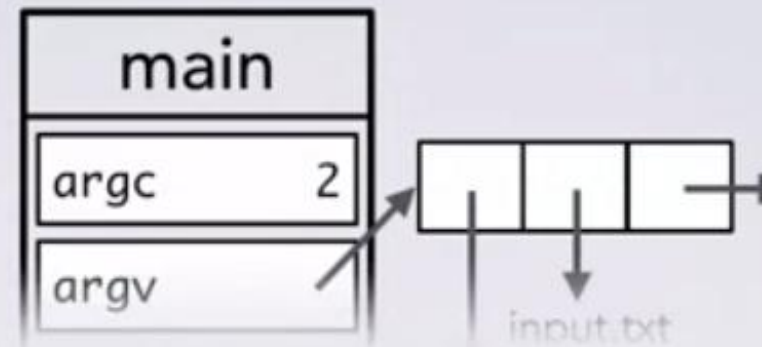


```

int main (int argc, char ** argv) {
→  if (argc != 2) { /* omitted */ }
    FILE * f = fopen(argv[1], "r");
    if (f == NULL) { /* omitted */ }
    int c;
    int letters = 0;
    while ( (c = fgetc(f)) != EOF ) {
        if (isalpha(c)) {
            letters++;
        }
    }
    printf("%s has %d letters in it\n", argv[1], letters);
    return EXIT_SUCCESS;
}

```



### Output

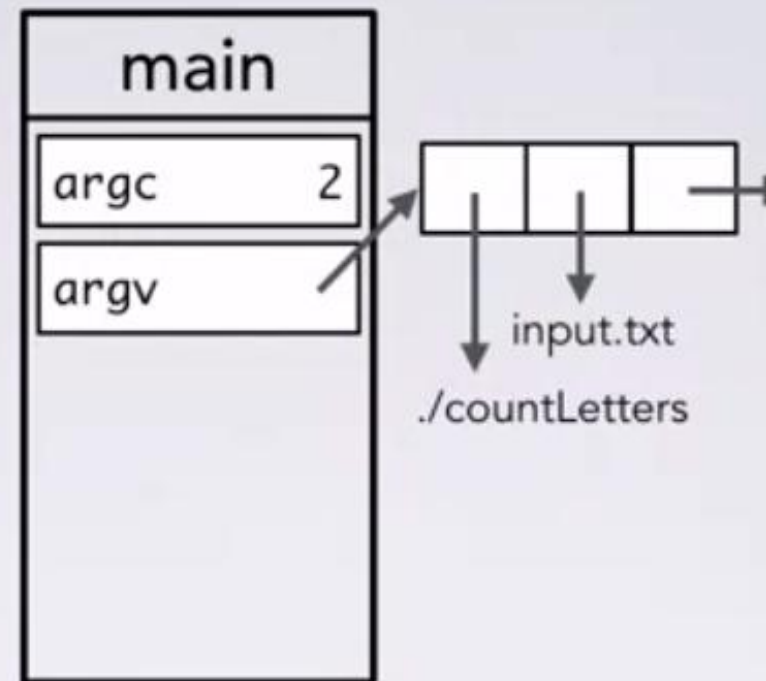
--

As always, our execution arrow begins at the start of main.

```

int main (int argc, char ** argv) {
    if (argc != 2) { /* omitted */ }
    FILE * f = fopen(argv[1], "r");
    if (f == NULL) { /* omitted */ }
    int c;
    int letters = 0;
    while ( (c = fgetc(f)) != EOF ) {
        if (isalpha(c)) {
            letters++;
        }
    }
    printf("%s has %d letters in it\n", argv[1], letters);
    return EXIT_SUCCESS;
}

```



### Output

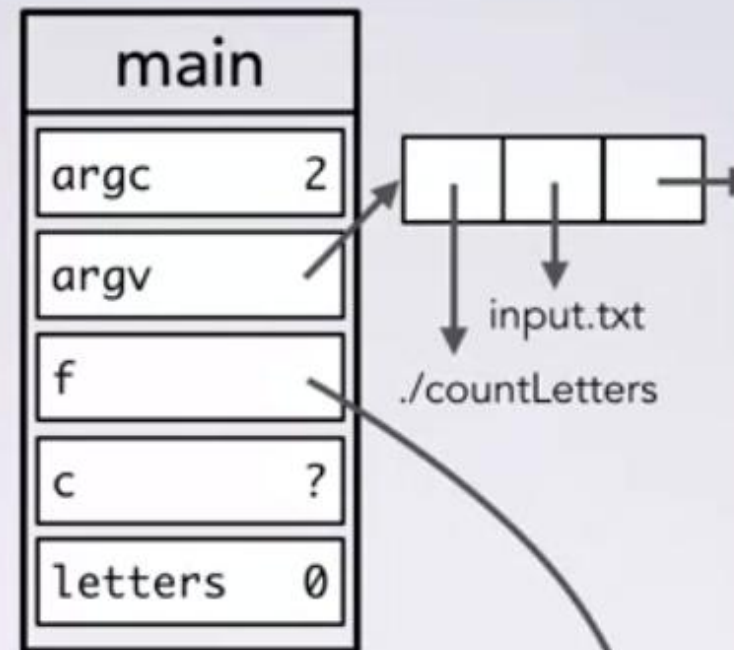
--

Now that we have command line arguments, `main`'s frame begins with `argc` and `argv`.

```

int main (int argc, char ** argv) {
    if (argc != 2) { /* omitted */ }
    FILE * f = fopen(argv[1], "r");
    if (f == NULL) { /* omitted */ }
    int c;
    int letters = 0;
    while ( (c = fgetc(f)) != EOF ) {
        if (isalpha(c)) {
            letters++;
        }
    }
    printf("%s has %d letters in it\n", argv[1], letters);
    return EXIT_SUCCESS;
}

