C programming

Deva

January 26, 2025

Contents

1 Introduction

1.1 History

History of C language is interesting to know. Here we are going to discuss a brief history of the c language.

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the founder of the c language.

It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL.

1.2 Prerequisites for writing first C program

1.2.1 Header Files:

The #include directives at the beginning of the program are used to include header files. Header files provide function prototypes and definitions that allow the C compiler to understand the functions used in the program.

1.2.2 Main Function:

Every C program starts with the main function. It is the program's entry point, and execution starts from here. The main function has a return type of int, indicating that it should return an integer value to the operating system upon completion.

1.2.3 Variable Declarations:

Before using any variables, you should declare them with their data types. This section is typically placed after the main function's curly opening brace.

1.2.4 Statements and Expressions:

This section contains the actual instructions and logic of the program. C programs are composed of statements that perform actions and expressions that compute values.

1.2.5 Comments:

Comments are used to provide human-readable explanations within the code. They are not executed and do not affect the program's functionality. In C, comments are denoted by // for single-line comments and /* */ for multi-line comments.

1.2.6 Functions:

C programs can include user-defined functions and blocks of code that perform specific tasks. Functions help modularize the code and make it more organized and manageable.

1.2.7 Return Statement:

Use the return statement to terminate a function and return a value to the caller function. A return statement with a value of 0 typically indicates a successful execution in the main function, whereas a non-zero value indicates an error or unexpected termination.

1.2.8 Standard Input/Output:

C has library functions for reading user input (scanf) and printing output to the console (printf). These functions are found in C programs and are part of the standard I/O library (stdio.h header file). It is essential to include these fundamental features correctly while writing a simple C program to ensure optimal functionality and readability.

1.2.9 Additional Information:

There is some additional information about the C programs. Some additional information is as follows:

1.2.10 Preprocessor Directives:

C programs often include preprocessor directives that begin with a # symbol. These directives are processed by the preprocessor before actual compilation and are used to include header files, define macros, and perform conditional compilation.

1.2.11 Data Types:

C supports data types such as int, float, double, char, etc. It depends on the program's requirements, and appropriate data types should be chosen to store and manipulate data efficiently.

1.2.12 Control Structures:

C provides control structures like if-else, while, for, and switch-case that allow you to make decisions and control the flow of the program.

1.2.13 Error Handling:

Robust C programs should include error-handling mechanisms to handle unexpected situations gracefully. Techniques like exception handling (using try-catch in C++) or returning error codes are commonly employed.

1.2.14 Modularization:

As programs grow in complexity, it becomes essential to modularize the code by creating separate functions for different tasks. This practice improves code reusability and maintainability.

Remember, the architecture and complexity of a C program can vary significantly depending on the specific application and requirements. The outline is a general overview of a simple C program's structure.

1.3 First Program

Listing 1: Example C++

```
#include <stdio.h>
int main(){
        printf("Hello␣C␣Language");
        return 0;
}
```

Let us first study the various parts of this C program:

1.3.1 #include <stdio.h>:

In this line, the program includes the standard input/output library (stdio.h) due to the pre-processor directive. For input and output tasks, the stdio.h library contains methods like printf and scanf.

1.3.2 int main() ...

:

It is the main function which is the entry point of the C program. The program starts executing from the beginning of the main function.

1.3.3 printf(”Hello World!”);

:

Use the printf() function to print formatted output to the console. In this example, the string ”Hello, C Language” is printed, followed by a newline character (n) which moves the pointer to the following line after the message is displayed.

1.3.4 return 0;

;

When the return statement is 0, the program has been completed. When determining the state of a program, the operating system frequently uses the value returned by the main function. A return value of 0 often indicates that the execution was successful.

After compilation and execution, this C program will quit with a status code 0 and output ”Hello, C Language” to the terminal.

The ”Hello, C Language” program is frequently used as an introduction to a new programming language since it introduces learners to essential concepts such as text output and the structure of a C program and provides a rapid way to validate that the working environment is correctly set up.

To write, compile, and run your first C program, follow these steps:

1. **Open a text editor** Open a text editor of your choice, such as Notepad, Sublime Text, or Visual Studio Code. It will be where you write your C code.

2. **Write the C program** Now, copy and paste the following code into the text editor:

   Listing 2: Example C++

   ```
   #include <stdio.h>
   int main(){
           printf("Hello␣C␣Language");
           return 0;
   }
   ```

3. **Save the file** After that, save the file with a .c extension such as first_program.c. This extension indicates that it is a C source code file.

4. **Compile the program** Now, compile the program in the command prompt.

5. **Run the program** After successful compilation, you can run the program by executing the generated executable file. Enter the following command into the terminal or command.

prompt: `./first_program` The program will execute, and you will see the output on the console:

Output:

Listing 3: Bash

```
Hello, C Language
```

2 Header Files

Header files in C are files with a `` `.h` `` extension that contain function declarations, macro definitions, type definitions, and other information that you want to share across multiple source files. They are used to promote code modularity and reusability by separating declarations from their implementations.

