# CA341 Comparative Languages
# Comparing Procedural and Object-Oriented Programming

---

Joao Pereira
joao.pereira2@mail.dcu.ie
19354106

Name(s): Joao Pereira

Programme: Computer Applications and Software Engineering

Module Code: ca341

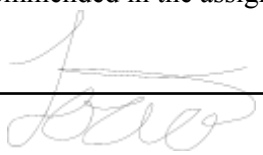
Assignment Title: Comparing Procedural and Object-Oriented Programming

Submission Date: 12/11/2021

Module Coordinator: Dr Brian Davis

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name(s):  Date: 15/11/2021

# Table of Contents



# Introduction - Choice of languages

## Procedural

For Imperative programming, I chose to pick C as my language of choice. I chose C due to the fact it is one of the languages I am most familiar and comfortable with. C is also a very strong procedural language in terms of speed, memory and overall performance and due to this fact it is a solid choice when it came to deciding what language to select.

## Object-Oriented Language

For Object-Oriented Programming, I selected Python as my language of choice. Python is by far my favourite programming language. I am most comfortable with Python and how it operates. It is quick, easy and highly effective in terms of the assignment outline. Another note to add is I do not have knowledge of any other Object-Oriented programming languages except minimal experience with Java. Therefore Python was realistically my only option.

# Comparative Analysis

## Procedural Code Analysis

Procedural languages are programming languages that use functions that then can be called back at any point throughout the code to perform the task it is intended for.

I began by creating a TreeNode struct to assign data types to variables that would be necessary to be used for further on.

```
typedef struct TreeNode {
    struct TreeNode *left;
    struct TreeNode *right;
    char *name;
    char *phone;
    char *address;
} TreeNode;
```

Once the TreeNode struct was created, I created multiple functions that pointed back to the TreeNode struct. Each function has pointers with assigned variables. The rest of the BST tree is then implemented with all the functions necessary.

```
TreeNode *newNode(char *title, char *mobile, char *dir){

    TreeNode *new_node = (TreeNode *)malloc(sizeof(TreeNode));
    new_node->left = NULL;
    new_node->right = NULL;
    new_node->name = title;
    new_node->phone = mobile;
    new_node->address = dir;
    return new_node;

}
// Insert function to insert name into BST. With aid of https://www.log2base
TreeNode* insertName(TreeNode* root, char *title, char *mobile, char *dir){
    if(root == NULL)
        return newNode(title, mobile, dir);
    else if(root->name < title)
        root->right = insertName(root->right, title, mobile, dir);
    else if(root->name > title)
        root->left = insertName(root->left, title, mobile, dir);
    return root;
}
```

Finally, the main function is developed in order to display a working PhoneBook. A list of contacts is displayed in the terminal along with the operation of each function and its error cases.

```
int main(){
    TreeNode *root;

    printf("--------Welcome to the phonebook--------\nHere is a list of co

    root = insertName(root, "Leighton Adkins", "0868122244", "Dublin 11");
    root = insertName(root, "Umair Lake", "0869861452", "Dublin 24");
    root = insertName(root, "Heath Mair", "0883418472", "Dublin 4");
    root = insertName(root, "Sofia Stark", "0814672830", "Dublin 1");
    root = insertName(root, "Jean-Luc Wiggins", "0859786732", "Dublin 3");
    root = insertName(root, "Austen Daugherty", "0833336666", "Dublin 2");
    root = insertName(root, "Romario Cox", "0812345678", "Dublin 14");
    root = insertName(root, "Jolyon Sinclair", "0884628422", "Dublin 11");
    root = insertName(root, "Ajay Chamberlain", "0872548194", "Dublin 9");
    root = insertName(root, "Cathy Hood", "0874528573", "Dublin 8");
    root = insertName(root, "Lyra Herrera", "0845389502", "Dublin 6");
    root = insertName(root, "Guy Smart", "0875492641", "Dublin 2");
```

Disclaimer: due to lack of time throughout the past days, it was impossible to find time to create a full interface with user commands so instead I displayed a phonebook with the functions showing that they fully operate

# Object-Oriented Code Analysis

To start off, an Object-Orientated (OO) language is a programming language that consists of classes, methods, objects and many more for the purpose to structure code and be able to call back to classes and objects in order to compute a solution. I started off in Python by assigning three classes. A Node class, a Name class and a Phone class.

