CSC111 – Admin

- Assignment 1 Due Sunday.
- Last Zoom class (hopefully) – In person lectures start Monday -> DTB A120.
- Labs -> ECS 242.

Casting - Review

- Forcing a value to be a specified type
- explicit casting:

```
int i = 5;
(double)i → 5.0

double d = 7.9;
(int)d → 7
```

- implicit casting:

```
int i = 5.7;  // value of i is 5
double d = 7; // value of d is 7.0
```

Precedence table (Review of operators)

Precedence	Description	Associativity
Highest	Operations enclosed in brackets ( ), `++`/`--` postfix	right to left
	`+`/`-` unary operator, `++`/`--` prefix, **(type) cast**	right to left
	`*, /, %`	left to right
	`+, -`	left to right
Lowest	`=, +=, -=, *=, /=, %=`	right to left

Functions – Review

```c
#include <stdio.h>

void print_number();

int main ( ) {
   print_number();
   return 0;
}

void print_number () {
   int num = 10;
   printf("%d\n", num);
}
```

Function PROTOTYPE

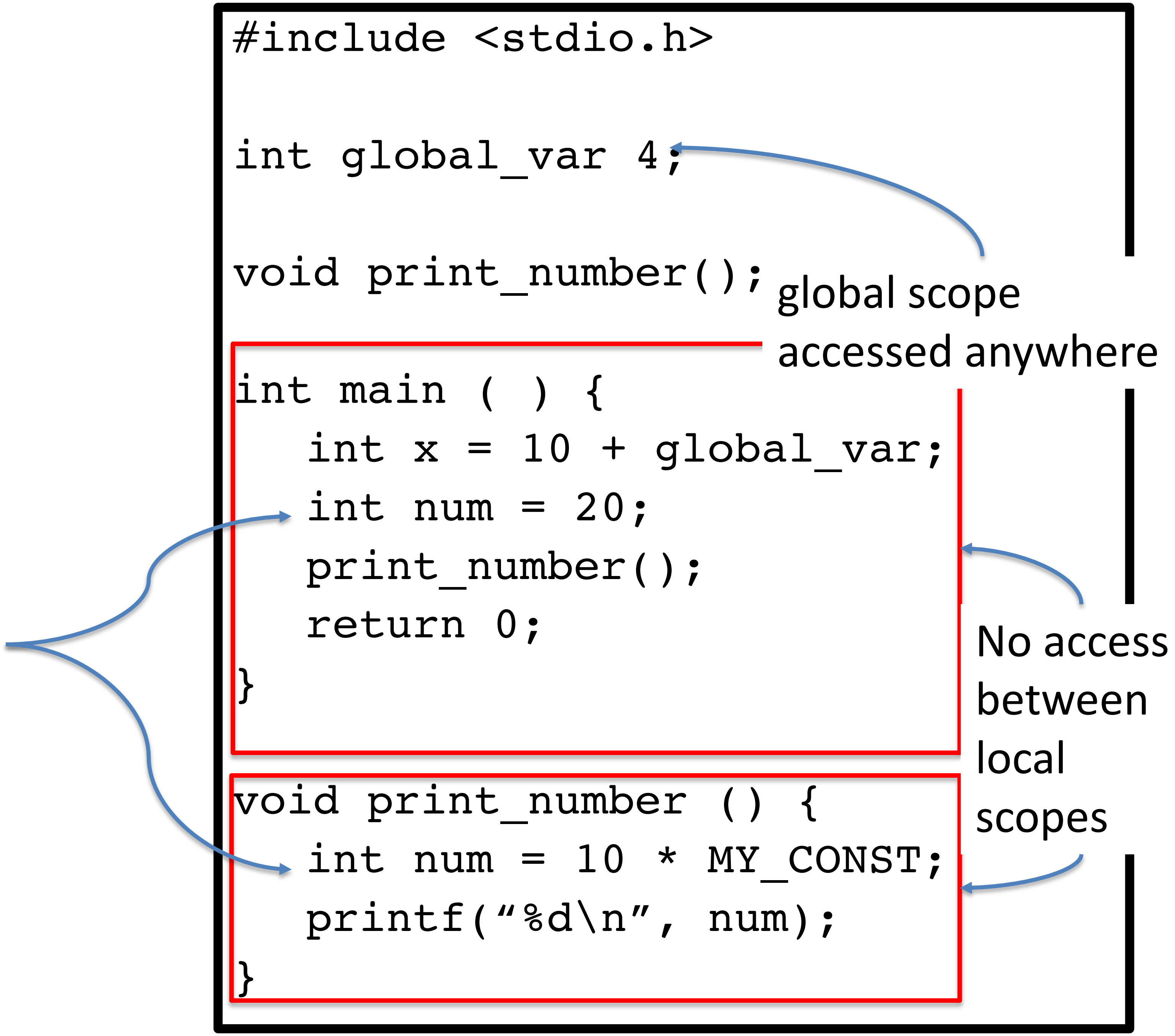
Function CALL

Function DEFINITION

Function Arguments and Relational Operators

GLOBAL SCOPE

LOCAL SCOPES



```
#include <stdio.h>

int global_var 4;

void print_number();

int main ( ) {
   int x = 10 + global_var;
   int num = 20;
   print_number();
   return 0;
}

void print_number () {
   int num = 10 * MY_CONST;
   printf(“%d\n”, num);
}
```

global scope accessed anywhere
Different variables
No access between local scopes

