University of Global Village (UGV), Barishal

Department of Computer Science and Engineering

C Programming Guide Book

By

Galib Jaman

Lab Instructor

Department of Computer Science and Engineering

February 2025

Contents

Chapter 1

Introduction to C Programming

Let's think about a simple question. How does a computer understand anything? How does a computer differentiate between a number and a character? The simple answer is it doesn't. Then, how does it work? To answer this simple concept we've to first understand the fundamental building blocks of a computer. Electric circuits!
Now, let's think about a simple electric circuit. It doesn't understand numbers or characters either. But what it does understand is the presence or absence of electricity. If there is electricity, it's a 1. If there is no electricity, it's a 0. That's it! That's the fundamental building block of a computer. It's called a bit. A bit is the smallest unit of data in a computer. It can have only two values, 0 or 1. A group of 8 bits is called a byte. A byte can represent 256 different values (2^8).
That's why every instruction we give to a computer needs to be converted into a series of 0s and 1s. This is called machine language. It's the lowest level of programming languages. Let's take a simple example. Let's say we want to output "Hello, World!" to the screen. In machine language, it would look something like this:

Machine Language Code

```
01001000 01100101 01101100 01101100 01101111 00101100 00100000 01010111
01101111 01110010 01101100 01100100 00100001
```

It's not very readable, is it? That's where the programming languages such as C come in.

1.1 What is C?

C is a general-purpose, procedural computer programming language developed by Dennis Ritchie at Bell Labs in the early 1970s. It was designed to be a small, efficient language that could be used for a wide range of applications. C is a low-level language, which means that it provides a lot of control over the hardware of the computer. This makes it a powerful language, but also more difficult to learn and use than higher-level languages like Python or Java. Let's take a look at a simple "Hello, World!" program written in C:

```c
#include <stdio.h>

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

Listing 1.1: Hello World in C

This program will output "Hello, World!" to the screen. It's much easier to read and write than the machine language equivalent! Computers don't understand C directly, so we need to compile the C code into machine code before we can run it. This is done using a compiler. The compiler takes the C code as input and produces an executable file that the computer can run.

1.2 Installing an IDE and a Compiler

To write and run C programs, you need two things: an Integrated Development Environment (IDE) and a C compiler. An IDE is a software application that provides a set of tools for writing, compiling, and debugging code. A C compiler is a program that translates C code into machine code that the computer can run. There are many IDEs and compilers available for C programming. We will be using the Code::Blocks IDE and the GCC compiler in this guide. Here's how you can install them:

1. **Download Code::Blocks:**

   - Go to the official Code::Blocks website: `http://www.codeblocks.org/`
   - Navigate to the "Downloads" section.
   - Click on the "Download the binary release" link.
   - Choose the appropriate version for your operating system (Windows, macOS, or Linux).
   - Download the installer that includes the GCC compiler (usually named something like `codeblocks-XX.XXmingw-setup.exe` for Windows).

2. **Install Code::Blocks:**

   - Run the downloaded installer.
   - Follow the on-screen instructions to complete the installation.

- Make sure to select the option to install the GCC compiler during the installation process.

3. **Configure Code::Blocks:**

   - Launch Code::Blocks after the installation is complete.
   - Go to `Settings` > `Compiler`.
   - Ensure that the selected compiler is ”GNU GCC Compiler”.
   - Click on `OK` to save the settings.

4. **Create a .c File:**

   - Go to `File` > `New` > `Empty File`.
   - Go to `File` > `Save File As` and save the file with a `.c` extension (e.g., `main.c`).
   - Save the file in a location where you can easily access it.

5. **Write and Run Your First Program:**

   - Write your C code in the editor.
   - Click on the `Build` and `Run` buttons to compile and execute your program.

1.3 Basic Structure of a C Program

A C program consists of a series of functions. The main function is the entry point of the program, where the execution begins. Here's the basic structure of a C program:

```c
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello, World!\n");
5      return 0;
6  }
```

Listing 1.2: Basic Structure of a C Program

Let's break down the structure of the program:

- **`#include <stdio.h>`:** This line includes the standard input/output library, which provides functions like `printf()` and `scanf()` for input and output operations. Take a look at line 4 of the program. We used `printf()` function is to output text to the screen. That's why we need to include the `stdio.h` library. The `#include` directive tells the compiler to include the contents of the specified header file (`stdio.h` in this case) in the program.

- `int main(){..}:` This is the main function of the program. It is the entry point of the program, where the execution begins. The function signature `int main()` indicates that the main function returns an integer value. The curly braces `{}` enclose the body of the function.
- `return 0:` This statement indicates that the program has executed successfully. The value 0 is returned to the operating system to indicate that the program terminated without errors.

1.4 Compiling and Running a C Program

Okay, now that we have written our first C program, let's compile and run it. Here's how you can do it using Code::Blocks:

1. **Build and Run Your Program:**
   - Click on the `Build > Build and Run` button to compile and execute your program.
   - You should see the output "Hello, World!" in the console window.

1.5 Comments in C

Comments are used to explain the code and make it more readable. They are ignored by the compiler and do not affect the execution of the program. There are two types of comments in C:

- **Single-line comments:** Begin with `//` and continue until the end of the line.
- **Multi-line comments:** Begin with `/*` and end with `*/`. They can span multiple lines.

Here's an example of how to use comments in C:

```c
#include <stdio.h>

int main() {
   // single-line comment
   printf("Hello, World!\n"); // single-line comment
   /*
      This is a multi-line comment
      It can span multiple lines
   */
   return 0;
}
```

Listing 1.3: Using Comments in C

Comments are an essential part of programming. They help you and others understand the code better. It's a good practice to add comments to explain the purpose of the code, especially for complex or non-obvious parts of the program.

1.6 Excersises and Solutions

Exercise 1: Write a C program to print "Hello, World!" to the screen.

