C Functions Notes

1. Introduction to Functions

- **Definition**: A function is a block of code that performs a specific task.
- **Purpose**: Functions help in code reusability, organization, and modular programming.

2. Function Declaration

- **Syntax**:

  cCopy

  ```c
  return_type function_name(parameter_list);
  ```

- **Example**:

  cCopy

  ```c
  int add(int a, int b);
  ```

3. Function Definition

- **Syntax**:

  cCopy

  ```c
  return_type function_name(parameter_list) {
      // body of the function
  }
  ```

- **Example**:

  cCopy

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

4. Function Call

- **Syntax**:

  cCopy

  ```c
  function_name(arguments);
  ```

- **Example**:

  cCopy

  ```c
  int result = add(5, 10);
  ```

5. Types of Functions

- **Built-in Functions**: Functions provided by the C standard library (e.g., `printf`, `scanf`).
- **User-defined Functions**: Functions created by the user to perform specific tasks.

6. Function Arguments

- **Passing Arguments**: Arguments can be passed by value or by reference.
  - **By Value**: A copy of the variable is passed.
  - **By Reference**: The address of the variable is passed.

7. Return Values

- Functions can return a value using the `return` statement.
- If no value is returned, the return type should be `void`.

8. Example of a Complete Program

cCopy

```c
#include <stdio.h>

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

// Main Function
int main() {
    int sum = add(5, 10);
    printf("Sum: %d\n", sum);
    return 0;
}

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

9. Common Errors

- Forgetting to declare a function before using it.
- Mismatched return types.
- Incorrect number of arguments in function calls.

Conclusion

Functions are essential for writing clean, efficient, and maintainable code in C. Understanding how to declare, define, and call functions is fundamental for any programmer.

Next Steps

- Practice writing your own functions.
- Explore built-in functions in the C standard library.

C Programming Notes

1. Introduction to C

- **Overview**: C is a general-purpose programming language that is widely used for system programming, embedded systems, and application development.
- **History**: Developed in the early 1970s by Dennis Ritchie at Bell Labs.
- **Features**:
  - Low-level access to memory
  - Simple and efficient
  - Structured programming language

2. Setting Up the Environment

- **Compilers**: Popular C compilers include GCC (GNU Compiler Collection), Clang, and Microsoft Visual C++.
- **IDE Options**: Integrated Development Environments like Code::Blocks, Dev-C++, and Visual Studio can be used for writing and compiling C code.

3. Basic Syntax

- **Structure of a C Program**: A simple C program includes:
  - Preprocessor directives (e.g., `#include <stdio.h>`)
  - The `main` function
  - Statements and expressions
- **Example**:

  cCopy

```c
#include <stdio.h>

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

4. Data Types

- **Basic Data Types**: `int`, `float`, `double`, `char`
- **Derived Data Types**: Arrays, Structures, Unions, Pointers
- **Enumeration**: Using `enum` to define a variable that can hold a set of predefined constants.

5. Operators

- **Arithmetic Operators**: `+`, `-`, `*`, `/`, `%`
- **Relational Operators**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical Operators**: `&&`, `||`, `!`

6. Control Structures

- **Conditional Statements**: `if`, `else`, `switch`
- **Loops**: `for`, `while`, `do while`

7. Functions

- **Definition**: A function is a block of code that performs a specific task.
- **Purpose**: Functions promote code reusability and modularity.

7.1 Function Declaration

- **Syntax**:

cCopy

```c
return_type function_name(parameter_list);
```

- **Example**:

cCopy

```c
int add(int a, int b);
```

7.2 Function Definition

- **Syntax**:

cCopy

```c
return_type function_name(parameter_list) {
    // body of the function
}
```

- **Example**:

cCopy

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

7.3 Function Call

- **Syntax**:

cCopy

```c
function_name(arguments);
```

- **Example**:

cCopy

```c
int result = add(5, 10);
```

7.4 Types of Functions

- **Built-in Functions**: Functions provided by the C standard library (e.g., `printf`, `scanf`).
- **User-defined Functions**: Functions created by the user to perform specific tasks.

7.5 Function Arguments

- **Passing Arguments**: By value or by reference.

7.6 Return Values

- Functions can return a value using the `return` statement.

7.7 Example of a Complete Program

cCopy

```c
#include <stdio.h>

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

// Main Function
int main() {
    int sum = add(5, 10);
    printf("Sum: %d\n", sum);
    return 0;
}

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

8. Common Errors

- Forgetting to declare a function before using it.
- Mismatched return types.
- Incorrect number of arguments in function calls.

Conclusion

Understanding the basics of C programming, including functions, is essential for writing efficient and maintainable code.

Next Steps

- Practice writing your own functions and programs.
- Explore more advanced topics in C programming.

C Functions Notes

1. Introduction to Functions

- **Definition**: A function is a block of code that performs a specific task.
- **Purpose**: Functions help in code reusability, organization, and modular programming.

2. Function Declaration

- **Syntax**:

  cCopy

  ```c
  return_type function_name(parameter_list);
  ```

- **Example**:

  cCopy

  ```c
  int add(int a, int b);
  ```

3. Function Definition

- **Syntax**:

  cCopy

  ```c
  return_type function_name(parameter_list) {
      // body of the function
  }
  ```

- **Example**:

  cCopy

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

4. Function Call

- **Syntax**:

  cCopy

  ```c
  function_name(arguments);
  ```

- **Example**:

  cCopy

  ```c
  int result = add(5, 10);
  ```

5. Types of Functions

- **Built-in Functions**: Functions provided by the C standard library (e.g., `printf`, `scanf`).
- **User-defined Functions**: Functions created by the user to perform specific tasks.

6. Function Parameters or Arguments

- **Passing Arguments**: We pass only variable names and their values as function parameters or arguments.
- **By Value**: A copy of the variable is passed.
- **By Reference**: The address of the variable is passed.

7. Declaration or Declaring Variables

- **Examples**:

cCopy

```c
int a;
int b;
string name[30];
char name[30];
float c;
double h;
char k;
```

8. Initialization or Initializing Variables

- **Examples**:

cCopy

```c
int a = 10;
int b = 20;
string name[30] = "Mahalakshmy";
char name[30] = "Mahalakshmy";
float c = 5.2;
double h = 9.9999;
char k = 'M';
```

9. Return Values

- Functions can return a value using the `return` statement.
- If no value is returned, the return type should be `void`.

10. Example of a Complete Program

cCopy

```c
#include <stdio.h>

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

// Main Function
int main() {
    int sum = add(5, 10);
    printf("Sum: %d\n", sum);
    return 0;
}

// Function Definition
```

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

11. Common Errors

- Forgetting to declare a function before using it.
- Mismatched return types.
- Incorrect number of arguments in function calls.

Conclusion

Functions are essential for writing clean, efficient, and maintainable code in C. Understanding how to declare, define, and call functions is fundamental for any programmer.

Next Steps

- Practice writing your own functions.
- Explore built-in functions in the C standard library.