Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages Assignment-01: Exploring Programming Paradigms

Vinayak R

21st January, 2024

Paradigm 1: Procedural

Procedural programming is centered around a series of instructions provided to the computer. These sets of instructions are commonly referred to as **procedures**. Procedural programming is considered a subset of the **Imperative Programming Paradigm**.

Procedural programming adopts a **top-down approach** due to the execution of instructions in well-defined sequences.

In procedural programming, our instructions are organized into smaller blocks of clearly defined code. These blocks of code are referred to as **functions**, with each function carrying out a distinct task. Executing a procedural program involves calling the appropriate function at the right time, specifically using the suitable function at the correct position within the extensive code. These functions often receive data and manipulate it to generate an output.

Structure of Procedural Programming

- Refer Figure 1

Language for Paradigm 1: Pascal

Pascal is an imperative and **procedural programming** language intended to encourage good programming practices using structured programming and data structuring.

Pascal programs start with the *program* keyword with a list of external file descriptors as parameters; then follows the main block bracketed by the *begin* and *end* keywords. *Semicolons* separate statements, and the *full stop* ends the whole program. *Letter case* is ignored in Pascal source.

Structure of Pascal Program

- Program name
- Uses command

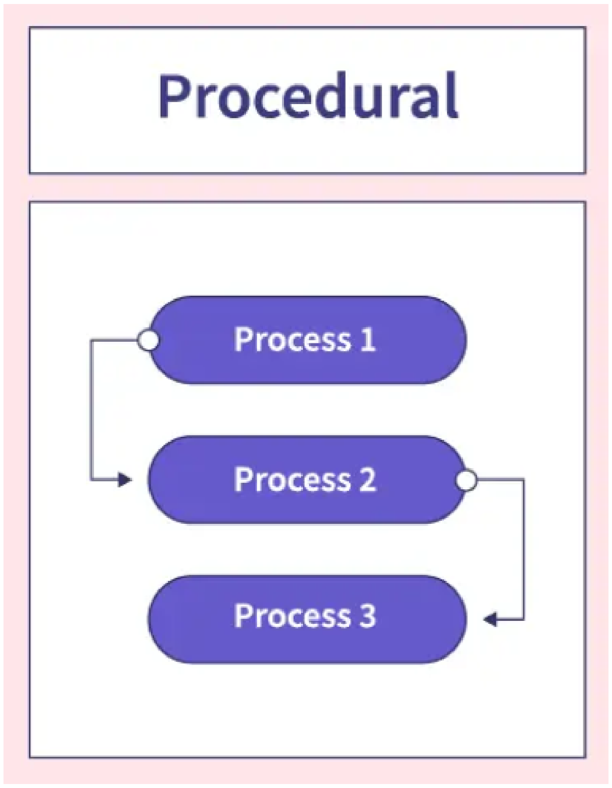


Figure 1: Structure of Procedural Programming

- Type declarations
- Constant declarations
- Variables declarations
- Functions declarations
- Procedures declarations
- Main program block
- Statements and Expressions within each block
- Comments

Example Code:

```
program HelloWorld(output);
begin
    WriteLn('Hello, World!')
end.
```

The text "Hello, World!" is printed as the output of the execution of above code.

