COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 8- Class 1: File I/O

Administrivia

- *Large Programming Assignment 1* is due on Friday!!
- SPA 1 & 2 Grades will be available tomorrow.

Today's Agenda

1. Quick Recap of past concepts
2. File I/O in C!!

Quick Review

What are Pointers?

- Pointers are variables that store the address of a memory cell that contains a certain data type.
- * indicates that variable holds a memory location of certain type
- & is the address

```
int m = 25; // stored in address AA0

    int *itemp = &m;
```

Stack	Space
AA3	
AA2	
AA1	itemp = AA0
AA0	m = 25

The Dreference Operator *

- We have seen so far in this course that everything is stored somewhere in memory.
- Each memory has its own unique address.
- The pointer variable holds the specific address.
- The dereference operator acts like a “magic key” that allows access to the value stored.
- * is known as deference in C.

The Address Operator &

- We have been using & in our programs ever since scanf was introduced.
- & means address of
- Holds a value in hexadecimal that represents the location in memory.
  - This done with the placeholder **%p**.
  - Hexadecimal is a base 16 number. This means there are 16 unique digits.
- Think about it. Every time we used `scanf("%d", &num)` we were telling the compiler to store the value at the *Memory Address* of the variable named num.

The Pointer Placeholder %p

- There exists a special placeholder that can display the memory address of a reference.

```c
int m = 25; // stored in address AA0

int *itemp = &m;

printf("The address of m is %p\n", &m);
printf("The address of itemp is %p\n", &itemp);
printf("itemp holds the value %p\n", itemp);
```

Functions with Parameters

- In past sessions, we have seen that variables have been passed by value.
- With pointers, we can now past variables by reference.
- Instead of making a local copy for the function, we can pass the memory location and perform computation on the variable in its original location. This is known as pass-by-reference.

Scope of Names

- Scope of a name refers to the region in a program where a particular meaning of a name is visible.
- Local and Global Variables
- When variables are being used, certain functions may not be able to access them due to where they were declared!
- Why can’t everything be global? Would that be easier?

```c
#include <stdio.h>

void increaseValue(int *num);
void calculate();

int var; // global variable BAD!!

int main(void){

    int num = 13;

    printf("num = %d\n", num);

return 0;

}

void calculate(){

    int num1;   // local variable
    int num2;   // local variable
    scanf("%d%d", &num1, &num2);

    int result = num1 + num2;

}
```

File I/O in C

Memory 101 Concepts

- Everything in memory has an address. (represented in hexadecimal)
- When we accessed a value from a variable name we were able to directly access that exact space in memory the value is stored at.
- In previous lessons, we have named spaces in memory which was used to access the values stored.
- In C, we can also access parts of memory indirectly through pointers!
- *Pointer* – a memory cell that sores the address (hexadecimal) of a data item

Files

- In C we can access files (such as text files)
- This access allows for reading and writing.
  - Reading – Input
  - Writing – Output
- There is a special kind of variable in C that allows us access for text files.
- *File Pointers!*

```
FILE *inp; // pointer to input file
FILE *outp; // pointer to output file
```

File Pointer Access

- There are two basic types of access we will learn in this class
  - *Reading* – this allows the program to collect input from a text file. Think of it like scanf for collecting input from the keyboard
  - *Writing* – this allows the program to write output to a text file. Think of it like printf for displaying output to the monitor

Other Types of File I/O Access

- There are other modes for FILE I/O Access besides r and w mode.
  - *a – append mode*
    - Adds content to the next available space in the File
  - *r+ – both reading and writing*
    - Acts as both r and w mode. Assumes that File exists in memory
    - If file does not exist then it doesn't work
  - *w+ – both reading and writing*
    - Acts as both and w mode. Doesn't assume that File exist in memory
    - If it does exist already, content will be deleted by setting the length to zero bytes
    - If it doesn't exist, it will create the File
  - *a+ – both reading and writing*
    - If file doesn't exist, it will create it
    - When reading, pointer starts at the beginning of the file content
    - Writing to file will only be appended

Syntax for Allowing Proper File Access

```
// preparing files for input and output
inp = fopen("indata.txt", "r");
outp = fopen("outdata.txt", "w");
```

Syntax for File Reading/Writing

```
// preparing files for input and output
inp = fopen("indata.txt", "r");
outp = fopen("outdata.txt", "w");
```

```
fscanf(inp, "%lf", &item); // reading file
fprintf(outp, "%f", item); // writing file
```

printf, scanf, fprintf, and fscanf

```c
FILE *inp; // pointer to input file
FILE *outp; // pointer to output file

// preparing files for input and output
inp = fopen("indata.txt", "r");
outp = fopen("outdata.txt", "w");

scanf("%lf", &item); // reading input from command line
fscanf(inp, "%lf", &item); // reading input from file

printf("%f", item); // printing information to command line
fprintf(outp, "%f", item); // writing file

fclose(inp);
fclose(outp);
```

printf, scanf, fprintf, and fscanf

```
FILE *inp; // pointer to input file
FILE *outp; // pointer to output file

// preparing files for input and output
inp = fopen("indata.txt", "r");
outp = fopen("outdata.txt", "w");
```

Notice the placeholder and variable address

```
scanf("%lf", &item); // reading input from command line
fscanf(inp, "%lf", &item); // reading input from file
```

Notice the placeholder and variable

```
printf("%f", item); // printing information to command line
fprintf(outp, "%f", item); // writing file

fclose(inp);
fclose(outp);
```

printf, scanf, fprintf, and fscanf

```c
FILE *inp; // pointer to input file
FILE *outp; // pointer to output file

// preparing files for input and output
inp = fopen("indata.txt", "r");
outp = fopen("outdata.txt", "w");
```

Notice the placeholder and variable address

```c
scanf("%lf", &item); // reading input from command line
fscanf(inp, "%lf", &item); // reading input from file
```

Notice the placeholder and variable

```c
printf("%f", item); // printing information to command line
fprintf(outp, "%f", item); // writing file

fclose(inp);
fclose(outp);
```

The only addition is the file pointer!

The `fscanf` Function

- Remember way back we talked about what the `scanf` function returns?
  - An integer value representing the number of values successfully processed.
- We just observed the similar syntax for the `scanf` and `fscanf` functions.
- Does `fscanf` return a similar value?
  - YES!!
  - It returns the number of values processed successfully. This also includes 0 if it was unable to process the first value being collected.

EOF Macro Constant

- C has a special ***predefined*** macro constant called EOF in the stdio header file.
- EOF stands for "*E*nd *O*f *F*ile"
  - The value of EOF is –1. 0 is still used if it can read something potential, BUT wasn't processed successfully.
- EOF is widely used to assist with reading an ENTIRE file.

