20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Abinesh G
CB.EN.U4CYS21001
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Procedural Programming Introduction

Introduction:

- Shifting gears: Move beyond procedural and object-oriented paradigms.
- Functional programming: Mathematical functions reign supreme!
- Focus: Composing pure functions, avoiding state changes and side effects.

Procedural Programming Key Principles

Key Principles:

- Immutability: Data is fixed, like constants in math equations.
- Pure Functions: Predictable and reliable operations, always the same output for the same input.
- First-Class and Higher-Order Functions: Treat functions like any other value, compose complex functions from simpler ones.
- Referential Transparency: Replace function calls with their results without affecting program behavior.
- Recursion: Solve repetitive tasks by a function calling itself with new arguments.
- Function Composition: Build complex logic from smaller, well-defined functions.

Example

Example: Calculating the factorial of a number:

```c
#include <stdio.h>

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1); // Recursive call
    }
}

int main() {
    int num = 5;
    int result = factorial(num);
    printf("The factorial of %d is %d\n", num, result);
    return 0;
}
```

Explanation: The factorial function uses a recursive approach to compute the factor
The main function calls factorial with a value, initiating the calculation. The function
proceeds sequentially, with control flow determined by if-else statements and the
recursive call.

Procedural - Applications

Applications:

- Data analysis and scientific computing: Manipulate and analyze large datasets.
- Web development and server-side applications: Build robust and scalable systems.
- Financial modeling and risk analysis: Develop precise and reliable models.
- Concurrent programming and real-time systems: Efficiently handle parallel tasks.
- Software development tools and libraries: Build reliable and expressive tools.
- Examples: Haskell, Erlang, Lisp (and many more!).

Procedural - Conclusion

Conclusion:

- Procedural programming offers a straightforward approach with clear control flow.
- Well-suited for projects with well-defined sequential tasks.
- Understanding its features and benefits is essential for effective programming.

Functional - Haskell

Benefits:

- Concise and Expressive: Clear and concise syntax for complex logic.
- Modular and Reusable: Pure functions promote modularity and code reuse.
- Reliable and Testable: Immutability, purity, and type safety enhance reliability and testability.
- Concurrent and Parallel: Well-suited for parallel and concurrent applications.
- Abstraction and Problem-Solving: Focus on high-level concepts and problem-solving.

Example

```haskell
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)

main :: IO ()
main = do
    let num = 5
    print $ "The factorial of " ++ show num ++ " is " ++ show (factorial num)
```

Explanation: The factorial function is defined purely, without side effects, using pattern matching for recursion. It takes an integer and returns an integer, adhering to Haskell's type system. The main function uses let to bind a value to num, then prints a formatted string incorporating the calculated factorial. The focus is on declaratively defining the factorial function, not the step-by-step execution

Comparison and Discussions

Procedural:

Strengths:

- Simple and straightforward concept.
- Efficient for sequential tasks.
- Good for beginners to learn programming fundamentals.

Weaknesses:

- Can be verbose and complex for large programs.
- Difficult to handle concurrency and state management.
- Less intuitive for non-sequential or data-driven problems.

When to use it?

- Projects with well-defined sequential tasks.
- Resource-constrained environments (embedded systems).
- Educational purposes to learn basic programming concepts.

Procedural - C

Benefits of Procedural-C:

- Clear structure and control flow.
- Efficient for sequential tasks.
- Portable across different platforms.
- Flexible memory management.
- Suitable for beginners to learn programming fundamentals.

Applications of Procedural-C:

- System programming (operating systems, device drivers).
- Embedded systems (microcontrollers, IoT devices).
- Scientific computing and simulations.
- Data processing and analysis.
- Educational purposes for teaching programming concepts.

Conclusion:

- Procedural-C: A powerful and versatile language for a wide range of applications.
- Well-suited for projects requiring clear structure, control, and efficiency.
- Understanding its features and benefits is essential for effective programming.

Functional - Applications

Applications:

- Data analysis and scientific computing: Manipulate and analyze large datasets.
- Web development and server-side applications: Build robust and scalable systems.
- Financial modeling and risk analysis: Develop precise and reliable models.
- Concurrent programming and real-time systems: Efficiently handle parallel tasks.
- Software development tools and libraries: Build reliable and expressive tools.
- Examples: Haskell, Erlang, Lisp (and many more!).

Functional - Conclusion

Conclusion:

- Functional programming offers a distinct approach to building software.
- Its emphasis on simplicity, purity, and composability leads to robust, modular code.
- Explore functional programming to expand your programming skillset and tackle new challenges.

Bibliography

- Google
- TheCloudStrap
- GeeksforGeeks
- ChatGPT
- Bard
- Notion
- etc