C Programming Tutorial

@ Coding Scape

1 Introduction

C is a general-purpose, procedural programming language that provides a powerful set of features for system programming and application development. It is known for its efficiency and control.

2 Basic Syntax

Definition: Basic syntax refers to the fundamental rules and structure that define how C programs are written. This includes the use of semicolons, curly braces, and the basic structure of a program.

2.1 Hello World Program

Example 1:

```
#include <stdio.h>

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

Example 2:

```
#include <stdio.h>

int main() {
    printf("Welcome to C Programming!\n");
    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    // Your code here
    return 0;
}
```

3 Variables and Data Types

Definition: Variables are named storage locations in memory that hold values. Data types define the type of data that a variable can hold, such as integers, floating-point numbers, or characters.

3.1 Variable Declaration and Initialization

Example 1:

```
#include <stdio.h>

int main() {
    int a = 5;
    float b = 3.14;
    char c = 'A';

    printf("Integer: %d\n", a);
    printf("Float: %.2f\n", b);
    printf("Character: %c\n", c);

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    double x = 9.81;
    long y = 1234567890;

    printf("Double: %.2f\n", x);
    printf("Long: %ld\n", y);

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    // Declare and initialize variables of different data types
    // Your code here

    return 0;
}
```

4 Control Flow

Definition: Control flow refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a programming language. It includes constructs like if-else statements and loops.

4.1 If-Else Statements

Example 1:

```
#include <stdio.h>

int main() {
    int num = 10;

    if (num > 0) {
        printf("Positive number\n");
    } else {
        printf("Non-positive number\n");
    }

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    int age = 20;

    if (age >= 18) {
        printf("Adult\n");
    } else {
        printf("Minor\n");
    }

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    int number;

    // Get the number from the user
    // Check if the number is even or odd
```

```
    // Your code here

    return 0;
}
```

5 Loops

Definition: Loops are used to repeatedly execute a block of code until a specified condition is met. Common loop types include for, while, and do-while loops.

5.1 For Loop

Example 1:

```
#include <stdio.h>

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

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    int i;

    // Print the multiplication table for a given number
    // Your code here

    return 0;
}
```

6 Functions

Definition: Functions are blocks of code designed to perform a specific task. They allow for code reusability and modularity by enabling code to be defined once and called multiple times.

6.1 Function Definition and Call

Example 1:

```
#include <stdio.h>

void greet() {
    printf("Hello from function!\n");
}

int main() {
    greet();
    return 0;
}
```

Example 2:

```
#include <stdio.h>

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

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

Unsolved Question:

```
#include <stdio.h>

int multiply(int a, int b) {
    // Your code here
}

int main() {
    int result = multiply(4, 5);
    printf("Product: %d\n", result);
    return 0;
}
```

7 Arrays

Definition: Arrays are collections of variables of the same type that are stored in contiguous memory locations. They are useful for managing multiple values under a single name.

7.1 Array Declaration and Access

Example 1:

```
#include <stdio.h>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    char name[] = "Coding";

    for (int i = 0; name[i] != '\0'; i++) {
        printf("%c", name[i]);
    }
    printf("\n");

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    int arr[10];

    // Initialize the array with squares of indices
    // Print the array elements
    // Your code here

    return 0;
}
```

8 Pointers

Definition: Pointers are variables that store the memory address of another variable. They are used for dynamic memory allocation, array manipulation, and function arguments.

8.1 Pointer Basics

Example 1:

```
#include <stdio.h>

int main() {
    int x = 10;
    int *p = &x;

    printf("Value of x: %d\n", x);
    printf("Value via pointer: %d\n", *p);

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    int arr[3] = {1, 2, 3};
    int *p = arr;

    for (int i = 0; i < 3; i++) {
        printf("%d ", *(p + i));
    }
    printf("\n");

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    int num = 50;
    int *ptr = &num;

    // Use pointer to modify the value of num
    // Print the modified value
    // Your code here

    return 0;
}
```

9 Structures

Definition: Structures are user-defined data types that group related variables of different types into a single unit. They are used to represent complex data.

9.1 Defining and Using Structures

Example 1:

```
#include <stdio.h>

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

int main() {
    struct Person p1 = {"Alice", 30};

    printf("Name: %s\n", p1.name);
    printf("Age: %d\n", p1.age);

    return 0;
}
```

Example 2:

```
#include <stdio.h>

struct Point {
    int x;
    int y;
};

int main() {
    struct Point p = {10, 20};

    printf("Point coordinates: (%d, %d)\n", p.x, p.y);

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

struct Book {
    char title[100];
    float price;
```

```
};

int main() {
    struct Book b;

    // Input and print details of the book
    // Your code here

    return 0;
}
```

10 File Handling

Definition: File handling refers to the operations of reading from and writing to files. It involves functions for opening, closing, reading, and writing files.

10.1 Reading and Writing Files

Example 1:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "w");

    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fprintf(file, "Hello, file!\n");
    fclose(file);

    return 0;
}
```

Example 2:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "r");
    char buffer[100];

    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
```

```
    while (fgets(buffer, sizeof(buffer), file)) {
        printf("%s", buffer);
    }

    fclose(file);

    return 0;
}
```

Unsolved Question:

```
#include <stdio.h>

int main() {
    FILE *file;

    // Open a file in append mode and write user input to it
    // Your code here

    return 0;
}
```