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

[22UHUC5000- Software Engineering and Project Management]

Odd Semester: September 2024-January 2025

Course Teacher: Dr. U.P.Kulkarni

2024- 2025

Submitted by
By

Vaibhavi A Shanbhag
2SD22CS120
5th Semester B division

Table of Contents

Assignment-1:

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

Theory:

1. **Function increment**: This function takes an integer parameter x. Inside the function, the value of x is incremented by. However, the change made to x within the function is local and does not affect the original value of a in the main function.
2. **Main function:** In the main function, an integer variable a is declared and initialized. The function increment is called with a as an argument. When the increment function is executed, the value of a is copied to x. The value of x is incremented within the increment function. However, since C uses call by value, the original value of a in the main function remains unchanged.

Program:

```
#include <stdio.h>

void increment (int x) {

    x++;

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

}

int main () {

    int a = 10;

    printf("Before increment: a = %d\n", a);

    increment(a);

    printf("After increment: a = %d\n", a);

    return 0;

}
```

Sample output:

```
Before increment: a = 10

Inside increment function: x = 11

After increment: a = 10
```

Assignmnet-2:

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.

Introduction to Usability

Usability measures how effectively, efficiently, and satisfactorily users can interact with a software product. Key attributes include learnability, efficiency, memorability, error handling, and user satisfaction. This report analyzes the usability of **Trello** and **Microsoft Teams**.

Usability Analysis of Two Major Software Products

1. **Trello**

    - **Learnability**: Trello's card-and-board system is intuitive, allowing users to create boards, lists, and cards easily. New users can quickly grasp task management concepts thanks to a straightforward interface and helpful onboarding tips.
    - **Efficiency**: The drag-and-drop functionality enhances workflow efficiency, allowing users to move cards effortlessly between lists to indicate progress. Keyboard shortcuts further speed up task management.
    - **Memorability**: Trello's visual layout aids memorability. The consistent design across boards makes it easy for users to navigate and recall how to organize tasks even after a break.
    - **Error Handling**: Trello provides basic notifications for errors, but users may find it challenging to recover deleted cards or lists. However, activity logs help track changes and provide some level of recovery.
    - **Satisfaction**: Users express high satisfaction with Trello's visual organization and flexibility. The ability to customize boards and utilize labels enhances the task management experience.

2. **Microsoft Teams**

    - **Learnability**: Microsoft Teams offers a relatively steep learning curve due to its wide array of features, including chat, video conferencing, and file sharing. However, it provides onboarding tutorials that help new users familiarize themselves with the platform.
    - **Efficiency**: Teams allows for efficient collaboration with integrated tools like calendars and file sharing. Users can easily schedule meetings and access shared documents within the same interface, which enhances productivity.
    - **Memorability**: The interface can be overwhelming for new users due to the multitude of features. However, consistent updates and a structured layout help regular users remember where to find functions.
    - **Error Handling**: Microsoft Teams provides clear error messages and prompts when issues arise, such as connectivity problems during meetings. Users can usually troubleshoot these issues with guided support.

    - **Satisfaction**: Overall user satisfaction is high due to the robust integration with other Microsoft products and features that support remote collaboration, though some users find the interface cluttered.

<u>Comparison of Usability</u>

- **Learnability**: Trello is easier to learn for new users compared to Microsoft Teams, which requires more time to navigate its extensive features.
- **Efficiency**: Both platforms excel in efficiency, but Microsoft Teams integrates more collaborative tools, enhancing productivity in communication and file sharing.
- **Memorability**: Trello's visual design supports better memorability for users, while Microsoft Teams can be more complex to navigate.
- **Error Handling**: Both platforms provide error notifications, but Microsoft Teams generally offers clearer guidance for troubleshooting issues.
- **Satisfaction**: User satisfaction is high for both products, with Trello praised for its simplicity and visual task management, while Microsoft Teams is appreciated for its comprehensive collaboration features.

Conclusion

Usability is crucial for the effectiveness of software applications. Trello stands out for its intuitive task management and visual organization, making it ideal for individuals and teams. Microsoft Teams excels in facilitating collaboration and integration with other tools, though it may present a steeper learning curve. Both applications effectively address user needs, highlighting the importance of tailoring usability features to specific functions and user groups.

Assignment-3:

Problem Statement: List the features of programming language and write programs to show how they help to write robust code.

Theory:

The C programming language includes several features that facilitate writing robust, efficient, and maintainable code. Here are some key features:

1. Strong Typing: C enforces type rules, helping to catch type errors at compile time.
2. Pointer Management: Provides direct memory management through pointers, which can be powerful but requires careful handling.
3. Modularity: Supports modular programming through functions and header files, promoting code reuse.
4. Error Handling: While C does not have built-in exception handling, it allows for custom error handling using return codes.
5. Code Readability: Clear syntax and structure promote readability and maintainability.
6. Standard Libraries: C offers a rich set of libraries that enhance functionality and reliability.

Example Programs :

1. Strong Typing

Code:

```
#include <stdio.h>

void add_numbers(int a, int b) {

   printf("Sum: %d\n", a + b);

}

int main() {

   add_numbers(5, 10);

   // add_numbers(5, "10"); // Uncommenting this will cause a compilation error

   return 0;

}
```

Explanation: C enforces type checking, which helps catch errors at compile time. The commented line demonstrates that passing the wrong type would cause a compilation error.