Pascal Data Types

- Refer Figure 2

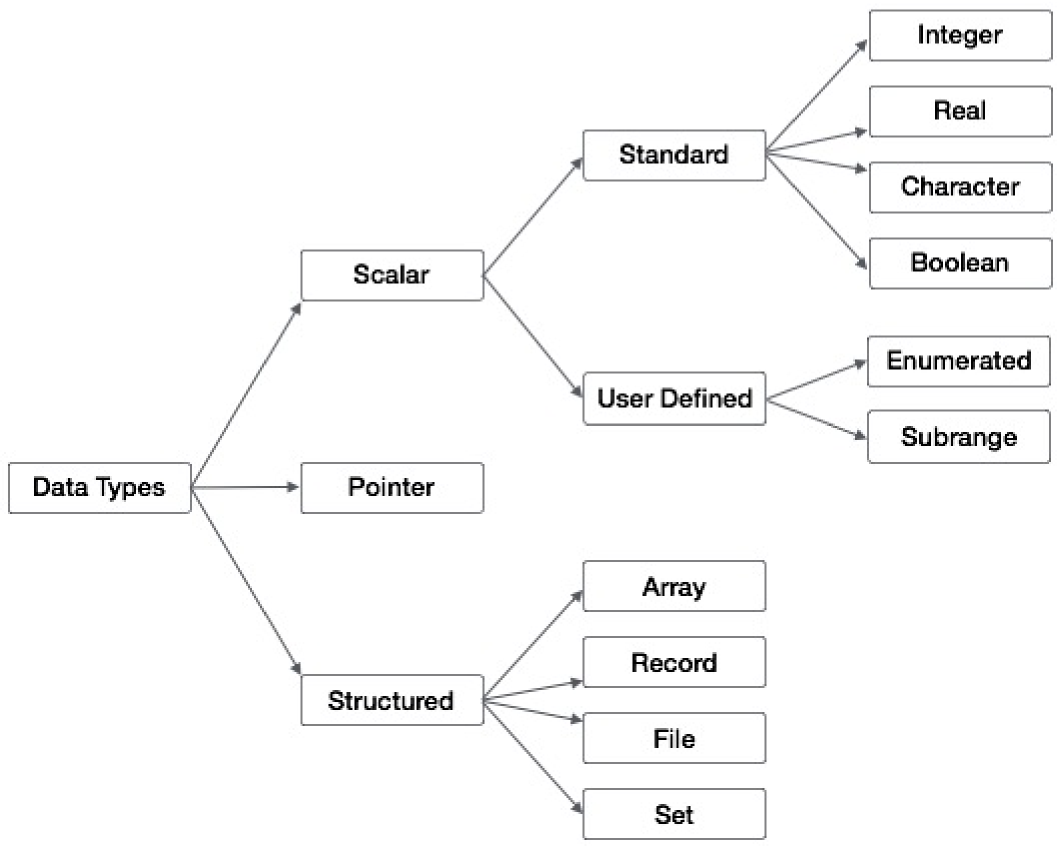


Figure 2: Data Types in Pascal

Paradigm 2: Dataflow

Dataflow programming (DFP) is a programming paradigm where program execution is conceptualized as data flowing through a series of **operations** or transformations. Each operation may be represented as a node in a graph. **Nodes** are connected by directed arcs through which data flows. A node performs its operation when its input data are available. It sends out the result on the output arcs. Dataflow programming aids **parallelization** without the added complexity that comes with traditional programming.

Simple program and its dataflow equivalent:

$$\begin{aligned} A &:= X + Y \\ B &:= Y / 10 \\ C &:= A * B \end{aligned}$$

Figure 3: A simple program

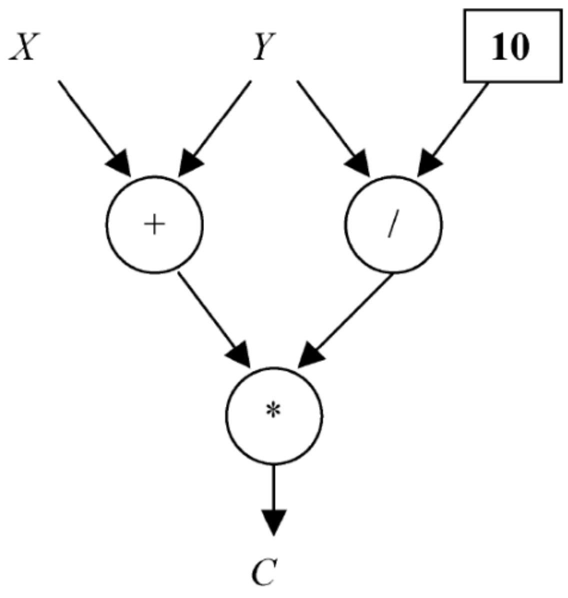


Figure 4: Dataflow equivalent

Consider the simple program shown in the above figure. In a traditional von Neumann architecture, these statements would be executed sequentially. Thus, the program would execute in three units of time.

In dataflow programming, we represent the program as a dataflow graph that has three inputs (X, Y, 10), three operations (+, /, *) and one output (C). This is a directed graph. Nodes represent instructions or operations. Arcs represent variables. Equivalently, arcs represent data dependencies between instructions. The value in Y is also duplicated since it's sent to two instructions.

In dataflow execution model, this program takes only two units of time. This is because the addition and division instructions can execute in parallel. For each of these instructions, their respective inputs are available at the start. They have no dependencies on each other and therefore can be parallelized.

Language for Paradigm 2: Blender Game Engine

In the context of game engines, Dataflow programming can be used to design game logic in a visual and intuitive way. This allows developers to create complex behaviours without necessarily writing code.

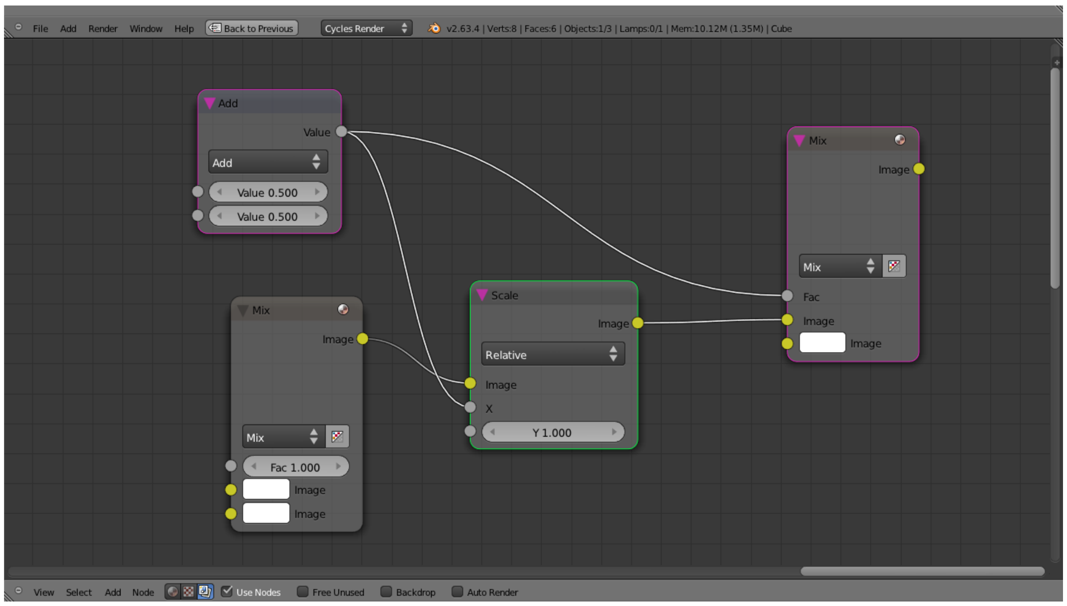

File Add Render Window Help Back to Previous
Cycles Render
v2.63.4 | Verts:8 | Faces:6 | Objects:1/3 | Lamps:0/1 | Mem:10.12M (1.35M) | Cube
Add
Value
Add
Value 0.500
Value 0.500
Mix
Image
Mix
Fac 1.000
Image
Image
Scale
Image
Relative
Image
X
Y 1.000
Mix
Image
Mix
Fac
Image
Image
View Select Add Node Use Nodes Free Unused Backdrop Auto Render

Figure 5: BGE

In Blender Game Engine, **logic bricks** were used as a visible programming interface. These logic bricks represented distinctive operations or behaviours, and you can join them in a visual way to define the flow of information and logic for your game. Each logic brick had inputs and outputs, and the connections between them determined the flow of data and control.