Node class:

Name class:

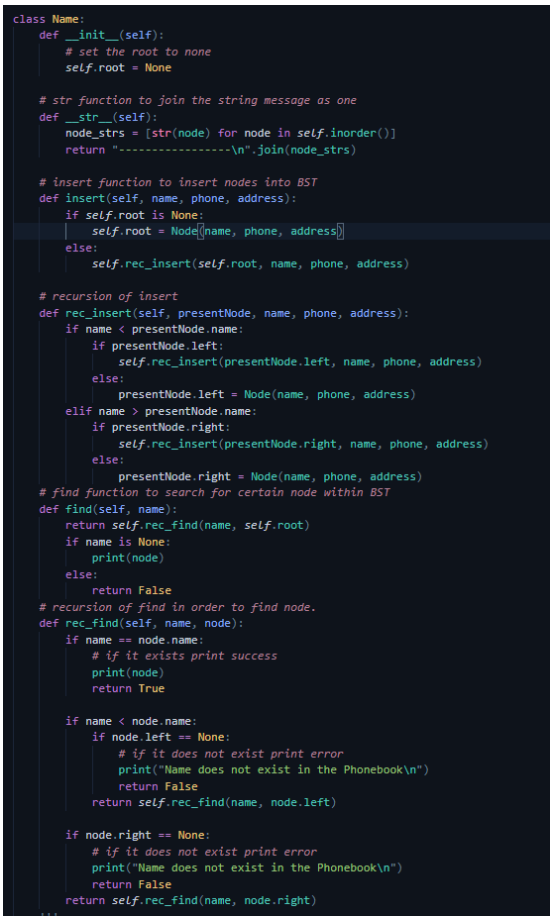
```
class Node:
    # intialize values to left, right, name, p
    def __init__(self, name, phone, address):
        self.left = None
        self.right = None
        self.name = name
        self.phone = phone

class Name:
    def __init__(self):
        # set the root to none
        self.root = None
```

Phone class:

```
class Phone:
    def __init__(self):
        # set the root to empty
        self.root = None
```

With the use of these three classes, I then performed various functions in each to be able to compute the results I was searching for. For example, for this assignment, it was outlined to ‘Implement a phonebook program that uses a binary tree to store, remove and search entries’ and to “Use 2 binary trees, one that uses name for insertion and searching and the other that uses the number for insertion

and searching". Therefore I developed an insert and a search function for both Name and Phone classes and further developed more functions such as a string function and a remove function in each class in aid to be able to perform the required outputs for both outputs. An example of Name class which is a copy of the phone class, except they use different variables(name, phone):



```
class Name:
    def __init__(self):
        # set the root to none
        self.root = None

    # str function to join the string message as one
    def __str__(self):
        node_strs = [str(node) for node in self.inorder()]
        return "----------------\n".join(node_strs)

    # insert function to insert nodes into BST
    def insert(self, name, phone, address):
        if self.root is None:
            self.root = Node(name, phone, address)
        else:
            self.rec_insert(self.root, name, phone, address)

    # recursion of insert
    def rec_insert(self, presentNode, name, phone, address):
        if name < presentNode.name:
            if presentNode.left:
                self.rec_insert(presentNode.left, name, phone, address)
            else:
                presentNode.left = Node(name, phone, address)
        elif name > presentNode.name:
            if presentNode.right:
                self.rec_insert(presentNode.right, name, phone, address)
            else:
                presentNode.right = Node(name, phone, address)
    # find function to search for certain node within BST
    def find(self, name):
        return self.rec_find(name, self.root)
        if name is None:
            print(node)
        else:
            return False
    # recursion of find in order to find node.
    def rec_find(self, name, node):
        if name == node.name:
            # if it exists print success
            print(node)
            return True

        if name < node.name:
            if node.left == None:
                # if it does not exist print error
                print("Name does not exist in the Phonebook\n")
                return False
            return self.rec_find(name, node.left)

        if node.right == None:
            # if it does not exist print error
            print("Name does not exist in the Phonebook\n")
            return False
        return self.rec_find(name, node.right)
```