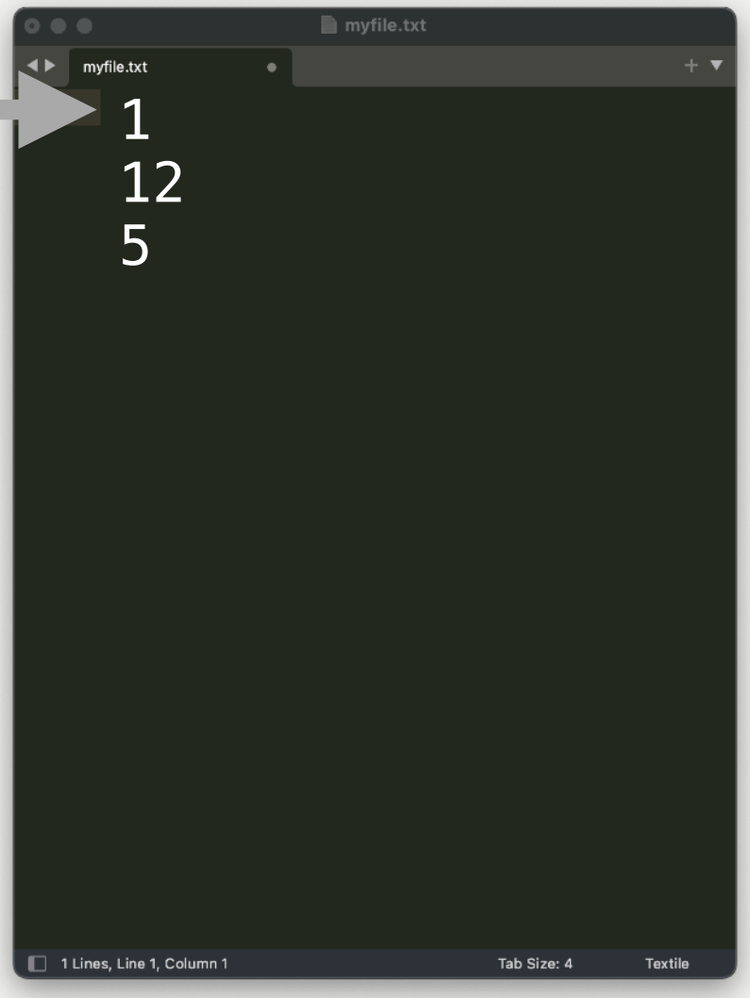
```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, "%lf", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

EOF Example

Here

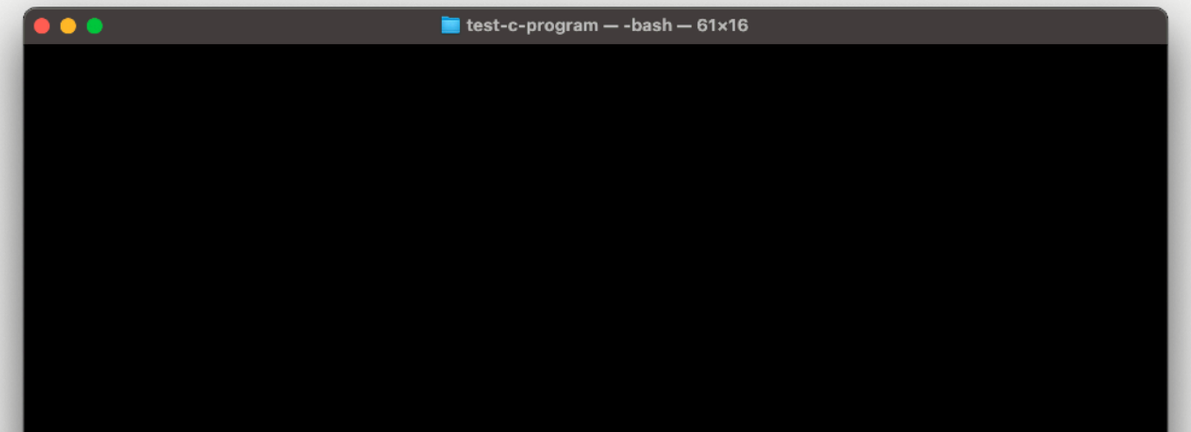
```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



File pointer has access to contents of the text file.

Stack	Space
AA1	inp = Some Address
AA0	

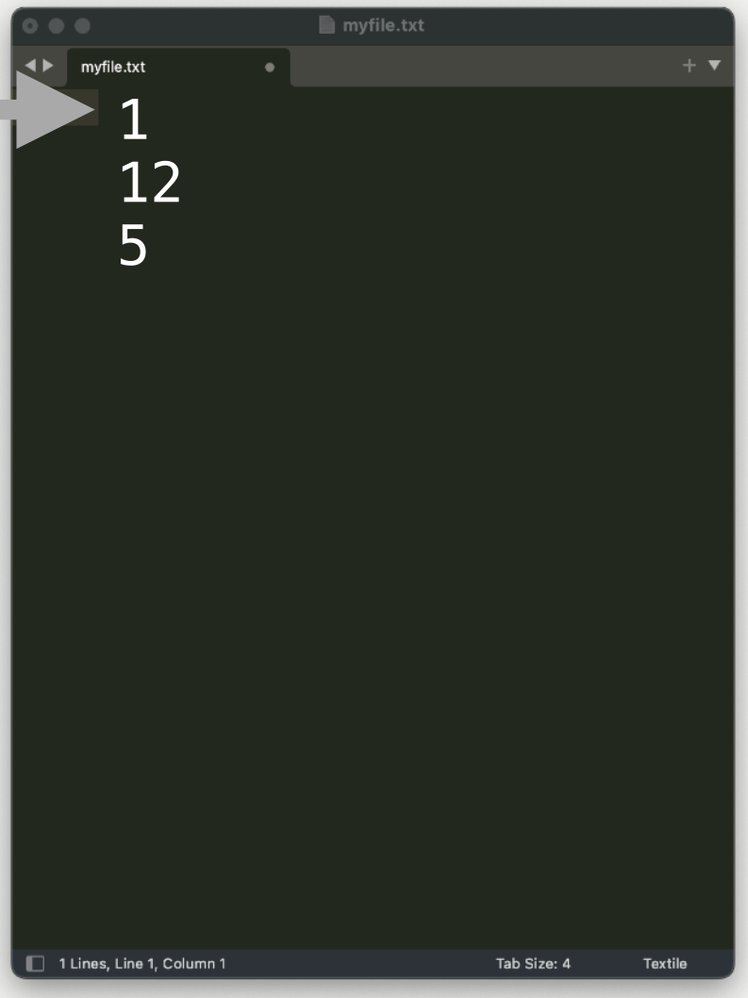


EOF Example

Here

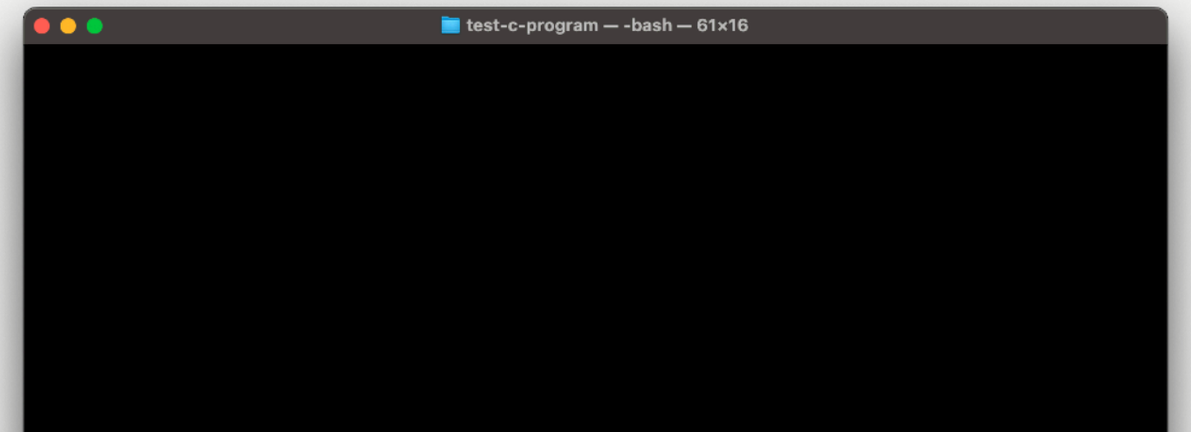
```c
FILE *inp = fopen("indata.txt", "r");
int item;
while(fscanf(inp, “%d", &item) != EOF){
    printf("item = %d\n", item);
}
fclose(inp);
```

inp



Integer variable called, item declared in stack space.

Stack	Space
AA1	inp = Some Address
AA0	item = ???

