**SDM College of Engineering and Technology**

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

Email: [principal@sdmcet.ac.in](mailto:principal@sdmcet.ac.in), [cse.sdmcet@gmail.com](mailto:cse.sdmcet@gmail.com)

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

**Department**

**of**

**COMPUTER SCIENCE AND ENGINEERING**

**Report on**

**Minor work**

**[22UHUC501 – SOFTWARE ENGINEERING & PROJECT MANAGEMENT]**

Odd Semester: Sep-Dec-2024

Course Teacher: Dr. U.P.Kulkarni

****

**2024- 2025**

Submitted by

By

**Nanda Basavaraj Gaddad**

**2SD23CS405**

**5th Semester B division**

**Table of Contents**

[Activity-1 : Write a C program to show that C programming Language support only Call by Value 3](#_Toc180437169)

[Problem Statement: 3](#_Toc180437170)

[Theory: 3](#_Toc180437171)

[Program: 3](#_Toc180437172)

[Sample input and output: 4](#_Toc180437173)

[Activity-2: Study the concept “USABLITY”, Prepare a report on USABILITY of at least TWO UIs of major software product you have seen 5](#_Toc180437174)

[Problem Statement: 5](#_Toc180437175)

[Theory: 5](#_Toc180437176)

[Activity-3: List all features of programming language and write PROGRAMS to show how they help to write ROBUST code 7](#_Toc180437177)

[Problem Statement: 7](#_Toc180437178)

[Theory: 7](#_Toc180437179)

[Program: 8](#_Toc180437180)

[Sample input and output: 13](#_Toc180437181)

[Activity-4: Study the ASSERTIONS in C language and its importance in writing RELIABLE CODE. Study POSIX standards and write a C program under Unix to show use of POSIX standard in writing portable code. 16](#_Toc180437182)

[Problem Statement: 16](#_Toc180437183)

[Theory: 16](#_Toc180437184)

[Program: 16](#_Toc180437185)

[Sample input and output: 17](#_Toc180437186)

# [Activity-1](#gjdgxs) : Write a C program to show that C programming Language support only Call by Value

## Problem Statement:

Write a C program to show that C programming Language support only Call by Value.

### Theory:

**Call by value** is a method of passing arguments to functions in programming, where the actual value of the argument is passed to the function. This means that the function operates on a copy of the variable, so any modifications made to the parameter inside the function do not affect the original value of the argument in the calling environment.

### Program:

The following business scenario demonstrates C programming language only supports Call by Value.

**Retail Store Temporary Price Adjustment:**

In a retail store's inventory system, a function is used to calculate the price of a product after applying a discount for promotional purposes. However, the store wants to ensure that the original product price stored in the inventory is not changed, since the discount is only temporary. The system should take the original price of a product, apply discount of 10% to a copy of the original price for promotional purposes and the discounted price is displayed to the customer, while the original price should remain unchanged in the system.

//C program for the above business scenario

#include <stdio.h>

// Function to apply discount to price

double applyDiscount(double price) {

    double discountRate = 0.10; // 10% discount

    price = price - (price \* discountRate); // Appling discount

    return price;

}

int main() {

    double originalPrice;

    printf("Enter the original price of the product: ");

    scanf("%lf", &originalPrice);

    printf("Original price: %.2lf\n", originalPrice);

    printf("Discounted price: %.2lf\n", applyDiscount(originalPrice));

    printf("Original price after discount calculation: %.2lf\n", originalPrice);

    return 0;

}

### Sample input and output:

Enter the original price of the product: 50

Original price: 50.00

Discounted price: 45.00

Original price after discount calculation: 50.00

Enter the original price of the product: 783.63

Original price: 783.63

Discounted price: 705.27

Original price after discount calculation: 783.63

Enter the original price of the product: 556.2

Original price: 556.20

Discounted price: 500.58

Original price after discount calculation: 556.20

# [Activity-2](#gjdgxs): Study the concept “USABLITY”, Prepare a report on USABILITY of at least TWO UIs of major software product you have seen

## Problem Statement:

Study the concept “USABLITY”, Prepare a report on USABILITY of at least TWO UIs of major software product you have seen

### Theory:

Usability in software refers to how easily and effectively users can interact with a software application to achieve their goals.

