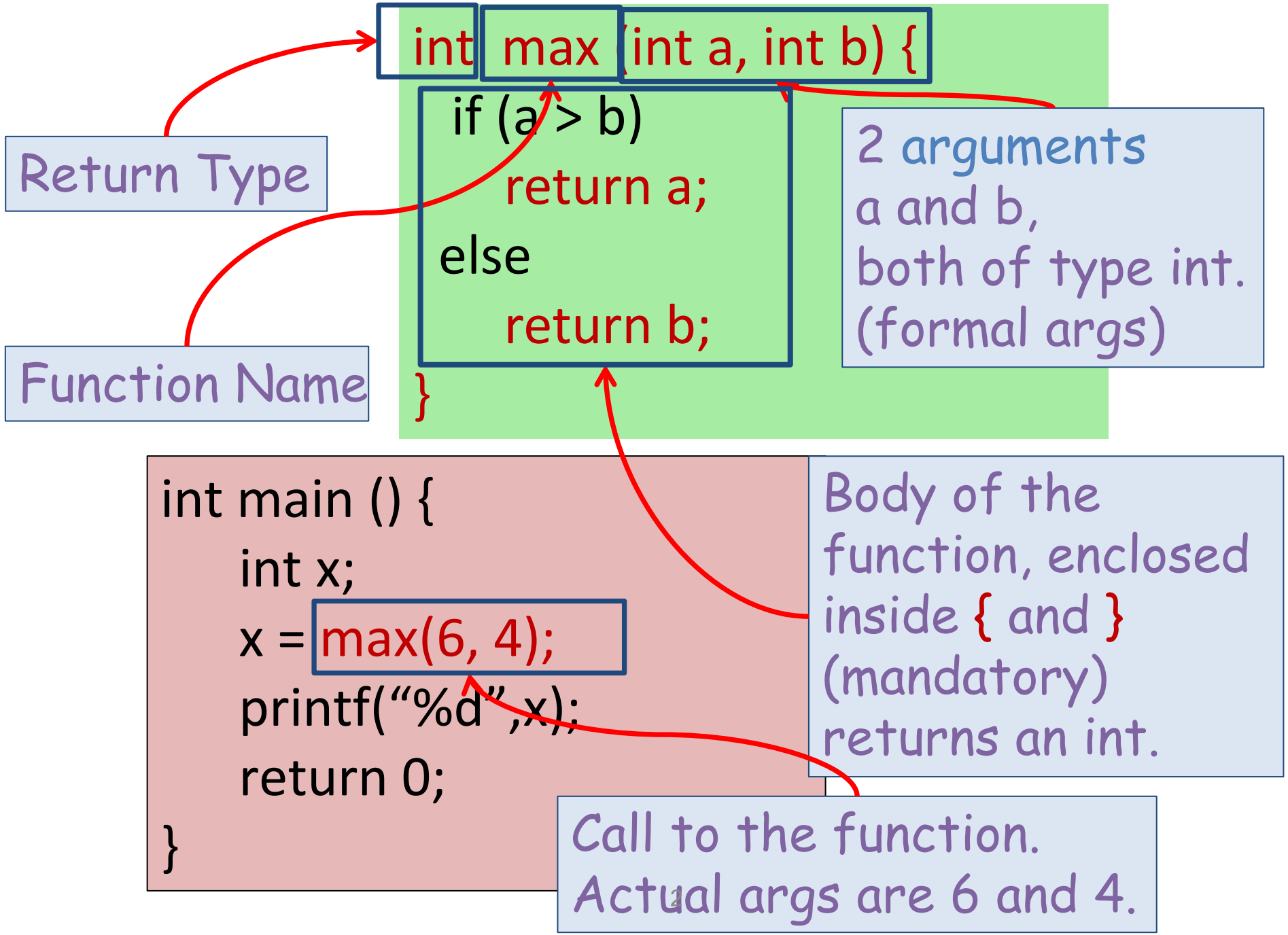
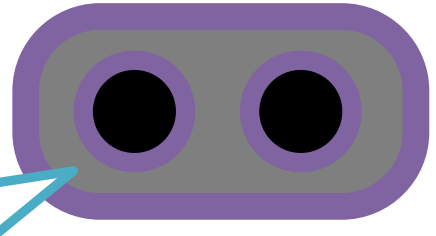
More about Functions

ESC101: Fundamentals of Computing

Nisheeth


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.

