# C Programming Basics for Security

**Objective**

Understand C operators, file handling, and control structures with a cybersecurity lens.

## File Handling with fopen

* "r" read (file must exist)
* "w" write (create new or truncate existing)
* "a" append (write at the end, keep existing contents)
* "r+" read & write (file must exist)
* "w+" read & write (truncate if exists)
* "a+" read & append

## Safe Input with fgets

Important: **fopen** sets **how** the file is accessed, while **fgets** just reads lines from it. So, do not confuse **fopen** modes ("r", "w", "a") with **fgets**.

A black screen with white text

AI-generated content may be incorrect.

A computer code on a black background

AI-generated content may be incorrect.

**fgets** means file get string. It reads a line of text (up to n-1 characters) into a buffer and automatically adds the null terminator \0.

Arguments:

**line**: buffer (char array) where the string will be stored.

**sizeof(line)**: maximum number of characters to read (prevents overflow).

**stdin**: the input stream (keyboard by default).

## Pointer & Address Operators

**\*** (dereference) and **&** (address-of). Why is it important?

Many classic **vulnerabilities** come from pointer misuse. So, understanding **\*** (dereference) and **&** (address-of) helps students see how attackers can control program memory**.**

## Struct Access

**Struct Access (. and ->).** Why are they important?

They help understand memory access**,** which is fundamental for buffer overflows, pointer manipulation, and memory corruption exploits. In forensics, for example, analysts need to interpret memory dumps that often map directly onto struct layouts. So they use it to interpret binary data structures (e.g. packets, processes).

**&** Get the address of something (address-of operator). It should return the memory address of a variable. It is often used when assigning a pointer or passing a variable by reference to a function

**\*** Go to the address and get the value. If **p** is a pointer to an int, then **\*p** gives you the int stored at that address.

. Access struct member directly.

**->** Access struct member through a pointer.

## Operators & Loops

**Unary Increment/Decrement (++, --)**

Commonly used in loops, these operators are also relevant because off-by-one errors in them often lead to vulnerabilities. Unary operators act on a single operand. For example, **-x** negates **x**, and **++x** increments **x** by one.

int x = 5;

int y = -x; // unary minus → makes x negative

In this example, the operator **-** is unary because it is applied to **one variable (x)**.

**Common binary operators:**

* Arithmetic: + - \* / %
* Relational: < > <= >= == !=
* Logical: && ||
* Bitwise: & | ^ << >>
* Assignment: =

**Relational & Logical Operators**

<, >, <=, >=, ==, !=, &&, ||, !

Why are they important?

Critical for **input validation** and **conditional checks**. Logic errors in conditions (e.g., == vs =) can introduce **authentication bypasses**.

**Assignment & Shorthand Assignment**

**=, +=, -=, etc**

Why is it important?

Mistakes like using = instead of == in conditionals are a classic **security bug**.

**Bitwise Operators**

**&, |, ^, ~, <<, >>**

Why is this important?

They are used heavily in **cryptography**, **network packet manipulation**, and **access control flags** (permissions, masks, etc.).

Example: **chmod** in Linux uses bitwise permission masks.

**Sizeof Operator**

sizeof()

Why is it important?

It is critical for **safe memory allocation**. So, misusing it can lead to memory corruption vulnerabilities.

**Ternary Operator**

Ternary operator works on three operands. There is only one ternary operator in C:

**?:**

In this code example:

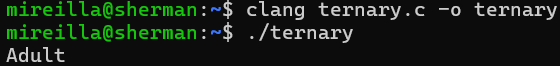
If condition (age >= 18) is true → "Adult”

Else → "Minor"

A computer screen with white text

AI-generated content may be incorrect.

Output



Useful shorthand in secure coding for **validations and sanitization**.

Example: int access = (user\_is\_admin ? 1 : 0);

Conditional Concepts in C Relevant to Cybersecurity

**if / else** statements are fundamental for access control, authentication, and input validation. For example, bugs like using **=** instead of **==** in conditionals can lead to authentication bypass vulnerabilities.

A **for** loop is a more efficient loop structure in 'C' programming

A computer code with text

AI-generated content may be incorrect.

The initial value of the for loop is performed only once.

The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.

The incrementation/decrementation increases (or decreases) the counter by a set value.

In C, the **for** loop can have multiple expressions separated by commas in each part.

A computer screen shot of a program code

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

## While loop and operator

Task: Read about the while loop and find out why this might be important in cybersecurity.