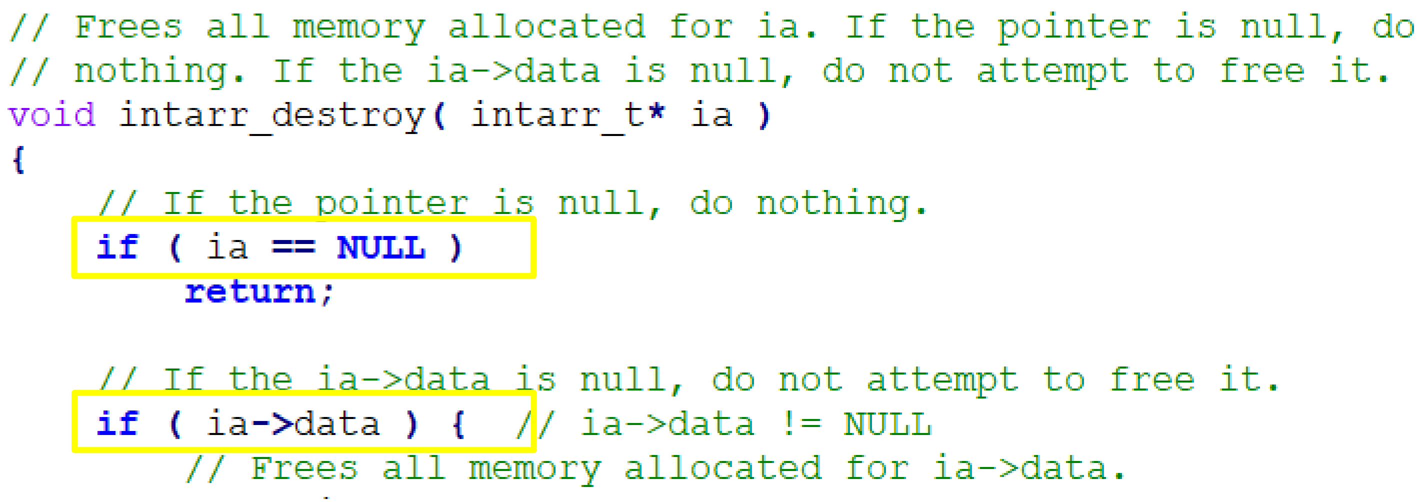
Lab 6

Helpful Tips

In general – for all labs/functions:
Parameter Validation

When a function is called, are its parameters valid?

