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

[22UHUC500- SOFTWARE ENGINEERING AND PROJECT MANAGEMENT]

Odd Semester: Jan-June-2022

Course Teacher: Dr. U.P.Kulkarni

2021- 2022

Submitted by

By

Mr. Suhas.S.Sambargi

2SD22CS109

5th Semester B division

Table of Contents

Termwork-1:

Problem Statement: **Write a C program to show that C programming Language supports only Call by Value.**

Theory: There are two parameter passing methods passing (or calling) methods to functions. They are Call by Value and Call by Reference. The most significant distinction between call by value and call by reference is that in the call by value, passing an actual value of the parameter. While, in a call by reference, passing the reference of the parameter; hence some change made to formal arguments will also reflect in actual arguments. C Language only supports Call by Value.This means that the called function is given the values of its arguments in temporary variables rather than the originals. This leads to some different properties than are seen with "call by reference". In C, both the calling and called functions don't share any memory they have their copy, along with the called function can't directly change a variable in the calling function; it may only alter its private, temporary copy. Call by value is an asset, however, not a liability. It usually leads to more compact programs with fewer extraneous variables, because parameters can be treated as conveniently initialized local variables in the called routine.

Program:

```c
#include <stdio.h>

void Value(int num)

{

    num = 7;

    printf("Value of number in function: %d\n", num);

}

int main()

{

    int num = 5;

    printf("Before function call, value of num: %d\n", num);

    Value(num);

    printf("After function call, value of num: %d\n", num);

    return 0;
}
```

```
Before function call, value of num: 5

Value of number in function: 7

After function call, value of num: 5
```

This demonstrates that changes made to the parameter inside the function do not affect the original argument in the caller function. When a variable is passed as an argument to a function in C, a copy of its value is created and passed to the function. Therefore, any modifications made to the parameter inside the function are only applied to the local copy, and the original value of the argument remains unaffected. This behavior is known as "Call by Value" in C.

Here the value of num in the main function is declared as 5. A function Value is called. In that function the value of the num is initialized to 7. After the execution of the function is finished the control is given back to the main function. In the main function if we print the value of num, its value will be the same as it was in the main function. The changes made in the function will not affect the value in the main function. Only the copy of the variable is sent to the function, not the actual variable.

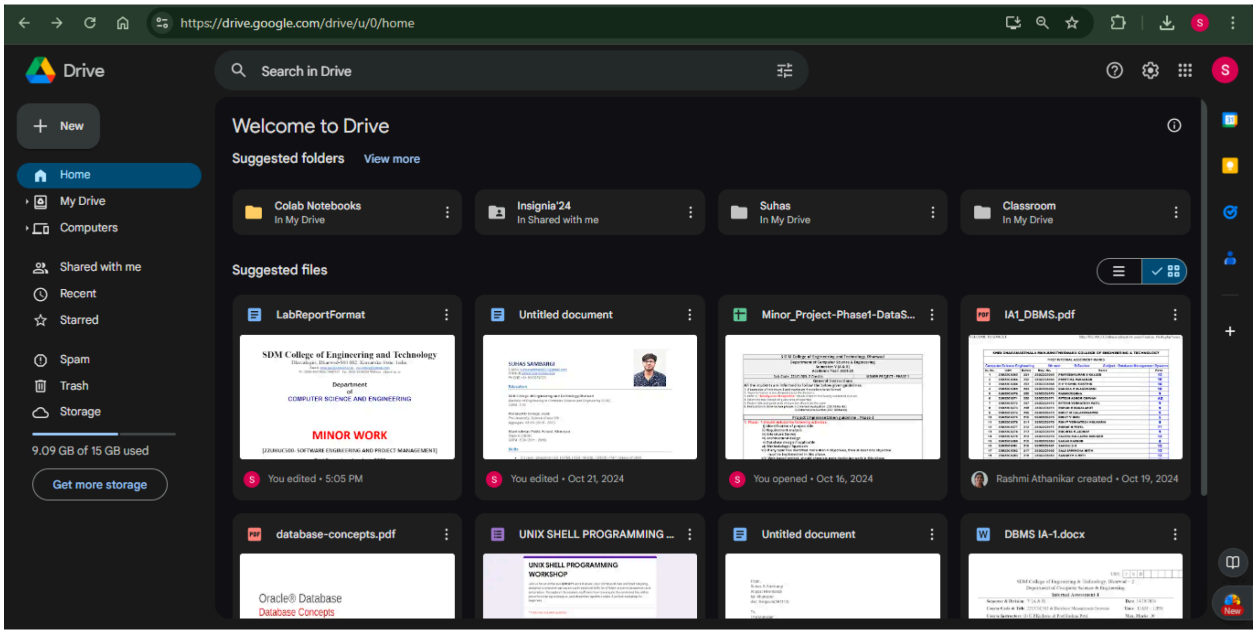
Termwork-2:

