ECS30: random numbers, user-defined functions

...

Dr. Rob Gysel
1/18/17

Random Number Generation

- Random number generator `rand`
  - Not actually random: `rand` gets next number (between 0 and `RAND_MAX`) in a fixed sequence

`notrandom.c`

Random Number Generation

- Random number generator `rand`
  - Not actually random: `rand` gets next number (between 0 and `RAND_MAX`) in a fixed sequence
  - Sequence is determined by *seed value* sent to `srand`
  - Seed by `time` to get truly random numbers

`random.c`

Random Number Generation

Specifying the range of random numbers

1. Seed with `srand` and `time`
2. Use `rand` to generate numbers
3. Use mod to restrict range, e.g. numbers between 0 and 9:

   ```
   rand() % 10
   ```

   Between 1 and 10:

   ```
   1 + rand() % 10
   ```

Function prototypes

- Usage:
  - `double z = pow(x, y);`
  - `z` is `x^y` after this statement
- Function prototype:
  - `double pow(double base, double exponent);`

Function prototypes

- Tells compiler how function is defined later
- Reminds programmers how to use a function

Also called *type signatures*

Function prototypes

- Usage:
  - `double z = pow(x, y);`
  - `z` is `x^y` after this statement
- Function prototype:
  - `double pow(double base, double exponent);`

function name:

identifier used to call the function

Function prototypes

- Usage:
  - `double z = pow(x, y);`
  - `z` is `x^y` after this statement
- Function prototype:
  - `double pow(double base, double exponent);`

Return type: type of variable returned

(`void` if nothing is returned)

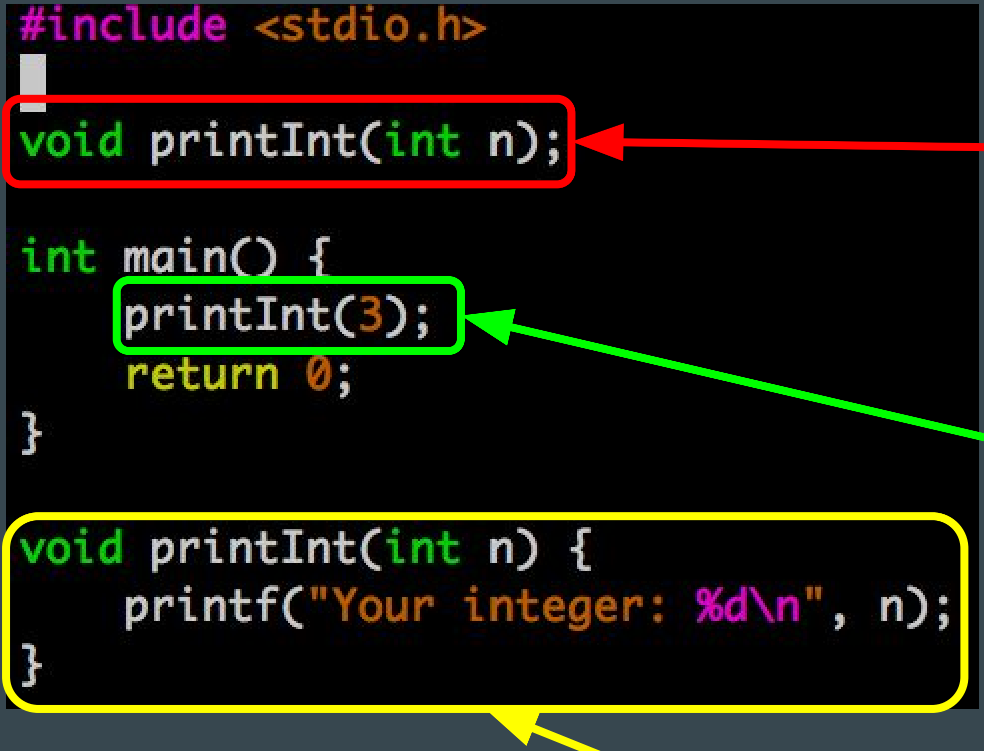
Function prototypes

- Usage:
  - `double z = pow(x, y);`
  - `z` is `x^y` after this statement
- Function prototype:
  - `double pow(double base, double exponent);`

