ESC101: Introduction to Computing

f(unction)

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --
- Take photos
  - Input: --
  - Output: Picture
- Talk (we can do that too!!)
  - Input: Phone number
  - Output: Conversation (if lucky)
- ...

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.
  - Email and Chat apps can share spell checker.
  - Phone and SMS apps can share dialer
- C facilitates this using Functions

Function

- An independent, self-contained entity of a C program that performs a well-defined task.
- It has
  - Name: for identification
  - Arguments: to pass information from outside world (rest of the program)
  - Body: processes the arguments do something useful
  - Return value: To communicate back to outside world
    - Sometimes not required

Why use functions?

Example : Maximum of 3 numbers

```
int main(){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */

   return 0;
}
```

```
int max(int a, int b){
   if (a>b)
     return a;
   else
     return b;
}

int main() {
   int a, b, c, m;

   /* code to read
    * a, b, c */

   m = max(a, b);
   m = max(m, c);
   /* print or use m */

   return 0;
}
```

This code can scale easily to handle large number of inputs (e.g.: max of 100 numbers!)

Why use functions?

- Break up complex problem into small sub-problems.
- Solve each of the sub-problems separately as a function, and combine them together in another function.
- The main tool in C for modular programming.

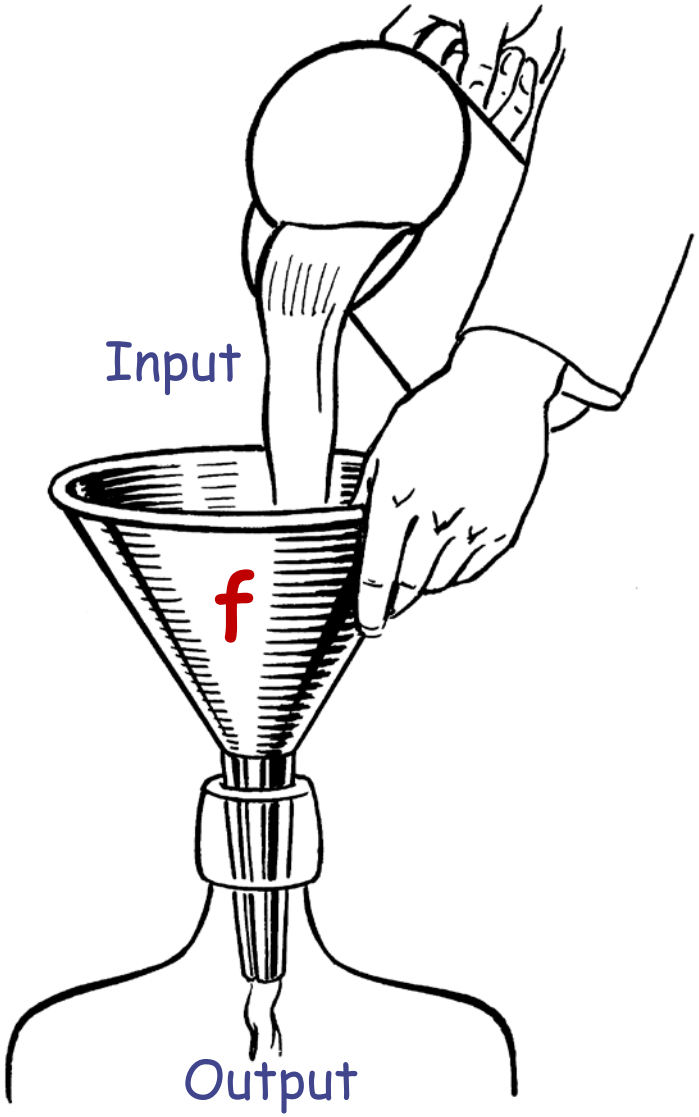
Advantages of using functions

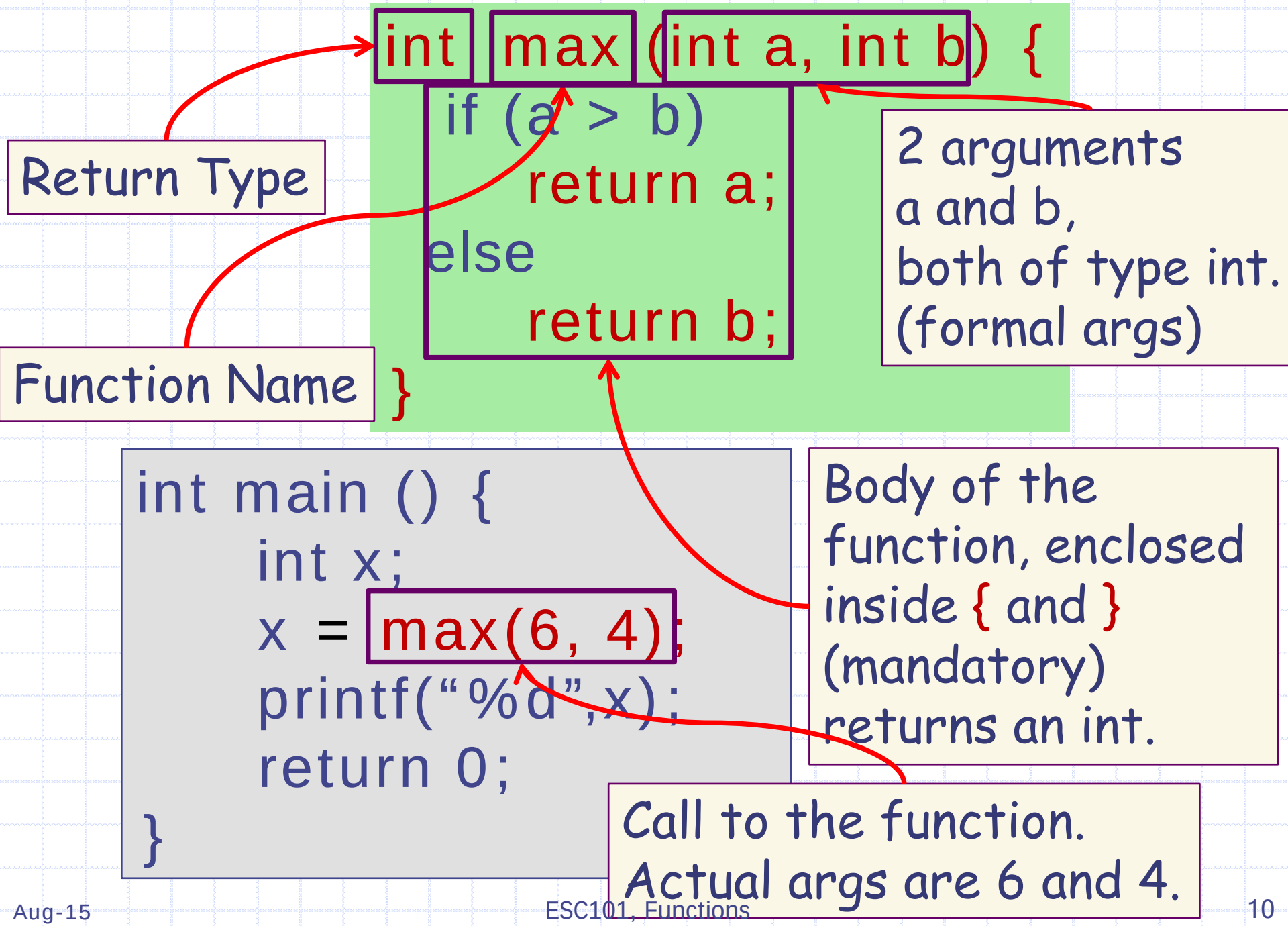
- **Code Reuse**: Allows us to reuse a piece of code as many times as we want, without having to write it.
  - Think of the `printf` function!
- **Procedural Abstraction**: Different pieces of your algorithm can be implemented using different functions.
- **Distribution of Tasks**: A large project can be broken into components and distributed to multiple people.
- **Easier to debug**: If your task is divided into smaller subtasks, it is easier to find errors.
- **Easier to understand**: Code is better organized and hence easier for an outsider to understand it.

We have seen functions before