Defining a function that take arguements

General form:

Concrete example:

Function prototype

```
void fn_name(type parameter_name);
```

```
void print_number(int num);
```

Must match

```
void fn_name (type parameter_name) {
    C statement 1;
    C statement 2;
}
```

```
void print_number (int num) {
    printf(“%d\n”, num);
}
```

Function definition

Documentation above EVERY function!

```
/* Purpose: print num in a field 8 spaces wide
 * Parameters: int num – a number
 */
void print_number (int num) {
   printf(“%8d\n”, num);
}
```

- Every function you write **SHOULD** have a purpose comment
- If the function takes parameter(s), you **MUST** list them and the purpose must describe how the parameter(s) is(are) used.

```c
#include <stdio.h>

void print_number(int num);

int main ( ) {
   int x = 12;
   print_number(x);
   print_number(x + 8);
   print_number(11);

   return 0;
}

/* Purpose: print num in a field
 *          8 spaces wide
 * Parameters: int num – a number
 */
void print_number (int num) {
   printf("%8d\n", num);
}
```

Function PROTOTYPE

Function CALLs
passing expected **argument**

Documentation

Function DEFINITION

Relational Operators and Boolean expressions

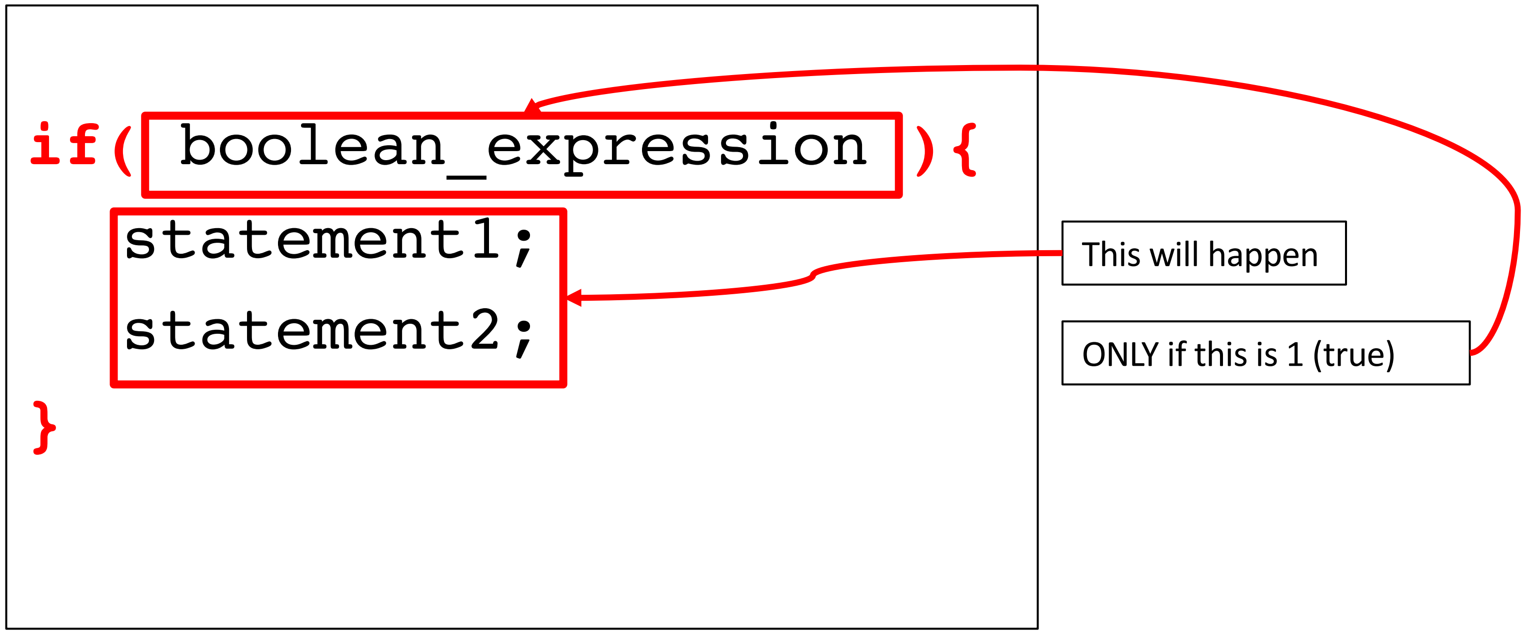
Relational Operator	meaning	Example Boolean expression	Result of Boolean expression
`>`	`greater than`	`x > y`	`**1** if x is greater than y, **0** otherwise`
`<`	`less than`	`x < y`	`**1** if x is less than y, **0** otherwise`
`>=`	`greater than or equal to`	`x >= y`	`**1** if x is greater than or equal to y, **0** otherwise`
`<=`	`less than or equal to`	`x <= y`	`**1** if x is less than or equal to y, **0** otherwise`
`==`	`equal to`	`x == y`	`**1** if x is equal to y, **0** otherwise`
`!=`	`not equal to`	`x != y`	`**1** if x is not equal to y, **0** otherwise`

