SDM College of Engineering and Technology

Dhavalagiri, Dharwad-580 002. Karnataka State. India.

Email: principal@sdmcet.ac.in, cse.sdmcet@gmail.com

Ph: 0836-2447465/ 2448327 Fax: 0836-2464638 Website: sdmcet.ac.in

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

MINOR WORK REPORT

[22UHUC500- Software Engineering and Project Management]

Odd Semester: September 2024-January 2025

Course Teacher: Dr. U.P.Kulkarni

2024- 2025

Submitted by
By

Vaishnavi Shetti

2SD22CS121

5th Semester B division

Table of Contents

1 Assignment-1

1.1 Problem Statement: **Write C program to show that c programming language supports only call by value**

1.2 Theory:

This program demonstrates that C programming language supports call by value. Here's why:

1. **Function Arguments:** When you pass a and b as arguments to the swap function, copies of their values are created and passed to the function. These copies are stored in the local variables x and y within the swap function.
2. **Swapping Within the Function:** The swap function modifies the values of x and y by swapping them. However, these modifications only affect the copies of the original values. The original values of a and b remain unchanged.
3. **Output:** The output of the program shows that the values of a and b remain the same before and after calling the swap function, confirming that the changes made within the function did not affect the original variables.

1.3 Program:

```
#include <stdio.h>

void swap(int x, int y) {

 int temp;

temp = x;

x = y; y = temp;

printf("Inside swap function: x = %d, y = %d\n", x, y);

}

 int main() {

int a = 10, b = 20;

 printf("Before calling swap function: a = %d, b = %d\n", a, b);

 swap(a, b);

 printf("After calling swap function: a = %d, b = %d\n", a, b);

return 0;

}
```

1.4 Sample output:

```
Before calling swap function: a = 10, b = 20

Inside swap function: x = 20, y = 10

After calling swap function: a = 10, b = 20
```

1.5 References:

1) Software Engineering: A Practitioner's Approach by Roger S. Pressman (9th edition)
2) C Programming Language by Brian Kernighan and Dennis Ritchie

2 Assignmnet-2:

2.1 Problem Statement: Study the concept "Usability", and prepare a report on "Usability" of at least two UI's of major software products you have seen.

2.2 Theory:

Usability is a measure of how easy a product is to use. It encompasses factors like learnability, efficiency, effectiveness, satisfaction, and accessibility. A usable product is intuitive, efficient, and enjoyable for users. In this report, we will analyze the usability of two major software products: Git Hub and Google Collab.

2.3 Usability of GitHub:

GitHub, the leading platform for version control and collaboration, has revolutionized how developers work together. However, its usability is a complex issue that depends on various factors, including user experience, features, and integration with other tools.

2.3.1 Strengths of GitHub's Usability

- ***Intuitive Interface***: GitHub's interface is generally user-friendly, with clear navigation and easily identifiable elements. Even for beginners, the platform's layout is intuitive, making it easy to learn and use.

- ***Powerful Features:*** GitHub offers a vast array of features, including version control, issue tracking, project management, and code review. These features are well-integrated and accessible through a consistent interface.

- ***Strong Community and Support:*** GitHub has a large and active community of developers, which provides extensive resources, tutorials, and support. This community can be invaluable for users who encounter issues or need help with specific tasks.

- ***Integration with Other Tools:*** GitHub seamlessly integrates with a wide range of development tools, such as IDEs, text editors, and continuous integration/continuous delivery (CI/CD) pipelines. This integration streamlines workflows and improves efficiency.

2.3.2 Areas for Improvement in GitHub's Usability

- ***Learning Curve***: While GitHub's interface is generally intuitive, there can be a learning curve for users who are new to version control or collaboration platforms. Understanding concepts like repositories, branches, and pull requests can take time.

- ***Complexity***: As GitHub has grown in features, it has become more complex. This can be overwhelming for some users, especially those who only need a subset of the platform's capabilities.

- ***Performance issue*****:** In some cases, GitHub can experience performance issues, particularly when dealing with large repositories or high traffic. This can impact the user experience and productivity.

2.4 Usability of Google Colab:

Google Colab is a cloud-based Jupyter notebook environment that provides a free platform for coding, executing, and sharing data science experiments. A User-Friendly Tool for Data Science and Machine Learning. Its user-friendly interface and powerful features make it a popular choice for researchers, students, and developers.

2.4.1 Key Strengths of Google Colab:

- ***Accessibility:*** Colab is accessible from any device with an internet connection, eliminating the need for local software installations.

- ***Free Usage:*** The basic version of Colab is free, making it a cost-effective option for individuals and small teams.

- ***Integration with Google Drive*****:** Colab seamlessly integrates with Google Drive, allowing users to store and manage their notebooks and data files.

- ***GPU and TPU Acceleration:*** Colab offers free access to GPUs and TPUs, which are essential for training deep learning models efficiently.

- ***Collaboration Features:*** Colab supports real-time collaboration, making it easy for teams to work on projects together.

- ***Sharing and Publishing*****:** Notebooks can be easily shared and published, making it a great platform for disseminating research and educational materials.

2.4.2 Areas for Improvement:

- ***Internet Connectivity:*** Colab heavily relies on a stable internet connection, which can be a limitation in areas with unreliable network access.

- ***Limited Customization*:** While Colab offers a basic level of customization, it may not meet the specific needs of advanced users who require more control over their environment.

- ***Storage Limitations*:** Free users have limited storage space, which can be a constraint for large datasets or projects.