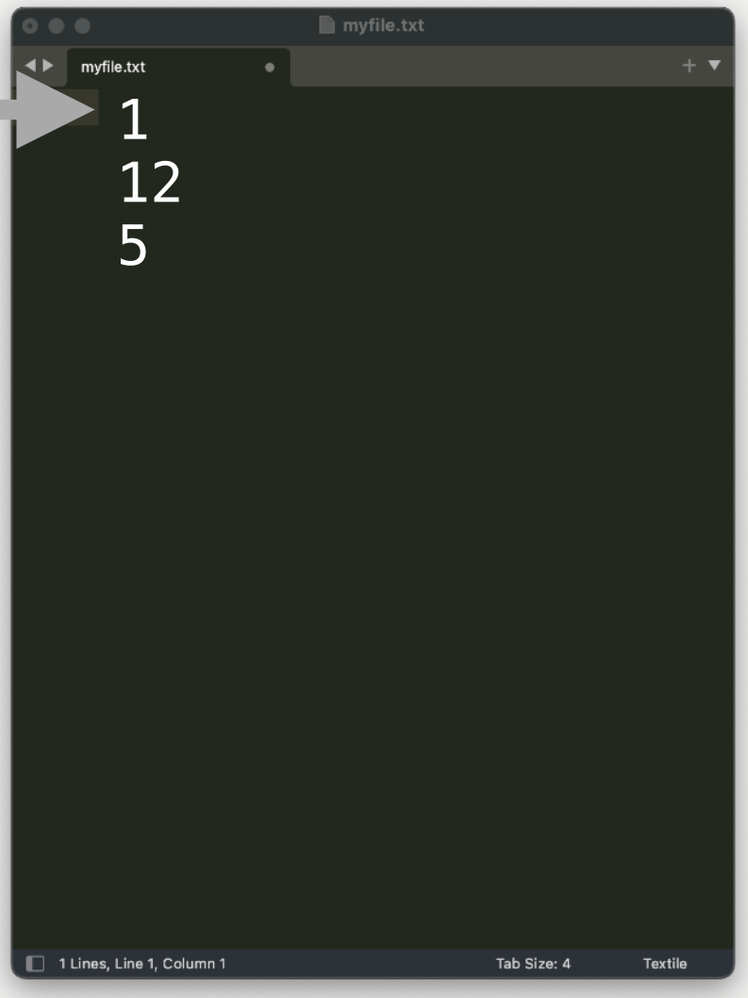


EOF Example

Here

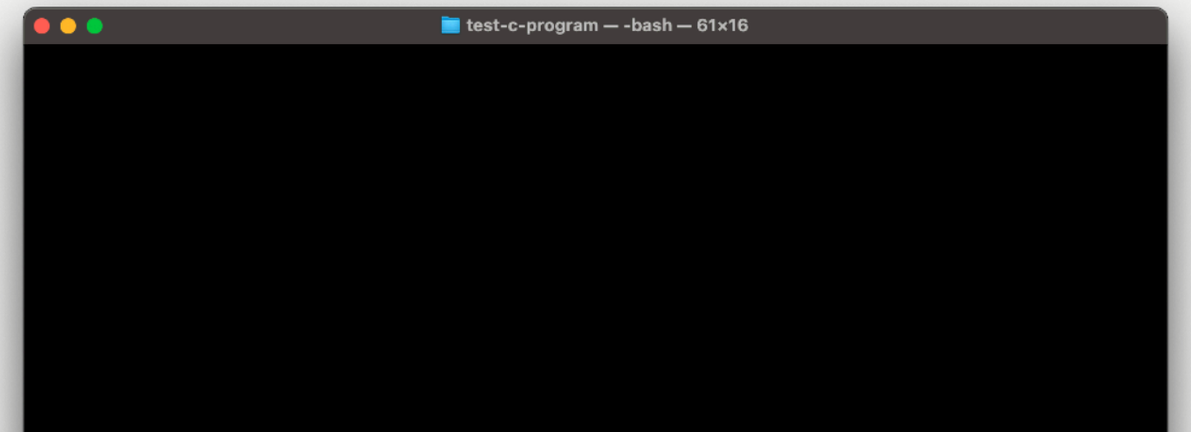
```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



fscanf is called and can read the value 1 successfully, which results in 1 being returned. The while loop condition is true.

Stack	Space
AA1	inp = Some Address
AA0	item = 1

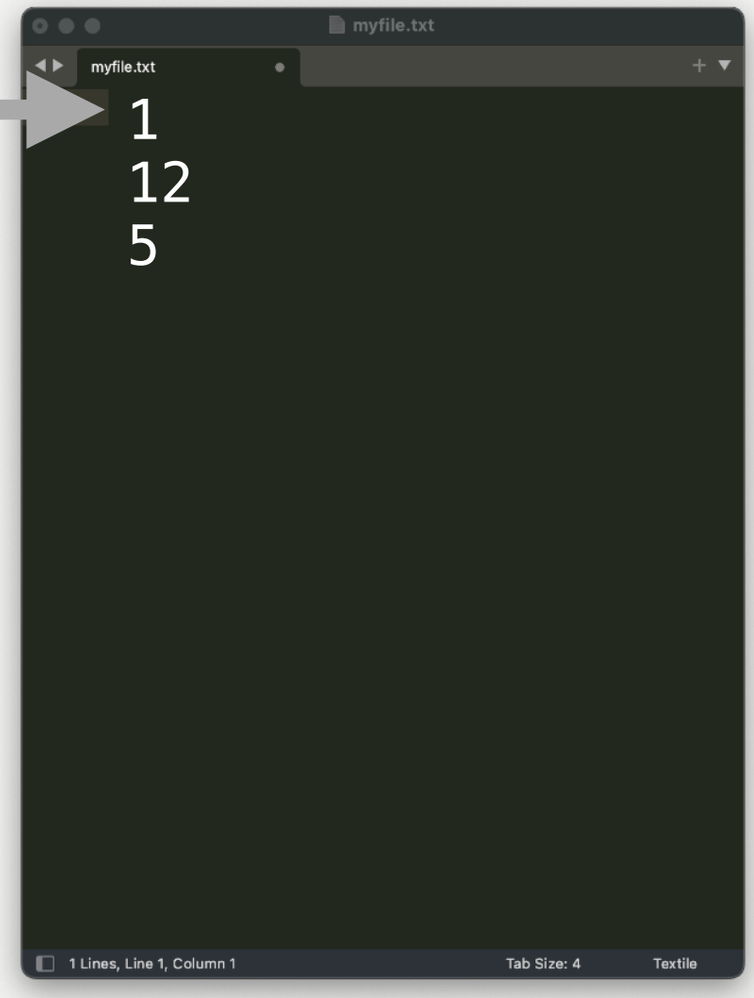


EOF Example

```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

printf("item = %d\n", item);

