Introduction to Programming with C

Matrix Software Dev Club - Beginner's Guide

About This Guide

This comprehensive guide is designed for complete beginners with no prior coding experience. It will serve as your introduction to the world of programming through the C language - a powerful foundation that will help you understand core concepts applicable to many other programming languages.

Table of Contents

1. Introduction to Programming

What is Programming?

Programming is the process of writing instructions that tell a computer exactly what to do. Think of it as giving directions to someone who will follow them precisely:

- Programs are sets of instructions written in a language computers can understand
- Programming allows us to solve problems, automate tasks, and build applications
- Learning to program means learning to think logically and break down problems into steps

Why Choose C as a First Language?

C was developed by **Dennis Ritchie** in 1972 at Bell Labs. Despite being nearly 50 years old, it remains one of the most important programming languages in the world.

Benefits of starting with C:

- **Foundation builder**: C teaches you fundamental programming concepts without hiding how things work
- **Widely used**: Operating systems, databases, and embedded systems are often written in C
- **Transferable knowledge**: Once you learn C, many other languages (C++, Java, JavaScript, Python) become easier to learn
- **Performance**: C programs are typically very fast and efficient
- **Understanding computers**: C helps you understand how computers actually work at a lower level

---

2. Setting Up Your Development Environment

Before writing code, you need the right tools. For C programming, you need two main components:

1. Text Editor or IDE (Integrated Development Environment)

Where you'll write your code. Options include:

- **Visual Studio Code** (Recommended) - Free, powerful, works on all platforms
- **CodeBlocks** - Specifically designed for C/C++
- **Sublime Text** - Fast, lightweight text editor
- **Vim/Emacs** - Advanced text editors (for more experienced users)

2. C Compiler

Converts your human-readable code into machine code the computer can execute:

- **GCC (GNU Compiler Collection)** - Standard C compiler for Linux/macOS
- **MinGW** - GCC for Windows
- **Clang** - Modern C compiler (alternative to GCC)

All-in-One Online Options (No Installation Required)

If you want to start immediately without installing anything:

- **Replit** (https://replit.com) - Create an account and start coding in your browser
- **Programiz Online Compiler** (https://www.programiz.com/c-programming/online-compiler/)
- **OnlineGDB** (https://www.onlinegdb.com/online_c_compiler)

Installation Instructions

Windows:

1. Download and install MinGW: https://sourceforge.net/projects/mingw/
2. Add MinGW to your PATH environment variable
3. Install VS Code: https://code.visualstudio.com/download
4. Install the C/C++ extension in VS Code

macOS:

1. Install Xcode Command Line Tools by opening Terminal and typing: `xcode-select --install`

2. Install VS Code: https://code.visualstudio.com/download
3. Install the C/C++ extension in VS Code

Linux:

1. Install GCC using your distribution's package manager:
   - Ubuntu/Debian: `sudo apt install build-essential`
   - Fedora: `sudo dnf install gcc`
2. Install VS Code (varies by distribution)
3. Install the C/C++ extension in VS Code

---

3. Your First C Program

Let's write the traditional first program that displays "Hello, World!" on the screen.

The Code

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Understanding Each Line

```c
#include <stdio.h>
```

- This line includes the Standard Input/Output library
- Libraries contain pre-written code we can use
- `stdio.h` gives us functions for input and output, like `printf()`

```c
int main() {
```

- Every C program starts execution at the `main()` function
- `int` means this function will return an integer value
- The curly braces `{ }` define the beginning and end of the function

```c
    printf("Hello, World!\n");
```

- `printf()` is a function that prints text to the screen
- Text must be enclosed in double quotes `""`

- `\n` is a special character that creates a new line (like pressing Enter)
- Every statement in C ends with a semicolon `;`

```
    return 0;
```

- Returns the value 0 to indicate the program finished successfully
- By convention, 0 means no errors occurred

Compiling and Running Your Program

Using the Command Line:

1. Save your code as `hello.c`
2. Open your terminal/command prompt
3. Navigate to the folder containing your file
4. Compile the code:

   ```
   gcc hello.c -o hello
   ```

5. Run the program:
   - Windows: `hello`
   - macOS/Linux: `./hello`

Using an IDE:

1. Open your IDE (like VS Code)
2. Create a new file and save it as `hello.c`
3. Type in the code
4. Use the build/run commands in your IDE (usually F5 or a play button)

Expected Output

```
Hello, World!
```

Exercise

Modify the program to display your name instead of "Hello, World!".