1. Sensor Brick (Input): Detects a key press.
2. Controller Brick (Processing): Performs a logical operation (e.g., AND, OR).
3. Actuator Brick (Output): Moves a character.

You could visually connect these logic bricks to create a flow where the character moves only when a specific key is pressed.

This visual approach made it easier for beginners to understand and implement game logic without delving into traditional programming. However, it had limitations, and for more complex games, developers often resorted to scripting in Python using the BGE API.

Blender provides a number of different editors for displaying and modifying different aspects of data.

- Text Editor: The Text Editor allowed users to write Python scripts for advanced game logic and functionality.
- Node Editor: The Node Editor is used to work with node-based workflows. - Refer Figure 6
- Logic Editor: The Logic Editor is used for creating game logic using a visual, node-based system. It allowed users to connect logic bricks to define the behaviour of game objects.

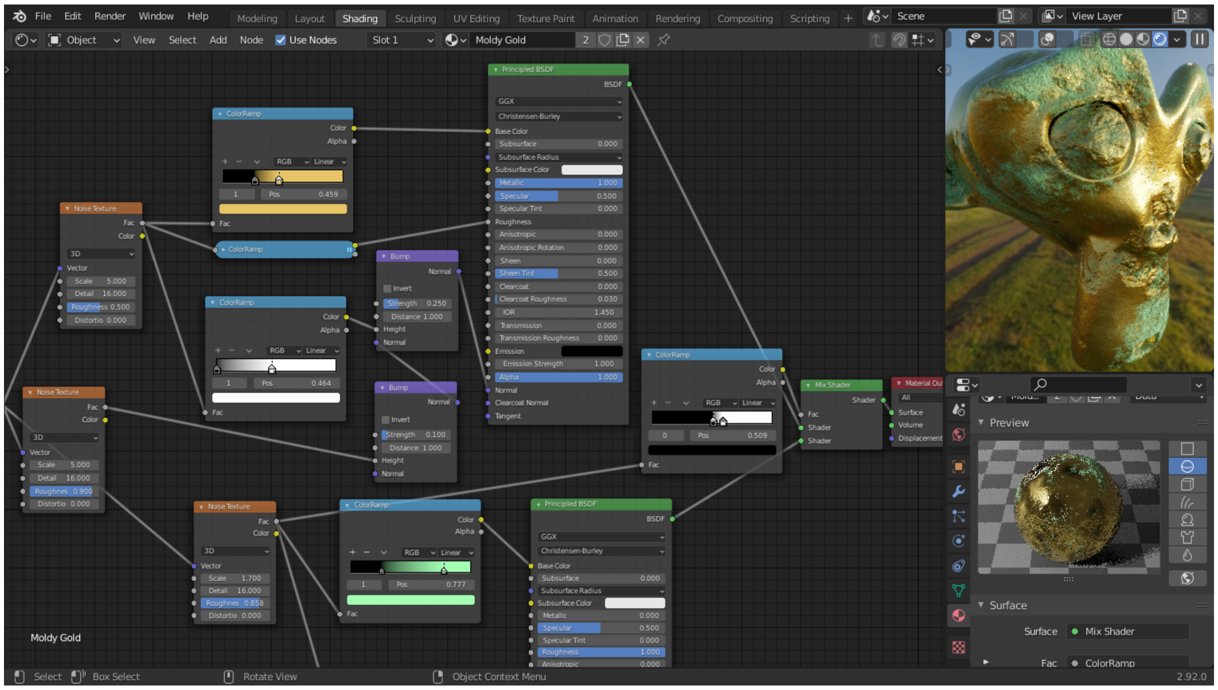


Figure 6: Node editor showcasing a Moldy Gold Material

Analysis

Features of Procedural Programming

- Top-Down Approach: The program is divided into smaller chunks that are executed in a sequential manner.
- Pre-defined Functions: Standardized instructions are frequently found within the programming language itself, typically in standard libraries that are integral to the language. These instructions serve specific purposes such as obtaining user input, presenting output on the screen, executing mathematical operations, and more. By utilizing these pre-defined functions, developers are able to automate various tasks without having to write the entire code from scratch.
- Local and Global Variable: Local variables are those that are declared within a local scope. These variables are not valid or accessible outside of the scope in which they are declared.
- A global variable is accessible throughout the entire program as it is declared outside of any functions or scopes defined within the program.

- Modularity: Modularity is the practice of using different functions and procedures simultaneously to accomplish a single, greater task. Each procedure performs a specific task that is different from other functions. But combining their results leads to accomplishing the greater task.
- Parameter Passing: Functions often work with data to give a certain result. We often supply such data input by passing them as parameters to a function. These parameters could be variables, values, memory addresses, etc.

Benefits of Procedural Programming

- Simplicity: Procedural programming is easy to understand and code due to its simple structure.
- Reusable Code: Functions can be reused within the program again and again. This makes coding easier. It also reduces the length of the code and the time needed to write the entire program.
- Smaller Memory Requirement: Procedural programming utilizes less memory than other paradigms. This increases the efficiency of our program.
- Easy Testability: The simplicity of procedural programming makes the code easy to test and debug. The shorter program length and the use of reusable code are great for quickly identifying sources of errors.
- General Programming: The procedural paradigm is great for general use and for smaller projects.