For example, a small online retail business that uses software to track and manage its stock. To run the business smoothly, the software must have good usability, ensuring that employees can quickly and easily manage system without errors or frustration. If the inventory management software has good usability, the employee will find it easy to learn how to use it, with clear labels, intuitive menus, and simple workflows. They can quickly add new products, update quantities, and save changes without needing extensive training.

Usability of major software product:

1. **Microsoft word**

Microsoft word is a widely used word processing application that offers a range of features for creating, editing, and formatting text documents. Usability features of Microsoft word are,

* Ease of Learning: Microsoft Word is designed with a familiar Ribbon interface, categorizing tools into easily identifiable tabs (Home, Insert, Layout, etc.). This layout allows new users to quickly locate features. Additionally, Word provides tutorials and a comprehensive help section, further easing the learning curve.
* Efficiency of Use: Word supports keyboard shortcuts that enhance user efficiency. For example, shortcuts like Ctrl + B (bold) and Ctrl + U (underline) enable experienced users to format text rapidly. The “Smart Lookup” feature allows users to research topics without leaving the document, streamlining workflow.
* Memorability: Regular users can quickly recall how to navigate Word, thanks to its consistent layout across versions. Features such as the Quick Access Toolbar allow users to save frequently used tools, enhancing memorability.
* Error Management: Word includes built-in spelling and grammar checkers, which alert users to potential errors in real-time, allowing for immediate corrections. The “Undo” function (Ctrl + Z) enables users to easily revert mistakes, providing confidence in document editing.

1. **Trello**

Trello is a project management tool that uses a visual board and card system to help users organize tasks and collaborate effectively. Its simplicity and flexibility make it popular among teams and individuals for managing projects. Usability features of Trello are,

* Ease of Learning: Trello’s interface is designed to be intuitive, with a simple drag-and-drop functionality. Users can create boards, lists, and cards easily, making it accessible for first-time users. The onboarding process includes helpful tips and guides, aiding new users in getting started quickly.
* Efficiency of Use: Trello allows users to organize tasks visually, enabling efficient project management. Users can quickly move cards between lists to indicate progress, add due dates, and attach files without navigating through complex menus. The use of labels and checklists further enhances task management.
* Memorability: The straightforward design of Trello makes it easy for users to remember how to navigate the interface. Frequent users can quickly adapt to their workflows, and the visual nature of boards aids in retaining information about project status.
* Error Management: Trello allows users to easily modify tasks by dragging and dropping cards, minimizing the chance of making errors. While there is no “Undo” button for all actions, users can archive cards and boards, preventing permanent loss of information.

# [Activity-3](#gjdgxs): List all features of programming language and write PROGRAMS to show how they help to write ROBUST code

## Problem Statement:

List all features of programming language and write PROGRAMS to show how they help to write ROBUST code

### Theory:

1. **Features of Java:**

* Object-Oriented: Encourages code reusability and modular design through classes and objects.
* Platform Independence: Java bytecode runs on any platform with a JVM, ensuring portability.
* Automatic Memory Management: Built-in garbage collector manages memory automatically to prevent leaks.
* Robustness: Strong typing, exception handling, and memory management enhance code stability and error prevention.
* Multithreading: Native support for concurrent programming via threads and executors.
* Security: Provides built-in security features like bytecode verification and a security manager.
* Rich Standard Library: Extensive libraries simplify common tasks, boosting development speed and reliability.

1. **Features of Python:**

* Dynamic Typing: Variables are not explicitly typed, enabling flexible and quick development.
* Easy-to-Read Syntax: Simple, readable syntax enhances code maintainability and collaboration.
* Cross-Platform Compatibility: Python code runs seamlessly across multiple operating systems.
* Robustness: Exception handling, garbage collection, and high-level data structures ensure stable and reliable code.
* Extensive Standard Library: Provides a wide array of modules to simplify development and ensure consistency.
* Interpreted Language: Code execution is done line-by-line, making debugging easier.
* Support for Multiple Paradigms: Supports object-oriented, functional, and procedural programming styles.
* Multithreading and Multiprocessing: Allows for concurrent and parallel execution through threads and processes.

