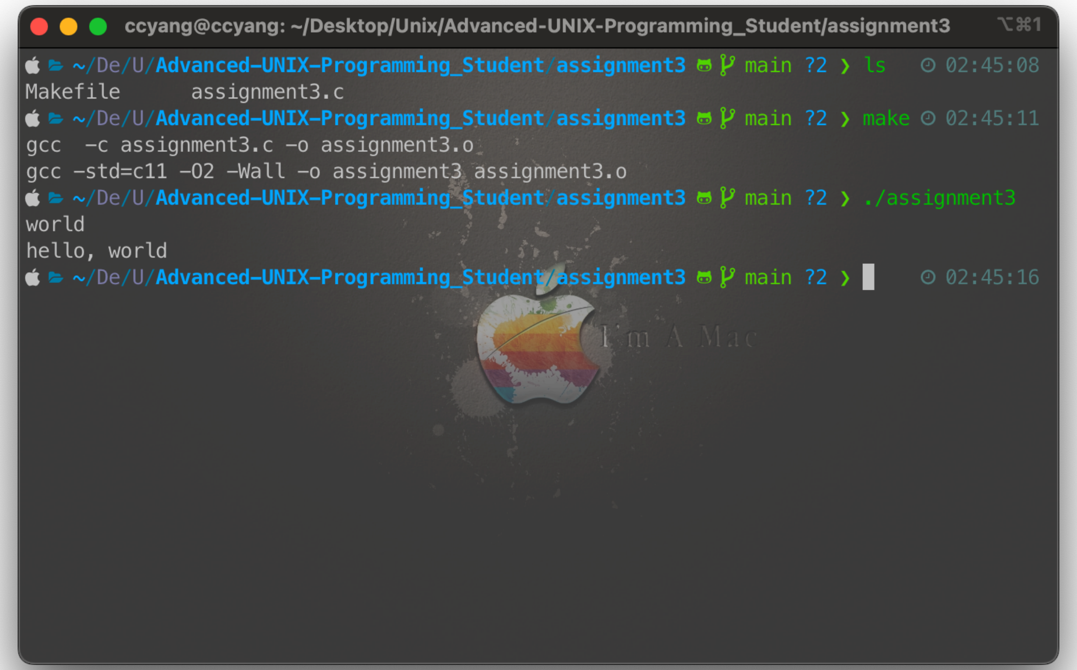
Unix Assignment3

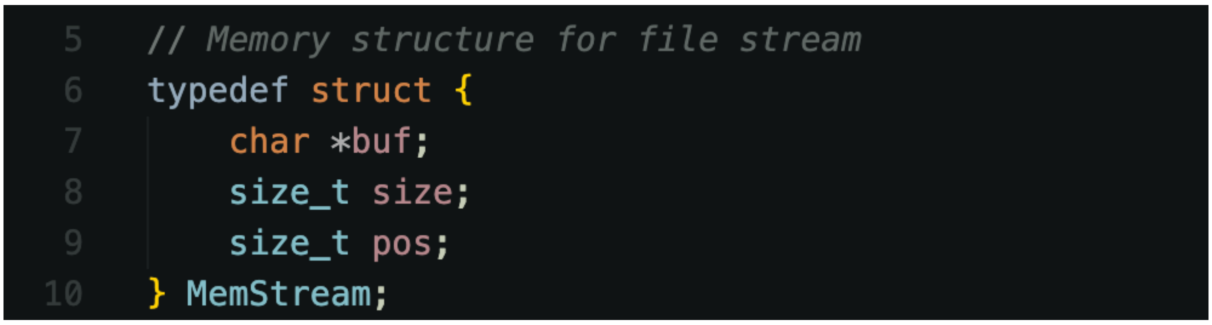
Team19 108072244 邱煒甯 108032053 陳凱揚 112065525 簡佩如

Result



```
ccyang@ccyang: ~/Desktop/Unix/Advanced-UNIX-Programming_Student/assignment3 ⌥⌘1
~/De/U/Advanced-UNIX-Programming_Student/assignment3  main ?2 ❯ ls      02:45:08
Makefile      assignment3.c
~/De/U/Advanced-UNIX-Programming_Student/assignment3  main ?2 ❯ make    02:45:11
gcc  -c assignment3.c -o assignment3.o
gcc -std=c11 -O2 -Wall -o assignment3 assignment3.o
~/De/U/Advanced-UNIX-Programming_Student/assignment3  main ?2 ❯ ./assignment3
world
hello, world
~/De/U/Advanced-UNIX-Programming_Student/assignment3  main ?2 ❯         02:45:16
```