Limitations of Procedural Programming

- Unsuitable for Complexity: Procedural programming is unsuitable for creating large-scale and complex applications.
- Focused on Operations: Procedural programming focuses on functions and procedures more than data. This method is unsuitable for projects where data is important.
- No Data Security: Data is exposed and accessible to multiple procedures in procedural programming. Thus, it is unsuitable for projects where we must protect data.
- No Portability: With procedural programming, code can be reused within a single project. However, they cannot be exported to other projects. So, we need to rewrite a lot of code when working on other projects.
- Cannot Model the Real World: Since procedures are prioritized over data and objects, this paradigm cannot model the real world properly.

Utilization of Procedural Programming

- Procedural programming is well-suited for small to medium-sized projects.
- It is good, especially when the tasks are straightforward and need a clear, linear flow.
- It is often employed in implementing algorithms, mathematical computations, and tasks that require a series of well-defined steps to achieve the desired outcome.
- Create compilers, operating systems, kernels, etc.,
- Code server-side applications as a web developer.

Features of the Pascal Language

- Pascal is a strongly typed language.
- It offers extensive error checking.
- It offers several data types like arrays, records, files and sets.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.

How Pascal fits within Procedural Programming paradigm?

- Procedure and Function Structure: In Pascal, the primary building blocks are procedures and functions. Procedures contain a series of instructions that perform a specific task, while functions return a value after computation. Both procedures and functions in Pascal follow a structured approach, defining a sequence of steps to accomplish a particular task.

Example:

```
program ProceduresAndFunctions;

procedure SayHello;
begin
  writeln('Hello, there!');
end;

function AddNumbers(a, b: integer): integer;
begin
  AddNumbers := a + b;
end;

begin
  SayHello; // Calling a procedure
  writeln('Sum:', AddNumbers(3, 4)); // Calling a function
end.
```

This code defines a procedure *SayHello* that prints "Hello, there!" to the console. It also defines a function *AddNumbers* that takes two integers as input and returns their sum.

- Modularity: Procedural programming emphasizes breaking down a problem into smaller, manageable modules or procedures. Pascal supports modular programming through the use of procedures and functions, allowing developers to create reusable and independent blocks of code. This modular approach enhances readability, maintainability, and code reusability.

Example:

```
program ModularProgramming;

procedure ProcessData;
begin
  // Code to process data
end;

procedure ValidateData;
begin
  // Code to validate input data
end;

begin
  ValidateData; // Modular function call
  ProcessData; // Modular function call
end.
```

These procedures *ProcessData* and *ValidateData* are modular, focusing on specific tasks (processing data and validating input data, respectively). They can be called independently, promoting code modularity and reusability.

- Structured Programming Constructs: Pascal enforces structured programming principles, advocating for clear, understandable code through the use of structured constructs like if-else statements, loops (such as while, repeat, and for loops), case statements, and nested structures. These constructs enable developers to write organized, logical code that is easier to comprehend and maintain.

Example:

```
program StructuredProgramming;

var
  num: integer;

begin
  writeln('Enter a number: ');
  readln(num);

  // if-else statement
  if num > 0 then
    writeln('The number is positive')
  else if num < 0 then
    writeln('The number is negative')
  else
    writeln('The number is zero');
  // For loop
  writeln('Counting from 1 to 5:');
  for num := 1 to 5 do
    writeln(num);
end.
```

These snippets demonstrate structured constructs. The if-else statement checks conditions and executes corresponding code blocks, while the for loop iterates from 1 to 5, displaying each number.

- Sequential Execution: Procedural programming focuses on the step-by-step execution of instructions. Pascal, being a procedural language, executes instructions in a sequential manner, following the order in which they are written in the code. This sequential flow of control is fundamental to procedural programming.

Example:

```
program SequentialExecution;

begin
  writeln('Step 1');
  writeln('Step 2');
  writeln('Step 3');
  writeln('Step 4');
end.
```

The code displays messages in a sequential order, showcasing how Pascal executes instructions in the order they are written.

- Data Abstraction: Pascal supports data abstraction by allowing the creation of user-defined data types using records, arrays, enumerations, and sets. These abstractions enable programmers to encapsulate related data and operations within a single entity, enhancing code organization and readability.

Example:

```
program DataAbstraction;

type
  // Record for representing a point in 2D space
  Point = record
    x, y: integer;
  end;

var
```

```
  p: Point;

begin
  p.x := 5;
  p.y := 10;

  writeln('Point coordinates: (', p.x, ', ', p.y, ')');
end.
```

This code defines a $Point$ record abstracting the concept of a point in 2D space by encapsulating x and y coordinates within a single entity.

- Emphasis on Procedures: Pascal encourages the use of procedures for grouping related operations. Procedures can take parameters as inputs, perform specific tasks, and return control to the calling code. This promotes a clear division of functionality and helps in managing complex programs by breaking them into smaller, manageable procedures.

Example:

```
program EmphasisOnProcedures;

procedure CalculateArea(radius: real);
var
  area: real;
begin
  area := 3.14 * radius * radius;
  writeln('Area of the circle with radius ', radius, ' is: ', area);
end;

begin
  CalculateArea(5.0);
end.
```

The $CalculateArea$ procedure emphasizes the procedural approach by encapsulating the logic to compute the area of a circle based on the given radius.

- Lack of OOP Features: Unlike object-oriented programming (OOP) languages such as Java or Python, Pascal lacks inherent support for OOP concepts like classes, objects, inheritance, and polymorphism. Its focus remains on procedural abstraction and structured programming methodologies.