2. Pointer Management

Code:

```
#include <stdio.h>

void safe_increment(int *ptr) {

   if (ptr != NULL) {
       (*ptr)++;
   } else {
       printf("Error: Null pointer passed to safe_increment.\n");
   }
}

int main() {
   int value = 5;
   safe_increment(&value);
   printf("Value after increment: %d\n", value);
   safe_increment(NULL); // Demonstrates error handling
   return 0;
}
```

Explanation: This program safely handles pointers by checking for NULL before dereferencing. It shows how pointer management can enhance robustness.

3. Modularity

Code:

```
#include <stdio.h>
void greet(const char *name) {
    printf("Hello, %s!\n", name);
}

int main() {
    greet("Alice");
    greet("Bob");
    return 0;
}
```

Explanation: The greet function demonstrates modular programming, making the code reusable and easier to manage. This promotes code organization and readability.

4. Error Handling

Code:

```
#include <stdio.h>
int divide(int a, int b) {
    if (b == 0) {
        printf("Error: Division by zero.\n");
        return -1; // Return an error code
    }
    return a / b;
}

int main() {
    int result = divide(10, 0);
    if (result == -1) {
        // Handle error
        return 1;
    }
    printf("Result: %d\n", result);
    return 0;
}
```

Explanation: The program uses a simple error handling mechanism with return codes to manage potential division errors, allowing for graceful failure.

5. Code Readability

Code:

```
#include <stdio.h>
void print_square(int number) {
    printf("Square of %d is %d\n", number, number * number);
}
int main() {
    for (int i = 1; i <= 5; i++) {
        print_square(i);
    }
    return 0;
}
```

Explanation: Clear function names and structured loops promote code readability, making it easy to understand the program's purpose and flow.

6. Standard Libraries

Code:

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

int main() {
    int *array = malloc(5 * sizeof(int));
    if (array == NULL) {
        printf("Memory allocation failed.\n");
        return 1; // Error handling for memory allocation failure
    }
        for (int i = 0; i < 5; i++) {
        array[i] = i * 2;
        printf("%d ", array[i]);
    }
    printf("\n");
   free(array); // Proper memory management
    return 0;
}
```

Explanation: This example demonstrates dynamic memory allocation and proper error handling. The program checks if memory allocation was successful and uses free to prevent memory leaks.

Conclusion:

The C programming language provides several features that contribute to writing robust code. Strong typing, careful pointer management, modular design, error handling, and the use of standard libraries all enhance the reliability and maintainability of C programs. These examples illustrate how these features can be employed to create safe and efficient applications.

Assignment-4:

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.

Theory:

Assertions are a debugging aid that tests assumptions made by the program. They are typically used to check conditions that should always be true at a specific point in the code. If an assertion fails, the program terminates, helping developers identify bugs during development.

Syntax for assertion in c language:

```
#include <assert.h>
assert(expression);
```

Importance of Assertions:

- **Early Detection of Errors**: Assertions help catch logical errors early in the development process.
- **Documentation**: They serve as documentation, clarifying the programmer's assumptions for future readers of the code.
- **Debugging Aid**: Assertions provide a simple way to validate assumptions and conditions without extensive logging or complex error handling.
- **Safety**: They ensure the code behaves as expected during development, leading to more reliable and maintainable software.

POSIX Standard

The **POSIX (Portable Operating System Interface)** standard is a set of standards specified by the IEEE for maintaining compatibility between operating systems. It defines the application programming interface (API), command line shells, and utility interfaces for software compatibility with variants of Unix and other operating systems.

Importance of POSIX:

Portability: Code that adheres to POSIX standards can run on different Unix-like operating systems with minimal changes.

Consistency: Provides a consistent interface for system calls and library functions.

Interoperability: Facilitates the development of applications that can work across various platforms.

C program :

```
 #include <stdio.h>

#include <stdlib.h>

int main() {

    const char* filename = "example.txt";

    FILE* file;
    char buffer[100];


    // Writing to the file

    file = fopen(filename, "w");

    if (file == NULL) {

        perror("Error opening file for writing");

        return EXIT_FAILURE;

    }


    fprintf(file, "Hello, POSIX file operations!\n");

    fprintf(file, "This is a simple example.\n");

    fclose(file); // Close the file after writing

 // Reading from the file

    file = fopen(filename, "r");

    if (file == NULL) {

        perror("Error opening file for reading");

        return EXIT_FAILURE;

    }

    printf("Contents of the file:\n");

    while (fgets(buffer, sizeof(buffer), file) != NULL) {

        printf("%s", buffer);

    }

    fclose(file); // Close the file after reading

return EXIT_SUCCESS;

}
```

References

1. https://www.geeksforgeeks.org/difference-between-call-by-value-and-call-by-reference/
2. https://www.nngroup.com/articles/usability-101-introduction-to-usability/
3. https://en.wikibooks.org/wiki/C_Programming
4. https://www.geeksforgeeks.org/assertions-cc/
   https://www.techtarget.com/whatis/definition/POSIX-Portable-Operating-System-Interface