---

4. Understanding Variables and Data Types

Variables are like labeled containers that store data in your program. They allow you to work with different types of information.

Common Data Types in C

Data Type	Description	Example	Size (bytes)
`int`	Whole numbers	`int age = 20;`	4
`float`	Decimal numbers (single precision)	`float price = 10.99;`	4
`double`	Decimal numbers (double precision)	`double pi = 3.14159265359;`	8
`char`	Single character	`char grade = 'A';`	1

Declaring Variables

The basic syntax for declaring variables is:

```
data_type variable_name = initial_value;
```

Examples:

```
int student_count = 25;        // Number of students
float temperature = 98.6;      // Body temperature in Fahrenheit
char first_initial = 'J';      // First letter of a name
double precise_value = 3.141592653589793;  // More precise decimal
```

Variable Naming Rules

- Can contain letters, digits, and underscores
- Must start with a letter or underscore
- Cannot use C keywords (int, if, while, etc.)
- Case-sensitive (`age` and `Age` are different variables)

Good Naming Practices

- Use descriptive names (`student_count` instead of just `x`)
- Use underscores for multi-word names (`first_name` or `user_age`)
- Be consistent with your style

Constants

Values that should never change during program execution:

```
#define PI 3.14159           // Preprocessor constant (no semicolon)

const double GRAVITY = 9.81;  // Constant variable (needs semicolon)
```

Using Boolean Values in C

C doesn't have a built-in boolean type in the original standard (C90). In C, zero is considered "false" and any non-zero value is "true".

In modern C (C99 and later), you can use the `stdbool.h` library:

```c
#include <stdio.h>
#include <stdbool.h>  // For bool, true, false

int main() {
    bool is_student = true;

    if (is_student) {
        printf("This person is a student.\n");
    }

    return 0;
}
```

---

5. Working with Input and Output

Communication between the user and your program happens through input and output operations.

Displaying Output with printf()

`printf()` allows you to display text and variable values:

```c
#include <stdio.h>

int main() {
    int age = 19;
    float height = 1.75;
    char initial = 'M';

    // Basic output
    printf("Hello, student!\n");

    // Displaying variable values
    printf("Age: %d years\n", age);
    printf("Height: %.2f meters\n", height);
    printf("Initial: %c\n", initial);

    // Multiple variables in one statement
    printf("Student profile: %c, %d years old, %.2f meters tall\n",
           initial, age, height);

    return 0;
}
```

Format Specifiers

Specifier	Used For	Example
`%d`	Integers	`printf("%d", 42);`
`%f`	Floating-point	`printf("%f", 3.14);`
`%.2f`	Float with 2 decimal places	`printf("%.2f", 3.14159);`
`%c`	Characters	`printf("%c", 'A');`
`%s`	Strings (text)	`printf("%s", "Hello");`

Getting Input with scanf()

`scanf()` reads input from the user:

```c
#include <stdio.h>

int main() {
    int age;
    float height;
    char initial;

    // Getting integer input
    printf("Enter your age: ");
    scanf("%d", &age);

    // Getting float input
    printf("Enter your height in meters: ");
    scanf("%f", &height);

    // Getting character input
    printf("Enter your first initial: ");
    scanf(" %c", &initial);  // Note the space before %c to consume newline

    // Display the input
    printf("\nYou entered:\n");
    printf("Age: %d years\n", age);
    printf("Height: %.2f meters\n", height);
    printf("Initial: %c\n", initial);

    return 0;
}
```

Important Notes About scanf()

- Always use the `&` symbol before variable names (except for strings)
- The format specifier must match the variable type

- Put a space before `%c` to skip any leftover newline characters
- `scanf()` can be tricky with string input; we'll cover safer methods later

Exercise

Write a program that asks the user for their name and age, then displays a greeting with this information.

---

6. Control Flow: Making Decisions

Control flow statements allow your program to make decisions and execute different code based on conditions.