Real-world projects or applications that use Procedural Programming Paradigm in Pascal

- Text Editors and Word Processors: Some text editors and word processors were developed in Pascal. While modern applications in this category often use a mix of paradigms, early text processing applications were written using Pascal's procedural features.

- System Utilities: Pascal was used to develop system utilities and tools. Tools for file management, disk utilities, and system administration were commonly written in Pascal.
- Database Management Systems: Some early versions of database management systems (DBMS) were implemented in Pascal. Pascal's structured nature made it suitable for developing the backend logic of database systems.
- Industrial Automation: Pascal has been used in industrial automation and control systems. Control software for manufacturing processes, machinery, and industrial automation systems has been developed using Pascal's procedural programming capabilities.
- Utilities in Embedded Systems: Pascal has found applications in developing utilities for embedded systems, especially in the context of microcontroller programming. Pascal's procedural nature allows for efficient low-level programming.

Features of Dataflow Programming

- A program is defined by its nodes, I/O ports and links.
  - A **node** can have multiple inputs and outputs. - Refer Figure 7

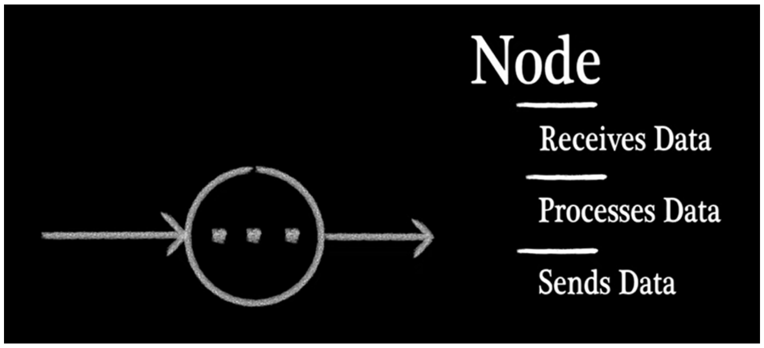


Figure 7: Node

  - Named **Ports**. - Refer Figure 8
  - **Links** connect nodes. Links can be viewed as buffers if they need to hold multiple data packets. - Refer Figure 9
- A **data packet** can take primitive/compound, structured/unstructured values, and even include other data packets.
- Execution can be either a **push or pull model**.
  - Push does eager evaluation and reacts to change. - Refer Figure 10
  - Pull does lazy evaluation and uses callbacks. - Refer Figure 11
- Execution can be **synchronous** or **asynchronous**. - Refer Figure 12
  - Synchronous execution: Each node take one "unit of time". Fast nodes will wait for slow nodes to complete before outputting the results.
  - Asynchronous execution: Outputs are never delayed and systems can run at full speed.