As we can see a class Name is assigned. Within this name class, the functions __init__, __str__, insert, rec_insert, find, rec_find if we to scroll further down we would notice a remove and an inorder function with a recursion inorder call. were

This exact code and format is copied onto the Phone class. Each function is purposed to do it's name identifies as. For example the find function searches the BST to check if the value exists and then prints either a success or failure depending on the input. I utilised recursion as a method to perform the tasks like use of recursion is something I am comfortable with using. what the

To finalise this procedural code analysis I implemented a main function to display the phonebook and all the functions created in operation.

Disclaimer: due to lack of time throughout the days, it was impossible to find time to create a full interface with user commands so instead I displayed a phonebook with the functions showing that they fully operate. past

## Procedural vs Object-Oriented

Both languages have their pros and cons when it comes to comparing them and their performance. To start off C uses structs and Python uses classes/objects. The main difference between both is that classes contain data and behaviour and structs are made up of data where they need to accept parameters built from the structure and in this case they were accepted through pointers.

## Memory Management

Python manages all of its memory through what is called "Blocks"(https://towardsdatascience.com/memory-management-in-python-6bea0c8aecc9#:~:text=The%20Python%20memory%20manager%20manages%20chunks%20of%20memory%20called%20%E2

%80%9CBlocks,object%20of%20the%20same%20size.). All the memory is controlled by a private heap which stores the data structures and objects.

On the other hand, in C, memory is managed and allocated through a library function named “malloc” which accesses memory through pointers (https://eleceng.dit.ie/frank/IntroToC/Memory.html).

Basically, a pointer contains an address to the memory. The address is formed by assigning & to a variable and * to where you want the memory to point to.

```
TreeNode *newNode(char *title, char *mobile, char *dir){

    TreeNode *new_node = (TreeNode *)malloc(sizeof(TreeNode));
```

To summarize memory management in both have their advantages and disadvantages. Python has the huge disadvantage of lacking efficiency in managing memory due to using a large amount of memory (blocks) which can then slow down the performance of creating and developing programs whilst C outperforms immensely. The fact C uses pointers for memory and the fact you can remove/free memory that is not needed to be performed in C is a massive benefit when it comes to memory performance. Overall C is much superior to python with memory.

*Disclaimer: Ran out of time to discuss topics on overall performance, data types and inheritance comparison between both languages. Each respectively was planned to be discussed however completion in time was unsuccessful.*

# References


- https://towardsdatascience.com/memory-management-in-python-6bea0c8aecc9#:~:text=The%20Python%20memory%20manager%20manages%20chunks%20of%20memory%20called%20%E2%80%9CBlocks,object%20of%20the%20same%20size.
- https://eleceng.dit.ie/frank/IntroToC/Memory.html
- https://www.youtube.com/watch?v=Vs8aN5wSyxA
- https://freecontent.manning.com/memory-efficiency-and-space/
- https://www.log2base2.com/data-structures/tree/insert-a-node-in-binary-search-tree.html
- https://www.programiz.com/dsa/binary-search-tree
- https://www.codesdope.com/blog/article/binary-search-tree-in-c/
- https://stackoverflow.com/questions/27285565/c-warning-incompatible-pointer-types-passing
- https://stackoverflow.com/questions/33715301/inserting-nodes-in-a-binary-search-tree-c/33715699
- https://www.youtube.com/watch?v=LSju119w8BE
- https://www.tutorialspoint.com/python_data_structure/python_tree_traversal_algorithms.htm


*Disclaimer: Ran out of time to construct reference links at a standard level of citation.*