CA341 Comparative Programming Languages

Assignment 1 - Comparing Procedural and Object-Oriented Programming

CASE3

Student Name: David Weir

Student Number: 19433086

13/11/2021

Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): David Weir

Programme: Comparative Programming Languages

Module Code: CA341

Assignment Title: Comparing Procedural and Object-Oriented Programming

Submission Date: 14/11/2021

Module Coordinator: Dr Brian Davis

Name(s): David Weir   Date: 14/11/2021

Comparative

I chose to implement my phonebook binary search trees using Python as my OOP language and GoLand (Go) as the procedural language. I decided to use Python as my choice of OOP language due to my familiarity with the language gained through my time studying in DCU. Most notably I gained knowledge on OOP and binary trees in CA268 in 2nd year.

I chose GoLang as my procedural language to provide an extra challenge in learning a new language while doing this assignment. I chose Go over C as they both have similar syntax style. While C is more efficient it is also a more difficult language to develop in, in my opinion. However, Go provides most of the same efficiency while being much easier to write a program in.

Procedural

Procedural programming is a programming paradigm based around sequences of instructions that are to be executed. It focuses on splitting up programs into procedures, a set of instructions executed by the program. A procedures data is stored locally and cannot be accessed from outside the procedure. There is minimum abstraction between the code and machine.

As said above procedures are blocks of instructions that can be called elsewhere in the program, that accepts arguments and returns a value to the caller. Procedural languages also follow structured programming practices and uses block-based control flow.

Object-Oriented Programming

OOP or Object Oriented Programming is a computer programming model / paradigm that organizes its code around objects instead of functions and logic. It is the most popular programming paradigm focused on classes and objects, it structures the program into easy to read, reusable pieces of code called classes. These classes contain methods that are localized to objects of that type. There are many positives to OOP which made programming in a OOP language easier including re-usability, code maintenance, easy reading and troubleshooting.

Comparing

My OO program begins by creating a Node class. This class stores the name, phone and address of the contact in the phonebook and a left and right child node which allows us to access the subtrees. The name and phone trees discussed later will be used to store these contact nodes. As we can see below the node collects the data in one area that can be reused as we build our trees.

```python
class Node:
    def __init__(self, name, phone, address):
        self.name = name
        self.phone = phone
        self.addr = address
        self.left = None
        self.right = None
```

```python
class NameTree:
    def __init__(self):
        self.root = None
```

```python
class PhoneTree:
    def __init__(self):
        self.root = None
```

We then create a tree to hold these nodes. As we see above we create a name and phone tree and we make our first method where we initialize the tree with an empty root node.
This is similar to how we use struct types in Go to group the same data together in fields. As we see below, we create a Node struct type, again storing the contacts phone, name and address with the left and right child nodes.

We also create a Tree struct type which stores a node from the Node struct storing it as the tree's root.

```go
type Tree struct {
    Root *Node
}

// Node structure with name phone and address strings and Left and Right child
nodes
type Node struct {
    Name    string
    Phone   string
    Address string
    Left    *Node
    Right   *Node
}
```

There are no major differences here between the procedural and OOP implementation instead we see how Go can support some of the ideas of Object-Oriented Programming.

Next, we look at how both paradigms deal with inserting, search and deletion. In OOP this is done with methods from within the tree classes while in procedural programming Go uses procedures.

To compare we will have a look at insertion. In Python, as mentioned above we use methods from within the tree.

As we can see the methods accept self, which is a keyword representing the instance of the tree class. Using it we can access the other attributes and methods of that tree class. It also accepts the name, phone and address of the contact. The insert method checks if the tree is empty, creating a node if it is and passing on the data if it is non-empty.