For functions that do not need to return anything i.e. void return type, you can either say return; or else not write return at all inside the function body in which case the entire body will get executed



- May write re on
- When Mr C (his dream world clone actually) sees a return statement, he immediately generates the output and function execution stops there.
- The dream ends and the original Mr C takes over

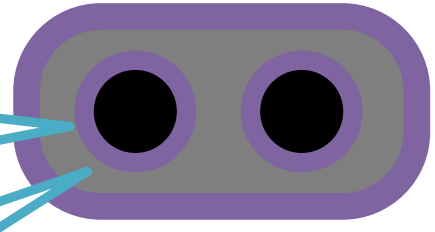
- If you return a float/double value from a function with int return type, automatic typecasting will take place.

- Be careful to not make typecasting mistakes

More

main() is also a function with return type int

main() is like a reserved function name. Cannot name your function main



- The value that is retu… normal variable of that same data type
- You can freely use returned values in expressions
  - Be careful of type though

```
int sum(int x, int y){
    return x + y;
}
```

```
int main(){
    printf("%d", sum(3,4) - sum(5,6));
    return 0;
}
```

Can even use within printf

Function and Expression

A function call is an *expression*. Can be used anywhere an expression can be used subject to type restrictions

Example below: assume we have already written the max and min functions for two integer arguments

printf("%d", max(5,3));	prints 5
max(5,3) – min(5,3)	evaluates to 2
max(x, max(y, z)) == z	checks if z is max of x, y, z
if (max(a, b)) printf("Y");	prints Y if max of a and b is not 0.

Nested Function

Not just main function but other functions can also call each other

A declaration or definition (or both) must be visible before the call

Help compiler detect any inconsistencies in function use

Compiler warning, if both (decl & def) are missing

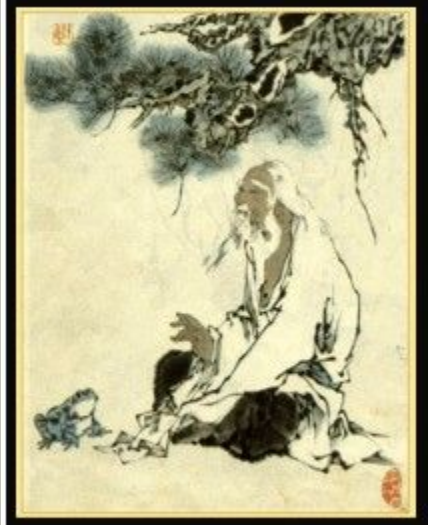
```
#include<stdio.h>
int min(int, int); //declaration
int max(int, int); //of max, min

int max(int a, int b) {
   return (a > b) ? a : b;
}

// this “cryptic” min, uses max :-)
int min(int a, int b) {
   return a + b – max (a, b);
}

int main() {
  printf(“%d”, min(6, 4));
}
```

