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 and Project Management
SEMESTER: 5th DIVISION: CSE 'B'
COURSE TEACHER: Dr. U.P Kulkarni

[Academic Year- 2024-25]

Date of Submission: 10-10-2024

Submitted

By

Ms. Sumedha K Bagalkoti USN : 2SD22CS111

Contents

Problem Statement 1: "Write a C Program to show that C Programming Language supports only call by value?"

```
#include <stdio.h>
 void increment(int x) {
    x++;
    printf("Inside increment: x = %d\n", x);
}
int main() {
    int a = 5;
    printf("Before increment: a = %d\n", a);
    increment(a);
    printf("After increment: a = %d\n", a);
    return 0;
}
```

In this example, the increment function takes an integer argument x. Inside the function, the value of x is incremented. However, since the function receives a copy of the original variable a, the incrementation performed within the function doesn't affect the value of a in the main function. Th
erefore the function operates on a copy of the original variable.

Problem Statement 2: "Study the concept 'USABILITY'. Prepare a report on USABILITY of at least two UI's of major software products you have seen."

Definition:

Usability measures how easy it is for a user to achieve their goals when interacting with a product. A usable product is intuitive, efficient, and satisfying to use.

Key Components of Usability:

1. Learnability: How easy it is for a new user to learn the basics of the product and start using it effectively.
2. Efficiency: How quickly and accurately users can perform tasks with the product once they have learned how to use it.
3. Memorability: How easy it is for users to remember how to use the product after a period of not using it.
4. Error Prevention: How well the product helps users avoid errors and recover from them when they do occur.
5. Satisfaction: How pleasant it is for users to interact with the product.

Software Product 1: Notepad

Key features of Notepad

1. Simplicity: Notepad is designed for basic text editing with no formatting options. It is extremely lightweight and simple, which makes it highly usable for quick tasks like note-taking, or editing plain text files. Users don't need any training to use Notepad, as the interface is minimal—just a blank screen and a few options for saving and editing text.
2. Speed: Due to its simplicity, Notepad loads quickly and is easy to use for short tasks. It's a go-to tool for users who need to quickly note down information or work with unformatted text.
3. No Distractions: Notepad lacks complex features like font styling, images, or tables, which can be seen as a benefit when the user needs a distraction-free environment for writing.

Strengths:

1. Simplicity
2. Lightweight
3. Speed
4. No Formatting
5. Universal Compatibility

Weaknesses:

1. Lack of Features
2. Poor for Large Documents
3. Basic Interface
4.

Software Product 2: Microsoft Word

Key features of Microsoft Word:

1. Feature-rich: Word is a full-fledged word processor that supports advanced document formatting, multimedia embedding, templates, and collaboration features. Its rich set of tools can be overwhelming for first-time users but is invaluable for creating complex documents like reports, resumes, and presentations.
2. Learnability: While Word has a learning curve compared to Notepad, Users can find formatting tools, styles, and editing options through labeled tabs like "Home," "Insert," and "Layout."
3. Efficiency: Once users become familiar with the software, they can work efficiently, using features like templates, and formatting styles to streamline document creation. Microsoft Word also provides real-time collaboration features, enabling multiple users to edit a document simultaneously.
4. Help and Support: Word offers extensive documentation, tutorials, and a "Help" feature to assist users in learning advanced functionalities, improving usability for more complex tasks.

Strengths:

1. Comprehensive Features
2. User-Friendly Interface
3. Advanced Document Formatting
4. Autosave and Document Recovery

Weaknesses:

1. Complexity
2. Resource-Intensive
3. Feature Overload

Problem Statement 3: "List all features of Programming language and write Programs to show they help to write ROBUST code."