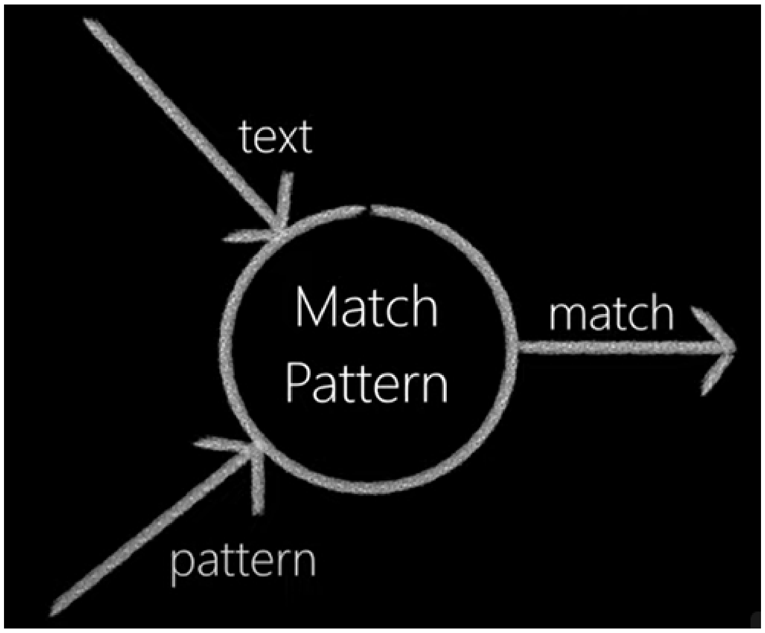


Figure 8: Port

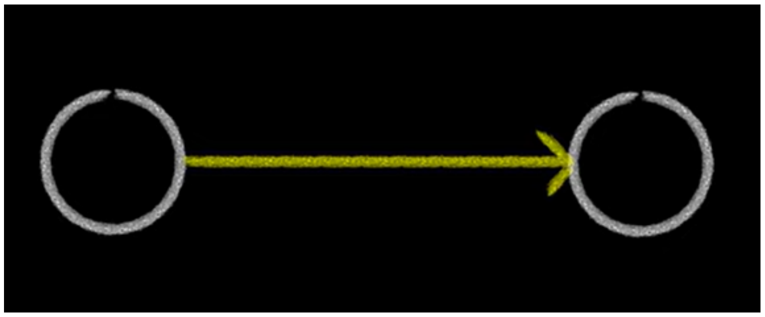


Figure 9: Links

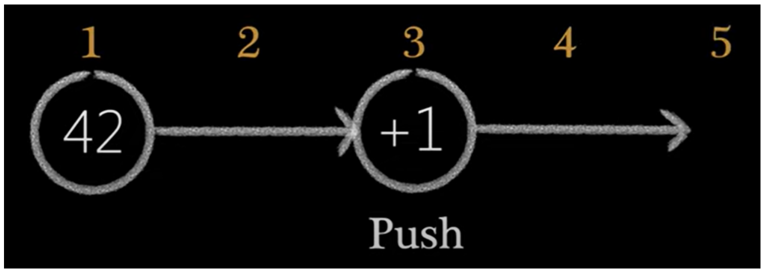


Figure 10: Push

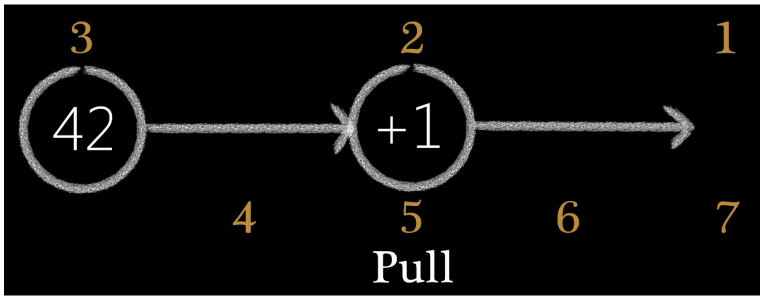


Figure 11: Pull

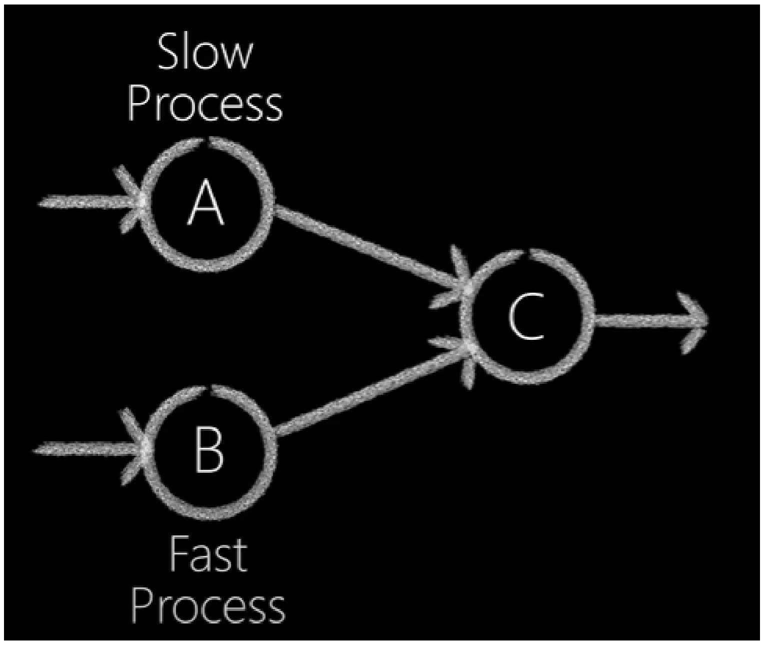


Figure 12: Synchronous and Asynchronous

Benefits of Dataflow Programming

- Parallel execution: Each node in the dataflow graph is independent of others and executes without side effects. A node executes as soon as its inputs are available. So, there's no need to deal with multiple threads, semaphores and deadlocks.
- Visual programming languages: DFP has enabled many visual programming languages that provide a more user-friendly interface so that even non-technical users can write programs.
- DFP is deterministic. This enables mathematical analysis and proofs.
- Loose coupling: DFP promotes loose coupling between software components that's well-suited for service-oriented architectures. Its functional style makes it easier to reason about system behaviour.

Limitations of Dataflow Programming

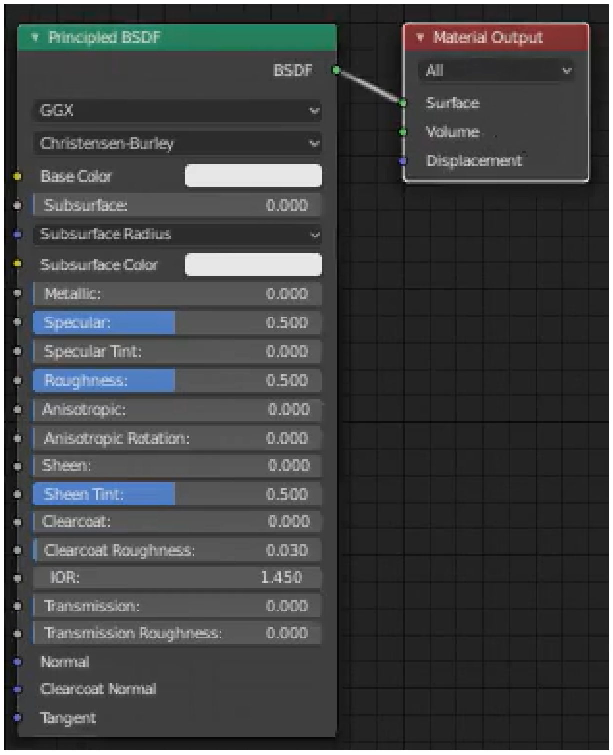
- Visual dataflow languages can be cumbersome for large programs with many nodes.
- Iterations and conditions are hard to represent visually.
- Debugging is difficult since multiple operations happen concurrently.
- DFP's deterministic nature can be a problem. Some use cases (such as ticket booking or database access) are inherently non-deterministic. To solve this, DFP must allow for non-deterministic merging of multiple input streams.

Utilization of Dataflow Programming

- Dataflow programming is commonly used in applications that involve real-time data processing, such as streaming analytics, real-time monitoring, and sensor data processing.
- In graphical applications, such as image processing or computer graphics, dataflow programming is well-suited for representing and managing complex computations with a visual structure.
- The dataflow paradigm is well-suited for distributed systems, where nodes can be distributed across multiple machines or nodes in a cloud environment.
- Dataflow programming is used in machine learning frameworks for defining and executing complex computation graphs.