```c
#include <stdio.h>

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

Listing 1.4: Solution to Exercise 1

Output

```
Hello, World!
```

Exercise 2: Write a C program to print a square of asterisks (*) to the screen using only one printf statement. The square should be 5x5 in size.

```c
#include <stdio.h>

int main() {
   printf("*****\n");
```

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

Listing 1.5: Solution to Exercise 2

Output

```
*****
*****
*****
*****
*****
```

The `\n` character is used to move the cursor to the next line. By repeating the pattern `"*****\n"` five times, we create a 5x5 square of asterisks.

Exercise 3: Write a C program to print a triangle of asterisks (*) to the screen using only one printf statement. The triangle should be 3 lines high.

```c
#include <stdio.h>

int main() {
    printf("  *\n");
    printf(" ***\n");
    printf("*****\n");
    return 0;
}
```

Listing 1.6: Solution to Exercise 3

Output

```
  *
 ***
*****
```

Chapter 2

Data Types and Variables in C

Let take a character ‘A’ and a whole number 65. They are not the same thing, right? But in the computer’s memory, they are stored as a series of 0s and 1s. For example the character ‘A’ is stored as 01000001 and the number 65 is also stored as 01000001. So, how does the computer know whether it’s a character or a number? The answer is **data types**.

2.1 Data Type

Data types tell the computer how to interpret the data stored in memory. Which one is ‘A’ and which one is 65 in this case. In C, there are several built-in data types that you can use to declare variables. Let’s take a look at some of the common data types in C:

- **int:** Used to store integer values (whole numbers). For example, 5, -3, 0, etc.
- **float:** Used to store floating-point values (real numbers). For example, 3.14, -0.5, 2.0, etc.
- **double:** Similar to float but with higher precision. Used to store double-precision floating-point values. For example, 3.14159, -0.12345, etc.
- **char:** Any single character enclosed in single quotes. For example, ’A’, ’b’, ’1’, etc.
- **bool:** Used to store boolean values (true or false). In C, boolean values are represented as 0 (false) or 1 (true). Although C does not have a built-in boolean data type, you can use the `stdbool.h` header file to define boolean values.
- **void:** Represents an empty data type. It is commonly used as the return type of functions that do not return a value.

2.2 Variables

Imagine you have a box. You can put anything you want in that box. You can put a number, a character, a string, etc. In programming, a variable is like that box. It's a named storage location in memory where you can store a value. You can think of a variable as a box with a label on it. The label is the variable name, and the value inside the box is the data stored in the variable. Let's take a look at how you can declare variables in C:

```c
int age; // Declares an integer variable named age
float height; // Declares a float variable named height
char grade; // Declares a character variable named grade
```

Listing 2.1: Declaring Variables in C

Different data types require different amounts of memory to store the data. For example, an `int` variable requires 4 bytes of memory, a `float` variable requires 4 bytes, and a `char` variable requires 1 byte. The size of a variable depends on the data type and the system architecture (32-bit or 64-bit).

2.2.1 Declaration and Initialization

When you declare a variable, you are telling the compiler to reserve memory for that variable. You can also initialize the variable with an initial value at the time of declaration. Here's how you can declare and initialize variables in C:

```c
int age = 25;
float height = 5.8;
char grade = 'A';
```

Listing 2.2: Declaring and Initializing Variables in C

2.2.2 Variable Naming Rules

When naming variables in C, you need to follow certain rules:

- Variable names must begin with a letter or an underscore.
- Variable names can contain letters, digits, and underscores.
- Variable names are case-sensitive (e.g., `age`, `Age`, and `AGE` are three different variables).
- Variable names cannot be keywords or reserved words (e.g., `int`, `float`, `char`, etc.).

- Variable names should be descriptive and meaningful (e.g., `age`, `height`, `grade`, etc.).

Here are some examples of valid and invalid variable names:

```c
// Valid variable names
int age;
float height;
char grade;
int _count;
float totalAmount;

// Invalid variable names
int 1age; // Cannot start with a digit
float height@; // Cannot contain special characters
char grade%; // Cannot contain special characters
int total amount; // Cannot contain spaces
```

Listing 2.3: Valid and Invalid Variable Names

One more important thing to remember is values stored in variables can be changed during the execution of the program. For example, you can change the value of the `age` variable from 25 to 30 during the execution of the program like this:

```c
int age = 25; // Declare and initialize the age variable
age = 30; // Change the value of the age variable
```

Listing 2.4: Changing Variable Values

Notice that we didn't use the `int` keyword when changing the value of the variable. We only need to use the data type when declaring the variable, not when changing its value or using it in expressions.