Java is a popular programming language known for its robustness, platform independence, and object-oriented features. Here are some key features and code examples demonstrating their benefits:

1. Object-Oriented Programming (OOP):

- Encapsulation: Bundles data and methods within objects, promoting data security and modularity.
- Inheritance: Allows classes to inherit properties and methods from parent classes, promoting code reuse and extensibility.
- Polymorphism: Enables objects of different classes to be treated as if they were of the same type, providing flexibility and dynamic behavior.
- Example:

```
class Animal {
   void makeSound() {
      System.out.println("Generic animal sound");
   }
}
class Dog extends Animal {
   @Override
   void makeSound() {
      System.out.println("Woof!");

   }
}
class Cat extends Animal {
   @Override
   void makeSound() {
      System.out.println("Meow!");
   }
}
public class
 Main {
   public static void main(String[] args) {
         Dog d = new Dog();
         Cat c =  new Cat();
         d.makeSound();
         c.makeSoud();
      }
   }
}
```

This code demonstrates polymorphism and inheritance, where different animal objects can be treated as the same type and their specific sounds are called using the makeSound() method.

2. Automatic Memory Management (Garbage Collection):
Handles memory allocation and deallocation automatically, reducing the risk of memory leaks and errors.
Example:

```
public class Main {
   public static void main(String[] args) {
      String message = "Hello, world!";
      // No need to manually deallocate memory
   }
}
```

The Java garbage collector will automatically reclaim the memory used by the message object when it is no longer needed.

3. Exception Handling:
Provides mechanisms to handle errors and unexpected situations gracefully, preventing program crashes.
Example:

```
public class Main {
   public static void main(String[] args) {
      try {
         int result = 10 / 0;
         System.out.println(result);
      } catch (ArithmeticException e) {
         System.out.println("Error: Division by zero");
      }
   }
}
```

This code catches the Arithmetic Exception thrown when dividing by zero and provides a meaningful error message.

4. Strong Typing:
Enforces strict rules about data types, preventing unintended type conversions and errors.
Example:

```
public class Main {
   public static void main(String[] args) {
      int num = 5;
      double decimal = num / 2; // Implicit type conversion
      System.out.println(decimal);
   }}
```

In this example, the integer num is implicitly converted to a double before division, ensuring correct type compatibility.

Problem Statement 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."

Assertions in C and Their Importance:

Assertions in C are a powerful tool for debugging and ensuring the correctness of code. They are essentially conditions that are expected to be true at a particular point in the program. If an assertion fails, the program terminates with an error message.

Importance of Assertions:

1. Early Detection of Errors: Assertions can help identify and fix bugs early in the development process, preventing them from propagating to later stages.
2. Documentation: Assertions can serve as documentation, clarifying the expected behavior of the code and making it easier to understand.
3. Testing: Assertions can be used as part of unit testing to verify that code is functioning as expected.
4. Defensive Programming: Assertions can help make code more defensive by checking for invalid inputs and unexpected conditions.

Using assert.h:

To use assertions in C, you need to include the <assert.h> header. The assert takes a boolean expression as an argument. If the expression evaluates to false, the program terminates with an error message.

Example:

```
#include <stdio.h>
#include <assert.h>
int divide(int numerator, int denominator) {
   assert(denominator != 0);
   return numerator / denominator;
}
int main() {
   int result = divide(10, 0);
   printf("Result: %d\n", result);
   return 0;
}
```

In this example, the assert checks if the denominator is not zero. If it is, the program will terminate with an assertion failure.

POSIX Standard and Portable Code:

POSIX (Portable Operating System Interface) is a family of standards for operating systems that define a common API for various functions, including file I/O, process management, and networking. By adhering to POSIX standards, you can write C code that is more portable and can be compiled and run on different Unix-like systems without significant modifications.

Business Scenario: Cross-Platform File I/O:

Imagine a business that needs to develop a utility to write data to the files on different operating systems. Using POSIX functions, we can write a portable C program to achieve this:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
int main() {
    // POSIX file creation using open()
    int fd = open("example.txt", O_WRONLY | O_CREAT, 0644);
    if (fd == -1) {
        perror("Failed to open file");
        return 1;
    }
 // Data to write to the file
    const char *data = "Hello, POSIX!\n";
    // POSIX write system call
    if (write(fd, data, strlen(data)) == -1) {
        perror("Failed to write to file");
        close(fd);
        return 1;
    }
    // POSIX file close
    if (close(fd) == -1) {
        perror("Failed to close file");
        return 1;
    }
    printf("Data written to file successfully!\n");
    return 0;
}
```

This program uses POSIX functions like open, write, and close to read from an input file and write to an output file. By using these standard functions, the code can be compiled and run on various Unix-like systems without requiring significant changes.