How Blender Game Engine fits within Dataflow Programming paradigm?

- Node-Based System: The logic brick system in BGE can be considered a form of node-based programming, which is a common aspect of dataflow programming paradigms. - Refer Figure 13
- Logic Bricks: BGE's logic brick system can be seen as a form of visual programming that represents a simplified version of dataflow.
- Visual Programming: Users could create game logic by visually connecting these logic bricks in the Logic Editor. Each logic brick represented a specific function or condition, and connecting them determined the flow of data and control within the game.



GGX
Christensen-Burley
Subsurface: 0.000
Metallic: 0.000
Specular: 0.500
Specular Tint: 0.000
Roughness: 0.500
Anisotropic: 0.000
Anisotropic Rotation: 0.000
Sheen: 0.000
Sheen Tint: 0.500
Clearcoat: 0.000
Clearcoat Roughness: 0.030
IOR: 1.450
Transmission: 0.000
Transmission Roughness: 0.000
All

Figure 13: Default Nodes in BGE

- Event-Driven Model: BGE's logic was often event-driven, where sensors detected events. This event-driven model aligns with the principles of dataflow programming, where actions are triggered by the flow of data through the system.
- Parallelism and Asynchronous Execution: In a dataflow programming paradigm, nodes execute concurrently as data becomes available. The nature of event-driven programming allowed for asynchronous execution of logic based on different events occurring in the game.

Real-world projects or applications that use Dataflow Programming Paradigms in the Blender Game Engine

- Architectural Visualization: Some projects in architectural visualization used BGE for creating interactive walkthroughs and virtual tours. Dataflow programming in BGE facilitated the implementation of user interactions and animations within architectural models.
- Simulations and Training: BGE was employed for developing simulations and training applications. The visual dataflow approach made it accessible for developers to model and simulate different scenarios for training purposes.
- 3D Animated Stories and Comics: BGE was utilized to create interactive 3D animated stories and comics. Dataflow programming in BGE enabled the implementation of interactive storytelling elements and character animations.
- Virtual Reality (VR) Experiences: Some projects explored the use of BGE for creating simple virtual reality experiences. Dataflow programming facilitated the integration of user interactions and scene changes in VR environments.

Property	**Procedural**	**Dataflow**
Abstraction	Procedural programming focuses on organizing code into procedures and functions.	Dataflow programming emphasizes the flow of data and the operations performed on that data.
Modularity	Modularity is achieved through the use of functions and procedures.	Programs are often naturally modular due to the nature of nodes operating on data.
Reusability	Emphasizes reusable procedures or functions.	Emphasizes reusable data-driven components.
Expressiveness	Strong control over the sequence of operations.	Well-suited for parallel and asynchronous operations.
Ease of Debugging	Debugging is often more straightforward as execution follows a clear sequence.	Debugging can be challenging due to the implicit nature of execution.

Table 1: Differences between Procedural and Dataflow

- Artistic and Experimental Games: Artists and game developers used BGE for creating artistic and experimental games. The visual dataflow approach allowed for unconventional and creative implementations in the game design.

Comparison

Similarities between Procedural and Dataflow

- Both dataflow programming and procedural programming are forms of imperative programming, where the focus is on defining a series of steps to accomplish a particular goal.
- Both paradigms involve the transformation of data.
- Both paradigms support encapsulation of logic.
- Both paradigms encourage modularity.
- Both paradigms can be designed to be stateful.

Differences between Procedural and Dataflow

- Refer Table 1

Property	Pascal	BGE
Variables	Variables in Pascal are declared using explicit data types.	Variables in BGE are often associated with game objects and can be managed using logic bricks.
Memory Management	Pascal is generally compiled and has static memory management.	BGE manages memory dynamically.
Function Calls	Function calls are explicit and follow the declaration. - Refer Figure 14	BGE provides a visual scripting system called logic bricks, and functions are represented by nodes. - Refer Figure 15
Object-Oriented Programming	Pascal has limited support for object-oriented programming (OOP)	BGE supports object-oriented programming (OOP) concepts through Python scripting.
Application	Pascal is a general-purpose programming language .	BGE is specifically designed for real-time game development within the Blender environment.

Table 2: Differences between Pascal and BGE

Similarities between Pascal and BGE

- Both environments involve the storage and usage of variables. In Pascal, variables are declared with specific data types and can be used for various operations. In BGE, variables can be used within Python scripts to control game logic and behavior.
- Both environments support conditional statements (such as if-else) and loop structures (such as for and while).
- Both Pascal and BGE support mathematical operations.
- Both environments involve event handling. In Pascal, events can be managed through procedures or functions and in BGE, events can be triggered by user input or changes in the game state.
- Both Pascal and BGE allow programmers to define and control the flow of their programs.

