2. Function Declaration, Definition, and Calling

2.1 Function Anatomy

A C function consists of several parts:

```
return_type function_name(parameter_list) {
    // Function body
    // Local variables
    // Statements
    return value; // (if return_type is not void)
}
```

Components:

1. **Return Type:** Data type of the value returned (int, float, char, void, etc.)
2. **Function Name:** Identifier for the function
3. **Parameter List:** Input values (formal parameters)
4. **Function Body:** Statements enclosed in braces
5. **Return Statement:** Returns control and optionally a value

2.2 Function Declaration (Prototype)

Function prototypes declare the function's interface before its actual definition. They are necessary when a function is called before its definition in the source code. This allows the compiler to check for correct usage and enables forward referencing. If a function is defined *before* it is called, a prototype is not strictly necessary.

Syntax:

```
return_type function_name(parameter_types);
```

Examples:

```
// Function prototypes
int add(int a, int b);                  // Two int parameters, returns int
```

```
float calculate_area(float length, float width);  // Two float parameters, returns float
void print_message(void);                 // No parameters, no return value
char get_grade(int score);               // One int parameter, returns char
```

Important Notes:

- Prototypes end with semicolon (`;`)
- Parameter names are optional in prototypes (but recommended for clarity)
- Must match exactly with function definition

2.3 Function Definition

The function definition contains the actual implementation:

```
#include <stdio.h>

// Function prototype
int multiply(int x, int y);

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

// Function definition
int multiply(int x, int y) {
    int product = x * y;
    return product;
}
```

2.4 Function Calling

After a function has been declared (prototyped) and defined, it can be executed, or "called," from another part of the program (e.g., from `main()` or another function). When a function is called, the program's control is transferred to that function.

Syntax for Calling a Function:

```
function_name(arguments);
```

- `function_name`: The name of the function to be executed.
- `arguments`: The values (actual parameters) passed to the function. These must match the type and order of the parameters in the function's declaration.

Example: In the `main()` function of the previous example, `multiply(4, 5);` is a function call.

```
int main() {
    int result = multiply(4, 5); // Calling the 'multiply' function
    printf("Result: %d\n", result);
    return 0;
}
```

Here, `4` and `5` are the arguments passed to the `multiply` function. The return value of `multiply` is then stored in the `result` variable.

2.5 void Functions

Keep in mind : Functions that don't return a value use `void` as the return type:

```
#include <stdio.h>

void print_header(void);
void print_line(int length);

int main() {
    print_header();
    print_line(30);
    return 0;
}

void print_header(void) {
    printf("=== STUDENT MANAGEMENT SYSTEM ===\n");
}

void print_line(int length) {
    for (int i = 0; i < length; i++) {
        printf("-");
    }
    printf("\n");
}
```

Revision #2
Created 2025-09-06 10:14:52 UTC by DS
Updated 2025-09-06 10:33:14 UTC by DS