If it is non-empty, "binary_insert" finds where the node should be placed and inserts the node in the correct position.

```python
def insert(self, name, phone, address):
    if self.root is None:
        self.root = Node(name, phone, address)
    else:
        self.binary_insert(self.root, name, phone, address)

def binary_insert(self, curr, name, phone, address):
    if name > curr.name:
        if curr.right is None:
            curr.right = Node(name, phone, address)
        else:
            self.binary_insert(curr.right, name, phone, address)

    else:
        if curr.left is None:
            curr.left = Node(name, phone, address)
        else:
            self.binary_insert(curr.left, name, phone, address)
```

As we see below, Go and Python are both procedural, Python having common related pieces of placed inside functions, methods and classes, while Go has its procedures that act as functions.

This is emphasized as we look at the insert command in Go.

```
// insert a node to the tree using the name
func (t *Tree) insertName(name, phone string, address string) error {

    if t.Root == nil { // if the tree is empty create a new node
        t.Root = &Node{Name: name, Phone: phone, Address: address}
        return nil
    }

    // else insert a node
    return t.Root.insertName(name, phone, address)
}
func (n *Node) insertName(name, phone string, address string) error {
    // Function to insert node based on name

    // if it is empty we cannot add it
    if n == nil {
        return errors.New("Cannot insert an entry into a nil tree")
    }

    switch {

    // if the node exists already
    case name == n.Name:
        return nil

    // if the name is less than the current node and the left child is empty
insert a node on the left
```

```
    // else call insert on the left subtree
    case name < n.Name:

        if n.Left == nil {
            n.Left = &Node{Name: name, Phone: phone, Address: address}
            return nil
        }

        return n.Left.insertName(name, phone, address)

    // if the name is greater than the current node and the right child is empty
insert a node on the right
    // else call insert on the right subtree
    case name > n.Name:

        if n.Right == nil {
            n.Right = &Node{Name: name, Phone: phone, Address: address}
            return nil
        }

        return n.Right.insertName(name, phone, address)
    }

    return nil
}
```

As we can see it follows the same logic as the insert implementation in Python. Operating almost the exact same, the only differences being syntactical and the lack of classes in Go. In the procedural implementation instead use procedures. As we compare further we start to see the both implementations' code are seemingly identical it is only the use of classes versus structs.

Through this comparison we can see the major differences of the two implementations. In OOP classes, all parts of the program are divided into classes, with methods to group related code together similar to functions. A clear benefit we can see of this is of course making it easier to read and understand the program. It also allows us to call these methods and initialize these classes anywhere in the program

The benefit of this is obvious when we see the GoLang implementation of the phonebook. While it is divided into procedures and it's structs provide similar functionality to OOP classes, the divide is less obvious. Despite this Go's procedures and structs and Python's classes and methods are similar and perform almost the same roles in their respective languages.

OOP allows for the use of abstract classes (one of the "pillars of OOP"). It is used to hide classes to reduce complexity and improve efficiency.

GoLang offers abstraction as well at a higher order level, however, it only uses certain abstractions. You can use Go's interfaces to create common abstractions used by multiple type structs.

Conclusion

In conclusion, while both paradigms and languages have pros and cons, they also have many similarities. Personally, I found implementing this program in Go easier than in Python. I had specific problems implementing deletion in Python when the node to be deleted had 2 child nodes.

References

[1] Anon, Major Programming Paradigm, https://www.cs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html#object, Accessed 11/11/21

[2] Robert W. Sebesta. Concepts of programming languages. Eleventh Edition, Global Edition. Accessed 13/11/21

[3] Dr. David Sinclair. Object-Oriented Programming Paradigm. https://loop.dcu.ie/pluginfile.php/4159679/mod_resource/content/2/CA341_Object-Oriented_Programming_Paradigm.pdf . Accessed 14/11/21.

[4] Dhanvani, P. (2012) Difference between Procedure Oriented Programming and Object Oriented Programming | Procedure Oriented Programming vs. Object Oriented Programming. Available at: http://freefeast.info/general-it-articles/difference-between-procedure-oriented-programming-and-object-oriented-programming-procedure-oriented-programming-vs-object-oriented-programming Accessed 14/11/21.

[5] Adhikari, B. Object Oriented Programming Vs Procedural Programming. https://www.researchgate.net/publication/311587459_Object_Oriented_Programming_Vs_Procedural_Programming . Accessed 14/11/21