```



Output

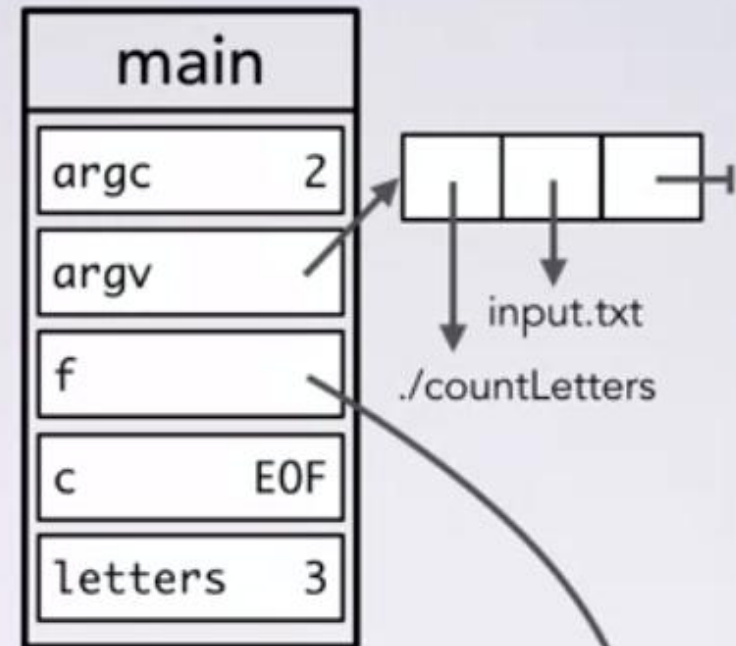
file: input.txt mode: read eof: false
ab4 c

We then do this assignment

```

int main (int argc, char ** argv) {
    if (argc != 2) { /* omitted */ }
    FILE * f = fopen(argv[1], "r");
    if (f == NULL) { /* omitted */ }
    int c;
    int letters = 0;
    while ( (c = fgetc(f)) != EOF ) {
        if (isalpha(c)) {
            letters++;
        }
    }
    printf("%s has %d letters in it\n", argv[1], letters);
    return EXIT_SUCCESS;
}

```



Output

```

file: input.txt
mode: read
eof: true

ab4
c

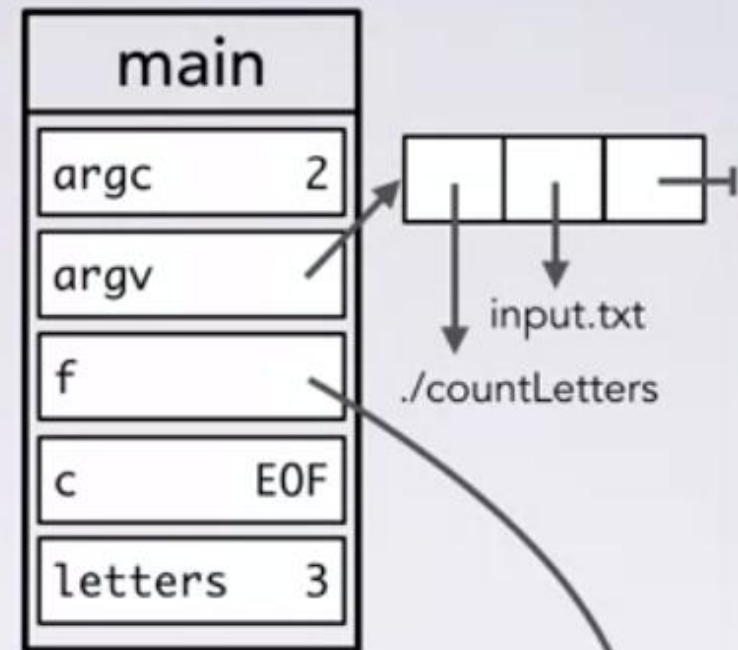
```

If we were to call `feof`,

```

int main (int argc, char ** argv) {
    if (argc != 2) { /* omitted */ }
    FILE * f = fopen(argv[1], "r");
    if (f == NULL) { /* omitted */ }
    int c;
    int letters = 0;
    while ( (c = fgetc(f)) != EOF ) {
        if (isalpha(c)) {
            letters++;
        }
    }
    printf("%s has %d letters in it\n", argv[1], letters);
    return EXIT_SUCCESS;
}

```



Output
input.txt has 3 letters in it

file: input.txt mode: read eof: true
ab4 c

So we print out input.txt  
has 3 letters in it.



```
#define LINE_SIZE 5
```

```
int main (int argc, char ** argv) {  
    if (argc != 2) { /* omitted */ }  
    FILE * f = fopen(argv[1], "r");  
    if (f == NULL) { /* omitted */ }  
    long total = 0;  
    char line[LINE_SIZE];  
    while (fgets(line, LINE_SIZE, f) != NULL) {  
        if (strchr(line, '\n') == NULL) {  
            printf("Line is too long!\n");  
            return EXIT_FAILURE;  
        }  
        total += atoi(line);  
    }  
    printf("The total is %ld\n", total);  
    return EXIT_SUCCESS;  
}
```

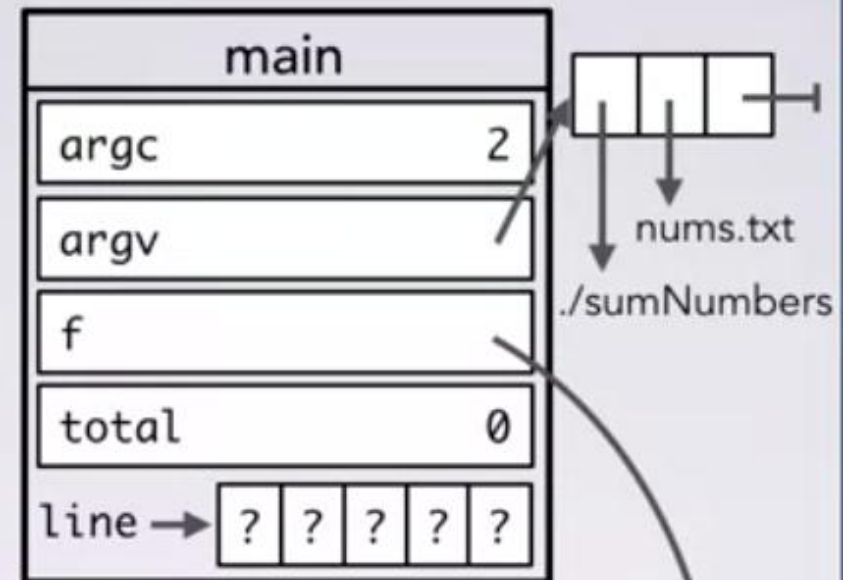
**Output**



0:35 / 5:01

```
#define LINE_SIZE 5
```

```
int main (int argc, char ** argv) {  
    if (argc != 2) { /* omitted */ }  
    FILE * f = fopen(argv[1], "r");  
    if (f == NULL) { /* omitted */ }  
    long total = 0;  
    char line[LINE_SIZE];  
    while (fgets(line, LINE_SIZE, f) != NULL) {  
        if (strchr(line, '\n') == NULL) {  
            printf("Line is too long!\n");  
            return EXIT_FAILURE;  
        }  
        total += atoi(line);  
    }  
    printf("The total is %ld\n", total);  
    return EXIT_SUCCESS;  
}
```