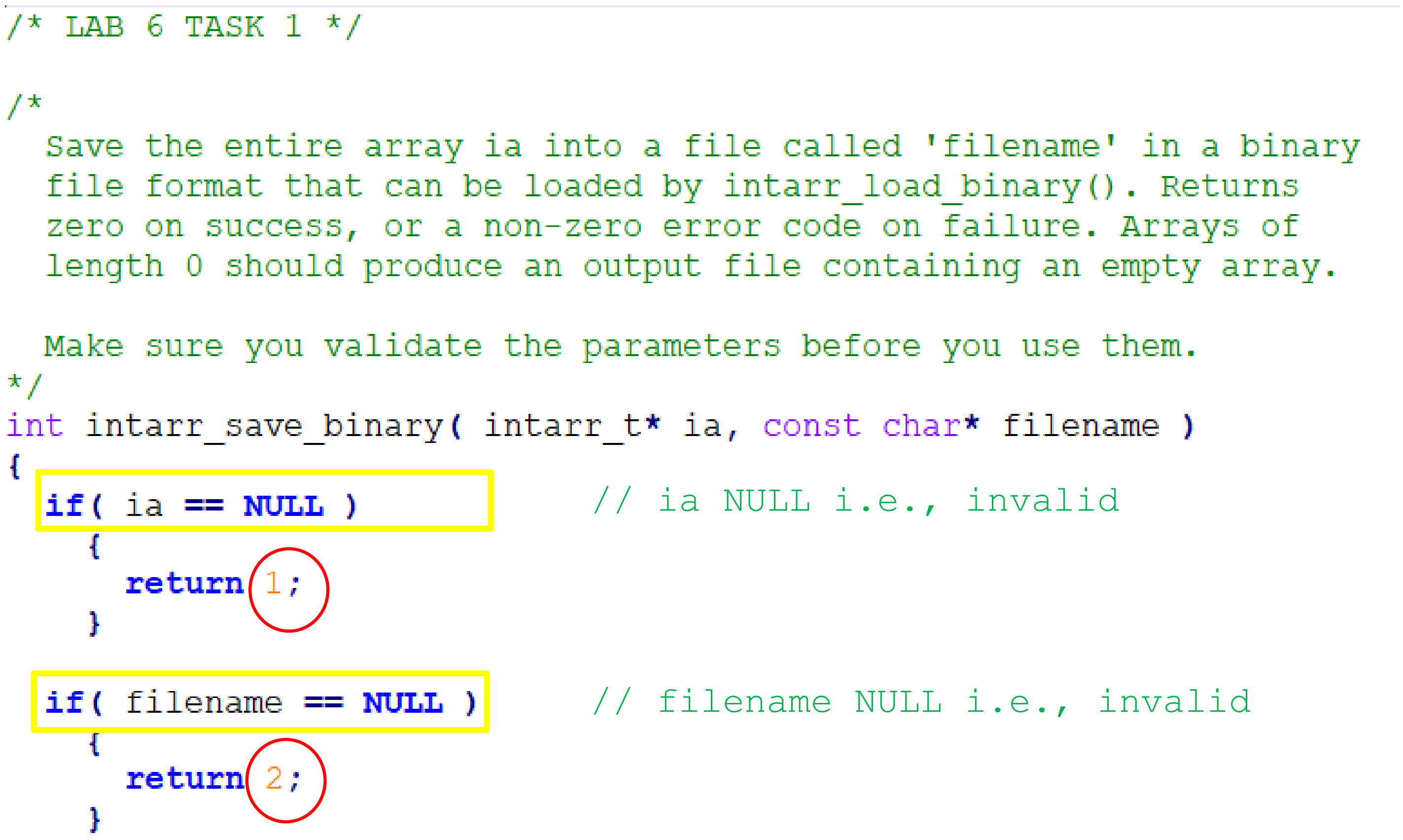
- Always check the validity of a function’s parameters
  - If a parameter is a pointer, is it NULL?
  - If a parameter is an array index, is its value “out of range”?
- Once validated, parameters can be used safely!
- Example 1 from Lab 5:



```
// Frees all memory allocated for ia. If the pointer is null, do
// nothing. If the ia->data is null, do not attempt to free it.
void intarr_destroy( intarr_t* ia )
{
    // If the pointer is null, do nothing.
    if ( ia == NULL )
        return;

    // If the ia->data is null, do not attempt to free it.
    if ( ia->data ) {  // ia->data != NULL
        // Frees all memory allocated for ia->data.
```

In general – for all functions:
Are the parameters of a function valid?

- Example 2 from Lab 6:



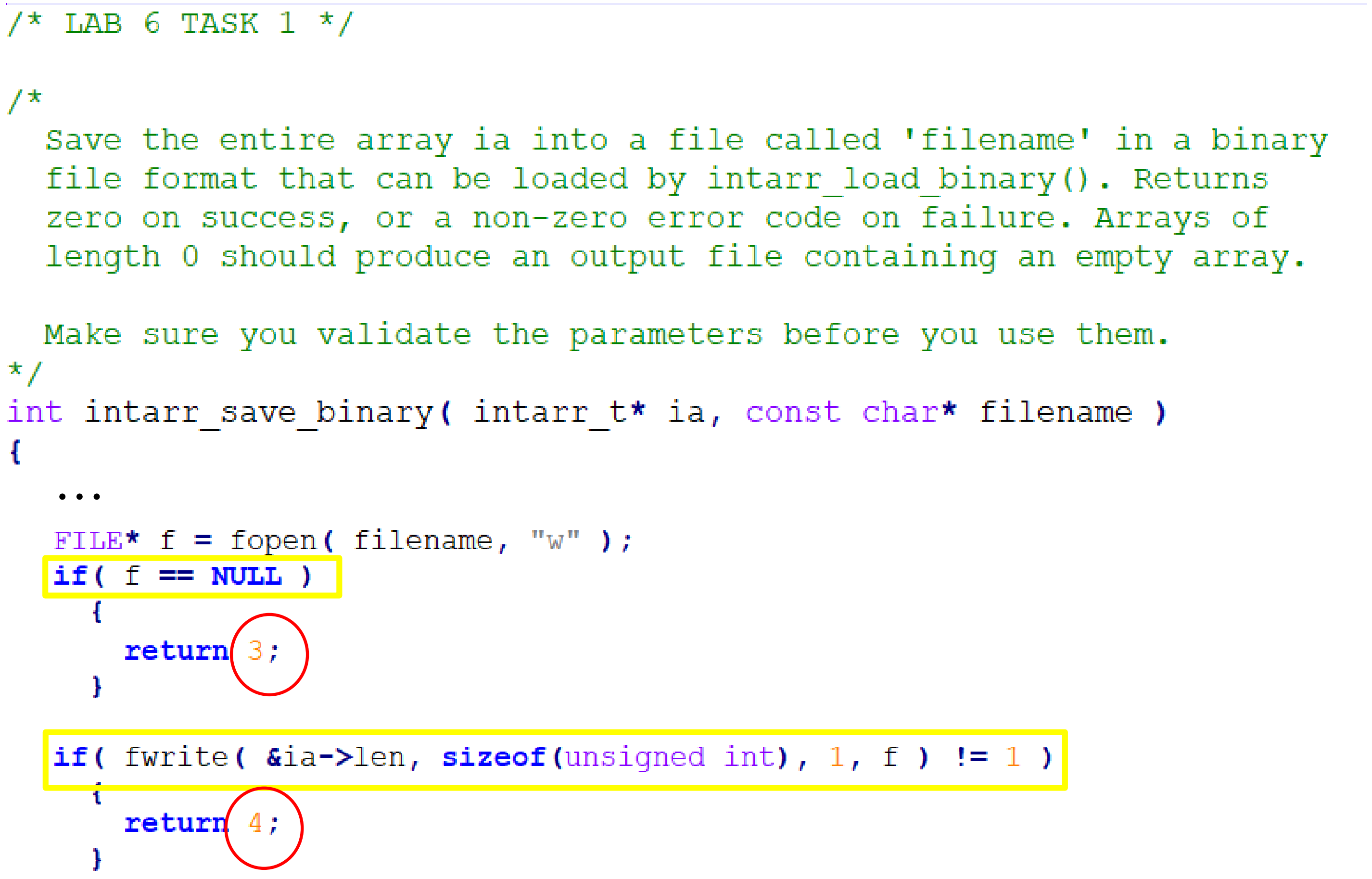
```
/* LAB 6 TASK 1 */

/*
  Save the entire array ia into a file called 'filename' in a binary
  file format that can be loaded by intarr_load_binary(). Returns
  zero on success, or a non-zero error code on failure. Arrays of
  length 0 should produce an output file containing an empty array.

  Make sure you validate the parameters before you use them.
*/
int intarr_save_binary( intarr_t* ia, const char* filename )
{
  if( ia == NULL )             // ia NULL i.e., invalid
    {
      return 1;
    }

  if( filename == NULL )       // filename NULL i.e., invalid
    {
      return 2;
    }
```

In general – for all functions:
Is the value the function returns valid?

- Always check the validity of the value returned by a function
  - If the function returns a pointer: is it NULL?
  - If the function returns a value: is it a value we are expecting?
- Once validated, the returned value can be used safely!
- However, sometimes, we cannot validate parameters/return value

In general – for all functions:
Is the value the function returns valid?