### Program:

**Program – 1:**

The Order processing System of a retail shop should take the order of a product from the customer, process the order and the system should check the stock availability, it should say that the stock is insufficient for a product The following Java Program demonstrates the scenario

import java.util.List;

import java.util.ArrayList;

import java.util.Scanner;

// Custom exception class for insufficient stock

class InsufficientStockException extends Exception {

public InsufficientStockException(String message) {

super(message);

}

}

// Product class to represent a product in the system

class Product {

private String name;

private int stockQuantity;

private double price;

public Product(String name, int stockQuantity, double price) {

this.name = name;

this.stockQuantity = stockQuantity;

this.price = price;

}

public String getName() {

return name;

}

public int getStockQuantity() {

return stockQuantity;

}

public double getPrice() {

return price;

}

// Method to process the order

public void processOrder(int orderQuantity) throws InsufficientStockException {

if (orderQuantity > stockQuantity) {

throw new InsufficientStockException("Insufficient stock for product: " + name);

}

stockQuantity -= orderQuantity;

System.*out*.println("Order processed! " + orderQuantity + " units of " + name + " sold.");

}

}

public class OrderProcessingSystem {

public static void main(String[] args) {

// Creating a list of products

List<Product> productList = new ArrayList<>();

productList.add(new Product("Laptop", 10, 75000.00));

productList.add(new Product("Smartphone", 20, 30000.00));

productList.add(new Product("Tablet", 15, 45000.00));

// Display available products

System.*out*.println("Welcome to the Order Processing System.");

System.*out*.println("Available Products:");

for (int i = 0; i < productList.size(); i++) {

Product product = productList.get(i);

System.*out*.println((i + 1) + ". " + product.getName() + " - Available Stock: " + product.getStockQuantity() + ", Price: " + product.getPrice());

}

// Getting user input for product selection

Scanner scanner = new Scanner(System.*in*);

System.*out*.print("Select a product by entering the corresponding number: ");

try {

int productIndex = scanner.nextInt() - 1; // Adjust for zero-based index

if (productIndex < 0 || productIndex >= productList.size()) {

System.*out*.println("Error: Invalid product selection.");

return; // Exit if the selection is invalid

}

Product selectedProduct = productList.get(productIndex);

System.*out*.println("You selected: " + selectedProduct.getName());

System.*out*.println("Available Stock: " + selectedProduct.getStockQuantity());

System.*out*.print("Enter quantity to order: ");

int orderQuantity = scanner.nextInt();

// Process the order

selectedProduct.processOrder(orderQuantity);

} catch (InsufficientStockException e) {

System.*out*.println("Error: " + e.getMessage()); // Handle stock error gracefully

} catch (Exception e) {

System.*out*.println("Error: Invalid input. Please enter a valid order quantity."); // Handle invalid input

} finally {

System.*out*.println("Thank you for using the Order Processing System.");

scanner.close();

}

}

}

The robustness of the Order processing System program is exemplified through its comprehensive error handling and validation mechanisms. By incorporating a custom exception for insufficient stock, the program effectively manages potential runtime errors, ensuring that users receive clear feedback without crashing. while input validation checks prevent invalid selections and provide user guidance. The program also maintains resource efficiency by properly closing the `Scanner`, reducing the risk of resource leaks. Overall, these elements work cohesively to create a stable and reliable application that can handle unexpected conditions gracefully, enhancing the user experience and ensuring operational integrity.

**Program – 2**

There is a Cafe Order System, where a customer selects an item from a menu and also selects the quantity of that item, and the system checks item number entered by a customer, quantity of it is valid or not. If an invalid selection is made, the following program handles the error by displaying appropriate error messages.

