

Inv	Fn	args	vars	Ret
OS	main		a=24	

Resources

- Programming in C, Chapter 7
- Wikipedia: Call Stack (https://en.wikipedia.org/wiki/Call_stack)
- Wikipedia: Recursion (computer science) (https://en.wikipedia.org/wiki/Recursion_(computer_science))