2.5 References:

1) Google Colab Documentation: https://colab.research.google.com/

2) GitHub Help Documentation:https://support.github.com/

3) GitHub Essentials: A Beginner's Guide by David Gassner

4) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron

3 Assignment-3:

3.1 Problem Statement: **List all feature of programming language and Write Programs to show how they help to write ROBUST code**

3.2 Theory: Features of C Programming language

C is a powerful and versatile programming language that offers a wide range of features that contribute to writing robust and efficient code. Here are some key features and examples of how they can be used to enhance code quality.

3.2.1 Data types:

- Strong typing**:** Enforces type safety, preventing unexpected behavior and errors.

3.2.2 Pointers:

- Memory management: Allows for dynamic memory allocation and deallocation.

3.2.3 Control Flow Statements:

- Conditional statements (if-else, switch)**:** Enable decision-making and branching based on conditions.
- Loops (for, while, do-while): Allow for repetitive execution of code blocks.

3.2.4 Functions:

- Modularity: Break down code into smaller, reusable units.

- Abstraction: Hide implementation details, improving code readability and maintainability.

3.2.5 Arrays:

- Store multiple values of the same data type.

3.2.6 Structures:

- Group related data elements into a single unit.

3.2.7 Pointers to structures:

- Pass structures by reference to functions.

3.2.8 Preprocessor Directives:

- Conditional compilation and macro definitions.

3.2.9 Standard library:

- Provides a rich set of functions and data types.

3.3 Program:

3.3.1 Data types:

```
int main() {
    int age = 30;
    float pi = 3.14159;

    printf("Age: %d\n", age);
    printf("PI: %f\n", pi);

    return 0;
}
```

Sample Input: None

Sample Output: Age: 30

```
PI: 3.141590
```

3.3.2 Pointers:

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

int main() {
    int *ptr = (int*)malloc(sizeof(int));
    *ptr = 10;

    printf("Value: %d\n", *ptr);

    free(ptr);
    return 0;
}
```

Sample Input: None

Sample Output: Value: 10

3.3.3 Control Flow Statements:

```
int main() {
    int age = 25;

    if (age >= 18) {
        printf("You are an adult.\n");
    } else {
        printf("You are a minor.\n");
    }

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

    return 0;
}
```

Sample Input: None

Sample Output: You are an adult.

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

3.3.4 Functions:

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int result = factorial(5);
    printf("Factorial of 5: %d\n", result);
    return 0;
}
```

Sample Input: None

Sample Output: Factorial of 5: 120

3.3.5 **Arrays:**

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

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

    printf("\n");
    return 0;
}
```

Sample Input: None

Sample Output: 1 2 3 4 5

3.3.6 **Structures:**

```
#include <stdio.h>

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

int main() {
    struct Person p = {"Alice", 25};

    printPerson(&p);

    return 0;
}

void printPerson(struct Person *p) {
    printf("Name: %s\n", p->name);
    printf("Age: %d\n", p->age);
}
```

Sample Input: None

Sample Output: Name: Alice

Age: 25

3.3.7 **Preprocessor Directives:**

```
#include <stdio.h>

#define PI 3.14159

int main() {
    printf("Value of PI: %f\n", PI);

    return 0;
}
```

Sample Input: None

Sample Output: Value of PI: 3.141590

3.3.8 **Standard Library:**

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

int main() {
    char str1[] = "Hello";
    char str2[] = "world";
    strcat(str1, str2);
    printf("%s\n", str1);

    return 0;
}
```

Sample Input: None

Sample Output: Hello world

3.4 References:

1) C: The Complete Reference by Herbert Schildt:

2) C Programming Tutorial**:** https://www.cprogramming.com/tutorial/c-tutorial.html

4 Assignment-4

4.1 Problem Statement: **Study the "assertions" in C language and its importance in writing reliable code study POSIX standard and write a C program under unix to show use of POSIX standard in writing portable code.**

4.2 Theory:

Assertions in C

Assertions are a powerful tool in C programming that can help detect and diagnose errors early in the development process. They are essentially checks that are placed in the code to verify that certain conditions are true at runtime. If a condition fails, the assertion triggers an error message, often causing the program to terminate.

Syntax: assert(expression);

Importance of Assertions:

- **Early Error Detection:** Assertions can catch errors early in the development process, preventing them from propagating to later stages.
- **Debugging Aid:** Assertions can provide valuable information about the state of the program at the point of failure, making it easier to identify and fix bugs.
- **Documentation:** Assertions can serve as documentation, clarifying the assumptions and preconditions of the code.
- **Testing:** Assertions can be used to create unit tests and verify the correctness of code.

POSIX Standards and Portable Code

POSIX (Portable Operating System Interface) is a family of standards that define an API for operating systems, ensuring compatibility and portability across different platforms. By adhering to POSIX standards, C programs can be more easily ported to different Unix-like systems.

Key POSIX Standards:

- **POSIX.1:** Defines the basic operating system interface, including file I/O, process management, and signals.
- **POSIX.2:** Defines the shell and utilities interface, specifying the behavior of common command-line tools.
- **POSIX.1-2008:** The latest version of POSIX.1, incorporating additional features and enhancements.

4.3 Program:

4.3.1 Assertions in C and POSIX Standards for Portable Code

```
#include <assert.h>
```

```
int main() {
    int x = 10;
    assert(x > 0); // Asserts that x is positive
    printf("x is positive\n");
    return 0;
}
```

Sample input: None

Sample output: x is positive

4.3.2 Example of Using POSIX Standards for Portable Code:

```
#include <stdio.h>
#include <unistd.h> // POSIX standard for file I/O
int main() {
    FILE *fp = fopen("file.txt", "w");
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

    fprintf(fp, "Hello, world!\n");
    fclose(fp);
    return 0;
}
```

Sample input: None

Sample output: Hello, world!

4.4 References:

1. C Programming Language by Brian Kernighan and Dennis Ritchie
2. UNIX System Programming by W. Richard Stevens: