**SDM COLLEGE OF ENGINEERING AND TECHNOLOGY**

Dhavalagiri, Dharwad-580002, Karnataka State, India.

**Email: cse.sdmcet@gmail.com**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# A Report On
# Minor Work

**COURSE CODE: 22UHUC500**

**COURSE TITLE: SOFTWARE ENGINEERING & PROJECT MANAGEMENT**

**SEMESTER: V , DIVISION: B**

**COURSE TEACHER: Dr. U.P.KULKARNI**

**[ Academic Year- 2024-25]**

**Date of Submission:**

Submitted By

**Mr. Ritesh.V.Patil**

**USN: 2SD22CS072**

## TABLE OF CONTENTS

## Problem Statement-1:

**Write a C Program To Show That C language Supports Only Call By Value.**

## Theory :

In C, function arguments are always passed by value. This means that when you pass a variable to a function, what is passed is a copy of the value, not the actual variable itself. Any changes made to the parameter inside the function don't affect the original variable in the calling function.

## Program :

```
#include <stdio.h>

int main()
{
   int a = 5;
   printf("Before calling modifyValue, a = %d\n", a);

   // Calling the function with 'a'
   modifyValue(a);

   // 'a' remains unchanged, demonstrating that C uses call by value
   printf("After calling modifyValue, a = %d\n", a);

   return 0;
  }
// Function that attempts to modify the value of its argument
void modifyValue(int x)
 {
   x = 10;  // This change is local to the function and won't affect the original variable
   printf("Inside modifyValue function, x = %d\n", x);

}
```

## Output :

```
Before calling modifyValue, a = 5
Inside modifyValue function, x = 10
```

After calling modifyValue, a = 5

## Problem Statement-2:

**Study Concept Of "USABILITY" using 2 UI's of major software product.**

## Theory :

Usability refers to the quality of a system, product, or service that allows users to interact with it easily and efficiently, achieving their goals without unnecessary complications. It plays a crucial role in product design, especially in software development, web design, and interface development, as it directly impacts user satisfaction, productivity, and overall user experience (UX).

## Scenario :

Your company is about to launch a new e-commerce platform aimed at selling niche, eco-friendly products. You aim to create an easy-to-use website with a clean design, accessible across various devices, and capable of handling complex interactions such as filtering products, adding to cart, and providing customer reviews. For this purpose, the usability of UI tools is crucial to ensure a seamless experience for your users.

## I. Figma: Prototyping and Design Tool :

**User-Centric Design**: Figma allows your design team to prototype the entire user journey from homepage to checkout with interactive elements like buttons and drop-down menus. Since the platform enables real-time collaboration, stakeholders can provide instant feedback, making it easy to adjust based on usability tests.

**Cross-Platform Compatibility**: The design can be easily adapted to various devices (mobile, desktop, tablet) to ensure that the UI remains user-friendly across different screen sizes.

**Collaborative Design Process**: Designers, product managers, and developers can collaborate on the same file, and the tool provides cloud-based storage, allowing for seamless updates and version control.

Figma's **interactive prototypes** allow you to see how a user would interact with the website. This helps identify areas where users might face friction.

The **commenting feature** enables multiple stakeholders to leave feedback in real-time, reducing the risk of design misinterpretation.

Designers can simulate user interactions and **test complex workflows** (like product search, cart management, etc.) without writing code.

## II. Material-UI: React UI Framework :

**Pre-built Components**: Material-UI offers a library of pre-built, accessible, and responsive UI components such as buttons, input fields, and navigation menus. These components follow Google's Material Design guidelines, ensuring consistency and usability across all parts of the application.

**Customization**: While it comes with ready-to-use elements, Material-UI is also highly customizable, making it easier for developers to tweak components to match the aesthetic of your brand.

**Responsive Design**: The framework makes it easy to build responsive layouts, which are crucial for e-commerce sites where users may be accessing from multiple device types.

Developers can quickly build and **test new features**, such as filter options or checkout flows, without needing to code every component from scratch, which speeds up the iteration process.

**Material-UI** components are designed to be **consistent and accessible**, ensuring that users, regardless of their technical abilities or device types, have a smooth experience.

## Problem Statement-3:

**List all features of programming language & write programs to show how they help to write robust code.**

**1.Object-Oriented:** Java follows the object-oriented programming (OOP) paradigm, which promotes code modularity, reusability, and maintainability through concepts like classes, objects, inheritance, polymorphism, abstraction, and encapsulation.

```
class Employee {
   private String name;
   private int age;
   public Employee(String name, int age)  // Constructor{
      this.name = name;
      this.age = age;
   }
   public String getName()  // Getter methods (read-only access){
      return name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) // Getter methods (read-only access) {
      if (age > 18) {
         this.age = age;
      } else {
         System.out.println("Invalid age. Must be greater than 18.");
      }
   }
}
public class Main {
   public static void main(String[] args) {
      Employee emp = new Employee("John Doe", 25);
      System.out.println(emp.getName()); // Output: John Doe
      emp.setAge(15); // Output: Invalid age. Must be greater than 18.
```

```
    }
  }
```

**2.Secure:**Java provides several mechanisms that make applications secure, such as bytecode verification, memory management, and sandboxing for applets. Java's strong type-checking and runtime exception handling enhance security and robustness.

```
public class Main {
  public static void main(String[] args) {
  int[] array = new int[5];
        try {
          array[6] = 10; // Causes ArrayIndexOutOfBoundsException
            } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Array index is out of bounds!");
             }
         }
     }
```

```
Array index is out of bounds!
```

**3.Exception Handling:**Java emphasizes early error checking and runtime error handling. It includes strong memory management (automatic garbage collection) and exception handling mechanisms that make it more reliable and robust.

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
        try {
        BufferedReader reader = new BufferedReader(new FileReader("nonexistentfile.txt"));
            String line = reader.readLine();
            System.out.println(line);
        } catch (FileNotFoundException e) {
            System.out.println("File not found!");
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
        } finally {
            System.out.println("File operation completed.");
        }
    }
}
```

```
File not found!
File operation completed.
```

**4.Multithreaded:**Java supports multithreading, which allows the execution of multiple tasks simultaneously. This is useful in applications that need to perform multiple operations at once (e.g., web servers, gaming, etc.).

```
class Task extends Thread {
private String taskName;

 public Task(String taskName) {
 this.taskName = taskName;
}
   public void run() {
      for (int i = 0; i < 5; i++) {
         System.out.println(taskName + " - iteration: " + i);
      }
   }
}

public class Main {
   public static void main(String[] args) {
      Task task1 = new Task("Task 1");
      Task task2 = new Task("Task 2");

      task1.start();
      task2.start();
   }
}
```

**5.Automatic Garbage Collection:**Java has an automatic garbage collection mechanism that automatically manages memory by reclaiming objects that are no longer in use. This helps prevent memory leaks and makes the code more efficient and robust.

```
public class GarbageCollectionDemo {

   // A simple class to demonstrate object creation
   static class DemoObject {
      private int id;

      DemoObject(int id) {
         this.id = id;
         System.out.println("Object " + id + " created.");
```

```
        }

        // Called by garbage collector before reclaiming the object
        @Override
        protected void finalize() throws Throwable {
            System.out.println("Object " + id + " is being collected by the Garbage Collector.");
        }
    }

    public static void main(String[] args) {
        // Creating objects
        DemoObject obj1 = new DemoObject(1);
        DemoObject obj2 = new DemoObject(2);
        DemoObject obj3 = new DemoObject(3);

        // Dereference the objects (make them eligible for garbage collection)
        obj1 = null;
        obj2 = null;

        // Requesting JVM to run the garbage collector
        System.gc();  // It's not guaranteed to run immediately, but will trigger garbage collection

        System.out.println("End of main.");
         }
      }
```

```
Object 1 created.
Object 2 created.
Object 3 created.
Object 1 is being collected by the Garbage Collector.
Object 2 is being collected by the Garbage Collector.
End of main.
```

## Problem Statement-4:

**Study ASSERTIONS in C language & its importance in writing RELIABLE Code, study POSIX standard & write a C program under Unix to show use of POSIX standards in writing portable code.**

## Theory -ASSERTIONS:

Assertions in C are a debugging tool used to test assumptions made by the program during its execution. They are provided by the assert header file. An assertion is a macro that evaluates a condition; if the condition evaluates to false, the program prints an error message and terminates execution.

## Scenario :

Let's consider a business scenario of an inventory management system where each product must have a non-negative stock count.

## Program :

```
#include <stdio.h>
#include <assert.h>
// Function to update stock level
void updateStock(int stock) {
   // Assertion to ensure stock is non-negative
   assert(stock >= 0);
   printf("Stock updated to: %d\n", stock);
}
int main() {
   int productStock = 10;
   // Correct stock value
   updateStock(productStock);
   // Invalid stock value (this will cause assertion failure)
   productStock = -5;
   updateStock(productStock);  // Assertion failure: stock >= 0
   return 0;
}
```

## Output :

```
Stock updated to: 10
```

In this program, the assertion assert (stock>=0) ; ensures that the stock level is always non-negative. If someone attempts to update the stock to a negative number, the assertion fails, and the program terminates, preventing further problems down the line.

## Theory -POSIX:

The POSIX (Portable Operating System Interface) standard defines a set of system APIs, shell utilities, and services to enable compatibility between different Unix-like systems such as Linux, macOS, and BSD. Writing applications that conform to the POSIX standard ensures that your code can run on any POSIX-compliant operating system without modification, making it highly portable.

## Scenario :

In a business scenario, consider an online retail company handling customer orders. A Unix-based system is used to process multiple customer orders simultaneously. To efficiently handle these requests, the company wants to process orders concurrently. This can be achieved using POSIX-compliant multithreading (POSIX threads ).

In this scenario, multiple customers may place orders at the same time, and the system should handle each order in a separate thread. Using POSIX standards ensures that this code can run on any Unix-like system, making it portable and easy to maintain.

## Program :

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h> // for sleep()
void* process_order(void* order_id) // Function to simulate order processing{
    int id = *((int*)order_id);
    printf("Processing order %d...\n", id);
    sleep(2); // Simulate processing time
    printf("Order %d processed.\n", id);
    return NULL;
}

int main() {
    int num_orders = 5; // Number of customer orders
    pthread_t threads[num_orders]; // Array to store thread IDs
    int order_ids[num_orders]; // Array to store order IDs

    // Create threads to process orders
    for (int i = 0; i < num_orders; i++) {
        order_ids[i] = i + 1; // Order ID starts from 1
        if (pthread_create(&threads[i], NULL, process_order, &order_ids[i])) {
            fprintf(stderr, "Error creating thread for order %d\n", i + 1);
            return 1;
        }
    }

    // Join threads to ensure all orders are processed before exiting
    for (int i = 0; i < num_orders; i++) {
        if (pthread_join(threads[i], NULL)) {
            fprintf(stderr, "Error joining thread for order %d\n", i + 1);
            return 2;
        }
    }
    printf("All orders processed successfully.\n");
    return 0;
}
```

## Output :

```
Processing order 1...
Processing order 2...
Processing order 3...
Processing order 4...
Processing order 5...
Order 1 processed.
Order 2 processed.
Order 3 processed.
Order 4 processed.
Order 5 processed.
```

All orders processed successfully.