}

fclose(inp);
```

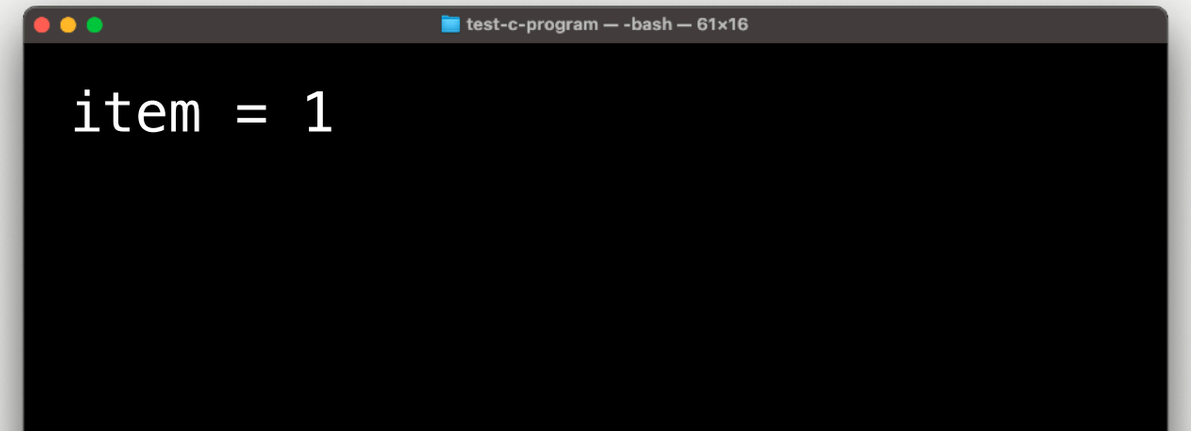
Here

inp



Display the value stored in item.

Stack	Space
AA1	inp = Some Address
AA0	item = 1



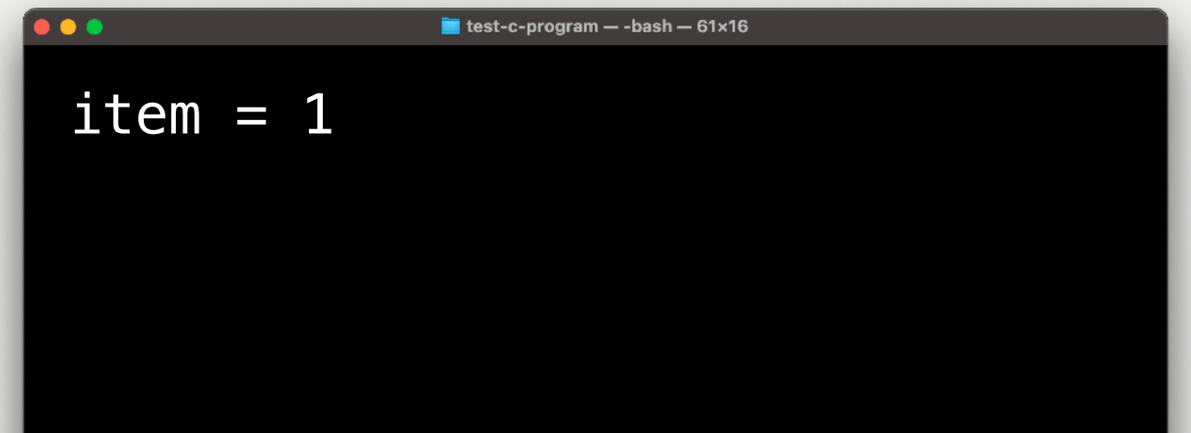
EOF Example

```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

Here

inp

Stack	Space
AA1	inp = Some Address
AA0	item = 1

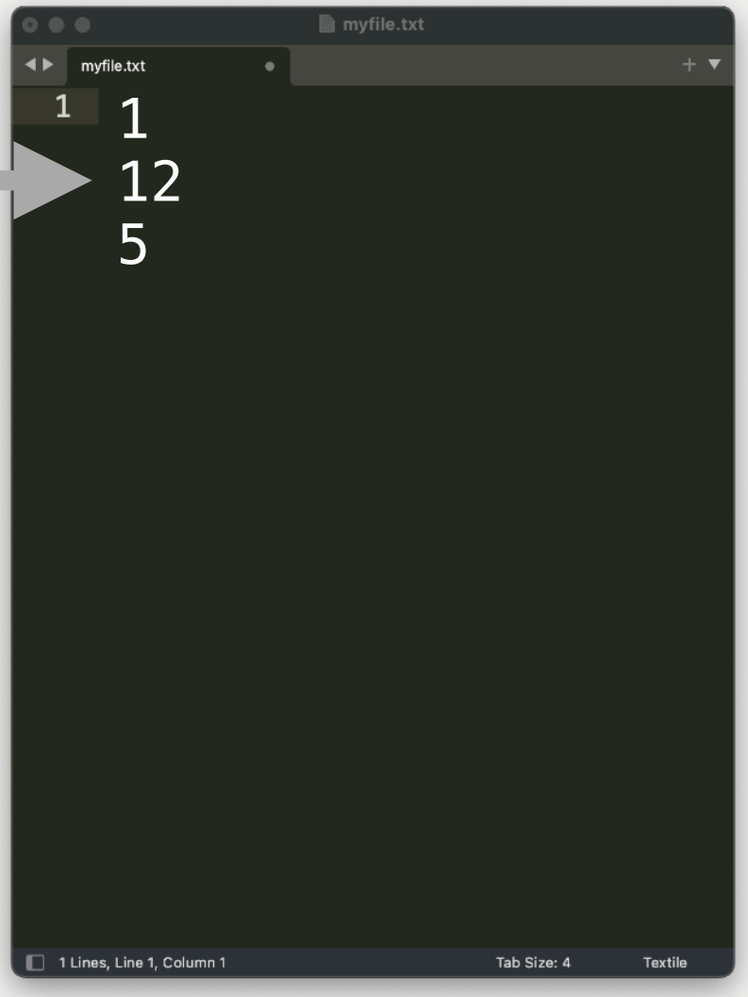


EOF Example

Here

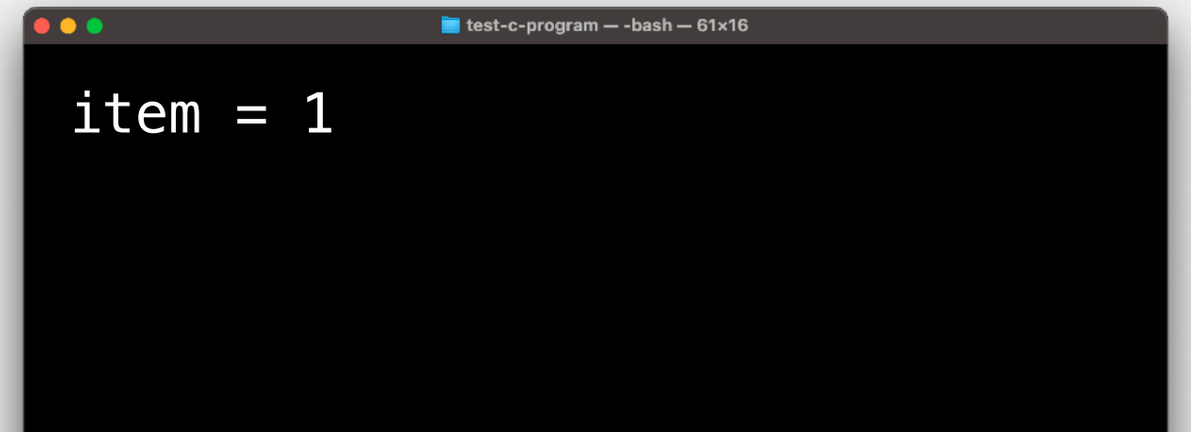
```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



True

Stack	Space
AA1	inp = Some Address
AA0	item = 1

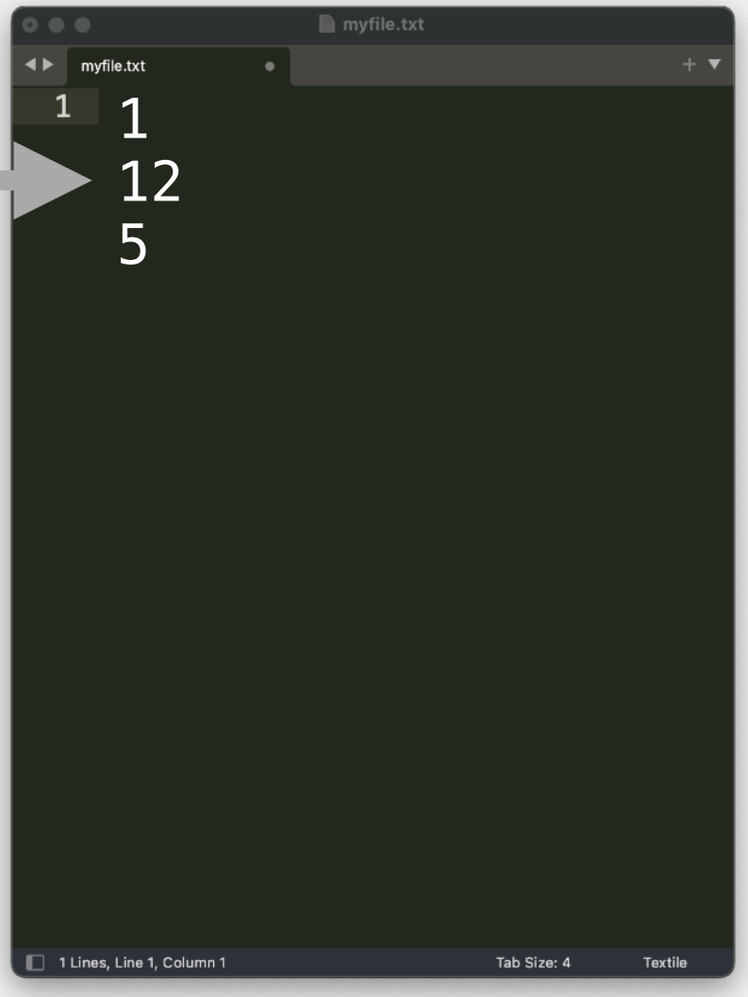


EOF Example

Here

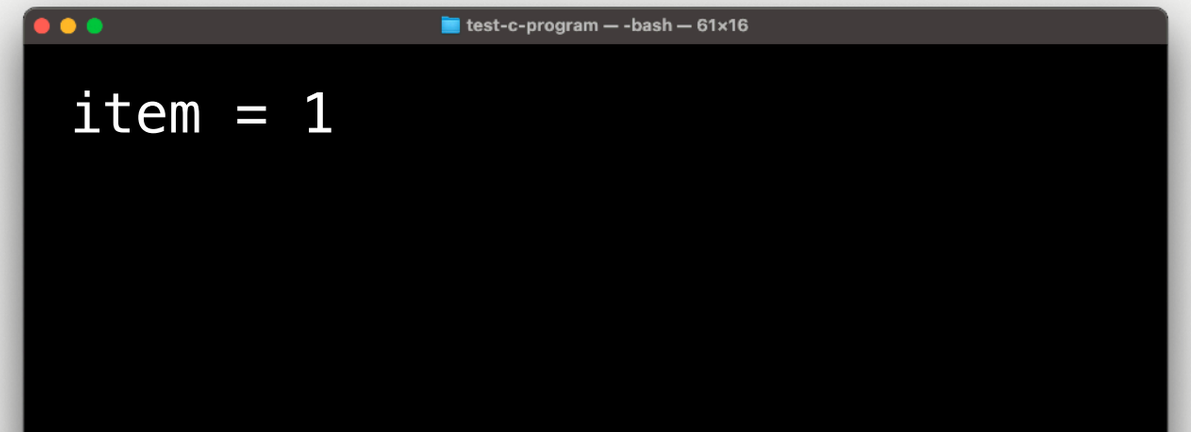
```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



