# Strings, Bugs, and Compiler Warnings

**Objective**

Practice secure string handling and learn how compiler warnings catch vulnerabilities.

## Basic C string functions

This program teaches **basic C string functions**:

* strcmp is used compare
* strcat is used to concatenate
* strcpy is used to copy
* strlen is used to find the length

Mini Labs: C String Functions

**1: String Comparison (strcmp)**

**Task**

1. Write a program that asks the user for **two strings**.
2. Use strcmp to compare them.
3. Print whether they are equal or not.

Solution:

A computer code on a black background

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

**2: String Concatenation (strcat)**

**Task**

1. Read two strings.
2. Join them using strcat.
3. Print the result.

**3: String Copy (strcpy)**

**Task**

1. Read one string into s1.
2. Copy it into s2.
3. Print both strings.

**4: String Length (strlen)**

**Task**

1. Read a string.
2. Use strlen to count characters.
3. Print the result.

**5: Combine Everything (Final Program)**

**Task**

* Combine **all four functions** into one program:
  + Compare two strings.
  + If not equal, concatenate them.
  + Copy result into a third string.
  + Print lengths of all.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

## Safe alternatives

Modify the tasks above to use **safe string functions**:

Replace **scanf("%s")** with **fgets**. fgets reads full line safely, including spaces. So, it prevents buffer overflow + allows spaces.

Replace **strcat** with **strncat**. **strncat** concatenates with length limit. **strncat** prevents writing past buffer end when concatenating

Replace **strcpy** with **strncpy**, which copies with length limit. **strncpy** prevents overflow when copying.

**s3[sizeof(s3)-1] = '\0';** ensures string is always null-terminated.

A screen shot of a computer program

AI-generated content may be incorrect.

**fgets** reads full line safely, including spaces. So, it prevents buffer overflow + allows spaces.

**strncat** concatenates with length limit. **strncat** prevents writing past buffer end when concatenating

**strncpy** copies with length limit. **strncpy** prevents overflow when copying.

**s3[sizeof(s3)-1] = '\0';** ensures string is always null-terminated

A screen shot of a computer code

AI-generated content may be incorrect.

## Compiler Warnings (-Wall -Wextra -Werror)

Many vulnerabilities start as “just warnings.” **-Wall -Wextra** are **compiler flags** for GCC/Clang. They do not change how the program run*s*, but they change how much the compiler warns you about possible problems. **-Wall -Wextra** makes the compiler act like a security assistant, catching unsafe coding before it ships. Good security practice: compile with warnings enabled, treat them as errors using **-Werror** during development.

**-Wall** enables a broad set of **common warnings**.

Examples:

* Using an **uninitialised variable**.
* Forgetting a return in a non-void function.
* Writing if (x = 5) instead of if (x == 5) (assignment instead of comparison).
* Declaring a variable and never using it.

**-Wextra** enables **additional warnings** that aren’t included in -Wall.  
Examples:

* Unused parameters in functions.
* Signed/unsigned comparison (e.g., comparing **int** with **size\_t**).
* Suspicious pointer arithmetic.

-**Werror** treats warnings as **errors** (forces you to fix them)

**Example**

This example shows that:

**Uninitialised variables are dangerous.** Using an uninitialised variable (x) leads to unpredictable results. In security terms, this could leak old stack memory, which might contain **passwords, keys, or other secrets**.

**Assignment vs comparison mistakes (= vs ==).**

if (z = 10) is legal C, but almost never what you want. This is a classic **logic bug**, the condition is always true. In security-critical code, such a mistake could cause authentication checks to always succeed (e.g., if (is\_admin = 1), it means every user becomes admin).

**Compilers do not save you unless you ask.**

Without -Wall -Wextra, this compiles quietly.

With warnings enabled, the compiler catches both problems.

Here is the demo\_warnings.c file.

A computer screen shot of a program code

AI-generated content may be incorrect.

Example output 1:

The code below shows that when compiled using Clang, Clang enables some critical warnings by default, only the most obvious errors.

A screenshot of a computer screen

AI-generated content may be incorrect.

Example output 2:

Because if (z = 10) always assigns 10 to z, which is nonzero, so the condition is always true. **GCC does not warn by default** about assignment-in-if. Without -Wall -Wextra, dangerous bugs slip through.



Example output 3: **-Wall -Wextra are necessary in GCC** to catch bugs that Clang might warn about automatically.

A screenshot of a computer program

AI-generated content may be incorrect.

Example output 4:

When you use -Wall -Wextra, you activate a **much larger set of diagnostic checks**. That is why the same code using clang produced **2 warnings** rather than the first **1 warning**.

A screenshot of a computer screen

AI-generated content may be incorrect.

In conclusion, it is good practice to always compile with at least -Wall -Wextra.

These flags **do not fix bugs**, but they often **catch them early**.

In security-sensitive code, ignoring warnings can lead to **exploitable flaws**.

-**Werror** treats warnings as **errors** (forces you to fix them).