2.2.3 Format Specifiers

When you want to print a variable of a specific data type using the `printf()` function or scan a value using the `scanf()` function which we will discuss later, you need to use format specifiers. Format specifiers are placeholders that tell the `printf()` and `scanf()` functions how to interpret the data. For example `%d` is used for integers. So, if you want to print an integer variable, you need to use the `%d` format specifier like this:

```c
int age = 25;
printf("My age is %d years.\n", age);
```

Listing 2.5: Using Format Specifiers

```
Output
My age is 25 years.
```

The `%d` format specifier is used to print integer values. Similarly, you can use other format specifiers for different data types. Heres a table to summarize all the concept we've discussed so far:

Data Type	Size (bytes)	Format Specifier	Example
int	4	%d	25, -3, 0
float	4	%f	3.14, -0.5, 2.0
double	8	%lf	3.14159, -0.12345
char	1	%c	'A', 'b', '1'
bool	1	%d	0 (false), 1 (true)

Table 2.1: Common Data Types in C

2.3 Constants

In addition to variables, you can also use constants in C. Constants are fixed values that do not change during the execution of the program. You can define constants using the `#define` directive or the `const` keyword. Here's how you can define constants in C:

```c
#define PI 3.14159 // Using #define directive
const int MAX_VALUE = 100; // Using const keyword
```

Listing 2.6: Defining Constants in C

Here's a example of how to use both variables and constants in a program:

```c
#include <stdio.h>

#define PI 3.14159
const int MAX_VALUE = 100;

int main() {
   int radius = 5;
   float area = PI * radius * radius;
   printf("The area of the circle is %f.\n", area);
   return 0;
}
```

Listing 2.7: Using Variables and Constants in C

Output

```
The area of the circle is 78.53975.
```

2.4 Taking User Input using scanf()

So far, we have been hardcoding the values of variables in our programs. But what if you want to take input from the user? You can use the `scanf()` function to read input from the user. Here's how you can use the `scanf()` function to read an integer value from the user:

```c
#include <stdio.h>

int main() {
   int age;
   printf("Enter your age: ");
   scanf("%d", &age);
   printf("Your age is %d.\n", age);
   return 0;
}
```

Listing 2.8: Reading Integer Input from User

Output

```
Enter your age: 25
Your age is 25.
```

The `scanf()` function takes two arguments: the format specifier `%d` and the address of the variable where the input will be stored using the `&` operator. The `&` operator is used to get the memory address of a variable. It is used to pass the address of the variable to the `scanf()` function so that it can store the input value in that memory location.

Don't use the `&` operator when using the `printf()` function to print the value of a variable. The `&` operator is only used with the `scanf()` function to read input from the user. If you use the `&` operator with the `printf()` function, it will print the memory address of the variable, not the value stored in the variable.

2.5 Arithmetic Operators

C provides a set of arithmetic operators that you can use to perform mathematical operations on variables. Here are some of the common arithmetic operators in C:

- + **(Addition):** Adds two operands.
- **- (Subtraction):** Subtracts the second operand from the first operand.
- * **(Multiplication):** Multiplies two operands.
- / **(Division):** Divides the first operand by the second operand.
- % **(Modulus):** Returns the remainder of the division of the first operand by the second operand. For example, $5\%2 = 1$.
- ++ **(Increment):** Increases the value of the operand by 1.
- – **(Decrement):** Decreases the value of the operand by 1.

Here's an example of how you can use arithmetic operators in C:

```c
#include <stdio.h>

int main() {
   int a = 5, b = 3;
   int sum = a + b;
   int difference = a - b;
   int product = a * b;
   int quotient = a / b;
   int remainder = a % b;
   printf("Sum: %d\n", sum);
   printf("Difference: %d\n", difference);
   printf("Product: %d\n", product);
   printf("Quotient: %d\n", quotient);
   printf("Remainder: %d\n", remainder);
   return 0;
}
```

Listing 2.9: Using Arithmetic Operators in C

Output

```
Sum: 8
Difference: 2
Product: 15
Quotient: 1
Remainder: 2
```

2.6 Exercises and Solutions

Exercise 1: Write a program to declare variables of different data types and display their sizes using sizeof().

```c
#include <stdio.h>

int main() {
   int a;
   float b;
   char c;
   printf("Size of int: %lu bytes\n", sizeof(a));
   printf("Size of float: %lu bytes\n", sizeof(b));
   printf("Size of char: %lu bytes\n", sizeof(c));
   return 0;
}
```

Listing 2.10: Solution to Exercise 1

Output

```
Size of int: 4 bytes
Size of float: 4 bytes
Size of char: 1 byte
```

The **`sizeof()`** operator is used to determine the size of a variable or data type in bytes. The **`%lu`** format specifier is used to print the size of the variable as an unsigned long integer.

Exercise 2: Write a program to calculate the area of a rectangle. Take the value of length and width from the user

```c
#include <stdio.h>

int main() {
   float length, width, area;
   printf("Enter the length of the rectangle: ");
   scanf("%f", &length);
   printf("Enter the width of the rectangle: ");
   scanf("%f", &width);
   area = length * width;
   printf("The area of the rectangle is %0.2f.\n", area);
   return 0;
}
```

Listing 2.11: Solution to Exercise 2

Output

```
Enter the length of the rectangle: 5
Enter the width of the rectangle: 3
The area of the rectangle is 15.00.
```

We can declare multiple variables of the same data type in a single line by separating them with commas. For example, `int a, b, c;` declares three integer variables `a`, `b`, and `c`. Similarly, we can declare and initialize variables of the same data type in a single line like this: `int a = 5, b = 10, c = 15;`.

The `%0.2f` format specifier is used to print the floating-point value with two decimal places. If you want to print the value with three decimal places, you can use the `%0.3f` format specifier and so on.