# Custom exception for invalid order

class InvalidOrderException(Exception):

    def \_\_init\_\_(self, message):

        super().\_\_init\_\_(message)

# Menu item class representing food or drink items

class MenuItem:

    def \_\_init\_\_(self, name, price):

        self.name = name

        self.price = price

    def process\_order(self, quantity):

        if quantity <= 0:

            raise InvalidOrderException("Order quantity must be greater than zero.")

        total\_price = self.price \* quantity

        print(f"Order processed: {quantity} x {self.name} = {total\_price:.2f}")

def display\_menu(menu\_items):

    print("Menu:")

    for i, item in enumerate(menu\_items, start=1):

        print(f"{i}. {item.name} - {item.price:.2f}")

def main():

    # Create a menu with available items

    menu\_items = [

        MenuItem("Coffee", 12.50),

        MenuItem("Tea", 10.75),

        MenuItem("Sandwich", 43.00)

    ]

    # Display menu items

    display\_menu(menu\_items)

    try:

        # Get user's menu choice

        choice = int(input("Select an item by entering the corresponding number: ")) - 1

        if choice < 0 or choice >= len(menu\_items):

            print("Error: Invalid menu selection.")

            return  # Exit if the selection is invalid

        selected\_item = menu\_items[choice]

        quantity = int(input(f"Enter the quantity for {selected\_item.name}: "))

        # Process the order

        selected\_item.process\_order(quantity)

    except InvalidOrderException as e:

        print(f"Error: {e}")  # Handle invalid order gracefully

    except ValueError:

        print("Error: Invalid input. Please enter a valid number.")  # Handle non-numeric input

    finally:

        print("Thank you for visiting the Café.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

The robustness of the Café Order Processing program is demonstrated through effective error handling and input validation. The program uses a custom exception (`InvalidOrderException`) to manage invalid order quantities, ensuring that users cannot place orders with negative or zero quantities. It also catches non-numeric input with a `ValueError`, preventing crashes and guiding users to enter valid data. The program validates the menu selection to ensure the user chooses from available options, and gracefully handles errors by providing clear feedback without terminating unexpectedly. These features work together to maintain stability and provide a seamless user experience even in the face of incorrect inputs.

### Sample input and output:

**Output of program – 1:**

Welcome to the Order Processing System.

Available Products:

1. Laptop - Available Stock: 10, Price: 75000.0

2. Smartphone - Available Stock: 20, Price: 30000.0

3. Tablet - Available Stock: 15, Price: 45000.0

Select a product by entering the corresponding number: 1

You selected: Laptop

Available Stock: 10

Enter quantity to order: 3

Order processed! 3 units of Laptop sold.

Thank you for using the Order Processing System.

Welcome to the Order Processing System.

Available Products:

1. Laptop - Available Stock: 10, Price: 75000.0

2. Smartphone - Available Stock: 20, Price: 30000.0

3. Tablet - Available Stock: 15, Price: 45000.0

Select a product by entering the corresponding number: 3

You selected: Tablet

Available Stock: 15

Enter quantity to order: 17

Error: Insufficient stock for product: Tablet

Thank you for using the Order Processing System.

Welcome to the Order Processing System.

Available Products:

1. Laptop - Available Stock: 10, Price: 75000.0

2. Smartphone - Available Stock: 20, Price: 30000.0

3. Tablet - Available Stock: 15, Price: 45000.0

Select a product by entering the corresponding number: 5

Error: Invalid product selection.

Thank you for using the Order Processing System.

**Output of program – 1:**

Menu:

1. Coffee - 12.50

2. Tea - 10.75

3. Sandwich - 43.00

Select an item by entering the corresponding number: 2

Enter the quantity for Tea: 2

Order processed: 2 x Tea = 21.50

Thank you for visiting the Café.

Menu:

1. Coffee - 12.50

2. Tea - 10.75

3. Sandwich - 43.00

Select an item by entering the corresponding number: 4

Error: Invalid menu selection.

Thank you for visiting the Café.