file: nums.txt  
mode: read  
eof: false

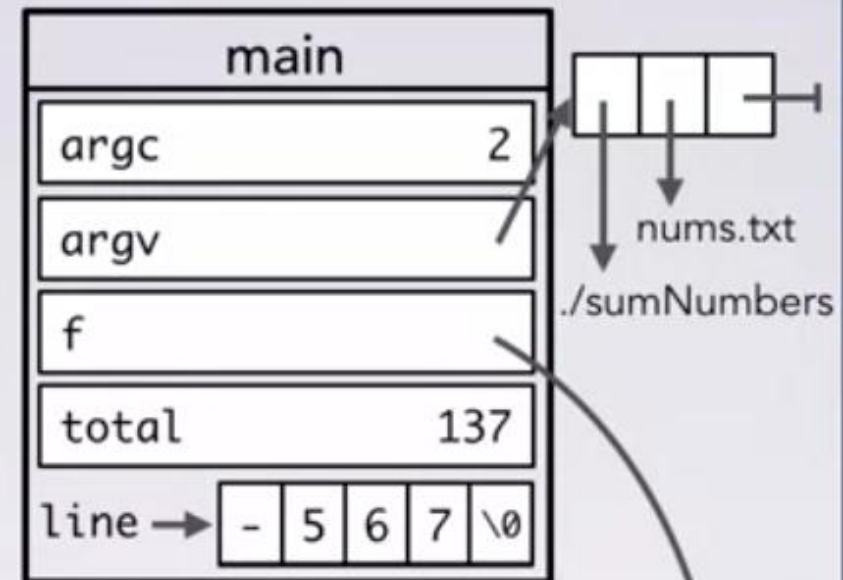
123  
14  
-56789  
77

### Output

--

```
#define LINE_SIZE 5
```

```
int main (int argc, char ** argv) {  
    if (argc != 2) { /* omitted */ }  
    FILE * f = fopen(argv[1], "r");  
    if (f == NULL) { /* omitted */ }  
    long total = 0;  
    char line[LINE_SIZE];  
    while (fgets(line, LINE_SIZE, f) != NULL) {  
        if (strchr(line, '\n') == NULL) {  
            printf("Line is too long!\n");  
            return EXIT_FAILURE;  
        }  
        total += atoi(line);  
    }  
    printf("The total is %ld\n", total);  
    return EXIT_SUCCESS;  
}
```



file: nums.txt  
mode: read  
eof: false

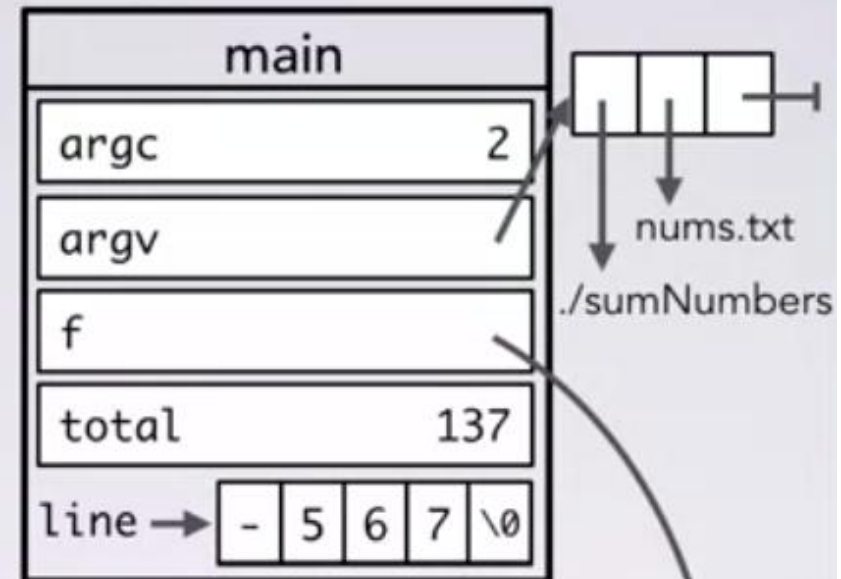
123  
14  
-56789  
77

### Output



```
#define LINE_SIZE 5
```

```
int main (int argc, char ** argv) {  
    if (argc != 2) { /* omitted */ }  
    FILE * f = fopen(argv[1], "r");  
    if (f == NULL) { /* omitted */ }  
    long total = 0;  
    char line[LINE_SIZE];  
    while (fgets(line, LINE_SIZE, f) != NULL) {  
        if (strchr(line, '\n') == NULL) {  
            printf("Line is too long!\n");  
            return EXIT_FAILURE;  
        }  
        total += atoi(line);  
    }  
    printf("The total is %ld\n", total);  
    return EXIT_SUCCESS;  
}
```