Exercise 3: Write a program to convert temperature from Celsius to Fahrenheit. Take the temperature in Celsius from the user and display the temperature in Fahrenheit. The formula to convert temperature from Celsius to Fahrenheit is: $F = \frac{9}{5} \times C + 32$

```c
#include <stdio.h>

int main() {
   float celsius, fahrenheit;
   printf("Enter the temperature in Celsius: ");
   scanf("%f", &celsius);
   fahrenheit = (9.0 / 5.0) * celsius + 32;
   printf("The temperature in Fahrenheit is %0.2f.\n", fahrenheit);
   return 0;
}
```

Listing 2.12: Solution to Exercise 3

Output

```
Enter the temperature in Celsius: 25
The temperature in Fahrenheit is 77.00.
```

Exercise 4: Write a program to calculate the area of a circle. Take the radius of the circle from the user. Use constant PI with a value of 3.14159. The formula to calculate the area of a circle is: $A = \pi \times r^2$

```c
#include <stdio.h>

#define PI 3.14159

int main() {
   float radius, area;
   printf("Enter the radius of the circle: ");
   scanf("%f", &radius);
   area = PI * radius * radius;
   printf("The area of the circle is %0.2f.\n", area);
   return 0;
}
```

Listing 2.13: Solution to Exercise 4

Output

```
Enter the radius of the circle: 5
The area of the circle is 78.54.
```

Exercise 5: Write a C program to convert specified days into years, weeks and days. Hint: 1 year = 365 days, 1 week = 7 days.

```c
#include <stdio.h>

int main() {
   int days, years, weeks, remainingDays;
   printf("Enter the number of days: ");
   scanf("%d", &days);
   years = days / 365;
   weeks = (days % 365) / 7;
   remainingDays = (days % 365) % 7;
   printf("Years: %d\n", years);
   printf("Weeks: %d\n", weeks);
   printf("Days: %d\n", remainingDays);
   return 0;
}
```

Listing 2.14: Solution to Exercise 5

Output

```
Enter the number of days: 3659
Years: 10
Weeks: 1
Days: 2
```

Chapter 3

Conditional Statements in C

Conditional statements allow you to make decisions in your program based on certain conditions. You can use conditional statements to execute different blocks of code depending on whether a condition is true or false. In C, you can use the `if`, `else`, and `else if` statements to implement conditional logic.

3.1 Comparison Operators

Comparison operators are used to compare two values and determine the relationship between them. Here are some common comparison operators in C:

- == **(Equal to):** Checks if two values are equal.
- != **(Not equal to):** Checks if two values are not equal.
- > **(Greater than):** Checks if the left operand is greater than the right operand.
- < **(Less than):** Checks if the left operand is less than the right operand.
- >= **(Greater than or equal to):** Checks if the left operand is greater than or equal to the right operand.
- <= **(Less than or equal to):** Checks if the left operand is less than or equal to the right operand.

3.2 Logical Operators

Logical operators are used to combine multiple conditions in a conditional statement. Here are some common logical operators in C:

- && **(Logical AND):** Returns true if both conditions are true.
- || **(Logical OR):** Returns true if at least one condition is true.

- **! (Logical NOT):** Returns true if the condition is false.

3.3 If Statement

The `if` statement is used to execute a block of code if a condition is true. Here's the syntax of the `if` statement in C:

```c
if (condition) {
   // Code to be executed if the condition is true
}
```

Listing 3.1: If Statement Syntax

Here's an example of how you can use the `if` statement in C:

```c
#include <stdio.h>

int main() {
   int age = 25;
   if (age >= 18) {
      printf("You are an adult.\n");
   }
   return 0;
}
```

Listing 3.2: Using If Statement in C

Output

```
You are an adult.
```

3.3.1 Else Statement

The `else` statement is used to execute a block of code if the condition in the `if` statement is false. Here's the syntax of the `else` statement in C:

```c
if (condition) {
   // Code to be executed if the condition is true
} else {
   // Code to be executed if the condition is false
}
```

Listing 3.3: Else Statement Syntax

Here's an example of how you can use the `else` statement in C:

```c
#include <stdio.h>

int main() {
   int age = 15;
   if (age >= 18) {
      printf("You are an adult.\n");
   } else {
      printf("You are a minor.\n");
   }
   return 0;
}
```

Listing 3.4: Using Else Statement in C

Output

```
You are a minor.
```

The `else` statement is optional but must be used with an `if` or `else if` statement. It cannot be used on its own.

3.3.2 Else If Statement

The `else if` statement is used to execute a block of code if the condition in the `if` statement is false and another condition is true. You can use multiple `else if` statements to check for multiple conditions. Here's the syntax of the `else if` statement in C:

```c
if (condition1) {
   // Code to be executed if condition1 is true
} else if (condition2) {
   // Code to be executed if condition2 is true
} else {
   // Code to be executed if all conditions are false
}
```

Listing 3.5: Else If Statement Syntax

Here's an example of how you can use the `else if` statement in C:

```c
#include <stdio.h>

int main() {
   int age = 15;
   if (age >= 18) {
      printf("You are an adult.\n");
```

```c
    } else if (age >= 13) {
        printf("You are a teenager.\n");
    } else {
        printf("You are a child.\n");
    }
    return 0;
}
```

Listing 3.6: Using Else If Statement in C

Output

```
You are a teenager.
```

The `else if` statement must be used after an `if` statement and before an `else` statement. You can have multiple `else if` statements to check for different conditions.

3.4 Switch Statement

The `switch` statement is used to execute different blocks of code based on the value of an expression. It is an alternative to using multiple `if` statements with `else if` statements. Here's an example of how you can use the `switch` statement in C:

```c
#include <stdio.h>

int main() {
    char grade = 'B';
    switch (grade) {
        case 'A':
            printf("Excellent!\n");
            break;
        case 'B':
            printf("Good!\n");
            break;
        case 'C':
            printf("Average!\n");
            break;
        case 'D':
            printf("Poor!\n");
            break;
        default:
            printf("Invalid grade!\n");
    }
    return 0;
}
```

Listing 3.7: Using Switch Statement in C

Output

```
Good!
```

The `break` statement is used to exit the `switch` statement after executing the code block for a particular case. If you omit the `break` statement, the code will continue to execute the code blocks for subsequent cases until it reaches a `break` statement or the end of the `switch` statement.

3.4.1 Why Use Switch Statement?

The `switch` statement is useful when you have multiple conditions to check against a single expression. It is more concise and easier to read than using multiple `if` statements with `else if` statements. The `switch` statement is also more efficient than using multiple `if` statements because the compiler can optimize the code better.

3.5 Exercises and Solutions

Exercise 1: Write a program to check if a number is positive, negative, or zero. Take the number from the user.

```c
#include <stdio.h>

int main() {
   int number;
   printf("Enter a number: ");
   scanf("%d", &number);
   if (number > 0) {
      printf("The number is positive.\n");
   } else if (number < 0) {
      printf("The number is negative.\n");
   } else {
      printf("The number is zero.\n");
   }
   return 0;
}
```

Listing 3.8: Solution to Exercise 1

Output

```
Enter a number: -5
The number is negative.
```

Exercise 2: Write a C program to find whether a given year is a leap year or not. Take the year from the user. A leap year is a year that is divisible by 4 but not divisible by 100, except for years that are divisible by 400.

```c
#include <stdio.h>

int main() {
   int year;
   printf("Enter a year: ");
   scanf("%d", &year);
   if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
      printf("%d is a leap year.\n", year);
   } else {
      printf("%d is not a leap year.\n", year);
   }
   return 0;
}
```

Listing 3.9: Solution to Exercise 2

Output

```
Enter a year: 2020
2020 is a leap year.
```

Exercise 3: Write a program to find the largest of three numbers. Take the numbers from the user.

```c
#include <stdio.h>

int main() {
   int num1, num2, num3;
   printf("Enter three numbers: ");
   scanf("%d %d %d", &num1, &num2, &num3);
   if (num1 >= num2 && num1 >= num3) {
      printf("%d is the largest number.\n", num1);
   } else if (num2 >= num1 && num2 >= num3) {
      printf("%d is the largest number.\n", num2);
   } else {
      printf("%d is the largest number.\n", num3);
```

```c
    }
    return 0;
}
```

Listing 3.10: Solution to Exercise 3

```
Output
Enter three numbers: 5 10 3
10 is the largest number.
```

Exercise 4: Write a C program to accept a coordinate point in an XY coordinate system and determine in which quadrant the coordinate point lies. Take the values of X and Y from the user.

```c
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter the values of X and Y: ");
    scanf("%d %d", &x, &y);
    if (x > 0 && y > 0) {
        printf("First quadrant.\n", x, y);
    } else if (x < 0 && y > 0) {
        printf("Second quadrant.\n", x, y);
    } else if (x < 0 && y < 0) {
        printf("Third quadrant.\n", x, y);
    } else if (x > 0 && y < 0) {
        printf("Fourth quadrant.\n", x, y);
    } else {
        printf("The point (%d, %d) lies on the origin.\n", x, y);
    }
    return 0;
}
```

Listing 3.11: Solution to Exercise 4

```
Output
Enter the values of X and Y: 5 -3
The point (5, -3) lies in the fourth quadrant.
```

Exercise 5: Write a C program to check whether a triangle can be formed with the given values for the angles/sides. To form a triangle, the sum of any two sides must be greater than the third side. Take the lengths of the three

sides from the user.

```c
#include <stdio.h>

int main() {
   int side1, side2, side3;
   printf("Enter the lengths of the three sides: ");
   scanf("%d %d %d", &side1, &side2, &side3);
   if (side1 + side2 > side3 && side2 + side3 > side1 && side1 + side3 > side2) {
      printf("A triangle can be formed.\n");
   } else {
      printf("A triangle cannot be formed.\n");
   }
   return 0;
}
```

Listing 3.12: Solution to Exercise 5

Output

```
Enter the lengths of the three sides: 5 10 15
A triangle cannot be formed.
```

Exercise 6: Write a C program to check whether a character is an alphabet, digit or special character. Take the character from the user.

```c
#include <stdio.h>

int main() {
   char ch;
   printf("Enter a character: ");
   scanf("%c", &ch);
   if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) {
      printf("%c is an alphabet.\n", ch);
   } else if (ch >= '0' && ch <= '9') {
      printf("%c is a digit.\n", ch);
   } else {
      printf("%c is a special character.\n", ch);
   }
   return 0;
}
```

Listing 3.13: Solution to Exercise 6

Output

```
Enter a character: 5
5 is a digit.
```

Chapter 4

Loops and Iteration in C

Suppose you want to print the numbers from 1 to 5. You can do this by writing five `printf()` statements like this:

```c
printf("1\n");
printf("2\n");
printf("3\n");
printf("4\n");
printf("5\n");
```

Output

```
1
2
3
4
5
```

It's one way of doing it. But what if you want to print the numbers from 1 to 100? or 1 to 1000? Writing that many `printf()` statements would be hard to say the least. This is where loops enters the picture like a superhero. Loops allow you to execute a block of code multiple times. Every loop has three main components:

- **Strating Point:** The starting point of the loop.
- **Ending Condition:** The condition that is checked before each iteration of the loop.
- **Increment/Decrement:** The value by which the loop variable is incremented or decremented after each iteration.