- main() is a special function. Execution of program starts from the beginning of main().
- scanf(...), printf(...) are standard input-output library functions.
- sqrt(...), pow(...) are math functions in math.h

Parts of a function




int max (int a, int b) {
if (a > b)
return a;
else
return b;
}
Return Type
Function Name
2 arguments
a and b,
both of type int.
(formal args)
int main () {
int x;
x = max(6, 4);
printf("%d",x);
return 0;
}
Body of the
function, enclosed
inside { and }
(mandatory)
returns an int.
Call to the function.
Actual args are 6 and 4.

Function Call

- A function call is an *expression*
  - feeds the necessary values to the function arguments,
  - directs a function to perform its task, and
  - receives the return value of the function.
- Similar to operator application

`5 + 3` is an expression of type integer that evaluates to `8`

`max(5, 3)` is an expression of type integer that evaluates to `5`

Function Call

- Since a function call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

```
printf("%d", max(5,3));
max(5,3) – min(5,3)
max(x, max(y, z)) == z

if (max(a, b)) printf("Y");
```

prints 5

evaluates to 2

checks if z is max of x, y, z

prints Y if max of a and b is not 0.

Returning from a function: Type

- Return type of a function tells the type of the result of function call
- Any valid C type
  - int, char, float, double, …
  - **void**
- Return type is void if the function is not supposed to return any value

```
void print_one_int(int n) {
    printf("%d", n);
}
```

Returning from a function: return statement

- If return type is not void, then the function should return a value:

  return return_expr;

- If return type is void, the function may *fall through* at the end of the body or use a return without return_expr:

  return;

```
void print_positive(int n) {
    if (n <= 0) return;
    printf("%d", n);
}
```

Returning through return

Fall through

Returning from a function: return statement

- When a return statement is encountered in a function definition
  - control is immediately transferred back to the statement making the function call in the parent function.
- A function in C can return only ONE value or NONE.
  - Only one return type (including void)