arguments

Function *parameters* values given by arguments

Function prototypes `functionPrototype.c`



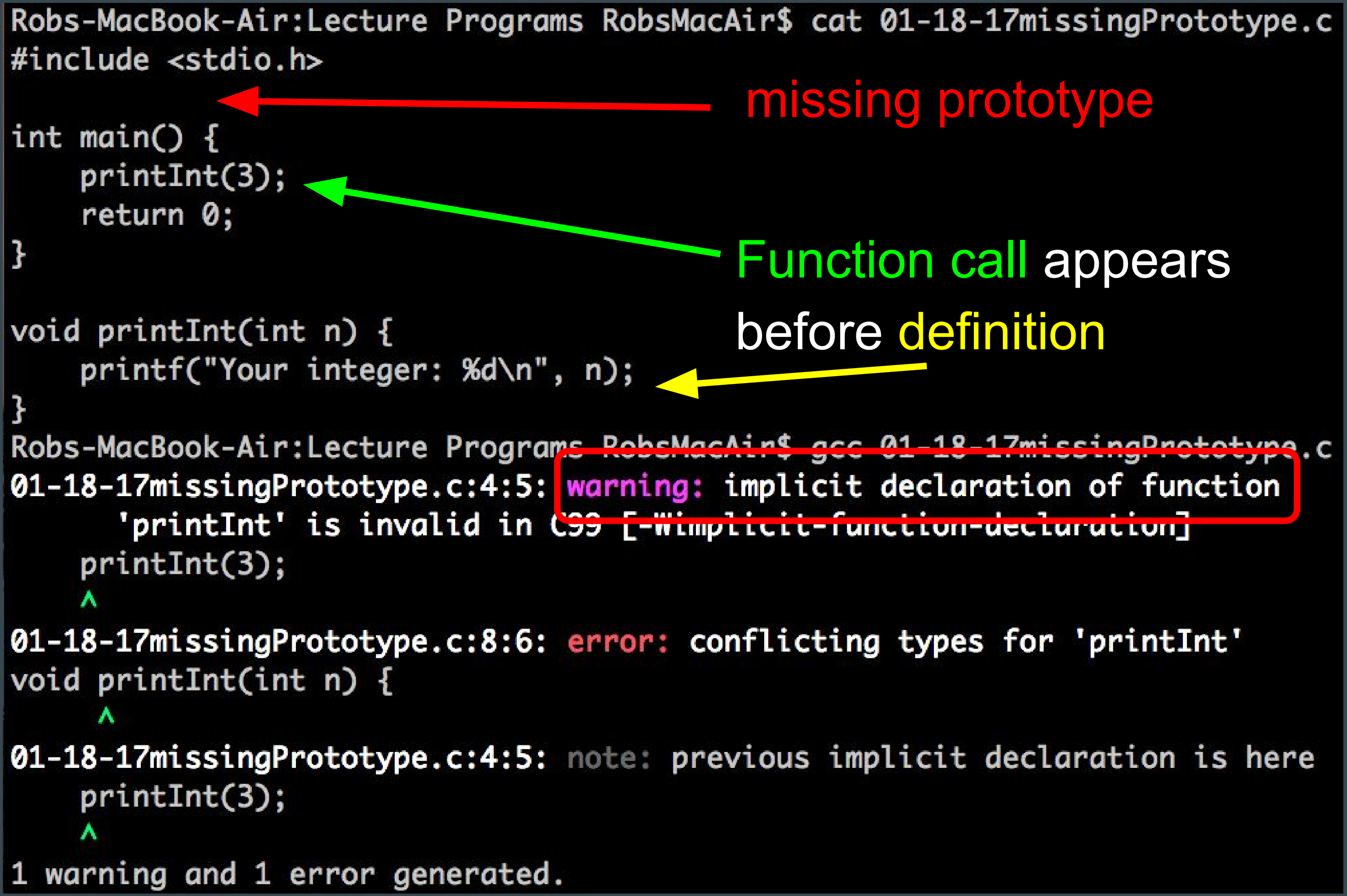
```
#include <stdio.h>

void printInt(int n);

int main() {
    printInt(3);
    return 0;
}

void printInt(int n) {
    printf("Your integer: %d\n", n);
}
```