Here's a flow chart of how a loop works:

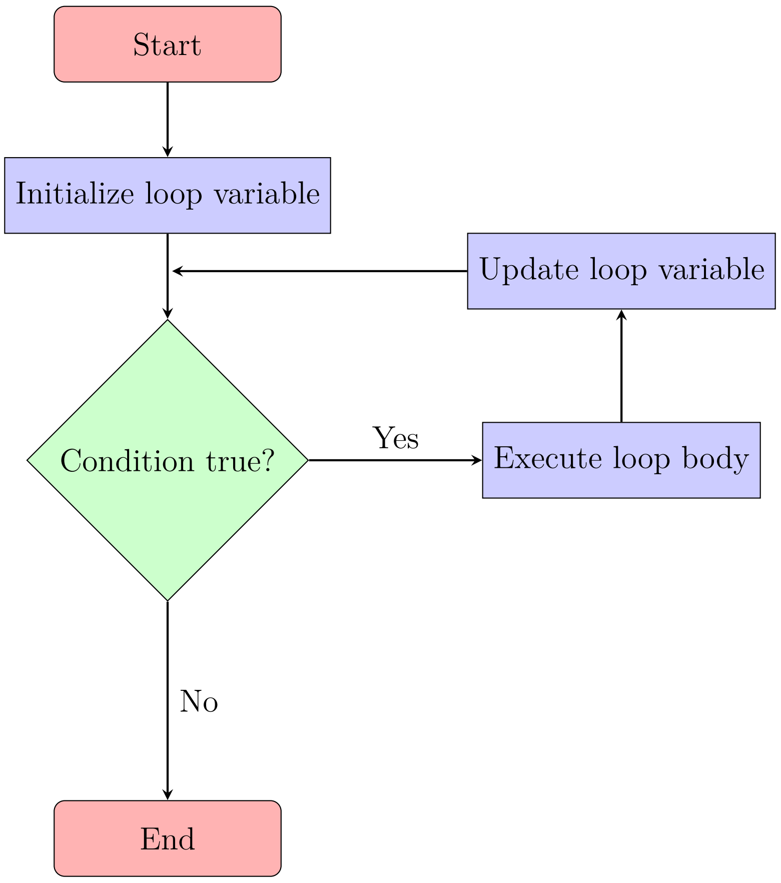

Figure 4.1: Flow chart of a loop

4.1 While Loop

The `while` loop is used to execute a block of code as long as a condition is true. Here's the syntax of the `while` loop in C:

```c
while (condition) {
    // Code to be executed
}
```

Listing 4.1: While Loop Syntax

Here's an example of how you can use the `while` loop in C to print the numbers from 1 to 5:

```c
#include <stdio.h>

int main() {
   int i = 1; // Part 1: Starting Point
   while (i <= 5) // Part 2: Ending Condition
   {
      printf("%d\n", i);
      i++; // Part 3: Increment
   }
   return 0;
}
```

Listing 4.2: Using While Loop in C

Output

```
1
2
3
4
5
```

Make sure that all the three components of the loop are present in the `while` loop. If you forget to update the loop variable, the loop will run indefinitely and cause an infinite loop.

4.1.1 When to Use While Loop?

The `while` loop is used when you don't know the number of iterations in advance and want to execute a block of code as long as a condition is true. For example, you can use a `while` loop to read input from the user until a specific value is entered. On of the most common use of `while` loop is the **game loop in game development**.

4.2 For Loop

The `for` loop is used to execute a block of code a specified number of times. It is more concise than the `while` loop when you know the number of iterations in advance. Here's the syntax of the `for` loop in C:

```c
for (initialization; condition; update) {
    // Code to be executed
}
```

Listing 4.3: For Loop Syntax

Here's an example of how you can use the `for` loop in C to print the numbers from 1 to 5:

```c
#include <stdio.h>

int main() {
   for (int i = 1; i ⩽ 5; i++) {
      printf("%d\n", i);
   }
   return 0;
}
```

Listing 4.4: Using For Loop in C

Output

```
1
2
3
4
5
```

We can declare the loop variable `int i` in this case, inside the `for` loop itself. This is called a **local variable** and it is only accessible within the loop.

Make sure that all the three components of the loop are present in the `for` loop and are separated by semicolons `;` **not commas** `,`.

4.2.1 When to Use For Loop?

The `for` loop is used when you know the number of iterations in advance and want to execute a block of code a specified number of times. It is more concise than the `while` loop when you know the starting point, ending condition, and increment/decrement value.

4.3 Do-While Loop

The `do-while` loop is similar to the `while` loop, but the condition is checked at the end of the loop. This means that the block of code is executed at least once, even if the condition is false. Here's the syntax of the `do-while` loop in C:

```c
do {
    // Code to be executed
} while (condition);
```

Listing 4.5: Do-While Loop Syntax

Here's the flow chart of how a `do-while` loop works:

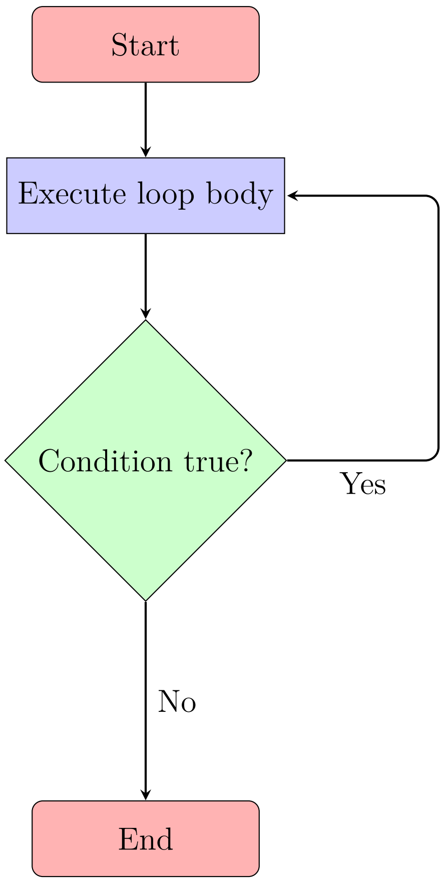

Figure 4.2: Flow chart of a do-while loop

Here's an example of how you can use the `do-while` loop in C to print the numbers from 1 to 5:

```c
#include <stdio.h>

int main() {
   int i = 1;
   do {
      printf("%d ", i);
      i++;
   } while (i <= 5);
   return 0;
}
```

Listing 4.6: Using Do-While Loop in C

Output

```
1 2 3 4 5
```

4.3.1 When to Use Do-While Loop?

The `do-while` loop is used when you want to execute a block of code at least once, even if the condition is false. It is useful when you want to execute the loop body before checking the condition.