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

Usability is a way to measure how easy a product is to use. It is a concept in design circles to ensure products—whether websites, mobile applications can be used as simply and painlessly as possible. Good usability means users can accomplish their tasks quickly, with minimal stress and errors, and ultimately feel satisfied in their interaction with a product. For websites in particular, usability is crucial. Visitors to a website can easily leave as soon as they encounter difficulty or confusion.

REPORT ON USABILITY OF TWO UIS OF MAJOR SOFTWARE PRODUCTS

1.GOOGLE DRIVE Google Drive is a cloud-based file storage and synchronization service developed by Google, enabling users to store, share, and collaborate on files across various devices.



1. Learnability: Google Drive is highly intuitive for first-time users. The user interface is simple, with clear icons for uploading, creating, and organizing files. The drag-and-drop feature for uploading files and folders is straightforward, making it easy for new users to learn.

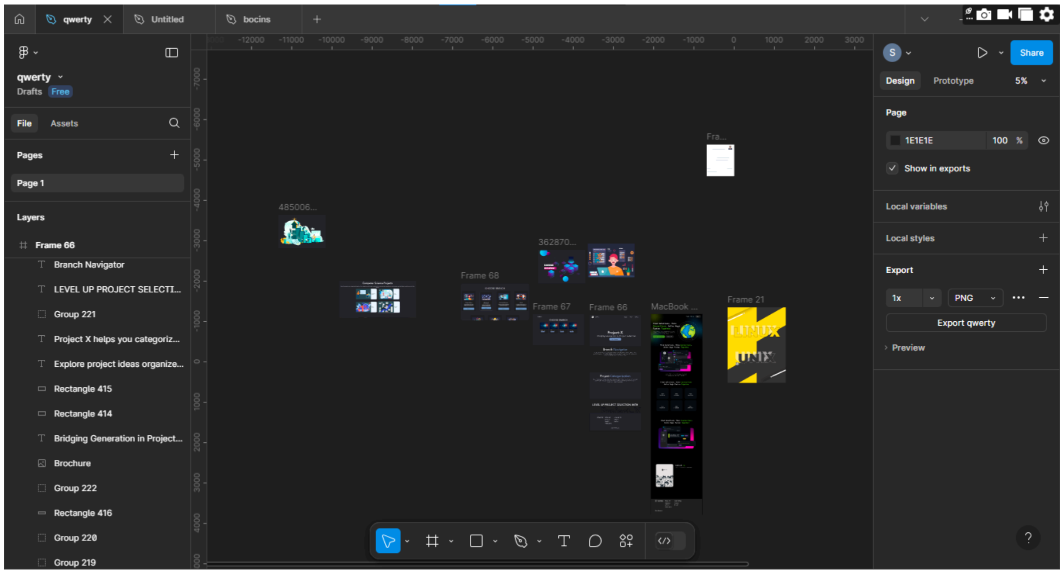
2. Efficiency: Google Drive is highly efficient for managing files. Users can quickly access their documents, spreadsheets, and presentations via integration with Google Workspace (Docs, Sheets, Slides). Real-time collaboration features allow multiple users to work on the same document simultaneously, improving workflow efficiency. Features like search filters, shortcuts, and file organization (folders, labels) contribute to fast file access.