Inception



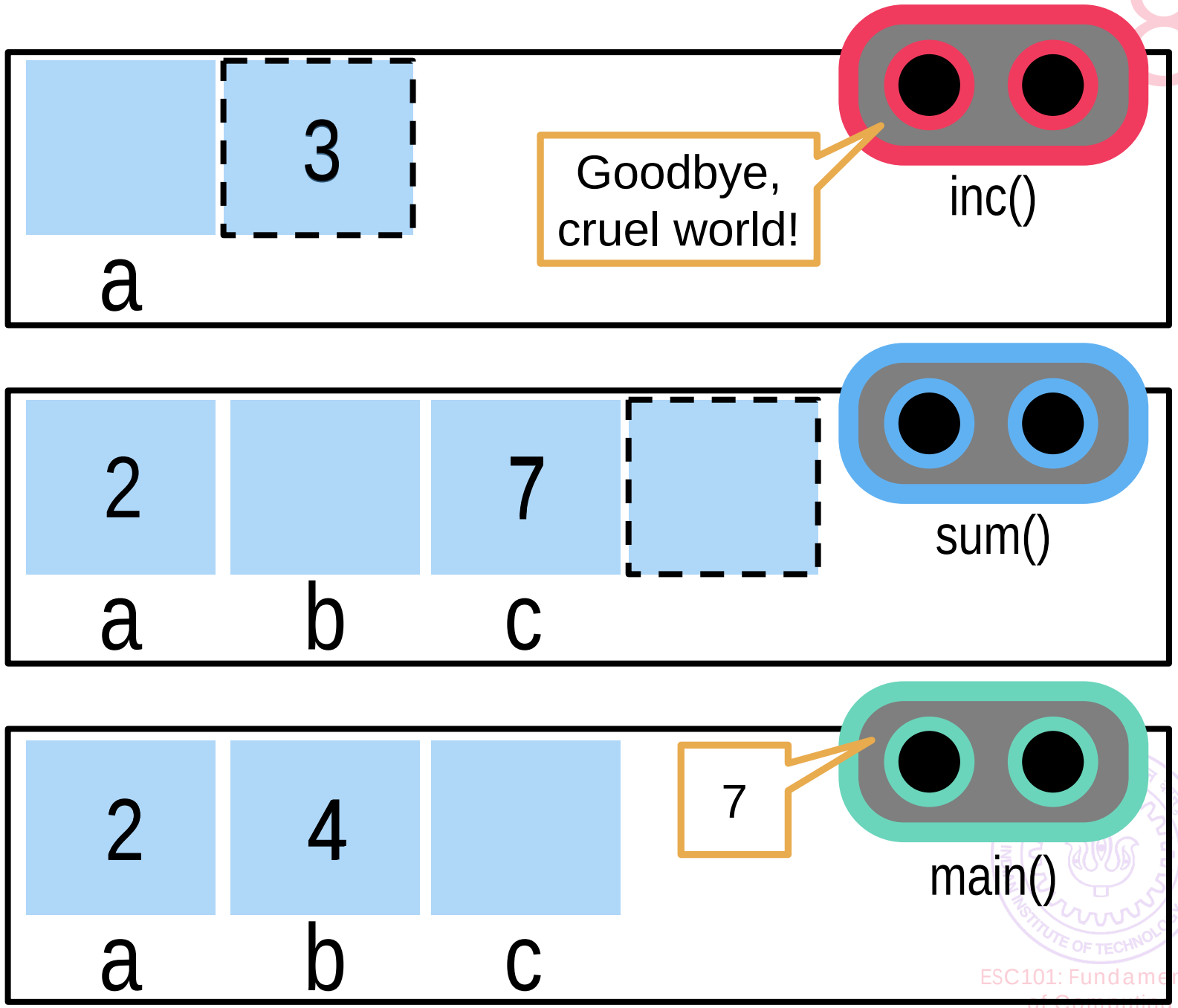
Once upon a time, Chuang Chou dreamed that he was a butterfly, a butterfly flitting about happily enjoying himself. He didn't know that he was Chou. Suddenly he awoke and was palpably Chou. He didn't know whether he were Chou who had dreamed of being a butterfly, or a butterfly who was dreaming that he was Chou.