4.4 Break and Continue Statements

The `break` and `continue` statements are used to control the flow of a loop. Here's how they work:

- **Break Statement:** The `break` statement is used to exit the loop immediately. It is used to terminate the loop prematurely.
- **Continue Statement:** The `continue` statement is used to skip the current iteration of the loop and continue with the next iteration. It is used to skip the remaining code in the loop body and move to the next iteration.

Here's an example of how you can use the `break` and `continue` statements in a loop:

```c
#include <stdio.h>

int main() {
   for (int i = 1; i <= 10; i++) {
      if (i == 3) {
         continue; // Skip the current iteration
      }
      if (i == 5) {
         break; // Exit the loop
      }
      printf("%d ", i);
   }
   return 0;
}
```

Listing 4.7: Using Break and Continue Statements in C

Output

```
1 2 4
```

Notice that the `continue` statement skips the number 3 and the `break` statement exits the loop when the number 5 is encountered even though our stopping condition is `i <= 10`.

4.5 Nested Loops

A nested loop is a loop inside another loop. You can use nested loops to perform more complex tasks that require multiple iterations. Let's say we want to print a pattern like this.

Output

```
*
* *
* * *
* * * *
* * * * *
```

Let's analyse go through the solution step by step.

Step 1: Take a closer look at the pattern. How many rows are there? There are 5 rows or in other words, we have 5 new lines. Isn't it? So, we need a loop that runs 5 times to print 5 new lines. So, let's print 5 new lines first.

```c
for (int i = 1; i <= 5; i++) {
    printf("\n"); // <--- New line
}
```

Step 2: But wait! We also need to print certain number of stars in each row before we print a new line. Okay, so how many stars do we need to print? Take a look at the pattern again. In the first row, we need to print 1 star, in the second row, we need to print 2 stars, and so on. So, we need to print `i` stars in the `i`th row. So our loop will start from 1 and run till the current row number. Let's print the stars now.

```c
for (int i = 1; i <= 5; i++) {

    // start: inner loop to print stars
    for (int j = 1; j <= i; j++) {
        printf("* ");
    }
    // end: inner loop

    printf("\n"); // <--- New line
}
```

4.6 Exercises and Solutions

Exercise 1: Write a program to print the numbers from 1 to 10 using a while loop.

Example Output

```
1 2 3 4 5 6 7 8 9 10
```

```c
#include <stdio.h>

int main() {
    int i = 1;
    while (i <= 10) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

Listing 4.8: Solution to Exercise 1

Exercise 2: Write a program in C to display n terms of natural numbers and their sum.

Example Output

```
Enter the number of terms: 5
The first 5 natural numbers are: 1 2 3 4 5
The sum of the first 5 natural numbers is: 15
```

```c
#include <stdio.h>

int main() {
   int n, sum = 0;
   printf("Enter the number of terms: ");
   scanf("%d", &n);
   printf("The first %d natural numbers are: ", n);
   for (int i = 1; i <= n; i++) {
      printf("%d ", i);
      sum += i;
   }
   printf("\nThe sum of the first %d natural numbers is: %d\n", n, sum);
   return 0;
}
```

Listing 4.9: Solution to Exercise 2

Exercise 3: Write a program in C to display the multiplication table for a given integer.

Example Output

```
Enter an integer: 7
Multiplication table for 7:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

```c
#include <stdio.h>

int main() {
   int n;
   printf("Enter an integer: ");
   scanf("%d", &n);
   printf("Multiplication table for %d:\n", n);
   for (int i = 1; i <= 10; i++) {
      printf("%d x %d = %d\n", n, i, n * i);
   }
   return 0;
}
```

Listing 4.10: Solution to Exercise 3

Exercise 4: Write a program in C to check if the number is a prime number or not. A prime number is a number that is greater than 1 and divisible by 1 and itself only.

Example Output

```
Enter a number: 7
7 is a prime number.
```

or

Example Output

```
Enter a number: 6
6 is not a prime number.
```

```c
#include <stdio.h>

int main() {
   int n, isPrime = 1;
   printf("Enter a number: ");
   scanf("%d", &n);

   // Check if the number is divisible by any number other than 1 and itself
   for (int i = 2; i <= n / 2; i++) {
      if (n % i == 0) {
         isPrime = 0;
         break;
      }
   }

   // Output the result
   if (n == 1) {
      printf("%d is not a prime number.\n", n);
   } else if (isPrime) {
      printf("%d is a prime number.\n", n);
   } else {
      printf("%d is not a prime number.\n", n);
   }
   return 0;
}
```

Listing 4.11: Solution to Exercise 4

Exercise 5: Write a program in C to display the Fibonacci sequence up to a certain number of terms. The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones, usually starting with 0 and 1.