The if Statement

The most basic decision-making statement:

```c
#include <stdio.h>

int main() {
    int age;

    printf("Enter your age: ");
    scanf("%d", &age);

    if (age >= 18) {
        printf("You are an adult.\n");
    }

    return 0;
}
```

if-else Statement

Allows you to execute one block if the condition is true and another if it's false:

```c
#include <stdio.h>

int main() {
    int score;

    printf("Enter your test score (0-100): ");
    scanf("%d", &score);

    if (score >= 60) {
        printf("You passed the test!\n");
    } else {
        printf("You did not pass the test.\n");
    }
```

```
    return 0;
}
```

if-else if-else Chain

For multiple conditions:

```c
#include <stdio.h>

int main() {
    int score;

    printf("Enter your test score (0-100): ");
    scanf("%d", &score);

    if (score >= 90) {
        printf("Grade: A\n");
    } else if (score >= 80) {
        printf("Grade: B\n");
    } else if (score >= 70) {
        printf("Grade: C\n");
    } else if (score >= 60) {
        printf("Grade: D\n");
    } else {
        printf("Grade: F\n");
    }

    return 0;
}
```

Comparison Operators

Operator	Meaning	Example
`==`	Equal to	`if (age == 21)`
`!=`	Not equal to	`if (choice != 'Y')`
`<`	Less than	`if (score < 60)`
`>`	Greater than	`if (temp > 100)`
`<=`	Less than or equal to	`if (grade <= 'C')`
`>=`	Greater than or equal to	`if (value >= 0)`

Logical Operators

Combine multiple conditions:

Operator	Meaning	Example

Operator	Meaning	Example
`&&`	AND (both must be true)	`if (age > 16 && has_license)`
`||`	OR (either can be true)	`if (is_holiday || is_weekend)`
`!`	NOT (inverts truth)	`if (!is_expired)`

```c
#include <stdio.h>

int main() {
    int age;
    char has_id;

    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Do you have ID? (Y/N): ");
    scanf(" %c", &has_id);

    if (age >= 18 && (has_id == 'Y' || has_id == 'y')) {
        printf("You can enter the venue.\n");
    } else {
        printf("You cannot enter the venue.\n");
    }

    return 0;
}
```

Switch Statement

For multiple choice situations:

```c
#include <stdio.h>

int main() {
    int day;

    printf("Enter day number (1-7): ");
    scanf("%d", &day);

    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
```

```
            case 4:
                printf("Thursday\n");
                break;
            case 5:
                printf("Friday\n");
                break;
            case 6:
                printf("Saturday\n");
                break;
            case 7:
                printf("Sunday\n");
                break;
            default:
                printf("Invalid day number\n");
        }

        return 0;
    }
```

Exercise

Write a program that asks for two numbers and an operation (+, -, *, /), then performs the calculation using if-else statements.

---

7. Loops: Repeating Tasks

Loops allow you to execute a block of code multiple times.

for Loop

Best when you know exactly how many times you want to repeat something:

```
#include <stdio.h>

int main() {
    // Print numbers from 1 to 5
    for (int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }

    printf("\n");
    return 0;
}
```

Syntax:

```
for (initialization; condition; update) {
    // Code to repeat
```

```
}
```

Parts:

- **Initialization**: Runs once at the beginning
- **Condition**: Checked before each iteration
- **Update**: Runs after each iteration

while Loop

Use when you don't know in advance how many iterations you need:

```c
#include <stdio.h>

int main() {
    int count = 1;

    while (count <= 5) {
        printf("%d ", count);
        count++;
    }

    printf("\n");
    return 0;
}
```

Syntax:

```c
while (condition) {
    // Code to repeat
}
```

do-while Loop

Similar to while, but always executes at least once:

```c
#include <stdio.h>

int main() {
    int number;

    do {
        printf("Enter a positive number: ");
        scanf("%d", &number);
    } while (number <= 0);

    printf("You entered: %d\n", number);
```

```
    return 0;
}
```

Syntax:

```
do {
    // Code to repeat
} while (condition);
```

Loop Control Statements

- **break**: Exits the loop immediately
- **continue**: Skips the rest of the current iteration

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        // Skip printing 5
        if (i == 5) {
            continue;
        }

        // Stop when we reach 8
        if (i == 8) {
            printf("Breaking the loop\n");
            break;
        }

        printf("%d ", i);
    }

    printf("\n");
    return 0;
}
```

Nested Loops

You can place one loop inside another:

```
#include <stdio.h>

int main() {
    // Print a small multiplication table
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 3; j++) {
            printf("%d x %d = %d\n", i, j, i*j);
```

```
        }
        printf("----------\n");
    }

    return 0;
}
```

Exercise

Write a program that calculates the sum of all numbers from 1 to n, where n is entered by the user.

---

8. Arrays: Working with Collections

Arrays allow you to store multiple values of the same data type under a single variable name.

Why Use Arrays?

- Store related data together
- Easily process multiple values using loops
- Efficiently manage large sets of data

Declaring and Initializing Arrays