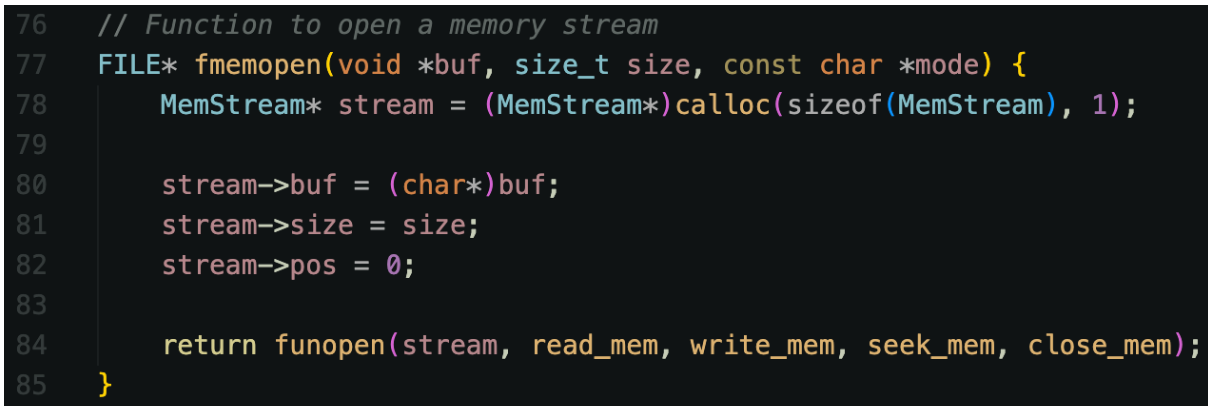
Implementation



```c
// Memory structure for file stream
typedef struct {
    char *buf;
    size_t size;
    size_t pos;
} MemStream;
```

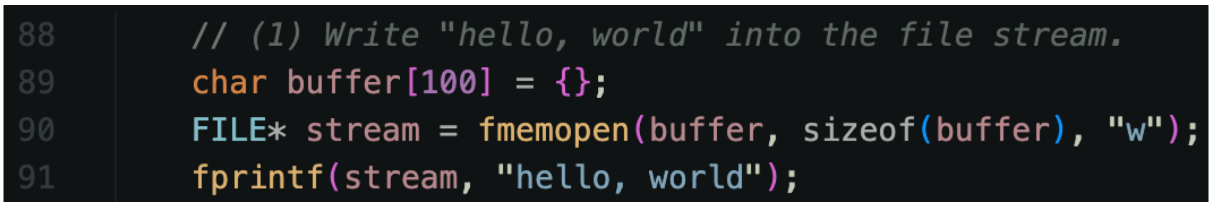
`MemStream` defines a memory structure for file stream, which includes `buf` (a pointer to memory), `size` (the size of memory), and `pos` (the current position).

(1) (1 pt) Write “hello, world” in the file stream.



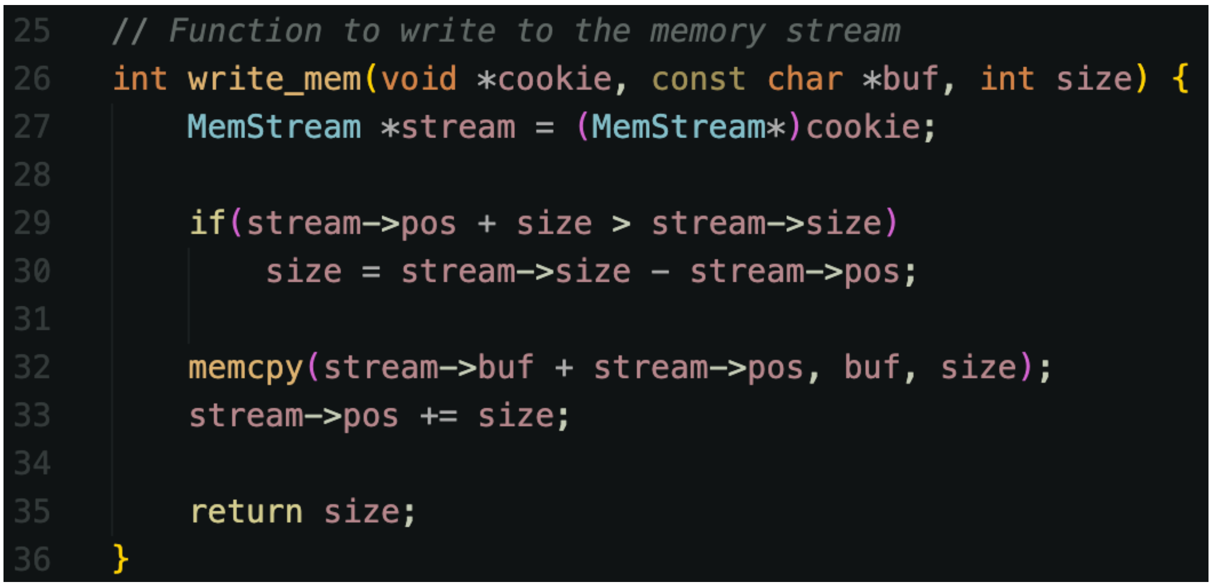
```c
76  // Function to open a memory stream
77  FILE* fmemopen(void *buf, size_t size, const char *mode) {
78      MemStream* stream = (MemStream*)calloc(sizeof(MemStream), 1);
79
80      stream->buf = (char*)buf;
81      stream->size = size;
82      stream->pos = 0;
83
84      return funopen(stream, read_mem, write_mem, seek_mem, close_mem);
85  }
```

We have defined the `fmemopen` function, which takes a pointer to memory (`buf`), the size of memory (`size`), and a `mode` , and it returns a pointer to a `FILE` structure. This `FILE` structure is actually customized using the `funopen` function, which maps standard file operations like `fprintf` , `fread` , `fseek` , and `fclose` to custom read and write functions, so we provide `read_mem` , `write_mem` , `seek_mem` and `close_mem` 4 function for `funopen` .



```c
88      // (1) Write "hello, world" into the file stream.
89      char buffer[100] = {};
90      FILE* stream = fmemopen(buffer, sizeof(buffer), "w");
91      fprintf(stream, "hello, world");
```

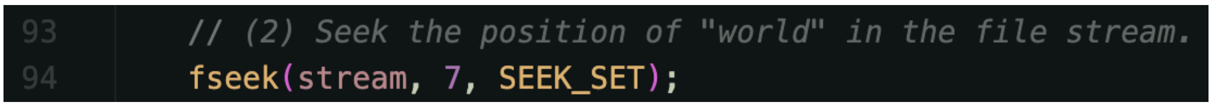
Next, we create a file stream named `stream` in line 90. This sets the stream to `w` mode, indicating it's opened in write mode. Subsequently, we use the `fprintf` function to write "hello, world" into the `stream` .



```c
// Function to write to the memory stream
int write_mem(void *cookie, const char *buf, int size) {
    MemStream *stream = (MemStream*)cookie;

    if(stream->pos + size > stream->size)
        size = stream->size - stream->pos;

    memcpy(stream->buf + stream->pos, buf, size);
    stream->pos += size;

    return size;
}
```

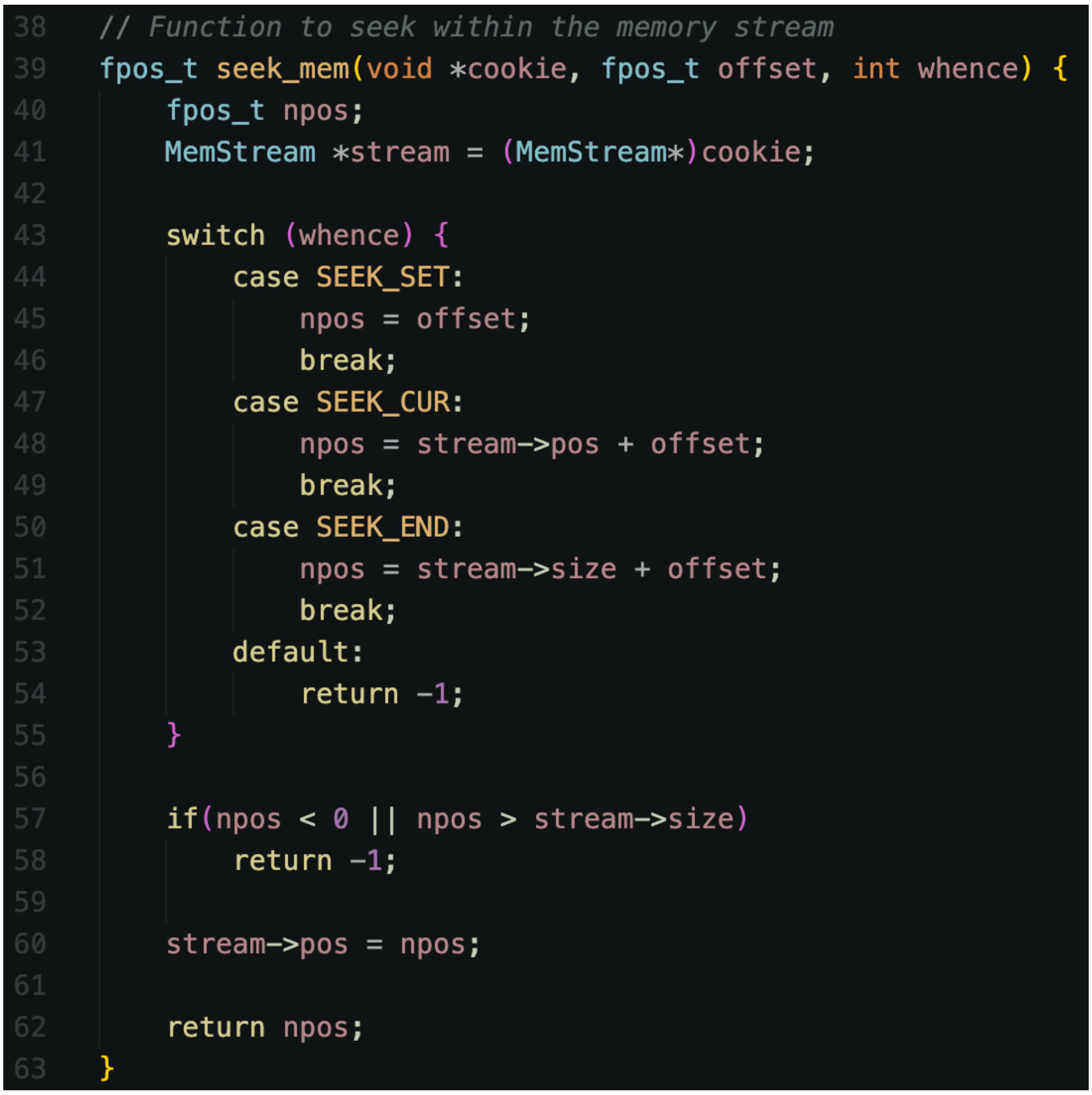
When the `fprintf` function is called, it invokes the `write_mem` function and passes `stream`, the address of the "hello, world" string, and the size of the string as parameters. Inside the `write_mem` function, it checks the current write position and the size of the data to be written. If it does, it adjusts the size to be write to match the remaining data size to prevent undefined behavior. Then, it uses the `memcpy` function to copy the data from the input buffer to the memory stream's position, and subsequently updates the position. Finally, the `write_mem` function returns the size of the written data.

(2) (1 pt) Seek the position of “world” in the file stream.



```c
    // (2) Seek the position of "world" in the file stream.
    fseek(stream, 7, SEEK_SET);
```