file: nums.txt  
mode: read  
eof: false

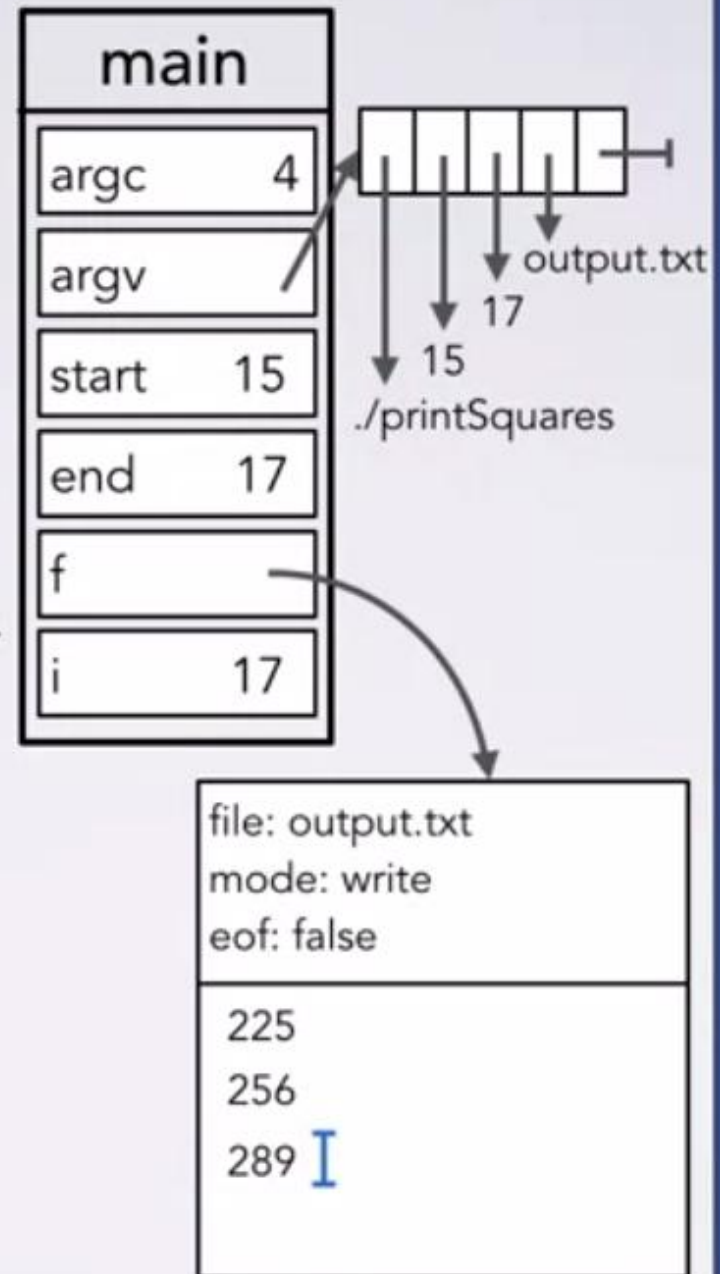
123  
14  
-56789  
77

### Output

```

int main (int argc, char ** argv) {
    if (argc != 4) { /* omitted */ }
    int start = atoi(argv[1]);
    int end = atoi(argv[2]);
    FILE * f = fopen(argv[3], "w");
    if (f == NULL) { /* omitted */ }
    for (int i = start; i <= end; i++) {
        fprintf(f, "%d\n", i*i);
    }
    //fclose discussed in next section
    if (fclose(f) != 0) { /* omitted */ }
    return EXIT_SUCCESS;
}

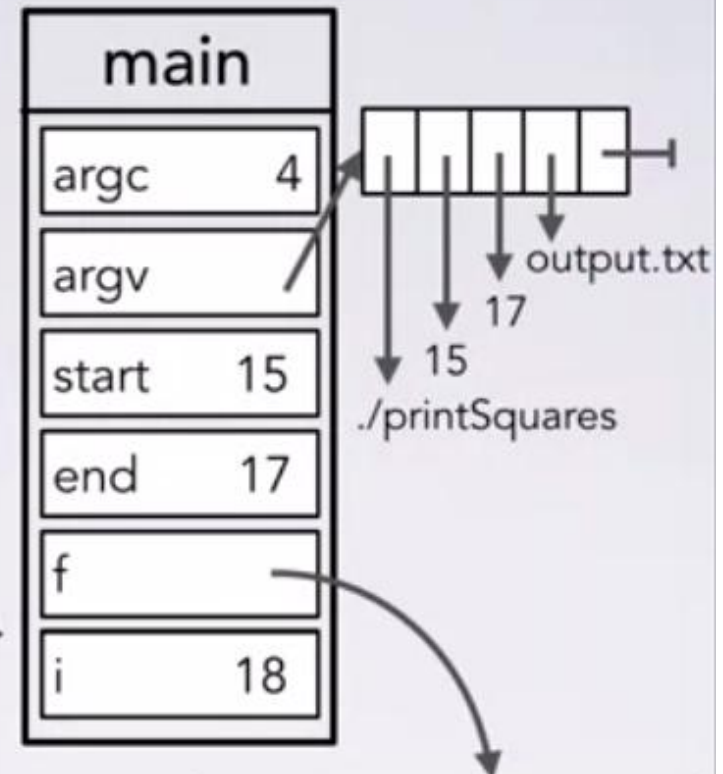
```



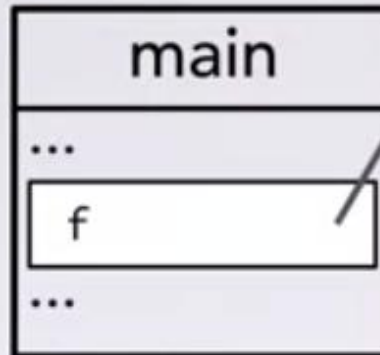
```

int main (int argc, char ** argv) {
    if (argc != 4) { /* omitted */ }
    int start = atoi(argv[1]);
    int end = atoi(argv[2]);
    FILE * f = fopen(argv[3], "w");
    if (f == NULL) { /* omitted */ }
    for (int i = start; i <= end; i++) {
        fprintf(f, "%d\n", i*i);
    }
    //fclose discussed in next section
    if (fclose(f) != 0) { /* omitted */ }
    return EXIT_SUCCESS;
}

```



→ ...  
if (fclose(f) != 0) { /\* omitted \*/ }  
...



file: output.txt  
mode: write  
eof: false

225  
256  
289

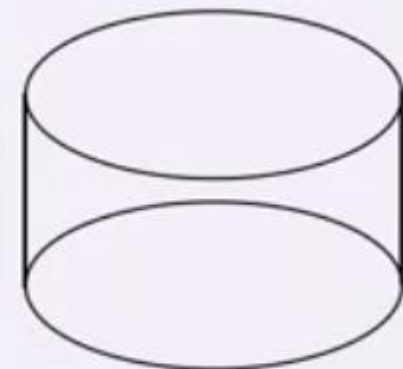


Kernel:

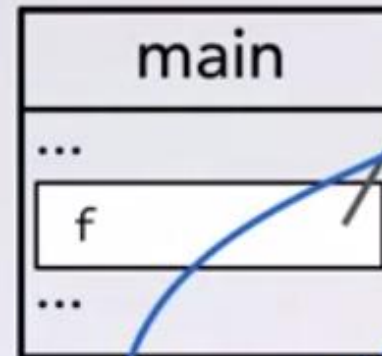
fd: 3  
disk loc: 1234  
mode: write



Hardware (disk):



...  
→ if (fclose(f) != 0) { /\* omitted \*/ }  
...



file: output.txt  
mode: write  
eof: false

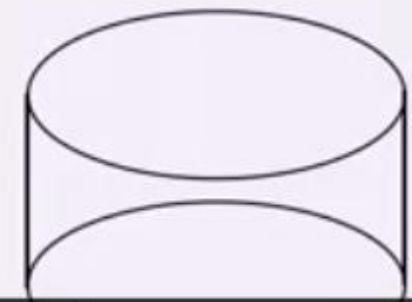
225  
256  
289

Kernel:

fd: 3  
disk loc: 1234  
mode: write  
3232 350a 3235  
360a 3238 390a

write (3, 12);

Hardware (disk):

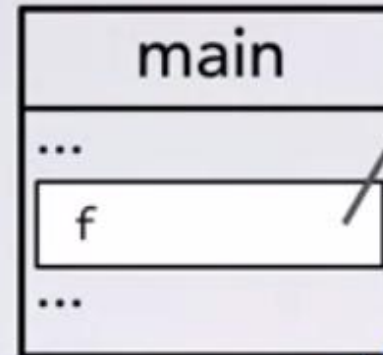


the bytes rather than the textual representa





```
...  
if (fclose(f) != 0) { /* omitted */ }  
...
```



file: output.txt
mode: write
eof: false
225
256
289

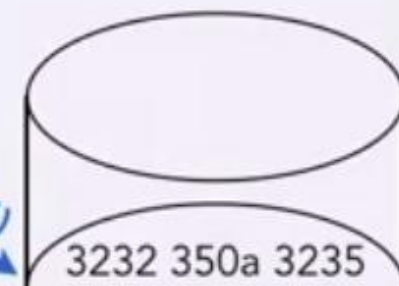
Kernel:

fd: 3
disk loc: 1234
mode: write
3232 350a 3235
360a 3238 390a

*close(3);*

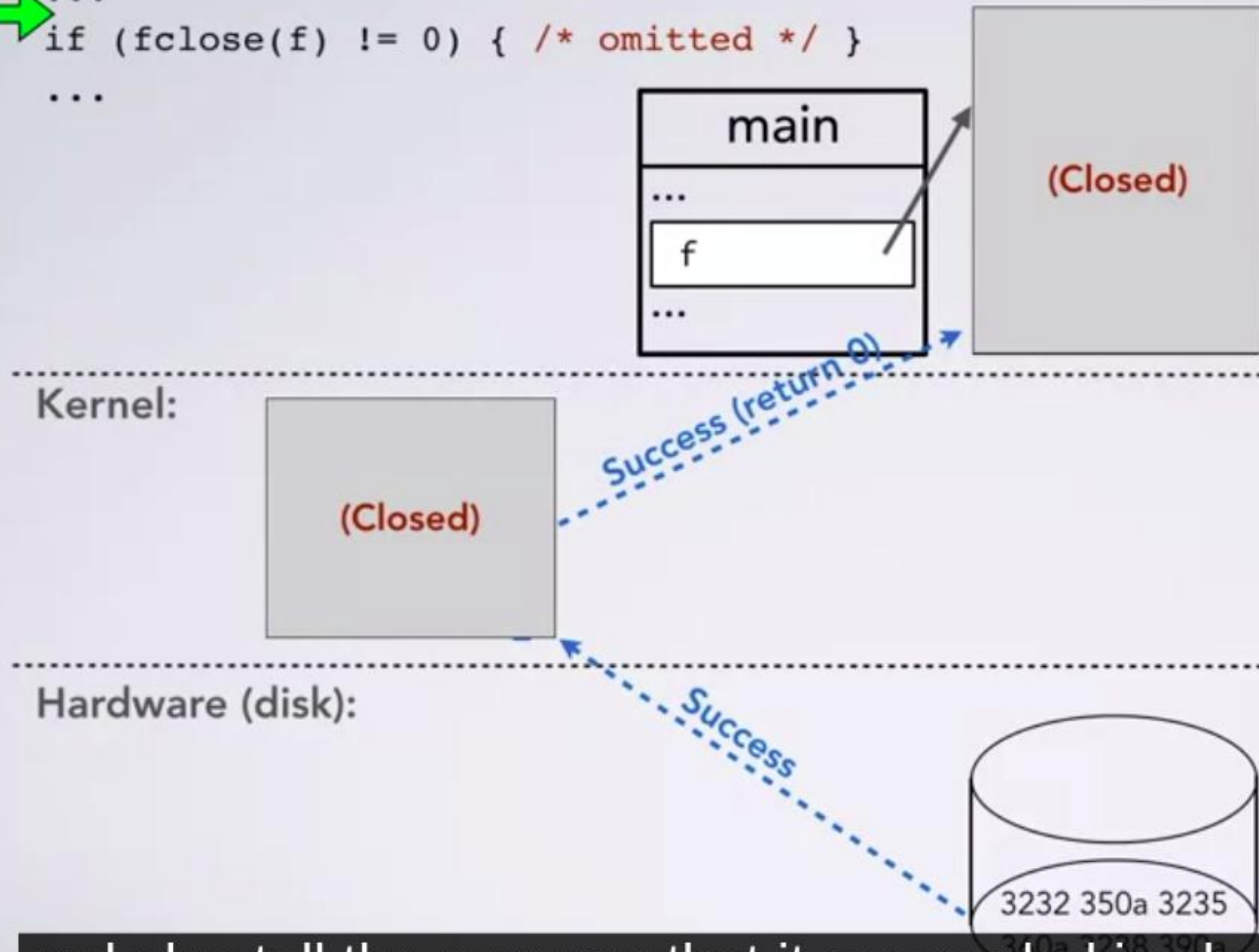
Hardware (disk):