```c
#include <stdio.h>

int main() {
    // Declare an array of 5 integers
    int scores[5];

    // Initialize at declaration
    int grades[5] = {85, 92, 78, 90, 88};

    // Partial initialization (remaining elements are 0)
    int values[5] = {10, 20};  // values = {10, 20, 0, 0, 0}

    // Array without size (compiler determines based on elements)
    int numbers[] = {1, 2, 3, 4, 5};  // Creates an array of size 5

    return 0;
}
```

Accessing Array Elements

Array indices start at 0:

```c
#include <stdio.h>
```

```c
int main() {
    int scores[5] = {85, 92, 78, 90, 88};

    // Access individual elements
    printf("First score: %d\n", scores[0]);   // 85
    printf("Third score: %d\n", scores[2]);   // 78

    // Modify an element
    scores[1] = 95;
    printf("Updated second score: %d\n", scores[1]);  // 95

    return 0;
}
```

Array Bounds

C does not check if you access elements outside the array bounds:

```c
int scores[5] = {85, 92, 78, 90, 88};

// WARNING: This is dangerous and can cause unpredictable behavior
scores[10] = 100;  // Accessing beyond array bounds
```

Always ensure your indices are within the valid range: 0 to size-1.

Processing Arrays with Loops

```c
#include <stdio.h>

int main() {
    int scores[5] = {85, 92, 78, 90, 88};
    int sum = 0;

    // Calculate sum of all scores
    for (int i = 0; i < 5; i++) {
        sum += scores[i];
    }

    float average = (float)sum / 5;
    printf("Average score: %.2f\n", average);

    return 0;
}
```

Inputting Values into an Array

```c
#include <stdio.h>

int main() {
    int size = 5;
    int numbers[size];

    // Get input from user
    for (int i = 0; i < size; i++) {
        printf("Enter number %d: ", i + 1);
        scanf("%d", &numbers[i]);
    }

    // Display the array
    printf("You entered: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    return 0;
}
```

Multi-dimensional Arrays

Arrays can have multiple dimensions:

```c
#include <stdio.h>

int main() {
    // 2D array (3 rows, 4 columns)
    int matrix[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // Access elements
    printf("Element at row 1, col 2: %d\n", matrix[1][2]);  // 7

    // Iterate through all elements
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 4; col++) {
            printf("%3d ", matrix[row][col]);
        }
        printf("\n");
    }

    return 0;
}
```

Common Array Operations

1. Finding the Maximum Value

```c
#include <stdio.h>

int main() {
    int numbers[] = {23, 45, 12, 67, 34, 9, 56};
    int size = 7;
    int max = numbers[0];  // Assume first element is max

    for (int i = 1; i < size; i++) {
        if (numbers[i] > max) {
            max = numbers[i];
        }
    }

    printf("Maximum value: %d\n", max);
    return 0;
}
```

2. Counting Occurrences

```c
#include <stdio.h>

int main() {
    int numbers[] = {5, 2, 8, 5, 1, 9, 5, 3, 5};
    int size = 9;
    int target = 5;
    int count = 0;

    for (int i = 0; i < size; i++) {
        if (numbers[i] == target) {
            count++;
        }
    }

    printf("The number %d appears %d times\n", target, count);
    return 0;
}
```

Exercise

Write a program that creates an array of 10 integers, asks the user to fill it, then finds and displays both the minimum and maximum values.

---

9. Strings: Working with Text

In C, strings are arrays of characters ending with a null character (\0).

Declaring and Initializing Strings

```c
#include <stdio.h>

int main() {
    // Method 1: Character array with explicit null terminator
    char name1[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

    // Method 2: Character array with string literal
    char name2[6] = "Hello";  // Compiler adds the null terminator

    // Method 3: Let compiler determine size
    char name3[] = "Hello";   // Creates a 6-character array

    // Print the strings
    printf("String 1: %s\n", name1);
    printf("String 2: %s\n", name2);
    printf("String 3: %s\n", name3);

    return 0;
}
```

Reading and Writing Strings

```c
#include <stdio.h>

int main() {
    char name[50];  // Buffer to store input, large enough for typical
names

    // Reading a string with scanf (stops at whitespace)
    printf("Enter your first name: ");
    scanf("%s", name);  // Note: no & required for arrays
    printf("Hello, %s!\n", name);

