HW01-5 Memory Leak

This program needs four files:

1. main.c
2. add.c
3. sub.c
4. header.h

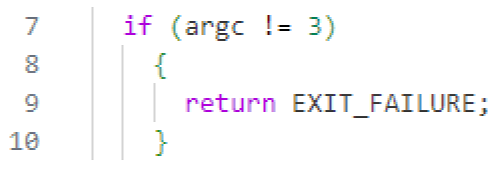
EXIT_FAILURE or EXIT_SUCCESS

When a program runs successfully, it should return `EXIT_SUCCESS`. If a program cannot do what it is supposed to do, the program should return `EXIT_FAILURE`.

Many reasons can make a program fail. Here are some examples:

- The program does not have sufficient numbers of arguments (`argc` is too small).
- The program needs to read inputs from a file but this file does not exist (`fopen` fails).

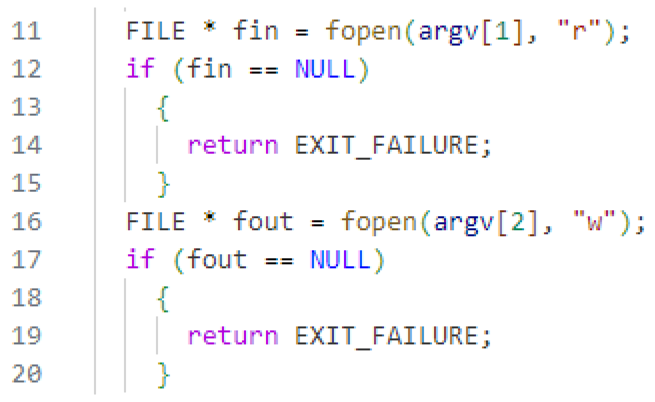
Application in the main.c:
1. Checking Command Line Arguments



```c
if (argc != 3)
  {
    return EXIT_FAILURE;
  }
```

If the provided command line arguments do not meet the requirement (not equal to 3), the program returns EXIT_FAILURE, indicating it cannot perform its intended task.

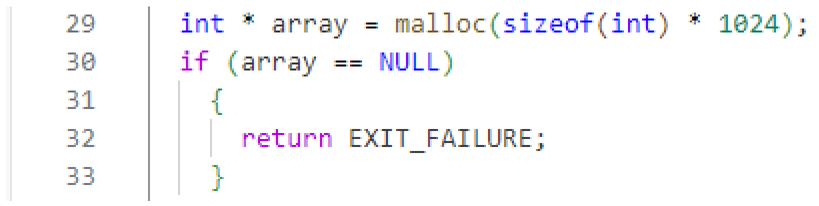
2. File Operation Error Handling



```c
FILE * fin = fopen(argv[1], "r");
if (fin == NULL)
  {
    return EXIT_FAILURE;
  }
FILE * fout = fopen(argv[2], "w");
if (fout == NULL)
  {
    return EXIT_FAILURE;
  }
```

If it's impossible to open the input or output file (for example, the file does not exist), the program returns EXIT_FAILURE, signaling an inability to read or write essential files.

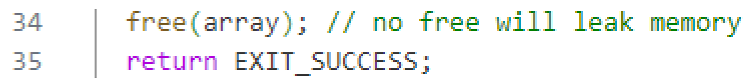
3. Dynamic Memory Allocation



```c
int * array = malloc(sizeof(int) * 1024);
if (array == NULL)
  {
    return EXIT_FAILURE;
  }
```

If memory allocation fails (due to insufficient memory, etc.), the program also returns EXIT_FAILURE.

4. Normal Termination



```c
free(array); // no free will leak memory
return EXIT_SUCCESS;
```

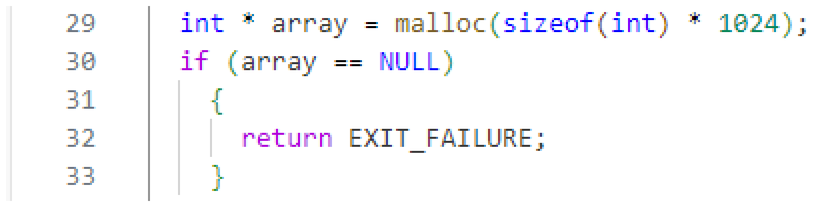
The program returns EXIT_SUCCESS after successfully completing all tasks (like file handling and memory management).

Memory Leak

1. Memory Leaks in C Programming
Memory leaks occur when a program allocates memory or other resources and fails to release them back to the system. This can lead to inefficient use of resources and potential exhaustion over time.

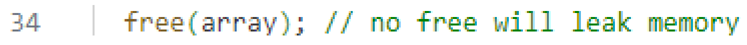
2. Analyzing main.c for Memory Leaks
main.c opens two files using fopen. If fclose is not used to close these files, the file descriptors remain open. This is a form of memory leak.

3. Dynamic Memory Allocation with malloc and free



```c
int * array = malloc(sizeof(int) * 1024);
if (array == NULL)
  {
    return EXIT_FAILURE;
  }
```

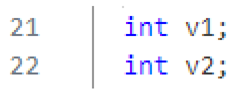
Memory is dynamically allocated for an array of 1024 integers using malloc.
The allocated memory is then freed using free.



```c
free(array); // no free will leak memory
```

If you forget to call free after you're done with the dynamically allocated memory, this will result in a classic memory leak. The allocated memory remains unused and is not returned to the system until the program terminates.

4. Static (automatic) memory allocation:



```c
int v1;
int v2;
```

When you declare local variables within a function (such as int v1, v2;), these variables are automatically allocated on the stack. When the function returns, these variables are automatically cleared, with no need for manual intervention.

5. This simplification

- Always Free Allocated Memory: It's crucial to use free for memory that you've allocated with malloc, calloc, or realloc to prevent memory leaks.

- Always Close Open Files: Similarly, fclose should be used to close files opened with fopen to avoid resource leaks.

- **Every byte of memory leak in your code will deduct one point from your total score. Therefore, even if your code has no logical errors, if there's a memory leak greater than 100 bytes, you will still receive zero points**