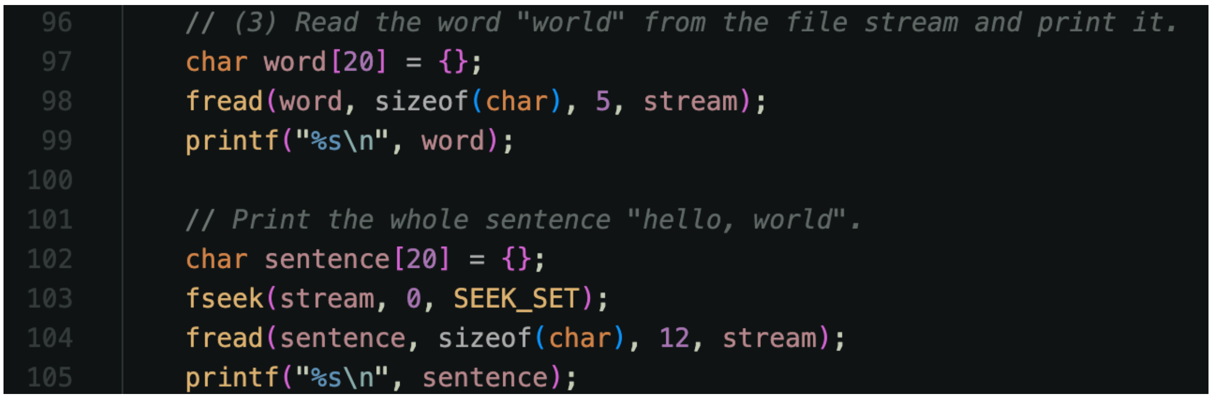
We use `fseek` to move the file pointer (file position indicator) to the beginning of the "world" string. Specifically, the `fseek` function will move the file pointer to the 7th position in the buffer array, which is the starting position of the "world" string.



```c
// Function to seek within the memory stream
fpos_t seek_mem(void *cookie, fpos_t offset, int whence) {
    fpos_t npos;
    MemStream *stream = (MemStream*)cookie;

    switch (whence) {
        case SEEK_SET:
            npos = offset;
            break;
        case SEEK_CUR:
            npos = stream->pos + offset;
            break;
        case SEEK_END:
            npos = stream->size + offset;
            break;
        default:
            return -1;
    }

    if(npos < 0 || npos > stream->size)
        return -1;

    stream->pos = npos;

    return npos;
}
```

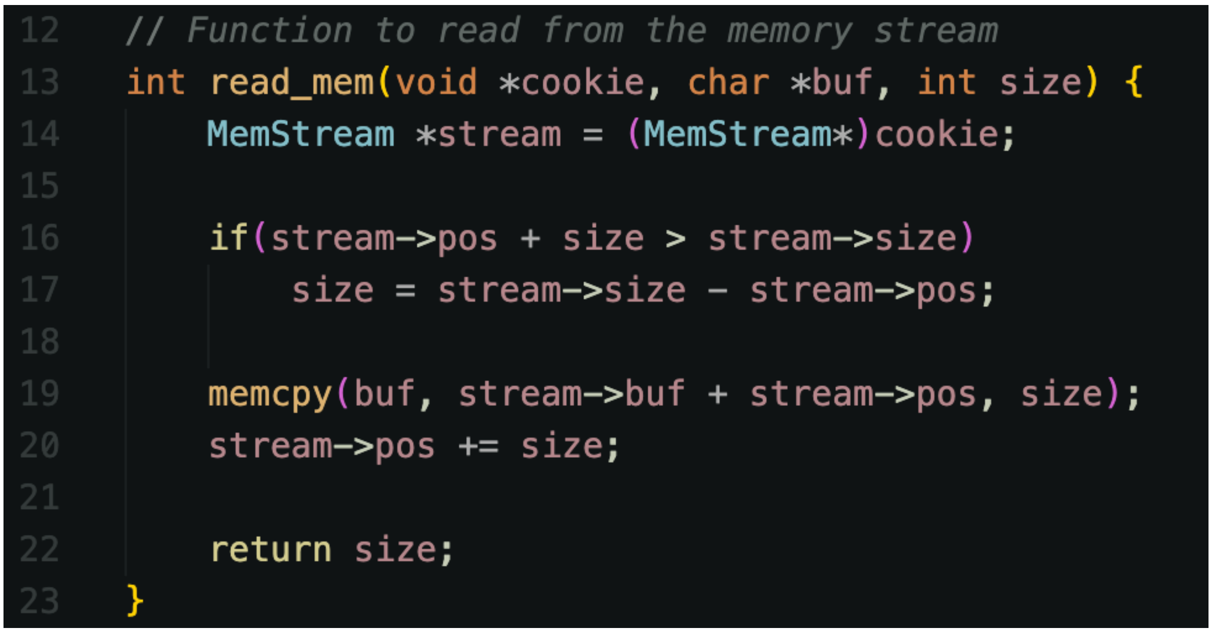
When the `fseek` function is called, it invokes the `seek_mem` function. Depending on the value of `whence`, it can be `SEEK_SET` (an offset counted from the beginning), `SEEK_CUR` (an offset counted from the current position), or `SEEK_END` (an offset counted from the end of the file). Based on the different `whence` values, the `seek_mem` function moves the file pointer to different positions.