fscanf is called and can read the value 12 successfully, which results in 1 being returned. The while loop condition is true.

Stack	Space
AA1	inp = Some Address
AA0	item = 12

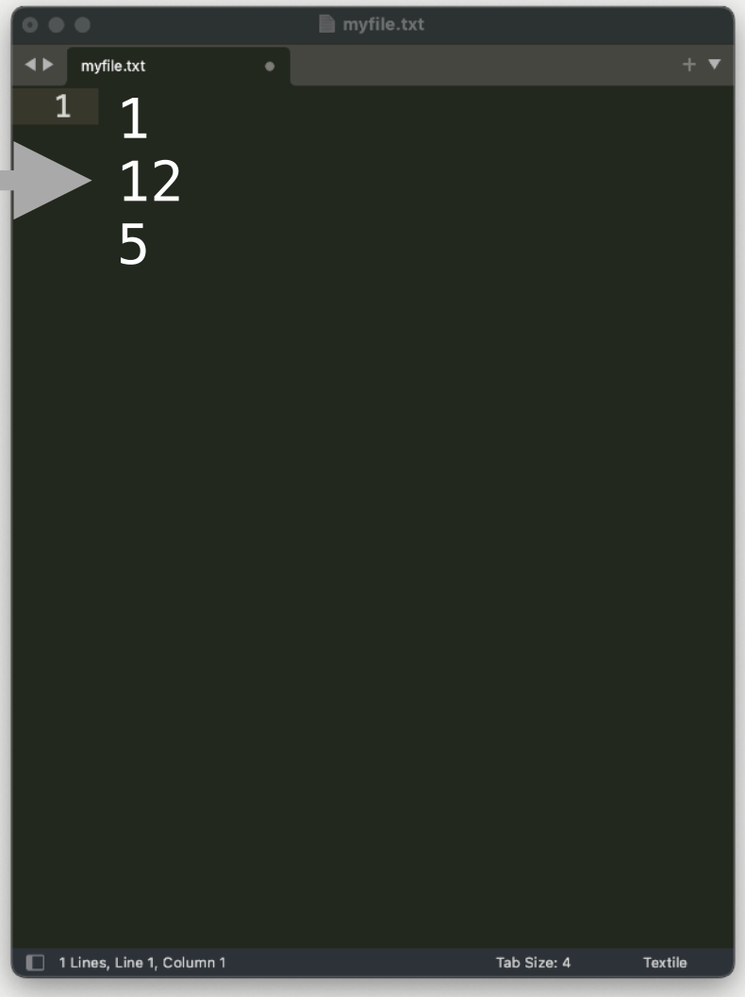


EOF Example

```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){
    printf("item = %d\n", item);
}

fclose(inp);
```

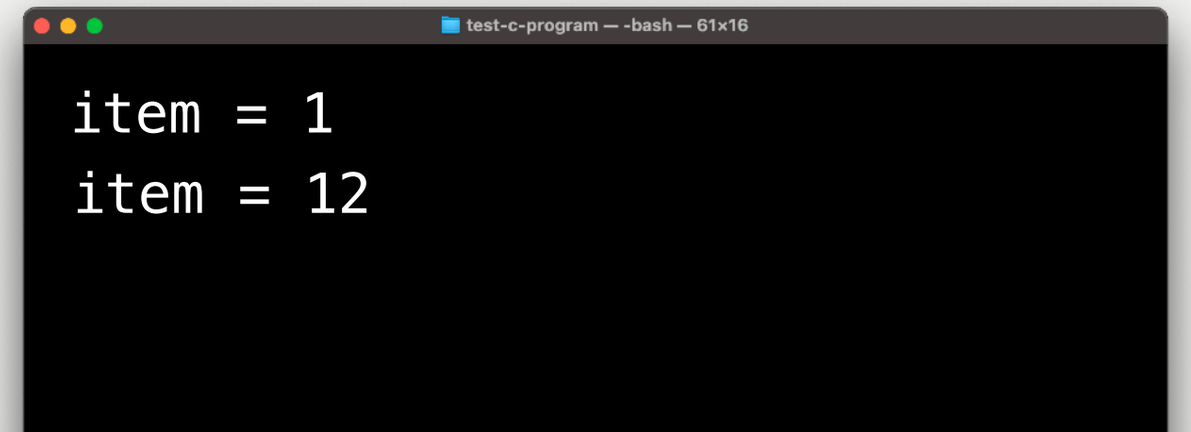
Here

inp



Display the value stored in item.

