C Style Guide

For CS1 Students
Spring 2025

General Naming Conventions

1. **Variable Names**:
   - Use **descriptive and meaningful names** for variables. Avoid cryptic names like `a1` or `temp2`.
   - For short-lived variables in loops, it's acceptable to use `i`, `j`, or `k`.
   - Examples:

```
int studentCount;  // Good
int sc;            // Avoid
```

2. **Function Names**:
   - Use **camelCase** for function names (e.g., `calculateSum`).
   - If the function name contains multiple words, capitalize the first letter of each subsequent word.
   - Examples:

```
void printArray();       // Correct
void printarray();       // Incorrect
```

3. **Constant and Macro Names**:
   - Use **ALL_CAPS_WITH_UNDERSCORES** for constants and macros.
   - Examples:

```
#define MAX_STUDENTS 100  // Good
```

4. **File Names**:
   - File names should be all lowercase with underscores separating words. Use `.c` for source files and `.h` for header files.
   - Example: `student_records.c`, `utility_functions.h`.

Code Formatting

1. **Indentation**:
   - Use consistent indentation throughout your code.
   - Prefer **4 spaces** per indentation level (do not use tabs for indentation).
   - Example:

```
if (condition) {
    printf("Condition is true.\n");
}
```

2. **Curly Braces**:
    - Place the opening curly brace `{` on the **same line** for functions and control structures.
    - Place the closing curly brace `}` on a **new line** and align it with the opening line of the block.
    - Examples:

```
// Function
void displayMessage() {
    printf("Hello, World!\n");
}

// If-else
if (x > 0) {
    printf("Positive number.\n");
} else {
    printf("Non-positive number.\n");
}
```

3. **Line Length**:
    - Keep lines under **80 characters** for readability. Break long lines into multiple lines using proper indentation.
4. **Spacing**:
    - Add spaces around operators for better readability:

```
int sum = a + b;  // Good
int sum=a+b;      // Avoid
```

    - Leave a blank line between logical sections of your code (e.g., between variable declarations and function logic).
5. **Function Definition and Call Formatting**:
    - Do not leave a space between the function name and the opening parenthesis.
    - Example:

```
int calculateSum(int a, int b) {  // Correct
    return a + b;
}
```

Commenting

1. **Inline Comments**:
    - Use `//` for single-line comments. Avoid `/* */` for inline comments.
    - Always leave a space after `//` before writing the comment:

```
int sum = 0; // Initialize sum
```

2. **Block Comments**:

- Use `/* */` for multi-line comments. Start with a brief description followed by detailed explanations if necessary:

```
/*
 * This function calculates the factorial of a number.
 * It uses a recursive approach.
 */
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

3. **Header Comments**:
   - Always include a header comment at the top of your source file with your name, course number, assignment title, and date.
   - Example:

```
/*
 * Author: John Doe
 * Course: CS2
 * Assignment: Array Operations
 * Date: Spring 2023
 */
```

4. **Comments for Logical Blocks**:
   - Use comments to explain the purpose of complex blocks of code.
   - Examples:

```
// Sort the array in ascending order
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            // Swap arr[j] and arr[j+1]
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
```

Best Practices

1. **Variable Declaration**:
   - Declare variables at the **top of a function**.
   - Group related variables together and initialize them if possible.
   - Example:

```
int sum = 0;
int count = 0;
```

2. **Avoid Magic Numbers**:
    - Use meaningful constants instead of hard-coded values.
    - Example:

```
#define MAX_LENGTH 100
char input[MAX_LENGTH];
```

3. **Error Handling**:
    - Always check the return value of functions that can fail (e.g., `malloc`, `scanf`, `fopen`).
    - Example:

```
FILE *file = fopen("data.txt", "r");
if (file == NULL) {
    printf("Error: Could not open file.\n");
    return 1;
}
```

4. **Avoid Global Variables**:
    - Minimize the use of global variables. Use `static` for variables that should remain private to a file or function.
5. **Functions**:
    - Each function should perform a single, well-defined task.
    - If a function grows too large, break it into smaller helper functions.
6. **Dynamic Memory**:
    - Always free dynamically allocated memory using `free()` to avoid memory leaks.
    - Example:

```
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) {
    printf("Memory allocation failed.\n");
    return 1;
}
// Use the array
free(arr);
```

7. **Testing**:
    - Test your program with edge cases, invalid inputs, and large data sets.

Additional Notes

- This style guide is intended to help you write **clean, readable, and maintainable C code**.
- By following these guidelines, your code will not only look professional but also be easier to debug, test, and understand for yourself and others.