3. Memorability: The consistent interface across all devices (desktop, mobile, web) ensures high memorability. Returning users can easily navigate Google Drive thanks to its minimal design and use of recognizable icons. Even after a long break, users can quickly remember how to upload, share, or organize files.

4. Error Handling: Google Drive has robust error-handling mechanisms. When uploading files, users are notified of any issues (e.g., connectivity problems, unsupported file types) through clear alerts. Additionally, Google Drive's version history and trash recovery options provide users with the ability to restore files and undo changes, which minimizes the impact of errors.

5. Satisfaction: User satisfaction with Google Drive is high due to its ease of use, integration with Google Workspace, and the ability to share files effortlessly. The platform's cross-device functionality and collaborative features offer users a smooth experience when working individually or with teams.

<u>2.FIGMA:</u>



Figma is a cloud-based design and prototyping tool that enables teams to create and collaborate on UI/UX designs in real time.

a. Learnability: Figma offers an intuitive interface that is easy to learn for users familiar with design tools like Sketch or Adobe XD. However, for users new to design software, there is a moderate learning curve due to the advanced tools and features. Figma provides tutorials and onboarding tips to assist beginners**.**

b. Efficiency: Once familiar with the tool, Figma is highly efficient for collaborative design work. Its real-time collaboration feature allows multiple users to work on the same design simultaneously. Design libraries, components, and responsive grid systems streamline the design process, improving productivity for teams working on large projects.

c. Memorability:Figma's layout and design tools are well-organized, which helps users retain knowledge after periods of inactivity. The drag-and-drop functionality, combined with easy-to-access components, makes it easier for users to resume their work without much relearning.

e. Satisfaction: Figma delivers a high satisfaction level, particularly for professional designers and teams. Its robust collaboration features, cloud-based accessibility, and design tools create an enjoyable and efficient design experience. Users appreciate the ability to work together seamlessly across different devices and operating systems.

Termwork-3:

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

Theory: There are different types of programming languages. Based on their features they are categorized as procedural programming language, object-oriented programming language, Functional programming language, Scripting language and Logic Programming language. The features are language specific but few common features of a programming language are

- Syntax: Each programming language has its own syntax or set of rules that govern how programs are written and structured. Enforcing type safety helps catch type-related errors during compilation rather than at runtime, reducing the chances of unexpected behavior or crashes.

  Example: Basic syntax of C

  ```
  //hello world

  #include <stdio.h>

  // Defining a main function

  void main()

  {

      // code

      printf("Hello World!");

  }
  ```

- Modularity: Modularity refers to the concept of dividing a system or a complex entity into smaller, self-contained modules or components. Each module performs a specific function or task and can be developed, tested, and maintained independently of the other modules.

  Example of a module in C

  ```
  //module to add two numbers
  void add ( int a, int b)
  {
      int sum;
      sum = a + b;
  ```

```
     printf("Sum of numbers= %d",sum) ;
       return 0;
}
```

- Libraries: Libraries are collections of precompiled code and resources that provide specific functionality or services to developers. They are designed to simplify software development by offering reusable components, functions, classes, or modules that can be incorporated into applications or projects.

```
//program to find square root of a number
#include <stdio.h>
#include <math.h>
int main()
 {
    float number = 25;
    float squareRoot = sqrt(number);
    printf("Square root of %.2f is %.2f\n", number,
squareRoot);
    return 0;
}
```

- Control Structures: Languages offer control structures to control the flow of program execution. Common control structures include conditionals (if-else statements, switch statements) and loops (for loops, while loops) that allow branching and repetition.

Example: Program showing the use of if else

```
//program to determine is a number is even or odd
#include <stdio.h>
int main()
 {
      int num;
      printf("Enter a number: ");
      scanf("%d", &num);
         if (num > 0)
```

```
        {
                    printf("The number is
positive.\n");
         }
 else if (num < 0)
    {
                    printf("The number is
negative.\n");
    }
else
{
                    printf("The number is zero.\n");
    }
return 0;
}
```

Termwork-4:

Problem Statement: Study the ASSERTIONS in C and its importance in writing reliable code.

Theory: Assertions are statements used to test assumptions made by programmers. They are an important tool for ensuring the correctness of your code in C. Assertions are statements that test for certain conditions that should always be true. If an assertion fails, the program will terminate immediately, allowing you to quickly identify and fix the problem.