- Example:



```
/* LAB 6 TASK 1 */

/*
  Save the entire array ia into a file called 'filename' in a binary
  file format that can be loaded by intarr_load_binary(). Returns
  zero on success, or a non-zero error code on failure. Arrays of
  length 0 should produce an output file containing an empty array.

  Make sure you validate the parameters before you use them.
*/
int intarr_save_binary( intarr_t* ia, const char* filename )
{
  ...
  FILE* f = fopen( filename, "w" );
  if( f == NULL )
    {
      return 3;
    }

  if( fwrite( &ia->len, sizeof(unsigned int), 1, f ) != 1 )
    {
      return 4;
    }
```

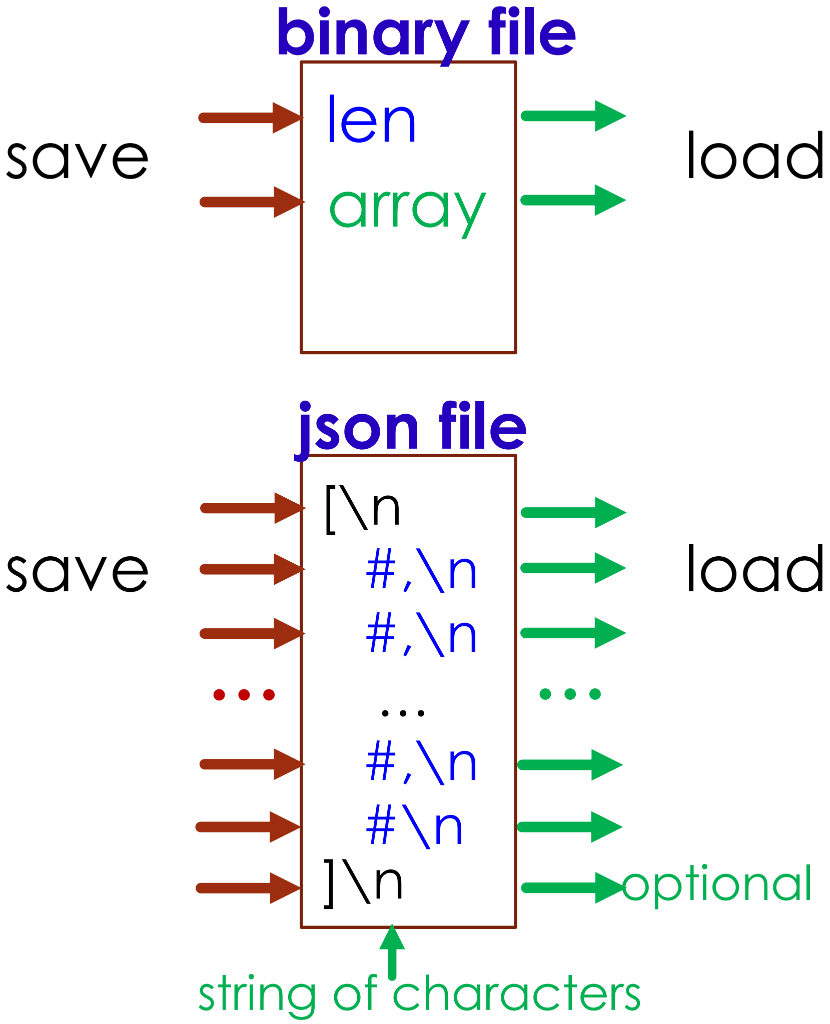
Concepts introduced in Lab 6

- External data representation -> files
  - Opening/creating files
  - Closing files
  - Writing to files
  - Reading from files

Using the C functions

Helpful Tips – Save/load content of our files

- Task 1
- Task 2



binary file

save

len

array

load

json file

save

[\n

#,\n

#,\n

…

#,\n

#\n

]\n

load

optional

string of characters

Each arrow → represents a call to one “write” function that writes to a file

Each arrow → represents a call to one “read” function that reads from a file

Explaining some of the Requirements

- Task 1 – Requirement 5

Performance hint: calls to fwrite() are relatively expensive. Try to use as few as you can.

  - This means: **1 call to `fwrite( )` for the whole array of #'s**

- Task 2 – Requirement 3

Hint: you should NOT create a single huge string in memory and write it out in one call to fwrite(). The string could require a huge amount of memory when your array is large. Since you chose an inefficient text format, you're not optimizing for speed so don't worry about using many calls to fwrite().

  - This means: **Within a `for` loop, you can call `snprintf( )` and `fwrite( )` for each number (#) separately**

Helpful Tips – For both tasks of Lab 6

- Make use of some of the functions you implemented in Lab 5 “**`intarr.c`**”

- How would you test your tasks?
  - Create a “**`testDriver.c`**”