*write(0x3232...390a to 1234)*

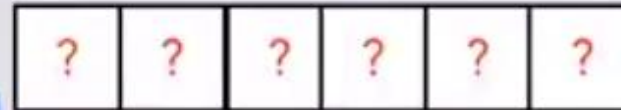
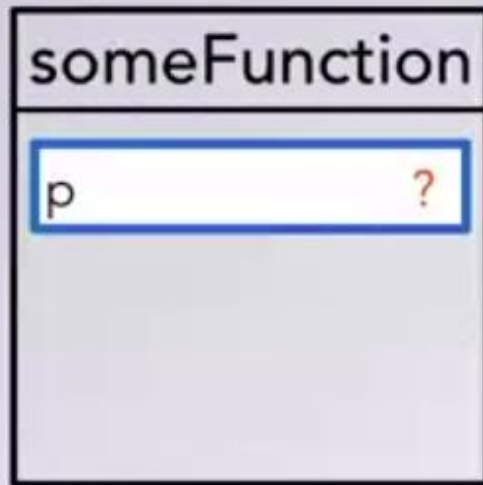


decide that it should write its buffered data

```
...  
→ if (fclose(f) != 0) { /* omitted */ }  
...
```



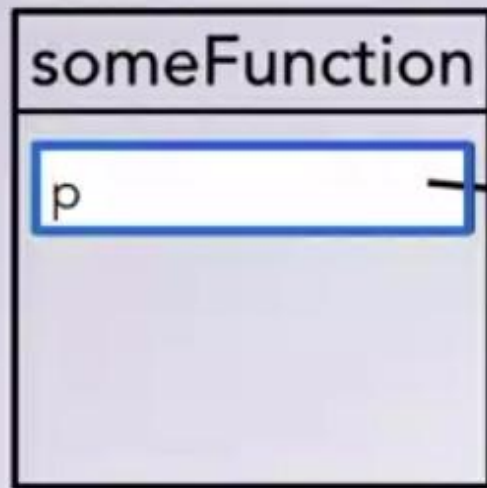
and also tell the program that it succeeded in closing



```
int * p; return value of malloc  
→ p = malloc (6 * sizeof(*p));
```

```
.....  
return;
```

So, let's draw that box.



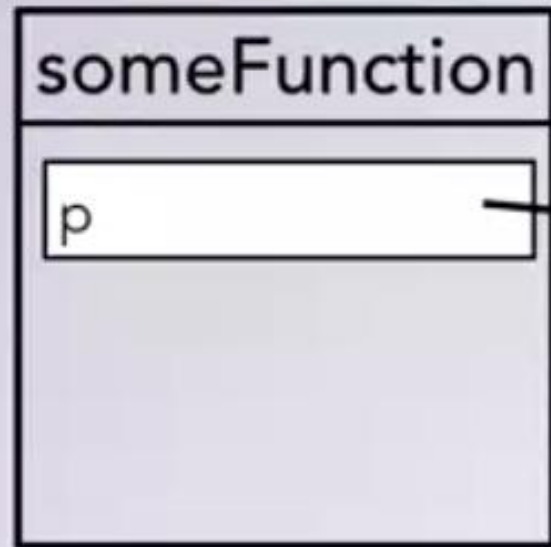
```
int * p; return value of malloc  
p = malloc (6 * sizeof(*p));
```



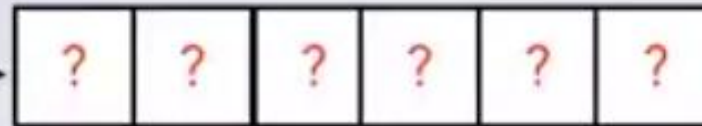
```
.....  
return;
```

which is the value of the function call in

Stack



Heap



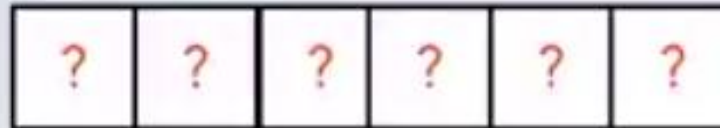
```
int * p;  
p = malloc (6 * sizeof(*p));
```

➡ .....  
return;



Stack

Heap



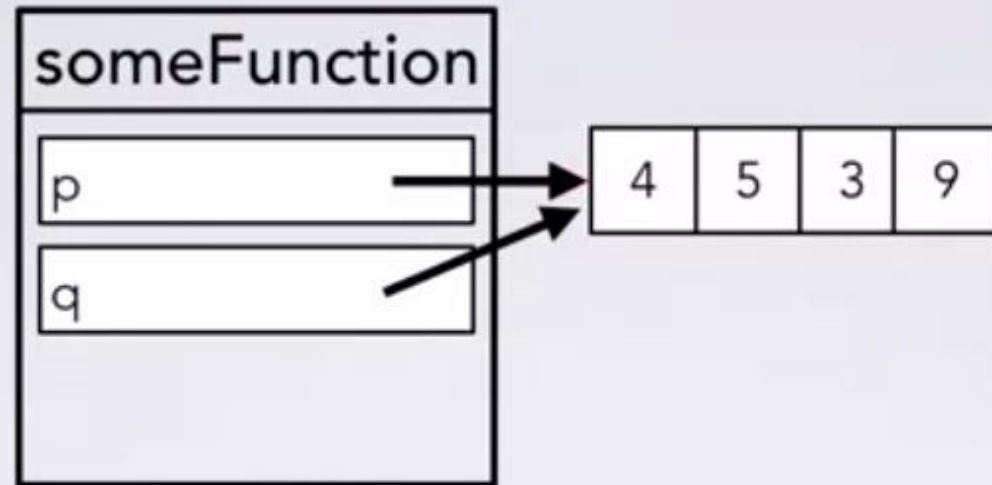
```
int * p;  
p = malloc (6 * sizeof(*p));
```

```
.....  
return;
```

```
int * p = malloc(4 * sizeof(*p));  
p[0] = 4;  
p[1] = 5;  
p[2] = 3;  
p[3] = 9;  
int * q = p;
```

//.....