Function prototype

Function call

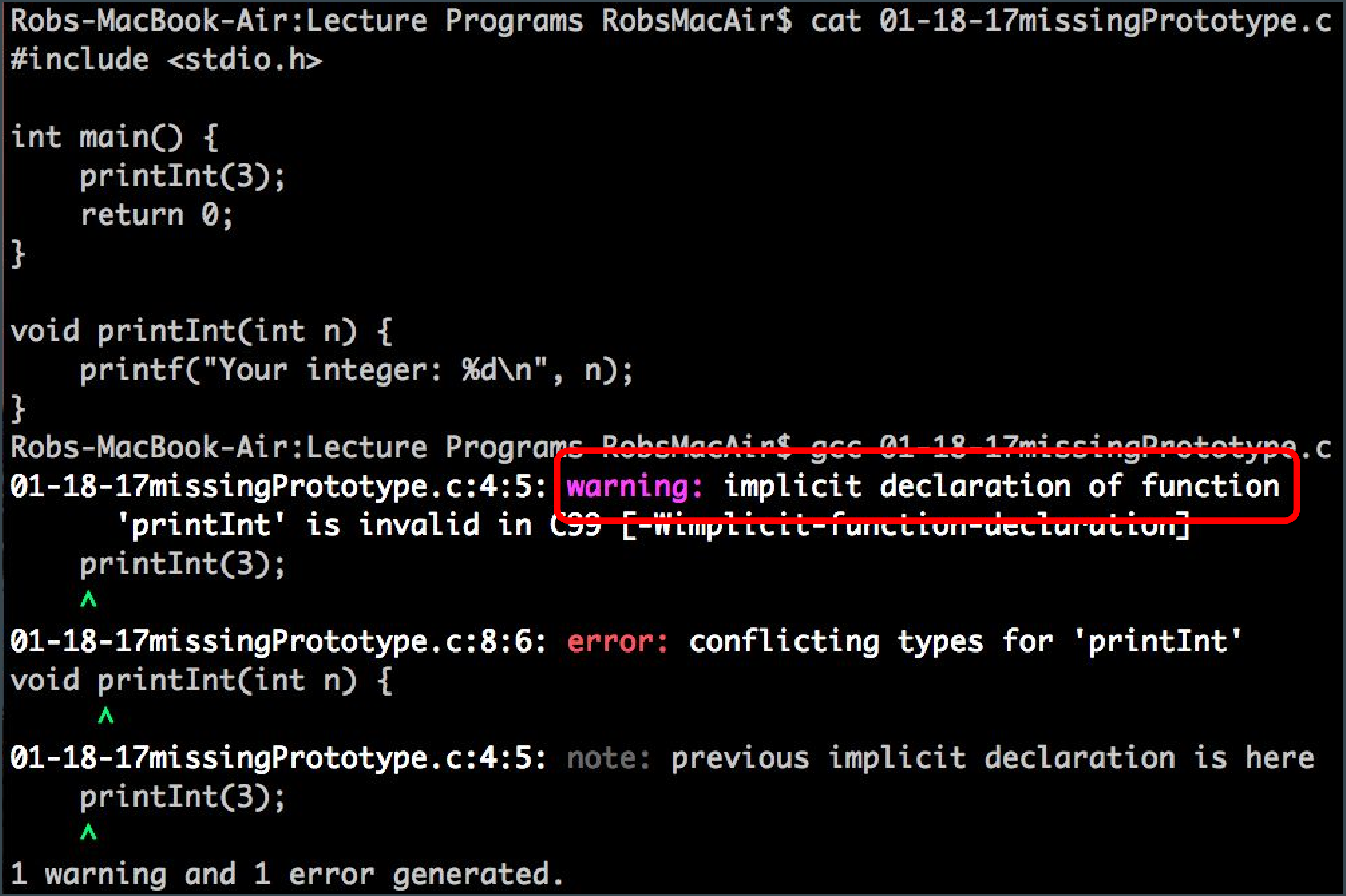
Function definition

Function prototypes `missingPrototype.c`



```
Robs-MacBook-Air:Lecture Programs RobsMacAir$ cat 01-18-17missingPrototype.c
#include <stdio.h>

int main() {
    printInt(3);
    return 0;
}

void printInt(int n) {
    printf("Your integer: %d\n", n);
}
Robs-MacBook-Air:Lecture Programs RobsMacAir$ gcc 01-18-17missingPrototype.c
01-18-17missingPrototype.c:4:5: warning: implicit declaration of function
      'printInt' is invalid in C99 [-Wimplicit-function-declaration]
    printInt(3);
    ^
01-18-17missingPrototype.c:8:6: error: conflicting types for 'printInt'
void printInt(int n) {
     ^
01-18-17missingPrototype.c:4:5: note: previous implicit declaration is here
    printInt(3);
    ^
1 warning and 1 error generated.
```

Function prototypes `missingPrototype.c`



```
Robs-MacBook-Air:Lecture Programs RobsMacAir$ cat 01-18-17missingPrototype.c
#include <stdio.h>

int main() {
    printInt(3);
    return 0;
}

void printInt(int n) {
    printf("Your integer: %d\n", n);
}
Robs-MacBook-Air:Lecture Programs RobsMacAir$ gcc 01-18-17missingPrototype.c
01-18-17missingPrototype.c:4:5: warning: implicit declaration of function
      'printInt' is invalid in C99 [-Wimplicit-function-declaration]
    printInt(3);
    ^
01-18-17missingPrototype.c:8:6: error: conflicting types for 'printInt'
void printInt(int n) {
     ^
01-18-17missingPrototype.c:4:5: note: previous implicit declaration is here
    printInt(3);
    ^
1 warning and 1 error generated.
```