(Zhuangzi)

izquotes.com

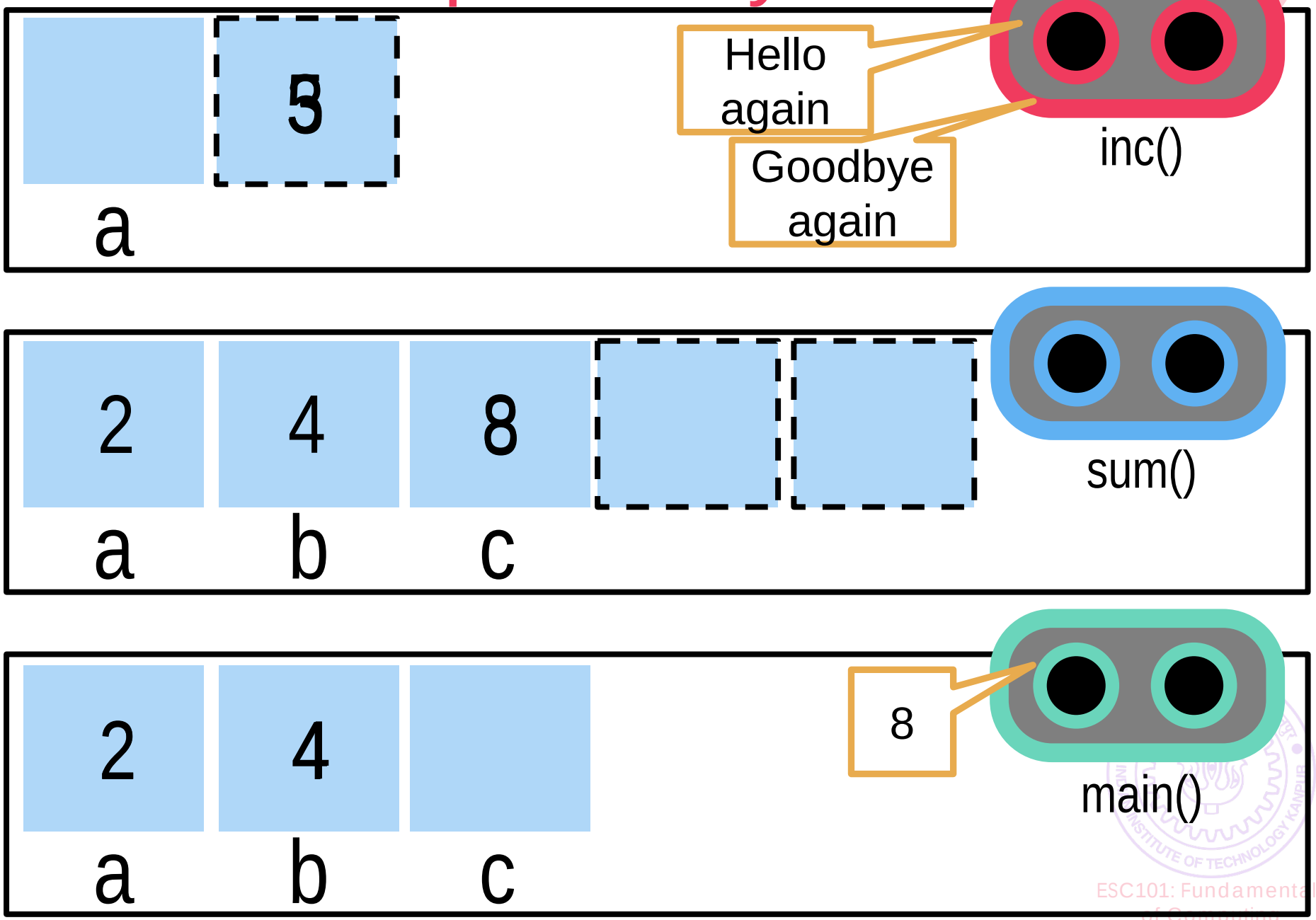
Inception

```
int inc(int a){
    return a+1;
}
int sum(int a, int b){
    int c = inc(a) + b;
    return c;
}
int main(void){
    int a = 2, b = 4, c;
    c = sum(a, b);
    printf("%d", c);
    return 0;
}
```