➡ free(p);

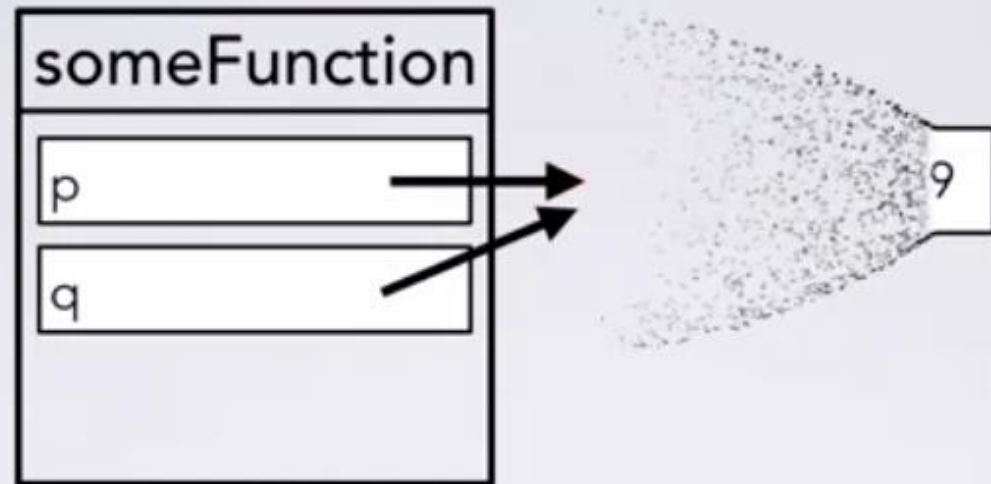


Free p does not actually affect the memory that p points to, but rather the memory that p points to.

```
int * p = malloc(4 * sizeof(*p));  
p[0] = 4;  
p[1] = 5;  
p[2] = 3;  
p[3] = 9;  
int * q = p;
```

```
//.....
```

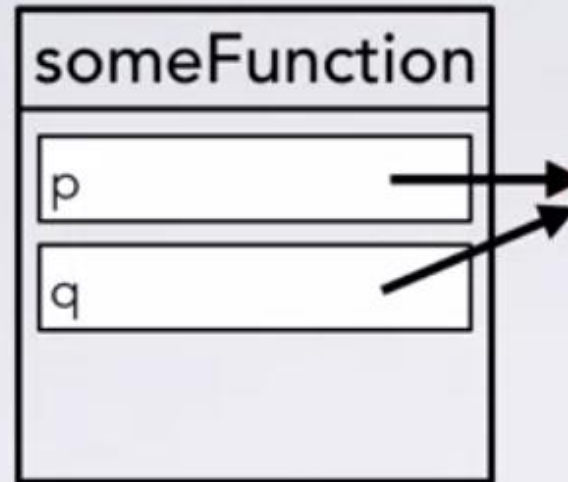
```
→ free(p);
```



Freeing this memory destroys that memory, leaving p dangling.

```
int * p = malloc(4 * sizeof(*p));  
p[0] = 4;  
p[1] = 5;  
p[2] = 3;  
p[3] = 9;  
int * q = p;  
  
//.....
```

```
free(p);
```



as with any dangling pointer.

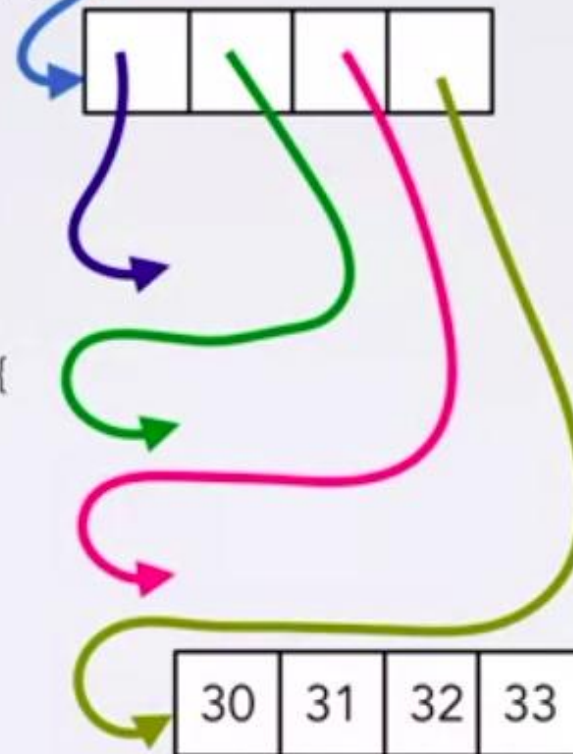
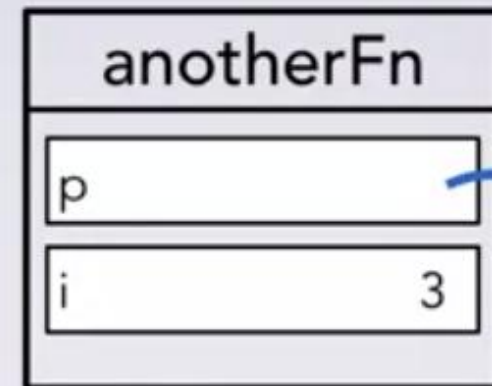
```

int ** p = malloc(4 * sizeof(*p));
for (size_t i = 0; i < 4; i++) {
    size_t s = i + 1;
    p[i] = malloc(s * sizeof(*p[i]));
    for (size_t j = 0; j < s; j++) {
        p[i][j] = i*10 + j;
    }
}

//.....

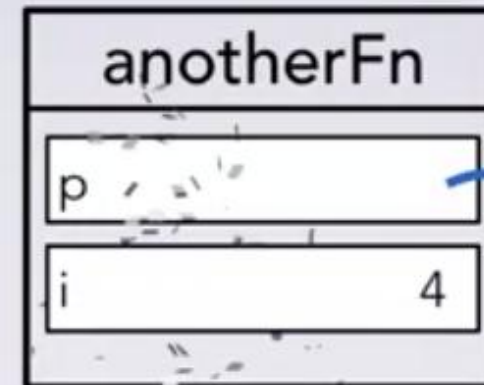
for (size_t i = 0; i < 4; i++) {
    → free(p[i]);
}
free(p);