Stack	Space
AA1	inp = Some Address
AA0	item = 12

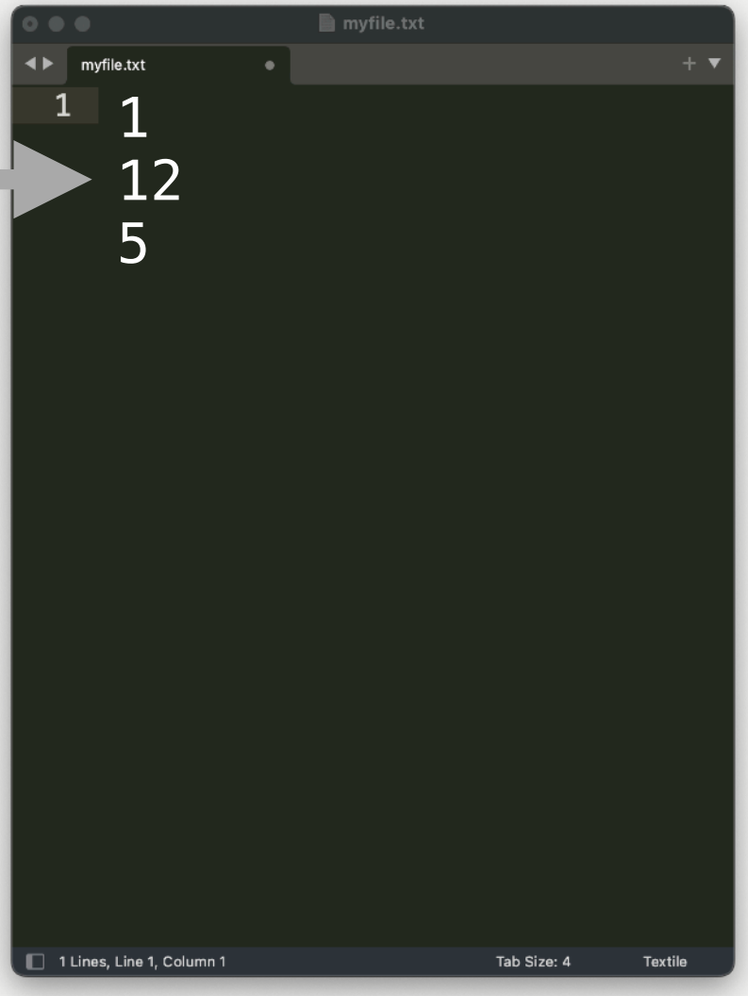


EOF Example

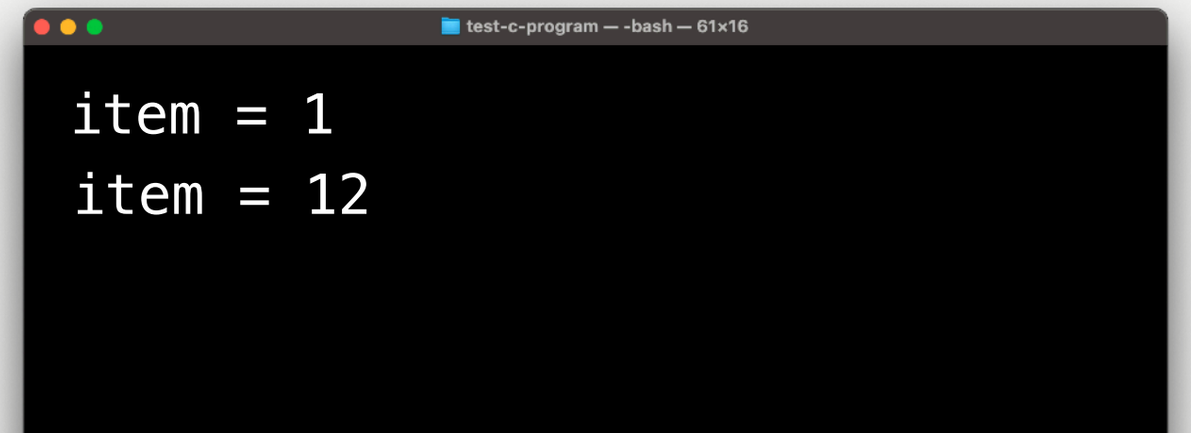
```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

Here

inp



Stack	Space
AA1	inp = Some Address
AA0	item = 12

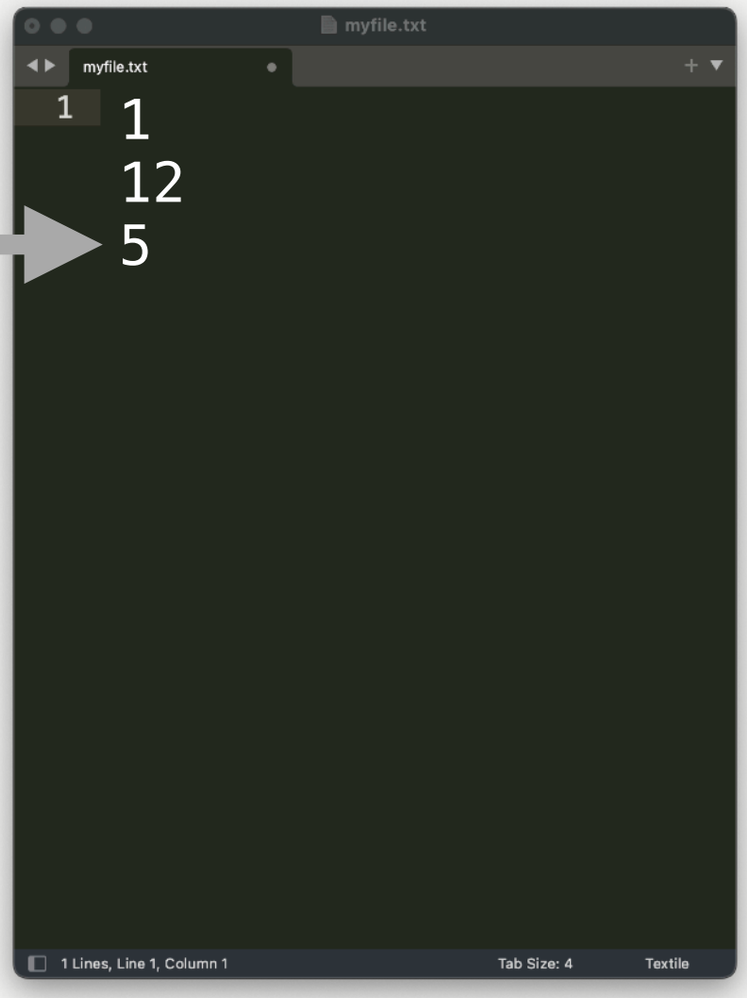


EOF Example

Here

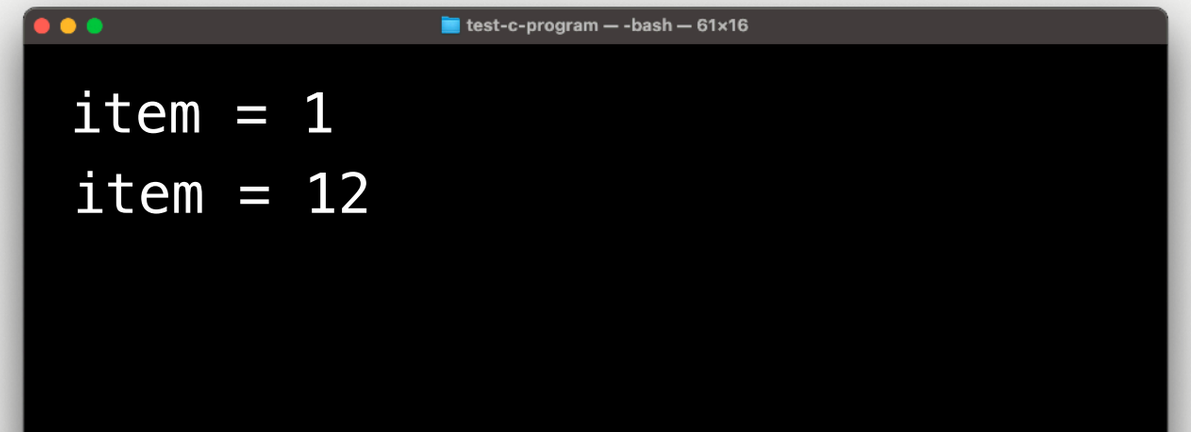
```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



fscanf is called and can read the value 5 successfully, which results in 1 being returned. The while loop condition is true.

