Here are the basic notes for the C programming language in Markdown format:

~~~markdown
# C Programming Language Notes

## Table of Contents
1. [Introduction](#introduction)
2. [Structure of a C Program](#structure-of-a-c-program)
3. [Data Types](#data-types)
4. [Variables](#variables)
5. [Operators](#operators)
6. [Control Statements](#control-statements)
    - [If-Else](#if-else)
    - [Switch](#switch)
    - [Loops](#loops)
7. [Functions](#functions)
8. [Arrays](#arrays)
9. [Pointers](#pointers)
10. [Structures](#structures)
11. [File Handling](#file-handling)
12. [Memory Management](#memory-management)

## Introduction
- **C** is a general-purpose, procedural programming language developed by
Dennis Ritchie in 1972.
- It is widely used for system programming, developing operating systems, and
embedded systems.

## Structure of a C Program
A basic C program structure includes:
- **Preprocessor Directives**: Instructions to the compiler before actual
compilation (e.g., `#include`).
- **Main Function**: Entry point of a C program (`int main()`).
- **Statements and Expressions**: Code logic.
- **Return Statement**: Ends the main function and returns a value to the
calling process (`return 0;`).

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

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

Data Types

C supports several data types to define variables:

- **Basic Types**: `int`, `float`, `double`, `char`.
- **Derived Types**: Arrays, Pointers, Structures, Unions.
- **Enumeration Types**: Enum.
- **Void Type**: For functions that do not return a value.

Variables

- **Variables**: Named memory locations used to store values.
- **Syntax**: `<data_type> <variable_name>;`

Example:

```
int age;
float salary;
char grade;
```

Operators

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

Example:

```
int a = 10, b = 20;
int sum = a + b;
int isEqual = (a == b);
```

Control Statements

If-Else

- Conditional branching.

Example:

```
int number = 10;
if (number > 0) {
    printf("Positive number\n");
} else {
    printf("Non-positive number\n");
}
```

Switch

- Multi-way branching.

Example:

```c
int day = 3;

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

Loops

- **For Loop**: Used for repeated execution of a block of code.
- **While Loop**: Executes as long as the condition is true.
- **Do-While Loop**: Similar to while loop, but ensures at least one execution.

Example:

```c
for (int i = 0; i < 5; i++) {
    printf("Iteration %d\n", i);
}

int j = 0;
while (j < 5) {
    printf("Count %d\n", j);
    j++;
}

int k = 0;
do {
    printf("Value %d\n", k);
    k++;
} while (k < 5);
```

Functions

- Functions are blocks of code that perform a specific task.
- **Function Declaration**: Tells the compiler about a function's name, return type, and parameters.
- **Function Definition**: Contains the actual code.

Example:

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

int main() {
    int sum = add(5, 3);
    printf("Sum: %d\n", sum);
    return 0;
}
```

Arrays

- **Array**: A collection of elements of the same data type.
- **Syntax**: `<data_type> <array_name>[size];`

Example:

```c
int numbers[5] = {1, 2, 3, 4, 5};
printf("First number: %d\n", numbers[0]);
```

Pointers

- **Pointer**: A variable that stores the memory address of another variable.
- **Syntax**: `<data_type> *<pointer_name>;`

Example:

```c
int x = 10;
int *ptr = &x;
printf("Value of x: %d\n", *ptr);
```

Structures

- **Structure**: A user-defined data type that allows grouping of variables of different types.
- **Syntax**: `struct <structure_name> { ... };`

Example:

```
struct Person {
    char name[50];
    int age;
};

int main() {
    struct Person person1;
    strcpy(person1.name, "John");
    person1.age = 30;

    printf("Name: %s, Age: %d\n", person1.name, person1.age);
    return 0;
}
```

File Handling

- **File Handling**: Enables reading from and writing to files.
- **Functions**: `fopen()`, `fclose()`, `fread()`, `fwrite()`, `fprintf()`, `fscanf()`, `fseek()`, etc.

Example:

```
FILE *fp;
fp = fopen("file.txt", "w");
if (fp != NULL) {
    fprintf(fp, "Hello, World!\n");
    fclose(fp);
}
```

Memory Management

- **Dynamic Memory Allocation**: Allocating memory during runtime.
- **Functions**: `malloc()`, `calloc()`, `realloc()`, `free()`.

Example:

```
int *ptr;
ptr = (int*) malloc(5 * sizeof(int)); // Allocates memory for 5 integers
if (ptr != NULL) {
    // Use the memory
    free(ptr); // Free the allocated memory
}
```