```





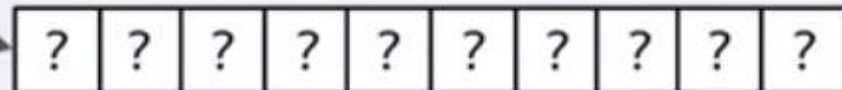
```
int ** p = malloc(4 * sizeof(*p));  
for (size_t i = 0; i < 4; i++) {  
    size_t s = i + 1;  
    p[i] = malloc(s * sizeof(*p[i]));  
    for (size_t j = 0; j < s; j++) {  
        p[i][j] = i*10 + j;  
    }  
}  
  
//.....  
  
for (size_t i = 0; i < 4; i++) {  
    free(p[i]);  
}  
free(p);
```



```
int main (void) {  
    int x = 0;  
    for (int i = 10; i < 100; i++) {  
        int * p = malloc(i * sizeof(*p));  
        x = doSomeComputation(x, i, p);  
    }  
    printf("Answer %d\n", x);  
    return EXIT_SUCCESS;  
}
```

main	
x	0
i	10
p	

Example without free



```

int main (void) {
    int x = 0;
    → for (int i = 10; i < 100; i++) {
        int * p = malloc(i * sizeof(*p));
        x = doSomeComputation(x, i, p);
    }
    printf("Answer %d\n", x);
    return EXIT_SUCCESS;
}

```

main	
x	55
i	11

Example without free

*Leaked memory!*

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

```
int main (void) {  
    int x = 0;  
    for (int i = 10; i < 100; i++) {  
        → int * p = malloc(i * sizeof(*p));  
        x = doSomeComputation(x, i, p);  
    }  
    printf("Answer %d\n", x);  
    return EXIT_SUCCESS;  
}
```

main	
x	55
i	11
p	

Example without free



```
int main (void) {  
    int x = 0;  
    for (int i = 10; i < 100; i++) {  
        int * p = malloc(i * sizeof(*p));  
        x = doSomeComputation(x, i, p);  
        free(p);  
    }  
    printf("Answer %d\n", x);  
    return EXIT_SUCCESS;  
}
```

main	
x	55
i	10
p	

Example with free

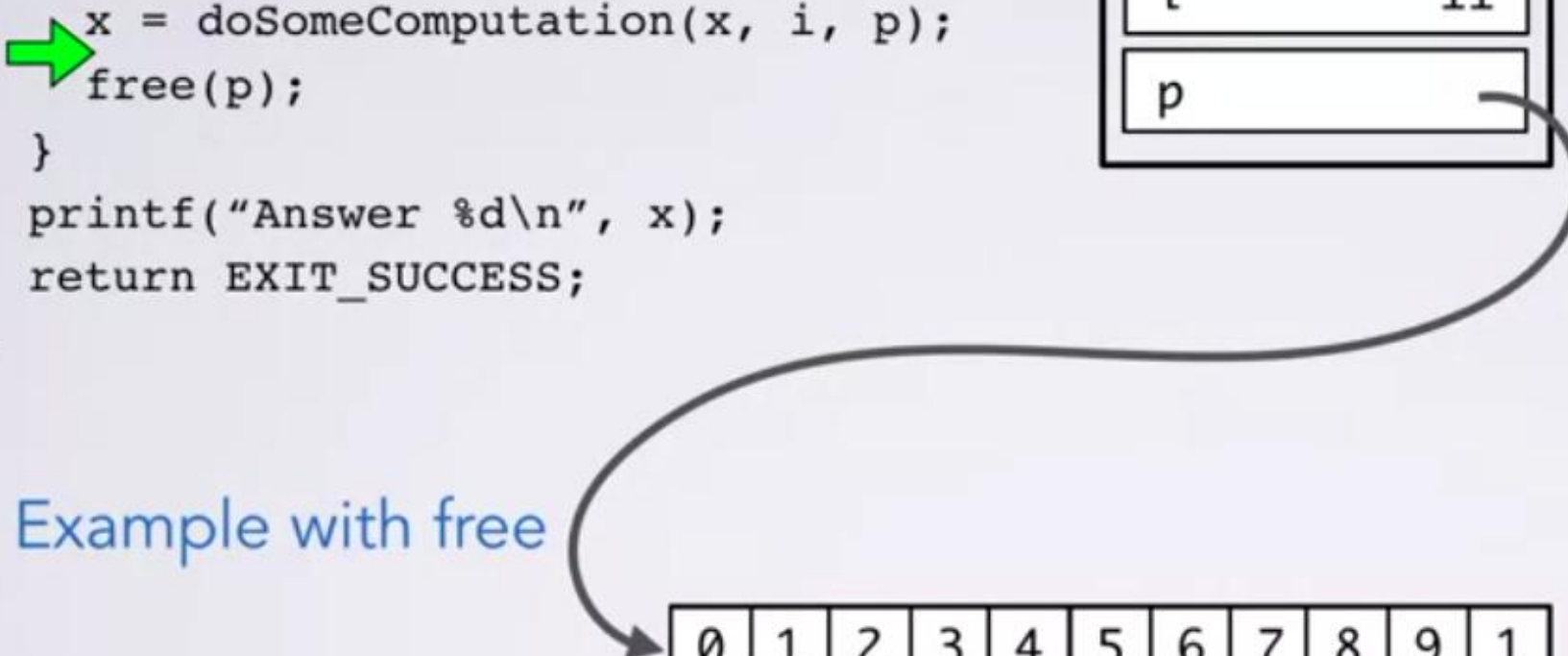




```
int main (void) {  
    int x = 0;  
    for (int i = 10; i < 100; i++) {  
        int * p = malloc(i * sizeof(*p));  
        x = doSomeComputation(x, i, p);  
        free(p);  
    }  
    printf("Answer %d\n", x);  
    return EXIT_SUCCESS;  
}
```

main	
x	928
i	11
p	

Example with free



0	1	2	3	4	5	6	7	8	9	1
---	---	---	---	---	---	---	---	---	---	---

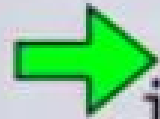
```
int main (void) {  
    int x = 0;  
    for (int i = 10; i < 100; i++) {  
        int * p = malloc(i * sizeof(*p));  
        x = doSomeComputation(x, i, p);  
        → free(p);  
    }  
    printf("Answer %d\n", x);  
    return EXIT_SUCCESS;  
}
```

main	
x	928
i	11
p	

Example with free



# Double Free



```
int * p = malloc(4 * sizeof(*p));
```

```
int * q = p;
```

```
...
```

```
free(q);
```

```
...
```

```
free(p);
```

# Free Memory Not in the Heap

```
→ int x = 3;  
  int * p = &x;  
  ...  
  free(p);
```