Same function called repeatedly

```
int inc(int a){
    return a+1;
}
int sum(int a, int b){
    int c = inc(a) +
inc(b);
    return c;
}
int main(void){
    int a = 2, b = 4, c;
    c = sum(a, b);
    printf("%d", c);
    return 0;
}
```



The 6 Basic Rules of Functions

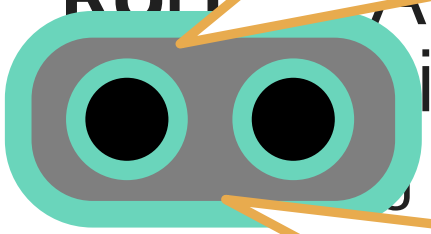
RULE 1: When we give a variable as input, the value stored inside that variable gets passed as an argument

RULE 2: When we give an expression as input, the value generated by that expression gets passed as argument

RULE 3: In case of a mismatch b/w type of arg promised and type of arg passed, typecasting will be attempted

WARNING: may cause loss of information or unexpected behavior

Yes, you may have multiple return statements but the dream world clone will die the moment any one of them is seen

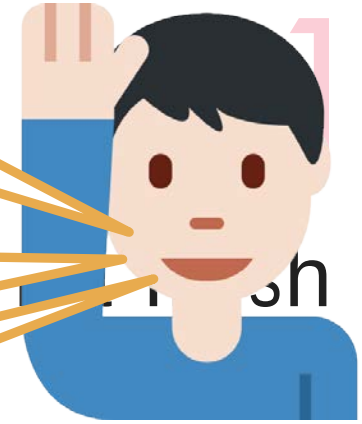


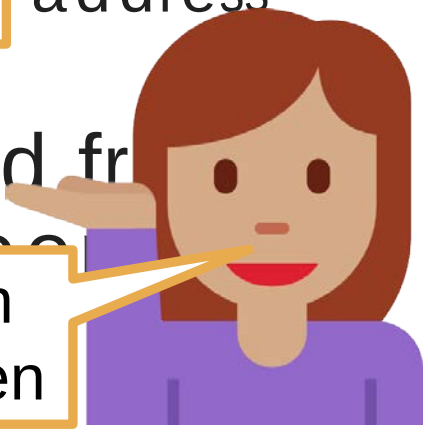
Careful, all return statements must return only one value, and that too of the type promised in the function

