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

CTA ACTIVITY REPORT

[22UHUC500- SOFTWARE ENGINEERING & PROJECT MANAGEMENT]

Odd Semester: Sep-Jan-2025

Course Teacher:Dr. Umakanth P Kulkarni

2024- 2025

Submitted by
By

Varsha
2SD22CS122
5th Semester B division

Table of Contents

Termwork-1: Write a C program to show that C programming Language supports only Call by Value.

Problem Statement:

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

Theory:

Business Scenario:

Imagine a bank application where we are attempting to update the balance of an account. However, the function receives only a copy of the original balance and not the actual value, so the real balance remains unchanged outside of the function.Here's a C program demonstrating that C supports call by value, meaning when arguments are passed to a function, the function operates on copies of the actual arguments rather than the originals.

Design:

Overview of Function Calling Mechanisms

Call by Value: A copy of the actual parameter is passed to the function.

Call by Reference: The actual parameter itself (memory reference) is passed to the function.

During Call by Value when a function is called in C, copies of the arguments are created and used within the function. Any changes made to the parameters inside the function do not affect the original variables.

Program:

C Program: Bank Balance Update (Demonstrating Call by Value)

```
#include <stdio.h>

// Function to update the balance (Call by Value)

void updateBalance(int balance) {

    // Adding $500 to the balance (Attempt to update balance)

    balance += 500;

    printf("Inside updateBalance: Updated Balance = $%d\n", balance);

}

int main() {

    // Initial balance of the account

 printf("Enter the Initial Balance of the account");

 scanf("%d\n", balance);

    // Print initial balance

    printf("Initial Balance = $%d\n", balance);
```

```
    // Call function to update the balance
    updateBalance(balance);
    // Print balance after calling updateBalance
    printf("After updateBalance function call: Balance = $%d\n", balance);
    return 0;
}
```

Sample input and output:

Input:

Enter the Initial Balance of the account =1000

Output:

Initial Balance = $1000

Inside updateBalance: Updated Balance = $1500

After updateBalance function call: Balance = $1000

References:

1. www.GeeksforGeeks.com
2. Learn-C.org

Termwork-2: prepare a report on USABILITY of at least two UIs of major software products that you have seen:

Problem Statement:

Study the concept of USABILITY. Prepare a report on USABILITY of at least two UIs of major software products that you have seen.

Theory:

Usability Report on Major Software Product UIs: A Business Scenario Approach.The usability of software products is crucial for user satisfaction, efficiency, and retention. Below is an analysis of the usability of two major software UIs based on principles of ease of use, navigation, learnability, and aesthetics.

Scenario Overview:

The fictional company "TechFlow" is evaluating the user interfaces (UI) of two major software products to determine which is best suited for its growing operational needs. TechFlow specializes in digital marketing and content creation, so usability is critical for efficient workflows. The products under consideration are:

1. Microsoft Teams (Collaboration & Communication)

2. Adobe Photoshop (Creative Design)

Design:

- User Interface (UI) Design:
- Unified Communication Platform:
- Collaborative Workspace:
- Customization & Personalization:

The design of Microsoft Teams(Collaboration & Communication) and Adobe Photoshop (Creative Design) is centered around enhancing collaboration, communication, and productivity within organizations. It incorporates several key principles and features, creating an intuitive and integrated user experience for remote or in-office teams.

Program:

```
#include <gtk/gtk.h>

// Callback function for button click

static void on_button_clicked(GtkWidget *widget, gpointer data) {

   g_print("Button Clicked! This is where you would send a message to Microsoft Teams.\n");

}

int main(int argc, char *argv[]) {

   GtkWidget *window;

   GtkWidget *button;

   // Initialize GTK

   gtk_init(&argc, &argv);

   // Create a new window

   window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

   gtk_window_set_title(GTK_WINDOW(window), "Teams Interaction Example");

   gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);

   g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
```

```
    // Create a button and add it to the window
    button = gtk_button_new_with_label("Send to Teams");
    g_signal_connect(button, "clicked", G_CALLBACK(on_button_clicked), NULL);
    gtk_container_add(GTK_CONTAINER(window), button);
    // Show all widgets
gtk_widget_show_all(window);
    // Start the main loop
    gtk_main();
    return 0;
}
```

Sample input and output:

Sample Input:

Running the program doesn't take any input from the user at the start. The input comes in the form of user interaction (i.e., clicking the button in the window).

Sample Output:

When the program starts, a window appears with a button labeled "Send to Teams".

If the user clicks the button, the following output will appear in the terminal:

Button Clicked! This is where you would send a message to Microsoft Teams.

References:

1. Udemy.com
2. StackOverflow.com

Termwork-3: List all the features of the programming language and write PROGRAMS to show how they help to write ROBUST code.

Problem Statement:

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

Theory:

Python is a versatile and widely-used programming language that is known for its simplicity and readability. Python is an incredibly versatile language with an extensive set of features that

make it ideal for a wide range of applications, from web development and data analysis to AI and scientific computing.