2.1 Why Use Header Files?

1. **Reusability:** You can reuse the declarations in multiple programs without rewriting them.
2. **Modularity:** Code becomes more organized by separating interface (declarations) from implementation (definitions).
3. **Readability:** Header files make your codebase easier to understand by summarizing what a program provides.
4. **Maintenance:** If you need to update a function's interface, you only have to modify the header file.

2.2 How Header Files Work

Inclusion in Source Files:

A header file is included in a source file (`` `.c` ``) using the `` `#include` `` directive. For example:

Listing 4: Example C++

```
#include "my_header.h"
```

This literally copies the contents of the header file into the source file at compile time.

2.2.1 Guards Against Multiple Inclusions:

Use **include guards** or `` `#pragma once` `` to prevent a header file from being included multiple times in the same program, which would lead to errors. For example: "'c

Listing 5: Example C++

```
#ifndef HEADER\_FILE\_NAME\_H
#define HEADER\_FILE\_NAME\_H

// Declarations go here

#endif // HEADER\_FILE\_NAME\_H
```

2.2.2 Separation of Declaration and Definition:

- Declaration: A function or variable is declared in the header file. For example:

Listing 6: Example C++

```
int add(int a, int b); // Declaration
```

- Definition: The actual implementation of the function is written in a source file. For example:

Listing 7: Example C++

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

2.3 Types of Header Files

2.3.1 1. Standard Header Files:

These are built into the C Standard Library and include:

- `` `<stdio.h>` `` for input/output functions.
- `` `<stdlib.h>` `` for general utilities like memory allocation.
- `` `<string.h>` `` for string manipulation functions.
- `` `<math.h>` `` for mathematical operations.

Example:

Listing 8: Example C++

```
#include <stdio.h>
printf("Hello,␣World!\n");
```

2.3.2 2. Custom Header Files:

These are user-defined and created for modular code. You can define your own functions, macros, or constants and include them in your projects.

2.3.3 Example of How Header Files Work

Here's a simple example to understand the concept:
Custom Header File: `` `my_header.h` ``

Listing 9: Example C++

```
#ifndef MY_HEADER_H
#define MY_HEADER_H

// Function declaration
int square(int x);

#endif // MY_HEADER_H
```

Implementation File: `` `my_header.c` ``

Listing 10: Example C++

```
#include "my_header.h"

int square(int x) {
        return x * x;
}
```

Main File: ‘main.c‘

Listing 11: Example C++

```
#include <stdio.h>
#include "my_header.h"

int main() {
        int number = 5;
        printf("Square␣of␣%d␣is␣%d\n", number, square(number));
        return 0;
}
```

Compilation and Execution

Listing 12: Bash

```
gcc main.c my_header.c -o program
```

2.4 Best Practices for Header Files

1. **Use Include Guards or `` `#pragma once` ``** Prevent multiple inclusions to avoid compiler errors.
2. **Keep Headers Lightweight** Avoid putting large implementations in the header file—use them only for declarations.
3. **Group Related Declarations** Group functions, macros, and type definitions logically for better organization.
4. **Document the Header File** Provide comments to explain the purpose of each declaration for clarity.

Header files are a cornerstone of clean and maintainable C programming. They are especially useful in large projects to manage code complexity and allow teamwork by separating interfaces from implementations.

3 Exercise: Using Custom Header Files

This exercise will guide you in creating and using a custom header file in C.

3.1 Task

1. Create a header file `math_operations.h` that declares two functions:

- A function to add two integers.
- A function to multiply two integers.

2. Implement the functions in `math_operations.c`.
3. Write a `main.c` program that uses these functions to perform operations and print the results.

3.2 Solution

3.2.1 Header File: `math_operations.h`

Listing 13: math_operations.h

```c
#ifndef MATH_OPERATIONS_H
#define MATH_OPERATIONS_H

// Function declarations
int add(int a, int b);
int multiply(int a, int b);

#endif // MATH_OPERATIONS_H
```

3.2.2 Source File: `math_operations.c`

Listing 14: math_operations.c

```c
#include "math_operations.h"

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

int multiply(int a, int b) {
        return a * b;
}
```

3.2.3 Main Program: `main.c`

Listing 15: main.c

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

int main() {
        int num1, num2;

        // Input two numbers
        printf("Enter␣the␣first␣number:␣");
        scanf("%d", &num1);
        printf("Enter␣the␣second␣number:␣");
        scanf("%d", &num2);

        // Use functions from the custom header
        int sum = add(num1, num2);
        int product = multiply(num1, num2);

        // Print results
        printf("Sum:␣%d\n", sum);
        printf("Product:␣%d\n", product);

        return 0;
}
```

3.3 Compilation and Execution

Compile and run the program using the following commands:

Listing 16: Compilation and Execution Commands

```
gcc main.c math_operations.c -o program
./program
```

3.4 Expected Output

```
Enter the first number: 5
Enter the second number: 10
Sum: 15
Product: 50
```

4 Best Practices for Header Files

- Use include guards or `#pragma once` to prevent multiple inclusions.
- Keep header files lightweight by avoiding large implementations.
- Group related declarations logically.
- Add comments to improve readability and maintainability.