The `void` keyword

`foobar.c`

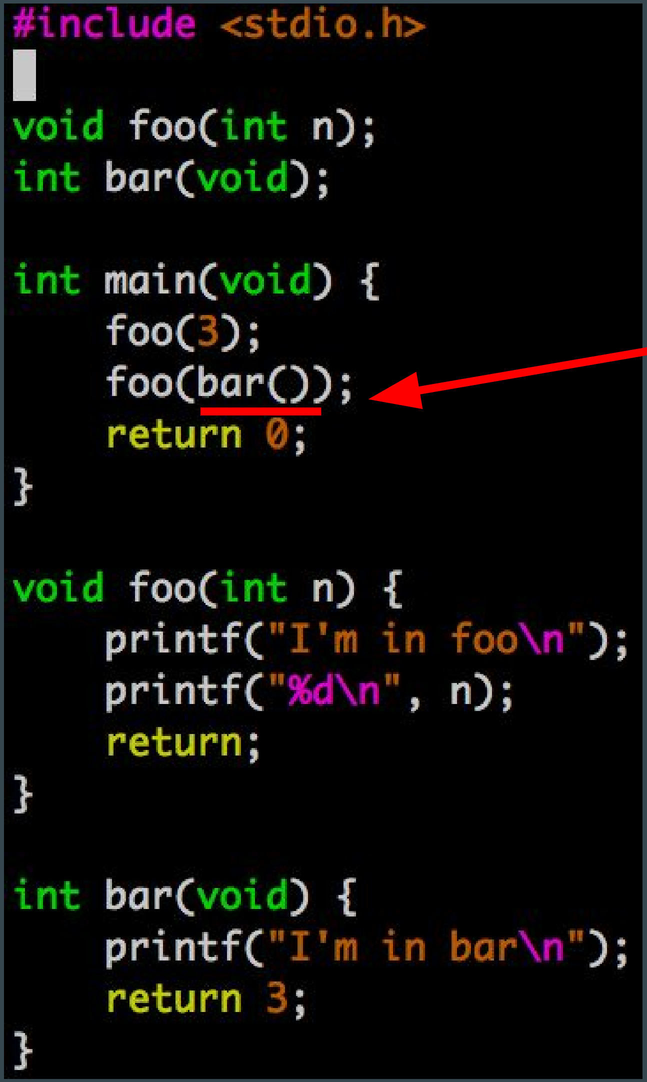
- As a return type
  - `void foo(int n);`
  - indicates foo returns nothing

The `void` keyword `foobar.c`

- As a return type
  - `void foo(int n);`
  - indicates foo returns nothing
- As a parameter type
  - `int bar(void);`
  - indicates bar does not take parameters (optional)

Composing Functions

foobar.c



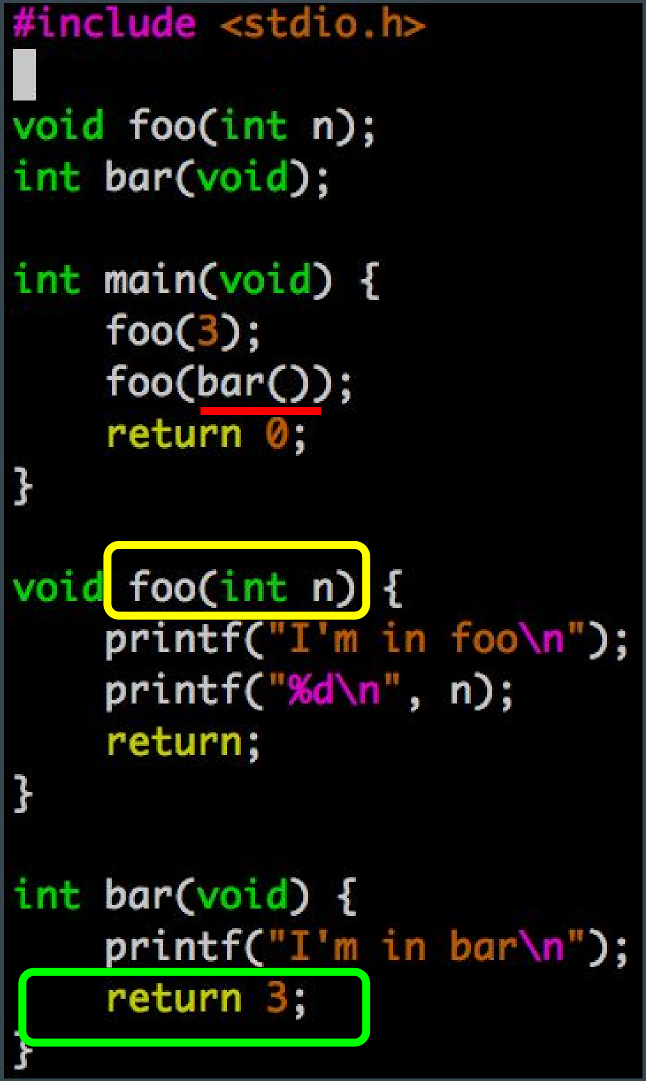
```
#include <stdio.h>

void foo(int n);
int bar(void);

int main(void) {
    foo(3);
    foo(bar());
    return 0;
}

void foo(int n) {
    printf("I'm in foo\n");
    printf("%d\n", n);
    return;
}

int bar(void) {
    printf("I'm in bar\n");
    return 3;
}
```