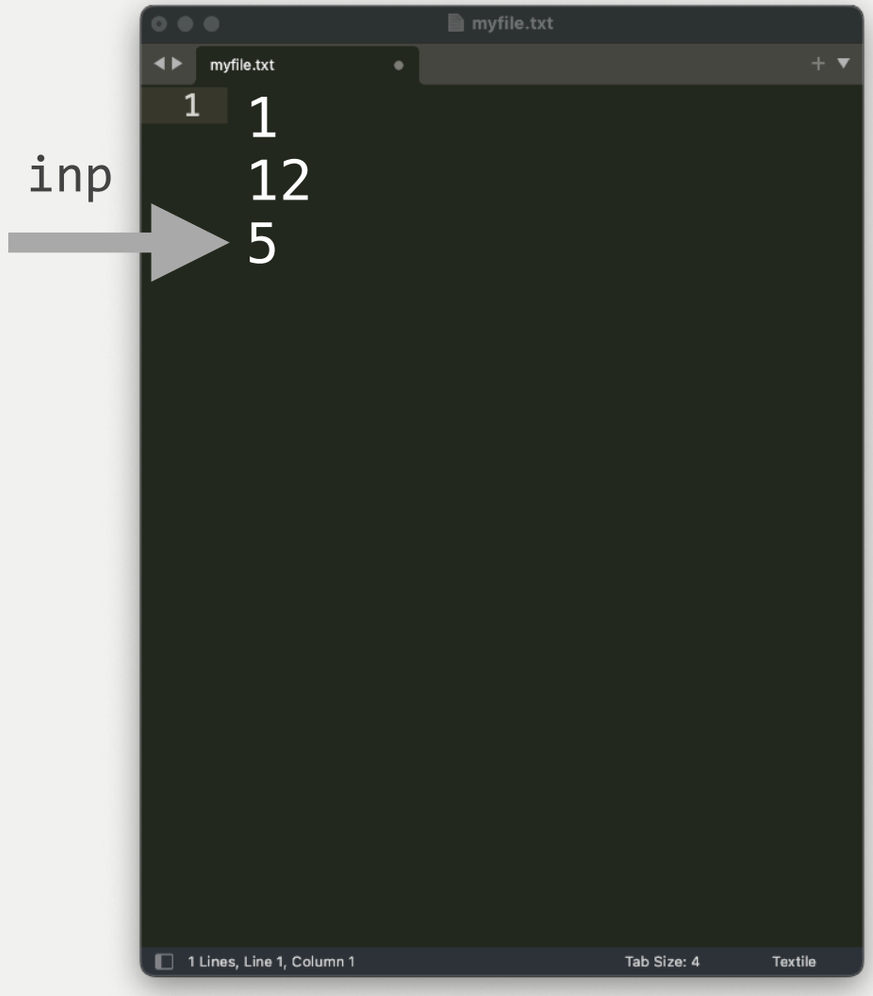
Stack	Space
AA1	inp = Some Address
AA0	item = 5



EOF Example

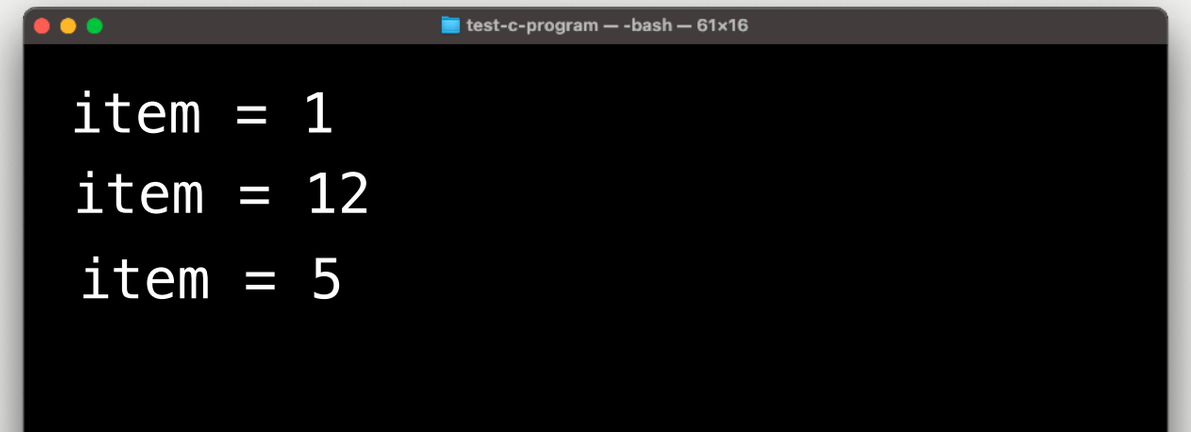
```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

printf("item = %d\n", item);

}

fclose(inp);
```

Here



Display the value stored in item.