Differences between Pascal and Blender Game Engine

- Refer Table 2

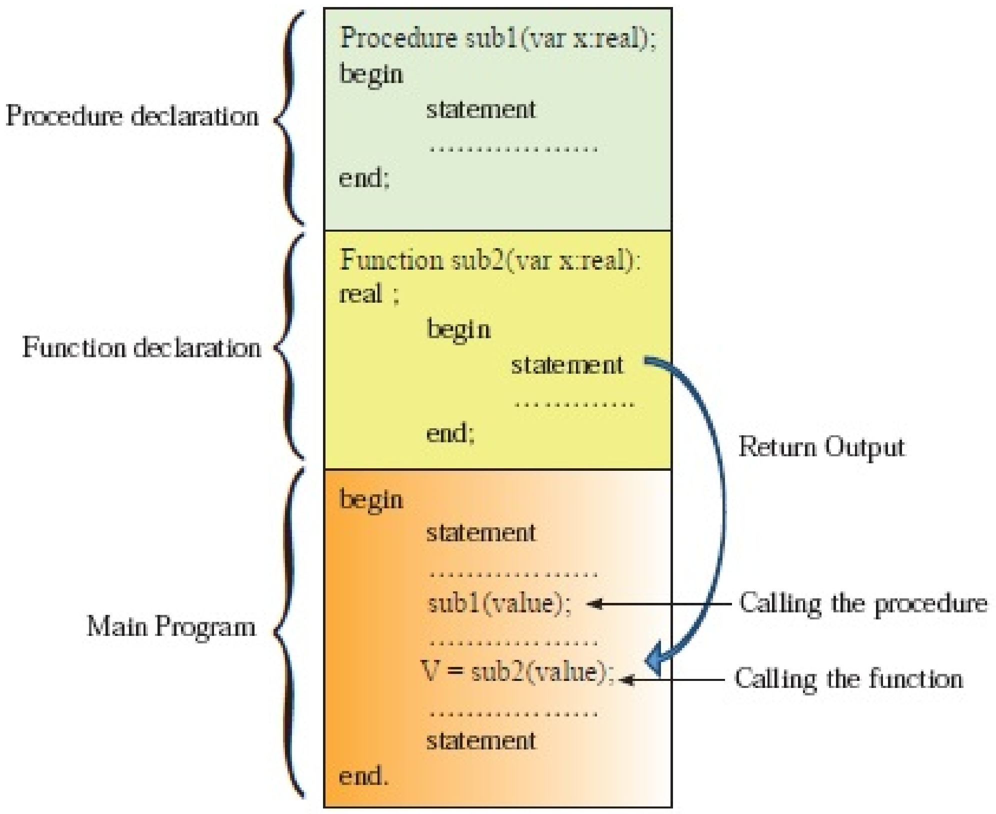
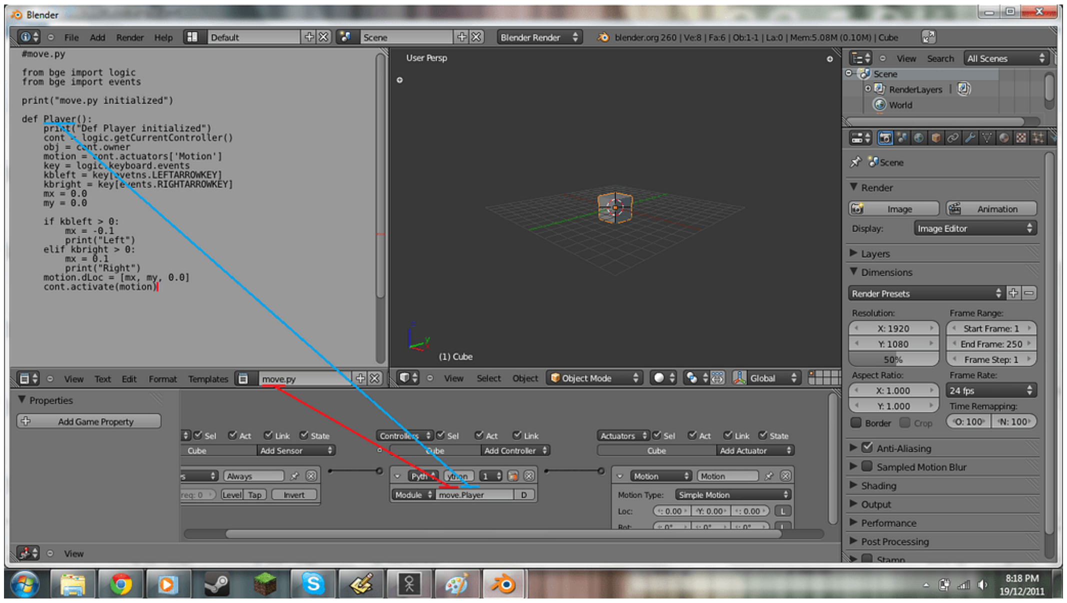


Figure 14: Function Call in Pascal



```
#move.py

from bge import logic
from bge import events

print("move.py initialized")

def Player():
    print("Def Player initialized")
    cont = logic.getCurrentController()
    obj = cont.owner
    motion = cont.actuators['Motion']
    key = logic.keyboard.events
    kbleft = key[evetns.LEFTARROWKEY]
    kbright = key[events.RIGHTARROWKEY]
    mx = 0.0
    my = 0.0

    if kbleft > 0:
        mx = -0.1
        print("Left")
    elif kbright > 0:
        mx = 0.1
        print("Right")
    motion.dLoc = [mx, my, 0.0]
    cont.activate(motion)
```

Figure 15: Function Call in BGE

Challenges Faced

Lack of resources for Dataflow - Blender Game Engine. Managed to understand few things with the prior knowledge that I had on Blender Software.

The Blender game engine (BGE) was removed from Blender 2.8. Referred to the old Documentation.

Theoretical knowledge was not sufficient to explain everything in detail. Practical experience is essential for mastering a programming paradigm.

Conclusion

In conclusion, this assignment has deepened my understanding of various programming paradigms, their characteristics, and the challenges involved in their exploration. It has set the foundation for further exploration and the application of these paradigms in real-world scenarios.

References

https://devopedia.org/dataflow-programmingqst-ans-6
https://www.youtube.com/watch?v=yKKy0vJ0CrY
https://docs.blender.org/manual/en/latest/
https://blenderartists.org/t/design-for-a-new-graphical-game-engine/455219
https://en.wikipedia.org/wiki/Procedural$_{p}rogramming$
https://programiz.pro/resources/what-is-procedural-programming/
https://www.geeksforgeeks.org/introduction-of-programming-paradigms/