False represented by 0
True represented by 1

Adding **relational operators** to precedence table

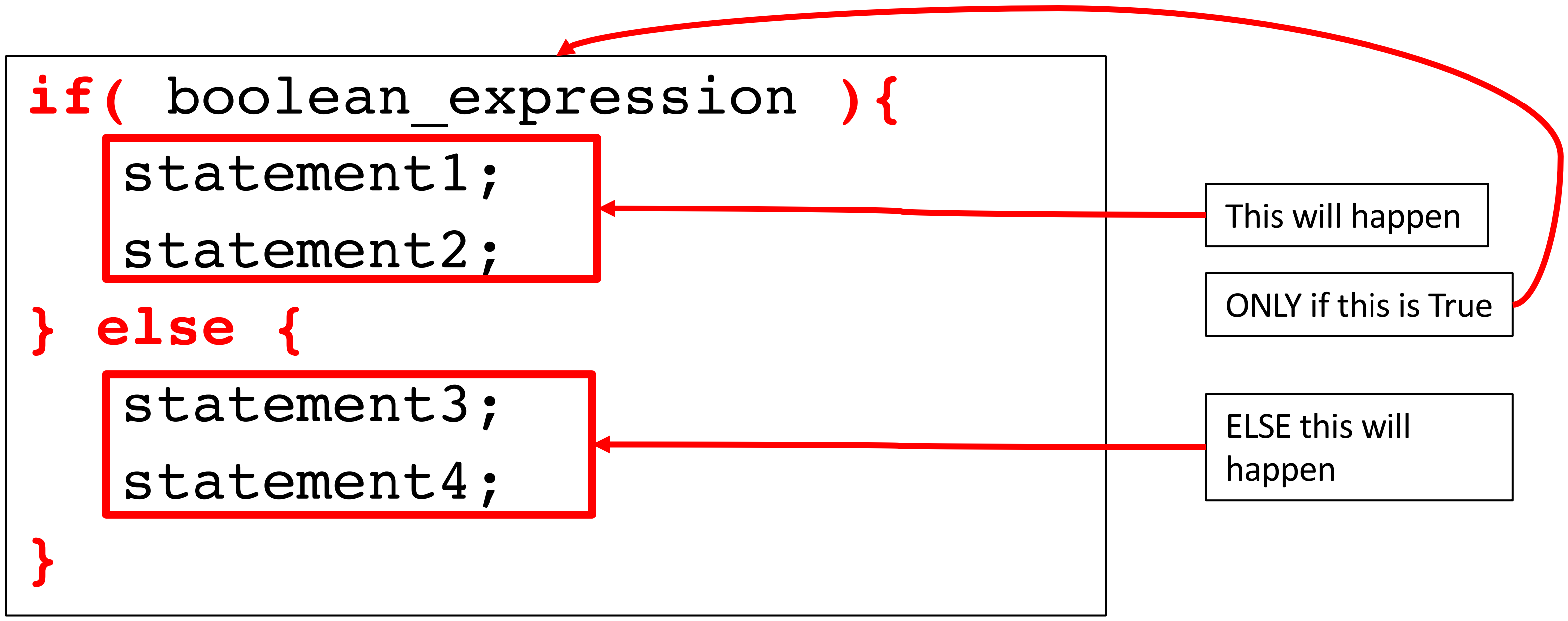
Precedence	Description	Associativity
Highest	Operations enclosed in brackets ( ), `++/--` postfix	left to right
	`+/-` unary operator, `++/--` prefix, (type) cast	right to left
	`*, /, %`	left to right
	`+, -`	left to right
	`<, <=, >, >=`	left to right
Lowest	`=, +=, -=, *=, /=, %=`	right to left