Stack	Space
AA1	inp = Some Address
AA0	item = 5

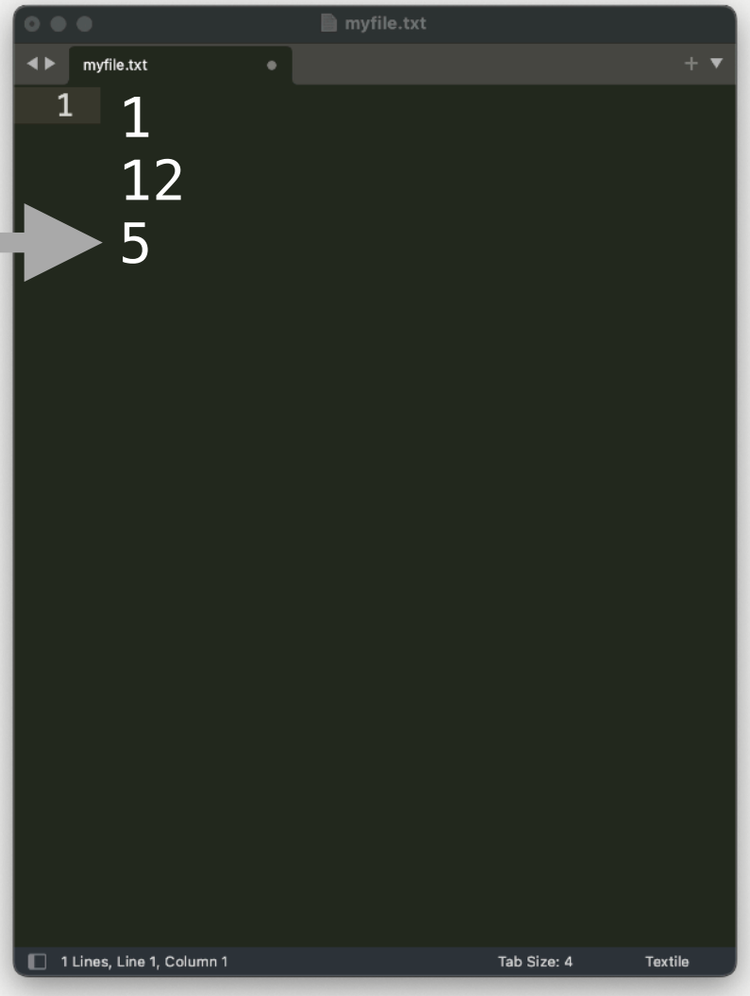


EOF Example

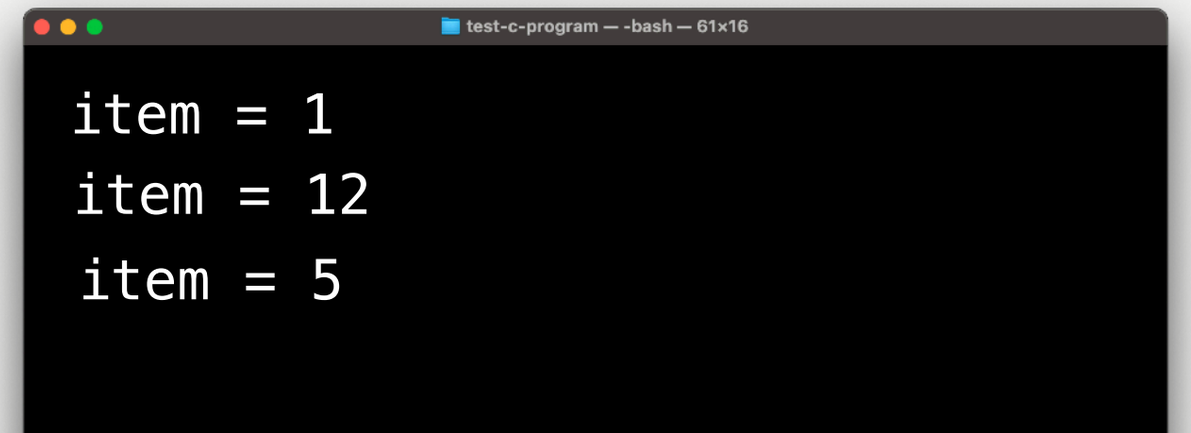
```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

Here

inp



Stack	Space
AA1	inp = Some Address
AA0	item = 5

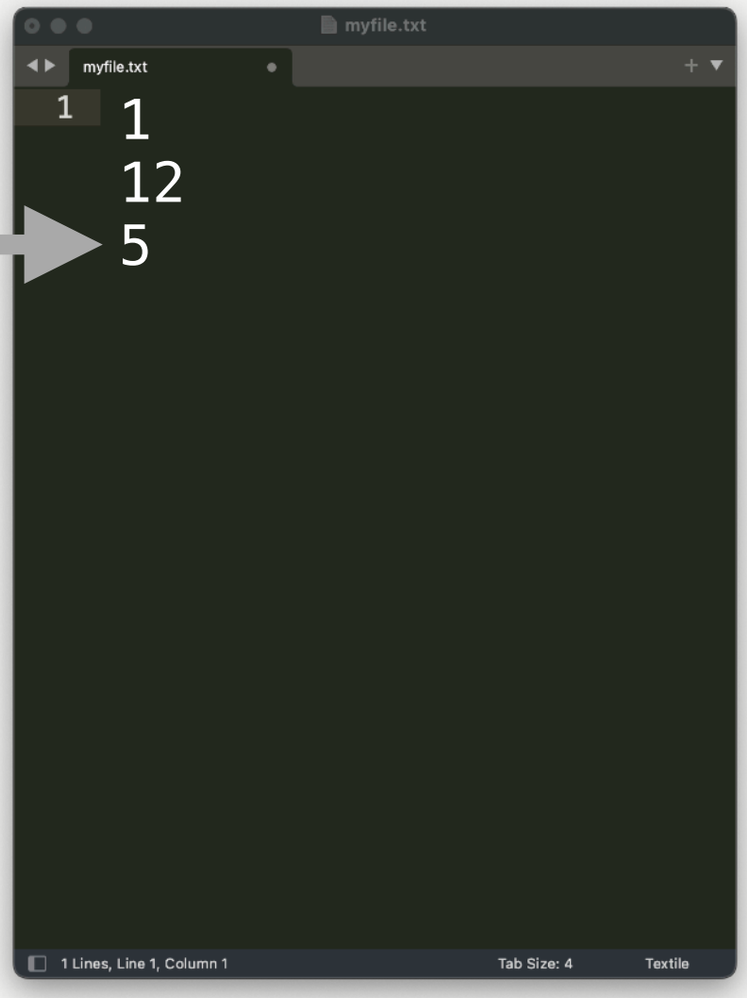


EOF Example

Here

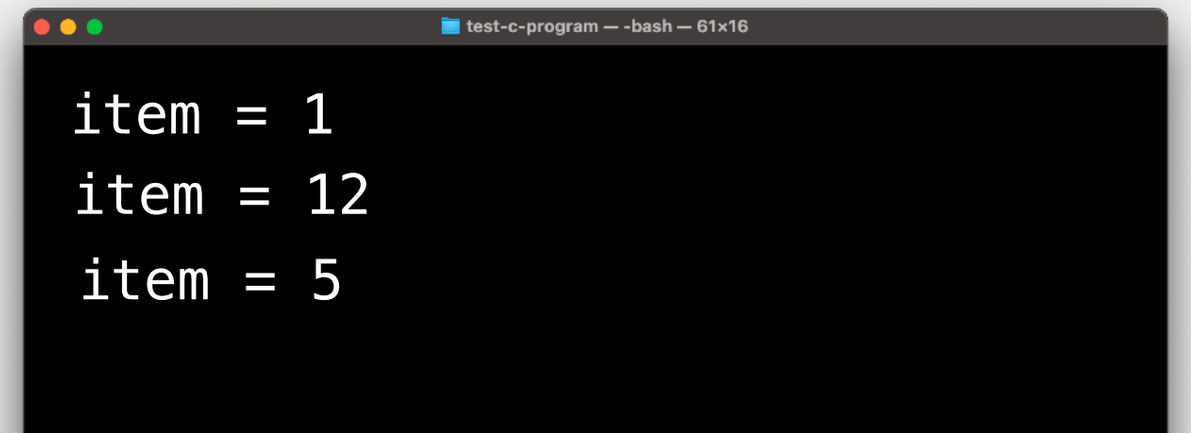
```c
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

inp



False

Stack	Space
AA1	inp = Some Address
AA0	item = 5



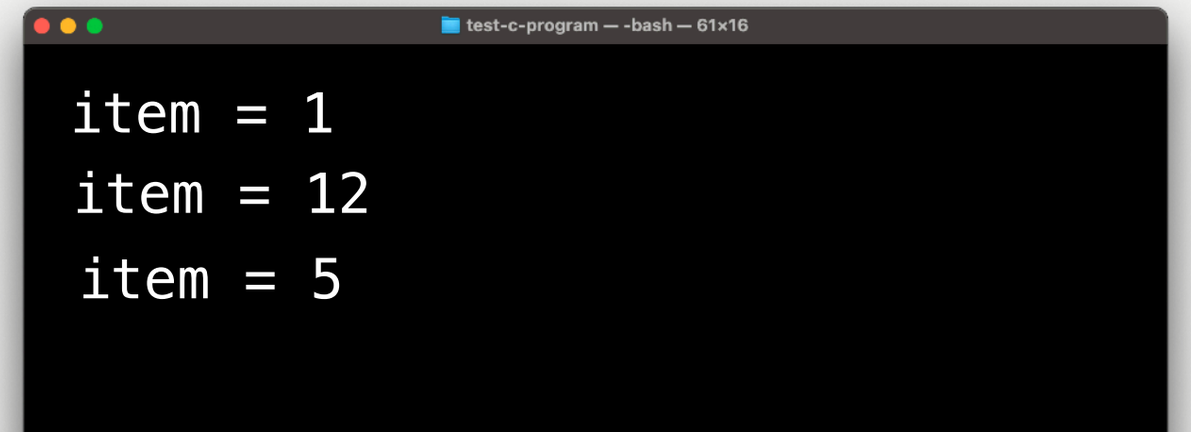
EOF Example

```
FILE *inp = fopen("indata.txt", "r");

int item;

while(fscanf(inp, “%d", &item) != EOF){

    printf("item = %d\n", item);

}

fclose(inp);
```

Here

File stream is now closed

Stack	Space
AA1	inp = Some Address
AA0	item = 5



One Last Thing…

- After you done accessing the file for reading or writing you must CLOSE the file.
- If you forget to close the file, the program will still run BUT leaves files open with access.
- It's a common mistake beginners make. Remember after opening to close the files.

```
fclose(inp);
fclose(outp);
```

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course