C Programming Training Module

A Comprehensive Guide for Beginners

Prepared by xAI Training Team May 07, 2025

Downloadable PDF version available for training purposes.

**Contents**

1. [Introduction to C](#_bookmark0) 2
   1. [Why Learn C?](#_bookmark1) 2
   2. [Setting Up the Environment](#_bookmark2) 2
2. [Basic C Syntax](#_bookmark3) 2
   1. [First C Program](#_bookmark4) 2
   2. [Variables and Data Types](#_bookmark5) 3
3. [Control Structures](#_bookmark6) 3
   1. [Conditional Statements](#_bookmark7) 3
   2. [Loops](#_bookmark8) 3
4. [Functions](#_bookmark9) 4
   1. [Defining Functions](#_bookmark10) 4
5. [Pointers](#_bookmark11) 4
   1. [Pointer Example](#_bookmark12) 4
6. [Arrays and Strings](#_bookmark13) 4
   1. [Array Example](#_bookmark14) 4
   2. [String Example](#_bookmark15) 5
7. [Structures](#_bookmark16) 5
   1. [Structure Example](#_bookmark17) 5
8. [File Handling](#_bookmark18) 5
   1. [File I/O Example](#_bookmark19) 5
9. [Conclusion](#_bookmark20) 6
10. [References](#_bookmark21) 6

1

# Introduction to C

C is a powerful, general-purpose programming language known for its efficiency and control over system resources. This training module introduces C funda- mentals, with practical examples to build programming skills.

## Why Learn C?

* + - **Efficiency**: Close-to-hardware performance for system programming.
    - **Foundation**: Basis for languages like C++ and operating systems.
    - **Versatility**: Used in embedded systems, OS development, and applications.

## Setting Up the Environment

Install a C compiler like GCC (GNU Compiler Collection). On Windows, use MinGW or WSL; on macOS/Linux, GCC is often pre-installed. Verify with:

gcc --version

1

Use an IDE like Code::Blocks or a text editor like VS Code.

# Basic C Syntax

C programs are structured around functions. Below is a simple ”Hello, World!” program.

## First C Program

1

**#include** <stdio.h>

**int** main() {

printf(”Hello, World!\n”);

**return** 0;

}

2

3

4

5

6

## Explanation:

* #include <stdio.h>: Imports input/output functions.
* int main(): Program entry point.
* printf: Outputs text to the console.
* return 0: Indicates successful execution.

## Variables and Data Types

C supports types like int, float, char, and double.

**#include** <stdio.h>

**int** main() {

**int** age = 25;

**float** salary = 50000.50;

**char** grade = ’A’;

printf(”Age: %d, Salary: %.2f, Grade: %c\n”, age, salary, grade)

;

**return** 0;

}

1

2

3

4

5

6

7

8

9

# Control Structures

Control structures manage program flow.

## Conditional Statements

Use if-else for decisions.

**#include** <stdio.h>

**int** main() {

**int** score = 85;

**if** (score >= 90) { printf(”Grade: A\n”);

} **else if** (score >= 80) { printf(”Grade: B\n”);

} **else** {

printf(”Grade: C\n”);

}

**return** 0;

}

1

2

3

4

5

6

7

8

9

10

11

12

13

## Loops

Loops repeat code. Below is a for loop example.

**#include** <stdio.h>

**int** main() {

**for** (**int** i = 1; i <= 5; i++) { printf(”Number: %d\n”, i);

}

**return** 0;

}

1

2

3

4

5

6

7

8

# Functions

Functions promote code reuse.

## Defining Functions

Declare functions with return types and parameters.

**#include** <stdio.h>

**int** add(**int** a, **int** b) {

**return** a + b;

}

**int** main() {

**int** sum = add(5, 3); printf(”Sum: %d\n”, sum); **return** 0;

}

1

2

3

4

5

6

7

8

9

10

11

# Pointers

Pointers store memory addresses, enabling efficient memory management.

## Pointer Example

1

**#include** <stdio.h>

**int** main() {

**int** num = 10;

**int** \*ptr = &num;

printf(”Value: %d, Address: %p\n”, \*ptr, ptr);

\*ptr = 20; // Modify value via pointer printf(”New Value: %d\n”, num);

**return** 0;

}

2

3

4

5

6

7

8

9

10

# Arrays and Strings

Arrays store multiple values; strings are character arrays.

## Array Example

1

**#include** <stdio.h>

**int** main() {

2

3

4

**int** numbers[5] = {1, 2, 3, 4, 5};

**for** (**int** i = 0; i < 5; i++) { printf(”Number: %d\n”, numbers[i]);

}

**return** 0;

}

5

6

7

8

9

## String Example

1

**#include** <stdio.h>

**#include** <string.h>

**int** main() {

**char** name[] = ”Alice”;

printf(”Name: %s, Length: %lu\n”, name, strlen(name));

**return** 0;

}

2

3

4

5

6

7

8

# Structures

Structures group related data.

## Structure Example

1

**#include** <stdio.h>

**struct** Person { **char** name[50]; **int** age;

};

**int** main() {

**struct** Person person = {”Bob”, 30};

printf(”Name: %s, Age: %d\n”, person.name, person.age);

**return** 0;

}

2

3

4

5

6

7

8

9

10

11

12

# File Handling

Read from and write to files.

## File I/O Example

1

**#include** <stdio.h>

**int** main() {

FILE \*file = fopen(”example.txt”, ”w”);

**if** (file == NULL) {

printf(”Error opening file!\n”);

**return** 1;

}

fprintf(file, ”Hello, C!\n”); fclose(file);

file = fopen(”example.txt”, ”r”);

**char** buffer[100]; fgets(buffer, 100, file);

printf(”File content: %s”, buffer); fclose(file);

**return** 0;

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

# Conclusion

This module covers C fundamentals, from syntax to pointers, structures, and file handling. Practice these examples and explore advanced topics like memory management and system programming.

# References

* C Programming Language, Kernighan & Ritchie
* C Reference: <https://en.cppreference.com/w/c>