Design:

Here is an extensive overview of the key features of Python:

1.Simple and easy to learn :

2.Exception Handling:

3. Interpreted Language:

4.High-Level Language:

5.Object-Oriented Programming (OOP) Support:

6. Dynamic Memory Allocation and Garbage Collection:

7.Scripting and Automation:

Program:

1.# Function to add two numbers

```
def add_numbers(a, b):

return a + b

num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

result = add_numbers(num1, num2)

print(f"The sum of {num1} and {num2} is: {result}");
```

2.#Here's a simple Python program demonstrating exception handling using try, except, and #finally blocks:

```
try:

number = int(input("Enter a number: "))

        result = 10 / number

        print(f"Result: {result}")

except ValueError:
```

```
        # This block handles invalid inputs that cannot be converted to an integer
        print("Error: Please enter a valid integer.")
    except ZeroDivisionError:
        # This block handles division by zero errors
        print("Error: Division by zero is not allowed.")
    finally:
        print("Execution completed.")
```

3.# A python program to show that it is executed line by line

```
print("Step 1: Program starts")
x = 10
print(f"Step 2: Variable x is set to {x}")
y = x + 5
print(f"Step 3: Variable y is set to {y}")
if y > 10:
    print("Step 4: y is greater than 10")
print("Step 5: Program ends");
```

4.#A simple program to add two numbers in python

```
print(3+4)
```

5.# OOP Concept: Class and Object

```
class Animal:
    def _init_(self, name):
        # OOP Concept: Encapsulation (data hiding)
        self.name = name
        # OOP Concept: Method (behavior of the object)
```

```
        def speak(self)
         raise NotImplementedError("Subclass must implement abstract method")
# OOP Concept: Inheritance
class Dog(Animal):
        def speak(self):
        # OOP Concept: Polymorphism (method overriding)
        return f"{self.name} says Woof!"
class Cat(Animal):
        def speak(self):
        return f"{self.name} says Meow!"
# Creating instances (objects) of classes
dog = Dog("Bob")
cat = Cat("Kitten")
# Calling methods of objects
print(dog.speak())  # Output: Bob says Woof!
print(cat.speak())  # Output: Kitten says Meow!
```

6.#A sample program for dynamic Memory allocation & garbage collection

```
import gc
class DynamicObject:
    def _init_(self, value):
        self.value = value
        print(f"Object with value {self.value} allocated")

    def _del_(self):
        print(f"Object with value {self.value} deallocated")
```

```
def create_objects():
    print("Creating objects dynamically...")
    obj1 = DynamicObject(10)
    obj2 = DynamicObject(20)
    obj3 = DynamicObject(30)
    print("Objects created.")
    # Delete an object manually (simulates freeing memory)
    del obj2
    print("Manually deleted obj2.")
    # Run garbage collection
    print("Forcing garbage collection...")
    gc.collect()
if _name_ == "_main_":
    gc.enable()  # Enable automatic garbage collection
    create_objects()
    print("Program Executed”);
```

Sample input and output:

1. Input: Enter first number = 2
   Enter second number = 5
   Output: The sum of 2 and 5 is 7
2. Input: Enter a number: 5
   Output:2.0
   Execution completed.
3. Output:Step 1: Program starts
   Step 2: Variable x is set to 10
   Step 3: Variable y is set to 15

Step 4: y is greater than 10

Step 5: Program ends

This output shows the code running sequentially, one line at a time.

4. Output: 7
5. Output: Bob says Woof! Kitten says Meow!
6. Output:Creating objects dynamically...

   Object with value 10 allocated

   Object with value 20 allocated

   Object with value 30 allocated

   Objects created.

   Manually deleted obj2.

   Object with value 20 deallocated

   Forcing garbage collection...

   Program finished.

References:

1) Real Python
2) Python.org

Termwork-4: 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.

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:

An assertion in C is a debugging tool used to test assumptions made by the program at runtime. It allows developers to specify a condition (usually an expression) that they believe should be

true at a particular point in the program. If the condition evaluates to true, the program continues execution normally. If the condition evaluates to false, the program prints an error message and usually terminates the execution.In C, assertions are implemented through the assert() macro.

Design:

1.Importance of Assertions in Writing Reliable Code

a. Identifying Logical Errors Early

b. Defensive Programming

c. Improving Code Robustness

d. Simplified Debugging

Program:

Here's a simple C program that demonstrates the use of POSIX standards to write portable code for Unix-like systems. The program retrieves the process ID (PID) and user ID (UID) using POSIX-compliant functions:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main() {
    // Get the process ID (POSIX-compliant function)
    pid_t process_id = getpid();
    // Get the user ID (POSIX-compliant function)
    uid_t user_id = getuid();

    // Display the process ID and user ID
    printf("Process ID: %d\n", process_id);
    printf("User ID: %d\n", user_id);
    return 0;
}
```

This example ensures that the code is portable across Unix-like operating systems, as it uses POSIX-compliant functions.

Sample input and output:

Output

Process ID: 12345

User ID: 1000

References:

1) Unix Programming , The Linux Documentation Project.
2) Unix Tutorials-GeeksforGeeks.