Uses of Assertion:

- Debugging
- Program correctness
- Testing and quality assurance

Key Uses of Assertions in C

1. Debugging

Assertions are invaluable for debugging during the development phase. They serve as checkpoints to ensure the program follows the intended logic. By placing assertions at key points, developers can catch logic errors early. When an assertion fails, the program stops executing, providing immediate feedback on where the error occurred, making debugging more efficient.

Example:

```
int x = -5;

assert(x > 0);  // This will fail and terminate the program, as x is negative.
```

In this case, the assertion checks whether the value of x is positive. If it is not, the program halts, signaling an issue in the logic that needs to be addressed.

2. Program Correctness

Assertions play a crucial role in ensuring the correctness of a program. After implementing an algorithm or function, developers can use assertions to verify that the program behaves as expected. By embedding checks that enforce logical consistency, assertions help prevent bugs from progressing unnoticed.

Example:

```c
void checkArraySorted(int *arr, int size) {
    for (int i = 0; i < size - 1; i++) {
        assert(arr[i] <= arr[i + 1]);  // Ensure the array
is sorted.
    }
}
```

This example ensures that an array is correctly sorted. If the array is not sorted, the assertion fails, and the program aborts, signaling a potential bug in the sorting logic.

3. Testing and Quality Assurance

In automated testing, assertions are vital for ensuring that the program produces the expected results. They allow developers to specify the correct behavior of a function or module and verify that it behaves as expected under different test cases. Assertions in test cases can help detect issues before the software is released, improving overall quality.

Example:

```c
void testFunction() {
    int expected = 5;
    int actual = someFunction();
    assert(expected == actual);  // Ensure the function
returns the expected value.
}
```

In this test case, the assertion checks if the output of `someFunction()` matches the expected result. If not, the assertion fails, indicating that the function is not behaving as intended.

Termwork-5:

Problem Statement: Study the POSIX standard system calls and write a program to demonstrate it.

Theory:The Portable Operating System Interface (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines both the system and user-level application programming interfaces (APIs), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems. POSIX is also a trademark of the IEEE. POSIX is intended to be used by both application and system developers. The POSIX standard defines a set of system calls that provide an interface for interacting with the operating system. These system calls allow programmers to perform various tasks such as file I/O, process management, inter-process communication, and more.

POSIX standard system calls

- **fork:** The fork() system call is a fundamental operation provided by the operating system that creates a new child process from the existing parent process. It essentially duplicates the entire parent process, including the code, data, file descriptors, and other resources, to create an exact copy in the child process.
- **execvp:** The execvp() system call is a function provided by the operating system that replaces the current process with a new program. It is typically used to execute an external program from within a C or C++ program. The "v" in execvp stands for "vector," indicating that the program and its arguments are passed as an array of pointers.
- **waitpid**: The waitpid() system call is used to wait for the completion of a specific child process and retrieve its termination status. It allows a parent process to pause its execution until one of its child processes exits or terminates.
- **open:** The open() system call is used to open a file or create a new file in the file system. It provides a way to access files for reading, writing, or both, depending on the specified flags.
- **close:** The close() system call is used to close a file descriptor after you have finished using it. It releases the resources associated with the file descriptor and makes it available for reuse by the system.

```
// Program to demonstrate fork(), execvp and waitid system
calls

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
    pid_t child_pid = fork();
    if (child_pid == -1){
        perror("fork");
        exit(1);
    }
    else if (child_pid == 0){
        char *args[] = {"ls", "-l", NULL};
        execvp("ls", args);
        perror("execvp");
        exit(1);
   }
    else{
        int status;
        waitpid(child_pid, &status, 0);
        if (WIFEXITED(status)) {
            int exit_status = WEXITSTATUS(status);
            printf("Child process exited with status: %d\n",
exit_status);
        }
        else if (WIFSIGNALED(status)){
            int signal_number = WTERMSIG(status);
                printf("Child process terminated by signal:
%d\n",
                    signal_number);
```

```
      }
  }
  return 0;}
```