(3) (1 pt) Read the word "world" from the file stream and print it. Then, print the whole sentence "hello, world".



```c
    // (3) Read the word "world" from the file stream and print it.
    char word[20] = {};
    fread(word, sizeof(char), 5, stream);
    printf("%s\n", word);

    // Print the whole sentence "hello, world".
    char sentence[20] = {};
    fseek(stream, 0, SEEK_SET);
    fread(sentence, sizeof(char), 12, stream);
    printf("%s\n", sentence);
```

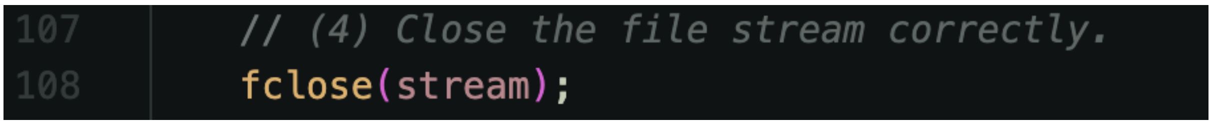
In the previous section, `fseek` was used to move the file pointer to the beginning of the "world" string. Then, `fread` is used to read 5 characters ("world") from the file stream into the `word` array. Finally, we print the read "world" string. We also read the whole sentence and print it using `fseek` and `fread`



```c
// Function to read from the memory stream
int read_mem(void *cookie, char *buf, int size) {
    MemStream *stream = (MemStream*)cookie;

    if(stream->pos + size > stream->size)
        size = stream->size - stream->pos;

    memcpy(buf, stream->buf + stream->pos, size);
    stream->pos += size;

    return size;
}
```

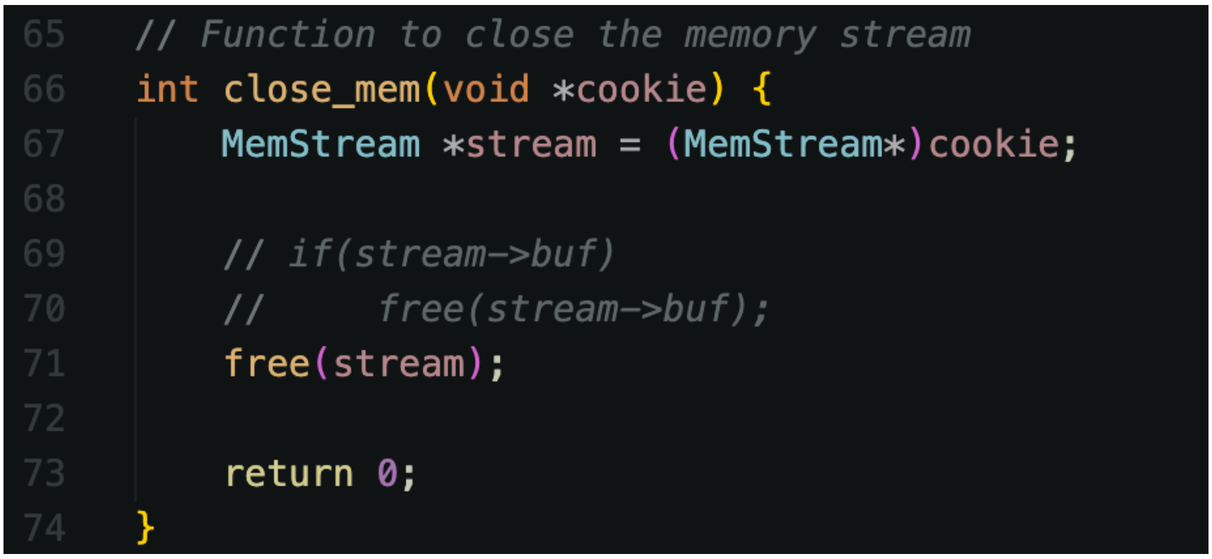
When the `fread` function is called, it invokes the `read_mem` function. The process is similar to the `write_mem` function. The difference is `read_mem` function copy the data from the file stream to the buffer.

(4) (1 pt) Close the file stream correctly.



```c
    // (4) Close the file stream correctly.
    fclose(stream);
```

`fclose` is responsible for closing the custom file stream. When the `fclose` function is called, it invokes the `close_mem` function, which is used to release memory and ensure there are no memory leaks.



```c
// Function to close the memory stream
int close_mem(void *cookie) {
    MemStream *stream = (MemStream*)cookie;

    // if(stream->buf)
    //     free(stream->buf);
    free(stream);

    return 0;
}
```

In the `close_mem` function, it should releases the `stream→buf` first (we comment it, because we don’t use dynamic allocate here). Then, it releases the stream structure because its memory is dynamic allocated.