    // Clear input buffer
    while (getchar() != '\n');

    // Reading a full line with fgets (includes spaces)
    char full_name[100];
    printf("Enter your full name: ");
    fgets(full_name, 100, stdin);
    printf("Hello, %s", full_name);  // fgets keeps the newline character

    return 0;
}
```

String Functions from <string.h>

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[20] = "Hello";
    char str2[20] = "World";
    char str3[40];

    // String length
    printf("Length of str1: %zu\n", strlen(str1));  // 5

    // String copy
    strcpy(str3, str1);
    printf("str3 after strcpy: %s\n", str3);  // Hello

    // String concatenation
    strcat(str3, " ");
    strcat(str3, str2);
    printf("str3 after strcat: %s\n", str3);  // Hello World

    // String comparison
    if (strcmp(str1, str2) == 0) {
        printf("str1 and str2 are equal\n");
    } else {
        printf("str1 and str2 are not equal\n");  // This will print
    }

    return 0;
}
```

Common String Operations

1. Converting to Upper Case

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int main() {
    char text[100] = "Hello, World!";
    int length = strlen(text);

    for (int i = 0; i < length; i++) {
        text[i] = toupper(text[i]);
    }

    printf("Uppercase: %s\n", text);  // HELLO, WORLD!
```

```
    return 0;
}
```

2. Counting Characters

```
#include <stdio.h>
#include <string.h>

int main() {
    char text[100];
    printf("Enter a string: ");
    fgets(text, 100, stdin);

    // Remove newline if present
    int len = strlen(text);
    if (text[len-1] == '\n') {
        text[len-1] = '\0';
        len--;
    }

    int letters = 0, digits = 0, spaces = 0, others = 0;

    for (int i = 0; i < len; i++) {
        if ((text[i] >= 'a' && text[i] <= 'z') || (text[i] >= 'A' &&
text[i] <= 'Z')) {
            letters++;
        } else if (text[i] >= '0' && text[i] <= '9') {
            digits++;
        } else if (text[i] == ' ') {
            spaces++;
        } else {
            others++;
        }
    }

    printf("Letters: %d\n", letters);
    printf("Digits: %d\n", digits);
    printf("Spaces: %d\n", spaces);
    printf("Other characters: %d\n", others);

    return 0;
}
```

String Input Safety

Always ensure your string buffer is large enough to hold the expected input:

```
#include <stdio.h>
#include <string.h>
```

```c
int main() {
    char name[30];

    // Safer input with fgets - limits input to buffer size
    printf("Enter your name: ");
    fgets(name, 30, stdin);

    // Remove newline character if present
    int len = strlen(name);
    if (name[len-1] == '\n') {
        name[len-1] = '\0';
    }

    printf("Hello, %s!\n", name);

    return 0;
}
```

Exercise

Write a program that asks the user to input a string and checks if it's a palindrome (reads the same forward and backward, ignoring spaces and case).

---

10. Functions: Building Blocks of Code

Functions are reusable blocks of code that perform specific tasks.

Why Use Functions?

- **Reusability**: Write code once, use it many times
- **Organization**: Break complex problems into smaller, manageable pieces
- **Readability**: Makes your code easier to understand
- **Maintenance**: Easier to fix or modify isolated pieces of code

Defining a Function

```c
return_type function_name(parameters) {
    // Function body
    // Code to execute
    return value;  // Optional
}
```

Example:

```c
#include <stdio.h>
```

```c
// Function definition
int add(int a, int b) {
    int sum = a + b;
    return sum;
}

int main() {
    // Function call
    int result = add(5, 3);
    printf("Sum: %d\n", result);

    return 0;
}
```

Function Prototypes

When defining functions after `main()`, you need to declare them first:

```c
#include <stdio.h>

// Function prototype (declaration)
int add(int a, int b);

int main() {
    int result = add(5, 3);
    printf("Sum: %d\n", result);

    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b;
}
```

Void Functions

Functions that don't return a value:

```c
#include <stdio.h>

// No return value
void greet() {
    printf("Hello! Welcome to C programming.\n");
}

int main() {
    greet();  // Call the function
```

```
    return 0;
}
```

Parameters vs. Arguments

- **Parameters**: Variables in the function definition
- **Arguments**: Values passed when calling the function

```
// Parameter 'name' is a placeholder
void greet_user(char name) {
    printf("Hello, %c!\n", name);
}

int main() {
    greet_user('J');  // 'J' is the argument
    return 0;
```