3. Basic Structure of a C Program

• THEORY EXERCISE: Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

1. Preprocessor Directives (Headers)

- **Purpose:** These lines, starting with a #, give instructions to the C preprocessor, which runs before the actual compilation. The most common directive is #include.
- **Headers:** Header files contain declarations of functions and variables that are predefined in the C standard library or by other programmers. Including a header file makes these declarations available to your program.
- **Example:**

```
#include <stdio.h>  // Standard Input/Output library (for functions like printf and scanf)

#include <stdlib.h> // Standard library (for general utility functions)

#include <math.h>   // Mathematical functions (like sqrt, sin, cos)
```

In the example above, #include <stdio.h> tells the preprocessor to include the stdio.h header file, which contains the declaration for the printf function we use later.

2. Comments

- **Purpose:** Comments are used to add explanations and notes to your code. They are ignored by the compiler but are crucial for making your code understandable to humans (including your future self!).
- **Types:**
    - **Single-line comments:** Start with // and continue until the end of the line.
    - **Multi-line comments:** Start with /* and end with */. Everything in between is considered a comment.
- **Example:**

```
// This is a single-line comment.


/*
This is a
```

```
multi-line
comment.
*/

int x = 5; // This line declares an integer variable named x.
```

3. The main Function - The Heart of the Program

- **Purpose:** The main function is the entry point of your C program. Execution begins here. Every standalone executable C program must have a main function.
- **Structure:**

```
int main() {
  // Code to be executed goes here
  return 0;
}
```

    - int: Specifies that the main function will return an integer value to the operating system.
    - main: The name of the function.
    - (): Parentheses indicate that it's a function and can potentially take arguments (though in this basic form, it doesn't).
    - {}: Curly braces enclose the body of the main function, containing the program's instructions.
    - return 0;: This statement typically indicates that the program executed successfully. A non-zero return value usually signals an error.

4. Data Types

- **Purpose:** Data types specify the kind of values that a variable can hold and the operations that can be performed on it. C is a strongly-typed language, meaning you need to declare the data type of a variable before you use it.
- **Basic Data Types:**
    - int: Integer numbers (whole numbers without a decimal point), e.g., -5, 0, 100.
    - float: Single-precision floating-point numbers (numbers with a decimal point), e.g., 3.14, -2.5, 0.001.

    - double: Double-precision floating-point numbers (provide more precision than float), e.g., 3.1415926535.
    - char: Single characters enclosed in single quotes, e.g., 'a', 'B', '7'.
    - void: Represents the absence of a type or can be used for generic pointers.
- **Type Modifiers:** These keywords can be used to modify the basic data types to create variations:
    - short: Reduces the amount of memory used for an int.
    - long: Increases the amount of memory used for an int or double.
    - unsigned: Allows only non-negative integer values.
    - signed: Allows both positive and negative integer values (this is the default for int).
- **Example:**

```
int age = 30;           // Declares an integer variable named 'age' and initializes it to 30.
float pi = 3.14159;     // Declares a floating-point variable named 'pi'.
double temperature = 25.5; // Declares a double-precision floating-point variable.
char initial = 'J';     // Declares a character variable named 'initial'.
unsigned int count = 100; // Declares an unsigned integer variable.
```

5. Variables

- **Purpose:** Variables are named storage locations in the computer's memory used to hold data. You can think of them as containers for storing values that can be accessed and modified during program execution.
- **Declaration:** Before you can use a variable, you must declare it, specifying its data type and name.

```
data_type variable_name;
data_type variable_name1, variable_name2; // Declaring multiple variables of the same type
```

- **Initialization:** You can assign an initial value to a variable at the time of declaration.

```
data_type variable_name = initial_value;
```

- **Assignment:** You can assign or change the value of a variable using the assignment operator (=).

```
int number;     // Declaration
number = 15;    // Assignment
```

```
float price = 99.99; // Declaration and initialization
price = 120.50;      // Re-assignment
```

- **Naming Rules:**
    - Variable names can consist of letters (a-z, A-Z), digits (0-9), and underscores (_).
    - The first character cannot be a digit.
    - Variable names are case-sensitive (myVariable is different from myvariable).
    - Reserved keywords (like int, float, return) cannot be used as variable names.

Understanding these basic building blocks is crucial for writing any C program. As you progress, you'll learn about more complex structures and concepts, but this foundation will serve you well! Let me know if you have any more questions.