However, verify that the float returned is not negative

However, verify that the int gives an index within bounds

However, verify that the address returned isn't NULL





Remember, Mr C terminates a function the moment any return statement is seen

RULE 6: All clones share the memory address space

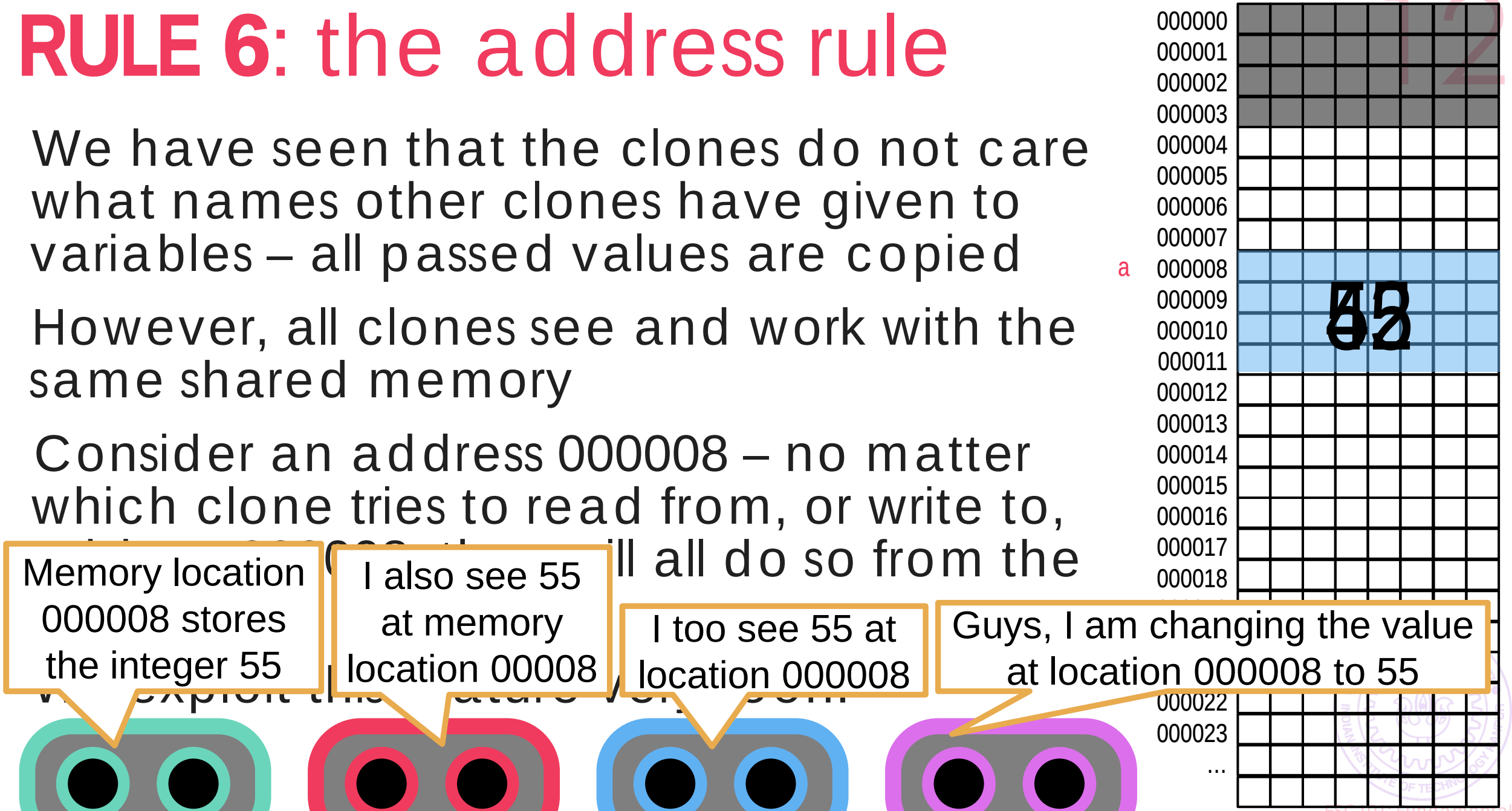
Let us look at this rule more closely

RULE 6: the address rule

We have seen that the clones do not care what names other clones have given to variables – all passed values are copied

However, all clones see and work with the same shared memory

Consider an address 000008 – no matter which clone tries to read from, or write to, ... ll all do so from the ... exploit this feature very much.