Conditional statements – if



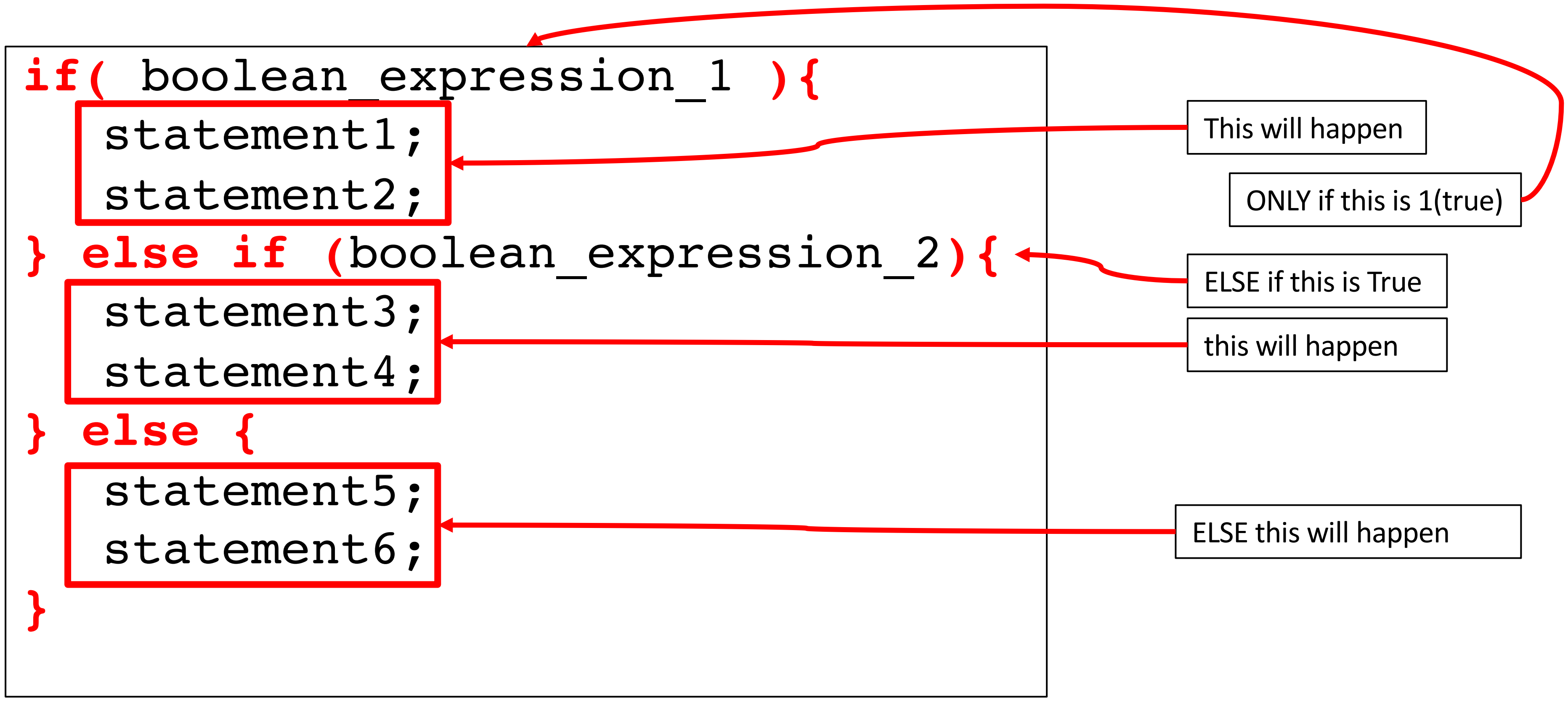
```
if( boolean_expression ){
    statement1;
    statement2;
}
```

Conditional statements – if/else



```
if( boolean_expression ){
    statement1;
    statement2;
} else {
    statement3;
    statement4;
}
```

Conditional statements – if/else if/else



```
if( boolean_expression_1 ){
    statement1;
    statement2;
} else if (boolean_expression_2){
    statement3;
    statement4;
} else {
    statement5;
    statement6;
}
```

GLOBAL SCOPE

LOCAL SCOPES

Cannot access y in these scopes

```
#include <stdio.h>

void print_number(int num);

int main ( ) {
    int x = 11;
    print_number(11);
    x = 10;
    print_number(10);
    return 0;
}
/* Purpose: ...
 *  Parameters: int num – a number
 */
void print_number (int num) {
    int x = 11;
    if(num == x) {
        int y = 2;
        num += (y + x);
        printf(“%d\n”, num);
    } else {
        printf(“%d\n”, num);
    }
    printf(“done\n”);
}
```

Optional { }

If code block within an `if` or an `else` is only one statement, the **{ }**s can be omitted.

```
if(boolean_expression){
    statement1;
}
```

Can be written as →

```
if(boolean_expression)
    statement1;
```

```
if(boolean_expression){
    statement1a;
} else {
    statement1b;
}
```

Can be written as →

```
if(boolean_expression)
    statement1a;
else
    statement1b;
```