Syntax, Input & Output

Syntax

- Brace Placement { }
- Consistent Naming Conventions
- Proper Use of Whitespace
- Commenting
- Consistent Use of Semicolons
- Consistent Code Organization

Brace Placement { }

Place opening braces { on the same line as the statement or control structure, and closing braces } on a separate line aligned with the beginning of the corresponding statement or control structure.

```
if (condition) {
    // Proper indentation
    printf("Hello, World!");
}
```

Consistent Naming Conventions

Use clear and consistent variable and function naming conventions. Common conventions include using lowercase letters for variable names and lowercase_with_underscores for function names.

```
int age;
void calculate_tax_amount();
```

Proper Use of Whitespace

Use whitespace to improve code readability. Add spaces around operators, after commas in function arguments, and between keywords and identifiers.

```
int result = x + y;
printf("Hello, World!");
```

Commenting

Use comments to explain the purpose and logic of your code. Add comments to describe complex algorithms, functions, and non-obvious parts of your code.

// for single-line comments

/* */ for multi-line comments.

```
// This function calculates the sum of two numbers.
int add(int a, int b) {
    return a + b;
}
```

Consistent Use of Semicolons

Use semicolons to terminate statements and ensure that they are consistently placed at the end of each statement.

```
int x = 42;  // Semicolon terminates the statement
printf("Hello, World!");
```

printf()

Defnition

- **printf()** is used to print output to the console or other standard output streams.
- It allows you to display text and values on the screen.
- Format specifiers, such as %d, %f, %s, are used to indicate the data type and the location where the corresponding values should be inserted in the output.

Syntax

```
int num = 42;
printf("The value of num is: %d\n", num);
```

scanf()

Defnition

- scanf() is used to read input from the user or standard input (e.g., keyboard).
- It is used with format specifiers to specify the type of data you want to read.
- The data entered by the user is stored in the specified variables.

Syntax

```
int age;
printf("Enter your age: ");
scanf("%d", &age);
printf("You entered: %d\n", age);
```

Getting User Input with scanf()

- Include the Required Header File

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

- Declare Variables

```
int age;
```

- Display a Prompt

```
printf("Enter your age: ");
```

- Read User Input

```
scanf("%d", &age);
```

- Use the User Input

```
printf("You entered: %d\n", age);
```

Getting User Input with scanf()

```c
#include <stdio.h>
int main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);
    printf("You entered: %d\n", age);
    return 0;
}
```

Printing Output with printf()

- Include the Required Header File
- Use the printf() Function: The printf()
- Compile and Run

```c
#include <stdio.h>
int main() {
    int num = 42;
    printf("The value of num is: %d\n", num);
    return 0;
}
```

Example 1: String Input and Output

Expected Input: "John"

Expected Output: "Hello, John!"

Example 2: Integer Input & output

Expected Input: 5

Expected Output: "You entered: 5"

Example 3: Float Input and Output

Expected Input: 3.14

Expected Output: "Value of Pi: 3.14"

Example 4: Taking Multiple Inputs in a Single Line

Expected Input: 10 20 30

Expected Output: "Sum of Inputs: 60"

Example 7: Arithmetic Operators

Expected Input: 10, 5

Expected Output:

"Addition: 15, Subtraction: 5,
Multiplication: 50, Division: 2.0"

Example 8: Comparison Operators

Expected Input: 10, 5

Expected Output:

"10 > 5: True, 10 < 5: False,
10 == 5: False, 10 != 5: True"

Example 9: Logical Operators

Expected Input: True, False

Expected Output:

"True and False: False,
True or False: True, not True: False"

Example 10: Taking Yes/No Input and Handling Case Sensitivity

Expected Input: Yes (or yes, YES, yEs, etc.)

Expected Output: "You entered: Yes"