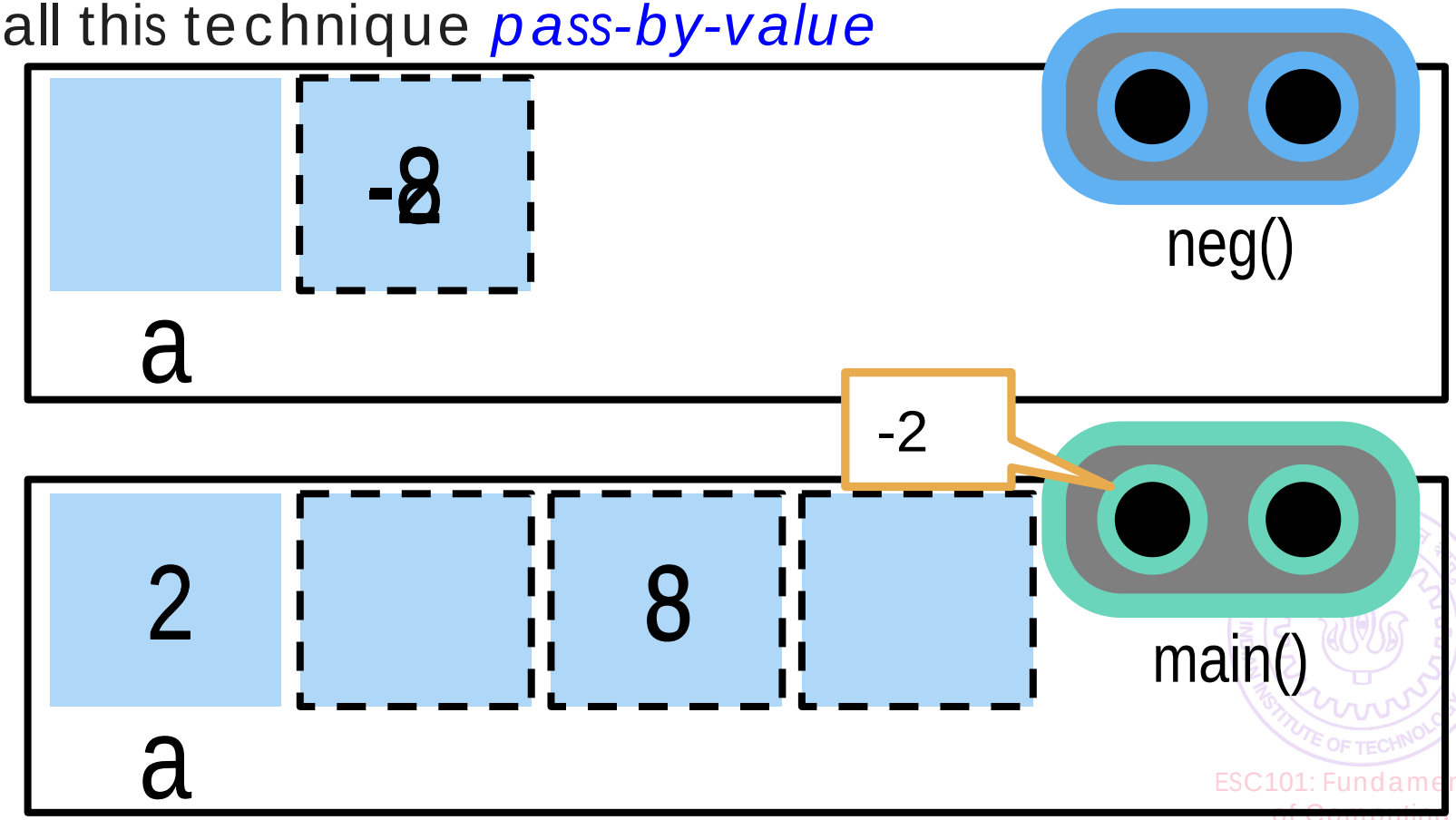
Passing simple variables/expressions

This is the case when the input to the function is either a variable (Rule 1) or an expression (Rule 2)

Rule 4 (fresh variables) will always apply no matter what is passed as input

Books, websites often call this technique *pass-by-value*

```
int neg(int a){
    return -a;
}
int main(void){
    int a = 2;
    printf("%d", neg(a));
    printf("%d", neg(4*2);
    return 0;
}
```



Summary

We have seen how normal variables (int, float, char) can be passed to functions (rule 1) and how expressions of these (rule 2) can be passed to functions

- Sometimes called pass-by-value

We have not yet seen how pointers (rule 1) and expressions that generate addresses (rule 2) can be passed to functions

- Sometimes called pass-by-pointer or pass-by-reference
- We will see this later

Remember - rule 4 always applies, no matter what!

Will see pass-by-array later

Next Class..

Static and global variables

Macros