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

Samiksha Sandeep Naik
2SD22CS085
5th Semester B division

Table of Contents

ASSIGNMENT-1

1.1 Problem Statement:

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

Business scenario: The program will take a number, increase it by 1, and print both the original and new numbers. We want to demonstrate that the original number does not change after the function is called.

1.2 Theory:

In C programming, functions use call by value, which means that when you pass an argument to a function, a copy of that argument is created. This copy is used within the function, so any changes made to it do not affect the original variable in the calling function.

1.3 Program:

```
#include <stdio.h>


 // Function to increment a number
      int increment(int num) {
       return num + 1; // Return the incremented value
      }
main() {
int    originalNumber    =    3;    //    Original    number
printf("Original Number: %d\n", originalNumber);
// Call the function to increment the number
int incrementedNumber = increment(originalNumber);


printf("Incremented Number: %d\n", incrementedNumber);
printf("Original   Number   after   incrementing:   %d\n",   originalNumber);   //
Remains unchanged
}
```

1.4 Sample input and output:

```
Original Number: 3
Incremented Number: 4
Original Number after incrementing: 3
```

1.5 References:

- Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language, 2nd Edition.

ASSIGNMENT-2

2.1 Problem Statement:

Study the concept "USABILITY". Prepare a report on USABILITY of at least TWO UIs of major software product you have seen.

2.2 Theory:

Usability refers to how easy and efficient it is for users to interact with a product, system, or service. In simple terms, usability is about making things user-friendly and intuitive. The important criteria for usability are often referred to as usability principles or factors. These key criteria include:

1. **Effectiveness**: How well users can complete tasks and achieve their goals with the system or product.
2. **Efficiency**: How quickly and easily users can accomplish tasks with minimal effort and time.
3. **Learnability**: How easy it is for new users to learn and use the system or product without a steep learning curve.
4. **Memorability**: How easily users can remember how to use the system after not using it for a while.
5. **Error Prevention and Recovery**: How well the system helps users avoid errors, and how easy it is to recover from them if they occur.

2.3 Design:

Two popular software interfaces: Google search and Microsoft Word

2.3.1 Google Search

Google Search is a widely used tool for finding information online. It's simple and user-friendly design ensures that anyone can use it with ease.

- **Ease of Use**: Google Search is simple, with a clear search bar and basic options for easy navigation.
- **Efficiency**: It provides fast results, and users can quickly filter information like images or news.
- **Learnability**: The interface is intuitive, requiring minimal effort to learn even advanced features.

- **Memorability**: Once learned, the interface is easy to remember, even after a long period of non-use.
- **Error Prevention and Recovery**: Google offers suggestions for mistakes and makes it easy to correct and adjust searches.

2.3.2 Microsoft Word

Microsoft Word is a widely used software for creating documents.

- **Ease of Use**: Microsoft Word's Ribbon toolbar makes basic tasks easy, but beginners may feel overwhelmed by its features.
- **Efficiency**: Word is fast with shortcuts and menus, but finding advanced features can take extra time.
- **Learnability**: Basic functions are easy to learn, but mastering advanced tools like mail merge and macros requires effort.
- **Memorability**: Simple tasks are easy to remember, but advanced features are often forgotten without regular use.
- **Error Prevention and Recovery**: Word helps avoid mistakes with checks, auto-save, and undo, offering recovery tools when needed.

2.4 References:

- **Nielsen, J. (1993).** *Usability Engineering*. Morgan Kaufmann.

ASSIGNMENT-3

3.1 Problem Statement:

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

3.2 Theory:

Robustness in code means that a program can handle unexpected problems, like errors or unusual inputs, without crashing or giving wrong results. A robust program keeps working properly, even when things go wrong, and gives clear messages to the user when there is an issue. The features of programming languages are :

- **Error Handling:** The ability to detect and respond to errors without crashing the program.
- **Memory Management**: Proper allocation and release of memory to avoid memory leaks or crashes.
- **Modularity:** Breaking the program into smaller, manageable, and reusable pieces (functions or modules).
- **Type Safety**: Prevents operations on incompatible data types, reducing errors during execution.
- **Input Validation**: Ensures that user inputs or external data are valid and safe before processing.

3.3 Program:

3.3.1 Program-1 Error handling

```
#include <stdio.h>


int main() {

   int a = 10, b = 0;

    if (b != 0) {

    int    result    =    a    /    b;

    printf("Result: %d\n", result);

 } else {
```

```
 printf("Error: Division by zero is not allowed.\n");
  }
  return 0;
}
```

3.3.2 Program-2 Memory management

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

int main() {
  int *arr = (int*) malloc(5 * sizeof(int)); // allocate memory
   if (arr == NULL) {
  printf("Memory allocation failed\n");
return 1;
  }
 // Initialize array
 for (int i = 0; i < 5; i++) {
    arr[i] = i * 2;
   }
    // Print array values
 for (int i = 0; i < 5; i++) {
    printf("%d ", arr[i]);
 }

   free(arr);  //  free  memory  to  prevent  leaks
  return 0;
}
```

3.3.3 Program-3 Modularity

```
#include <stdio.h>


// Function declaration

int sum(int a, int b);

int main() {
   int a = 10, b = 20;
   printf("Sum: %d\n", sum(a, b)); // calling function
   return 0;
}

// Function definition
int sum(int a, int b) {
   return a + b;
}
```

3.3.4 Program-4 Type-safety

```
#include <stdio.h>



int main() {

 int a = 10;

 float b = 5.5;

// type-safe operation

 float sum = a + b; // implicit type conversion

 printf("Sum: %.2f\n", sum);

  return 0;

}
```

3.3.5 Program-5 Input validation

```
#include <stdio.h>

int main() {
    int a, b;

    //      Input      first      number
    printf("Enter the numerator: ");
    scanf("%d", &a);

    // Input second number and validate
    printf("Enter  the  denominator:  ");
    scanf("%d", &b);

    if (b == 0) {
        printf("Error: Division by zero is not allowed.\n");
    } else {
        int     result     =     a     /     b;
        printf("Result: %d\n", result);
    }

    return 0;
}
```

3.4 Sample input and output:

Program-1: Error: Division by zero is not allowed.

Program-2: 0 2 4 6 8

Program-3: Sum: 30

Program-4: Sum: 15.50

Program-5:
```
Enter the numerator: 20

Enter the denominator: 0

Error: Division by zero is not allowed.
```

3.5 References:

- "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie

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 are used to test assumptions made by the programmer during development. They help identify bugs early by ensuring specific conditions (such as variable values or logic) hold true while the program is running.
- POSIX (Portable Operating System Interface) is a standard that defines a set of system calls, libraries, and guidelines for Unix-based operating systems. Its primary goal is to make applications portable across different Unix-like systems.

4.3 Program:

4.3.1 Program 1-Assertions

```
#include      <stdio.h>

#include <assert.h>


void check_number(int num) {

// Assert that the number is positive

  assert(num > 0); // If num is less than or equal to 0, the program
  will terminate

printf("The number %d is positive.\n", num);

}


int main() {

int num = -3;  // Test with a negative number


// Check if the number is positive
```

```
            check_number(num);


            return 0;
    }
```

4.3.2 Program 2-POSIX Program

```
            #include <stdio.h>
            #include <fcntl.h>    // For open()
            #include <unistd.h>   // For write(), read(), close()


            int main() {
             int fd;
             char buffer[20];


              // Open or create the file in write-only mode, 0644 sets file
              permissions
            fd = open("output.txt", O_CREAT | O_WRONLY, 0644);


            if (fd < 0) {
             perror("File open failed");
             return 1;
}


    // Write text to the file using POSIX write()
           write(fd, "Hello, POSIX!\n", 14);


// Close the file using POSIX close()
      close(fd);
    // Open the file again in read-only mode
        fd = open("output.txt", O_RDONLY);
        if (fd < 0) {
        perror("File open failed");
         return 1;
    }

    // Read the content and print it to the screen
         read(fd, buffer, 14);
         printf("File contents: %s", buffer);

    // Close the file
        close(fd);

        return 0;
}
```

4.4 Sample input and output:

Program1:

```
Assertion failed: num > 0, file filename.c, line 4
Aborted (core dumped)
```

Program 2:

```
Message written to file successfully.
```

After running the output.txt file ,we get

```
Hello, POSIX!
```

4.5 References:

- "Advanced Programming in the UNIX Environment" by W. Richard Stevens
- The POSIX.1 standard as part of the IEEE Std 1003.1-2017 document.