Example Output

```
Enter the number of terms: 10
Fibonacci sequence up to 10 terms: 0 1 1 2 3 5 8 13 21 34
```

```c
#include <stdio.h>

int main() {
   int n, t1 = 0, t2 = 1, nextTerm;
   printf("Enter the number of terms: ");
   scanf("%d", &n);
   printf("Fibonacci sequence up to %d terms: ", n);
   for (int i = 1; i <= n; i++) {
      printf("%d ", t1);
      nextTerm = t1 + t2;
      t1 = t2;
      t2 = nextTerm;
   }
   return 0;
}
```

Listing 4.12: Solution to Exercise 5

Exercise 6: Write a program in C to display the following pattern using nested loops.

Example Output

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```c
#include <stdio.h>

int main() {
   for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= i; j++) {
         printf("%d ", j);
      }
      printf("\n");
   }
   return 0;
}
```

Listing 4.13: Solution to Exercise 6

Exercise 7: Write a program in C to display the following pattern using nested loops.

Example Output

```
   1
  2 3
 4 5 6
7 8 9 10
```

```c
#include <stdio.h>

int main() {
   int n = 4;
   int k = 1;
   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n - i; j++) {
         printf(" ");
      }
      for (int j = 1; j <= i; j++) {
         printf("%d ", k++);
      }
      printf("\n");
```

```c
    }
    return 0;
}
```

Listing 4.14: Solution to Exercise 7

In the above solution, we use two nested loops. The outer loop runs from 1 to 4 (number of rows) and the inner loop runs from 1 to `n - i` to print the spaces before the numbers. The second inner loop runs from 1 to `i` to print the numbers. `k++` is used to print the current value of `k` and increment it by 1.

Exercise 8: Write a C program to calculate the factorial of a given number using a for loop. The factorial of a number is the product of all positive integers less than or equal to the number.

Example Output

```
Enter a number: 5
Factorial of 5 is 120
```

```c
#include <stdio.h>

int main() {
    int n, factorial = 1;
    printf("Enter a number: ");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        factorial *= i;
    }
    printf("Factorial of %d is %d\n", n, factorial);
    return 0;
}
```

Listing 4.15: Solution to Exercise 8

Exercise 9: Write a C program to reverse a given number using a do-while loop. Print the original and reversed numbers.

Example Output

```
Enter a number: 1284
Original number: 1284
Reversed number: 4821
```

```c
#include <stdio.h>

int main() {
   int n, reversedNumber = 0, remainder;
   printf("Enter a number: ");
   scanf("%d", &n);
   int originalNumber = n;
   do {
      remainder = n % 10;
      reversedNumber = reversedNumber * 10 + remainder;
      n /= 10;
   } while (n != 0);
   printf("Original number: %d\n", originalNumber);
   printf("Reversed number: %d\n", reversedNumber);
   return 0;
}
```

Listing 4.16: Solution to Exercise 9

Exercise 10: Write a C program to count the number of digits in a given integer.

Example Output

```
Enter an integer: 12948
Number of digits: 5
```

```c
#include <stdio.h>

int main() {
   int n, count = 0;
   printf("Enter an integer: ");
   scanf("%d", &n);
   while (n != 0) {
      n /= 10;
      count++;
   }
   printf("Number of digits: %d\n", count);
   return 0;
}
```

Listing 4.17: Solution to Exercise 10

Exercise 11: Create a program to find all prime numbers in a given range using nested loops and conditionals.

Example Output

```
Enter the lower limit: 10
Enter the upper limit: 50
Prime numbers between 10 and 50 are: 11 13 17 19 23 29 31 37 41 43 47
```

```c
#include <stdio.h>

int main() {
   int lower, upper, isPrime;
   printf("Enter the lower limit: ");
   scanf("%d", &lower);
   printf("Enter the upper limit: ");
   scanf("%d", &upper);
   printf("Prime numbers between %d and %d are: ", lower, upper);
   for (int i = lower; i <= upper; i++) {
      isPrime = 1;
      for (int j = 2; j <= i / 2; j++) {
         if (i % j == 0) {
            isPrime = 0;
            break;
         }
      }
      if (isPrime && i != 1) {
         printf("%d ", i);
      }
   }
   return 0;
}
```

Listing 4.18: Solution to Exercise 11

Exercise 12: Implement a program to repeatedly ask the user for a password until the correct one is entered.

Example Output

```
Enter the password: 1234
Incorrect password. Try again.
Enter the password: 5678
Incorrect password. Try again.
Enter the password: 9876
Correct password. Access granted.
```

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

```c

int main() {
   int password = 9876, input;
   do {
      printf("Enter the password: ");
      scanf("%d", &input);
      if (input != password) {
         printf("Incorrect password. Try again.\n");
      }
   } while (input != password);
   printf("Correct password. Access granted.\n");
   return 0;
}
```

Listing 4.19: Solution to Exercise 12

Exercise 13: Write a program in C to find the number and sum of all integers between two which are divisible by 7.

Example Output

```
Enter the lower limit: 10
Enter the upper limit: 50
Numbers between 10 and 50 divisible by 7: 14 21 28 35 42 49
Sum of numbers between 10 and 50 divisible by 7: 189
```

```c
#include <stdio.h>

int main() {
   int lower, upper, sum = 0;
   printf("Enter the lower limit: ");
   scanf("%d", &lower);
   printf("Enter the upper limit: ");
   scanf("%d", &upper);
   printf("Numbers between %d and %d divisible by 7: ", lower, upper);
   for (int i = lower; i <= upper; i++) {
      if (i % 7 == 0) {
         printf("%d ", i);
         sum += i;
      }
   }
   printf("\nSum of numbers between %d and %d divisible by 7: %d\n", lower,
       upper, sum);
   return 0;
}
```

Listing 4.20: Solution to Exercise 13