Composition: `foo(bar())`

- Innermost function called first

Composing Functions

`foobar.c`



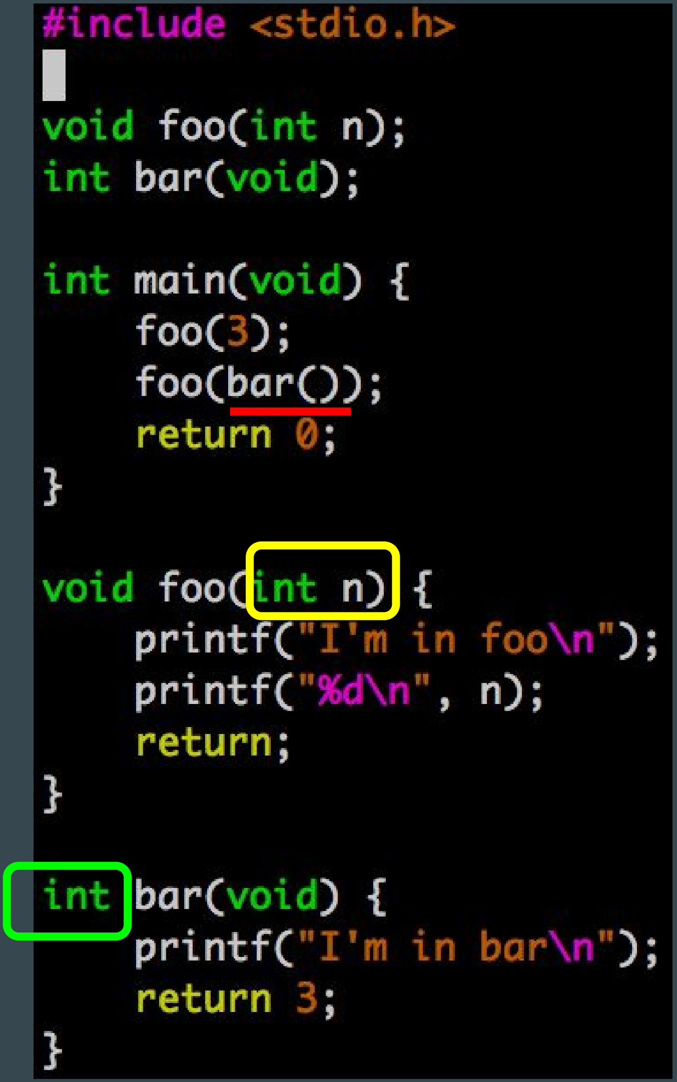
```
#include <stdio.h>

void foo(int n);
int bar(void);

int main(void) {
    foo(3);
    foo(bar());
    return 0;
}

void foo(int n) {
    printf("I'm in foo\n");
    printf("%d\n", n);
    return;
}

int bar(void) {
    printf("I'm in bar\n");
    return 3;
}
```

Composition: `foo(bar())`

- Innermost function called first
- Return value of bar passed as argument to foo

Composing Functions

`foobar.c`



```
#include <stdio.h>

void foo(int n);
int bar(void);

int main(void) {
    foo(3);
    foo(bar());
    return 0;
}

void foo(int n) {
    printf("I'm in foo\n");
    printf("%d\n", n);
    return;
}

int bar(void) {
    printf("I'm in bar\n");
    return 3;
}
```

Composition: `foo(bar())`

- Innermost function called first
- Return value of bar passed as argument to foo
- `bar`'s return type matches `foo`'s parameter type