DSA SQUAD BY SOURCIFY IN

DAY 2:

Functions

What are Functions in Programming?

Functions in Programming is a block of code *that encapsulates a specific task or related group of tasks. Functions are defined by a name, may have parameters and may return a value. The main idea behind functions is to take a large program, break it into smaller, more manageable pieces (or functions), each of which accomplishes a specific task.*

https://bit.ly/SourcifyINCommunity

Syntax

Function Declaration

A function must be declared before it is used, typically at the beginning of a program or in a header file.

Ex. `return_type function_name(parameter_list);`

Function Call

A function is called to execute its code.

Ex. `function_name(argument_list);`

Function Definition

A function is defined to specify what the function does. This includes the body of the function.

```
Ex. return_type function_name(parameter_list) {
    ( function body )
}
```

Parts of a Function

Return Type

Specifies the type of value the function returns. If the function does not return a value, the return type is void.

Ex. `int add(int a, int b);` (returns an integer)

`void display();` (returns nothing)

Parameter List

A comma-separated list of parameters. Each parameter has a type and a name.

Ex. `int add(int a, int b);` (parameters are int a and int b)

Function Body

Enclosed in curly braces {}, it contains the statements that define what the function does.

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

https://bit.ly/SourcifyINCommunity

Types of Functions

Based on Parameters and Return Type

1. No parameters, no return value:

Ex.
```
void display() {
    printf("Hello!\n");
}
```

2. **No parameters, with return value:**

Ex.
```
int getNumber() {
return 42;
}
```

3. With parameters, no return value:

Ex.
```
void setAge(int age) {
    printf("Age: %d\n", age);
}
```

4. With parameters, with return value:

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

Function Prototypes

Function prototypes are declarations of functions that specify their return type and parameter list. They enable the compiler to ensure that functions are called correctly.

Ex.
```
#include <stdio.h>


void greet(); (Function prototype)


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


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

https://bit.ly/SourcifyINCommunity

Scope and Lifetime of Variables

Local Variables

Declared inside a function and accessible only within that function.

Ex.
```
void exampleFunction() {
    int localVar = 10; // local variable
    printf("%d\n", localVar);
}
```

Global Variables

Declared outside any function and accessible to all functions in the program

Ex.
```
int globalVar = 20;  (global Variables)
void exampleFunction() {
    printf("%d\n", globalVar);
}
```

Static Variables

Declared with the static keyword, they retain their value between function calls and have a local scope.

Ex.
```
void counter() {
    static int count = 0;
    count++;
    printf("Count: %d\n", count);
}
```

Passing Arguments

By Value

Copies the value of an argument into the parameter of the function. Modifications to the parameter do not affect the argument.

Ex.
```
void modify(int x) {
    x = 10;
}
```

https://bit.ly/SuricifyINCommunity

```
int main() {
    int a = 5;
    modify(a);
    printf("%d\n", a); // Output: 5
    return 0;
}
```

By Reference (Using Return Value of Function)

Instead of using pointers, you can achieve a similar effect by returning a value from the function and assigning it back to the original variable.

Ex.
```
Int modify(int x) {
    X = 10;
    Return x;
}

Int main() {
    Int a = 5;
    A = modify(a);
    Printf("%d\n", a);
    Return 0;
}
```

https://bit.ly/SourcifyINCommunity

Inline Functions

Defined with the inline keyword, they suggest to the compiler to insert the function's code directly at the call site, potentially improving performance.

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

Tips for Functions in Programming:

- **Proper Use of Return Statements:** Ensure that all code paths in a function that should return a value do so.
- **Avoid Global Variables:** Functions should ideally rely on their input parameters and not on external variables.
- **Single Responsibility Principle:** Each function should do one thing and do it well.

Conclusion

Functions are essential for structuring and managing C programs effectively. By understanding their syntax, types, and usage, you can write more organized, reusable, and maintainable code.

https://bit.ly/SourcifyINCommunity