Menu:

1. Coffee - 12.50

2. Tea - 10.75

3. Sandwich - 43.00

Select an item by entering the corresponding number: 3

Enter the quantity for Sandwich: 5

Order processed: 5 x Sandwich = 215.00

Thank you for visiting the Café.

Menu:

1. Coffee - 12.50

2. Tea - 10.75

3. Sandwich - 43.00

Select an item by entering the corresponding number: 1

Enter the quantity for Coffee: 3

Order processed: 3 x Coffee = 37.50

Thank you for visiting the Café.

# [Activity-4](#gjdgxs): Study the ASSERTIONS in C language and its importance in writing RELIABLE CODE. Study POSIX standards and write a C program under Unix to show use of POSIX standard in writing portable code.

## Problem Statement:

Study the ASSERTIONS in C language and its importance in writing RELIABLE CODE. Study POSIX standards and write a C program under Unix to show use of POSIX standard in writing portable code.

### Theory:

In C language, Assertions are a debugging tool that helps developers catch logical errors during runtime. Defined in the assert.h header, the assert() macro evaluates a condition (expression). If the expression is false, the program prints an error message and terminates, allowing developers to quickly identify bugs.

Importance of Assertions in Reliable Code:

1. Early Detection of Errors: Assertions enable catching errors early in the development process, preventing them from going unnoticed and causing unpredictable behavior.
2. Document Assumptions: They help document the assumptions made in the code, making it easier for future developers to understand the intended logic.
3. Testing Edge Cases: By asserting conditions that must always hold true, developers can ensure that edge cases and potential issues are thoroughly tested.
4. Aid in Debugging: Assertions can pinpoint exactly where a problem occurs, making the debugging process faster and more efficient.

POSIX (Portable Operating System Interface) is a set of standards defined by the IEEE to ensure compatibility and portability between different operating systems. These standards specify the interfaces and behaviors of operating system components, such as:

1. System Calls and APIs: Defines functions for handling file systems, processes, signals, and more (e.g., open(), fork(), exec()).
2. Shell and Utilities: Standardizes command-line utilities (e.g., grep, awk, sed) and scripting to ensure uniform behavior across POSIX-compliant systems.
3. Threads (Pthreads): Provides a standard for multithreading, allowing consistent thread management across platforms.

The POSIX standards help developers write code that can run on different UNIX-like systems (e.g., Linux, macOS, BSD) without modification, promoting cross-platform compatibility and reducing system dependency issues.

### Program:

C program that demonstrates the use of POSIX-compliant functions to create a portable program that runs on UNIX-like systems. This example uses POSIX functions like fork(), exec(), and wait() to create a child process and execute another program (e.g., /bin/ls).

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Create a new process using the POSIX fork() system call

pid = fork();

if (pid < 0) {

// Fork failed

perror("Fork failed");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process: Use exec() to run a new program (e.g., /bin/ls)

printf("Child process: Executing /bin/ls\n");

execl("/bin/ls", "ls", NULL); // POSIX-compliant exec function

perror("exec failed"); // If exec() fails

exit(EXIT\_FAILURE);

} else {

// Parent process: Wait for child process to complete

wait(NULL); // POSIX-compliant wait function

printf("Parent process: Child process completed\n");

}

return 0;

}

* fork(): Creates a new process (child). The child process gets a copy of the parent’s memory.
* execl(): Replaces the child process with a new program (in this case, `/bin/ls`).
* wait(): Parent process waits for the child process to finish.

This program is POSIX-compliant, making it portable across all UNIX-like operating systems that follow POSIX standards. It can be compiled and run on Linux, macOS, and other POSIX-compliant systems.

### Sample input and output:

Assuming the current working directory has file1.txt, file2.txt and folder1. The output will be

Child process: Executing /bin/ls

file1.txt file2.txt folder1

Parent process: Child process completed

child process prints “Child process: Executing /bin/ls” and ls command is executed by the child process, and it lists the files and directories in the current directory (file1.txt, file2.txt, folder1). Once the child process completes, the parent process prints “Parent process: Child process completed”.