Comparing Procedural vs

Object Oriented Programming

**Natasha McCarthy & Jon Bryan Ortiz**

Student ID: 20335841 & 20319236

CA341 Comparative Programming Languages

06.11.2022

# 

# INTRODUCTION

For our implementation of a phonebook with a binary tree, we use Python and C implementation to analyse the differentiation of how these languages program the same problem.

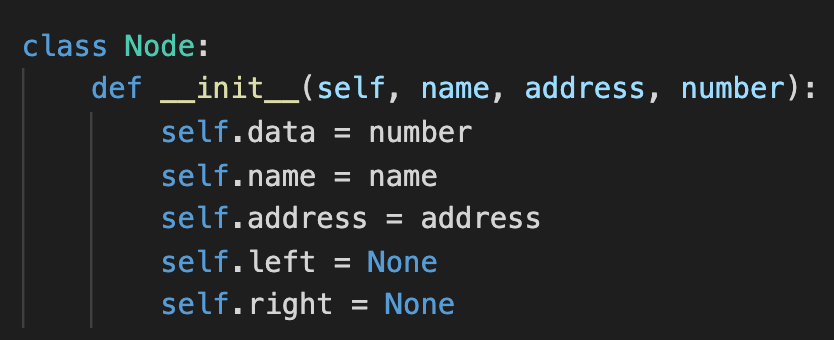
The reason why we chose Python as our Object Oriented programming language is that it is an easier programming language to understand and efficiently run. We wanted to implement C as our procedural language as we can see the way that the code maps the attributes to each node of a tree in comparison to Python. In this assignment, we will be comparing the implementation of both programs under the topics discussed.

# IMPLEMENTATION ANALYSIS:

When we implemented the program in both languages, we decided to read contact information from a text file in order to compare how both languages can read and handle extracting data from an external database source.

## OBJECT-ORIENTED:

For the object-oriented implementation of the phonebook including a binary tree, first, we must create a Node that can be inserted into the tree. To complete this, the Node class was implemented. The attributes of the class included the name, address, and phone number of the contact as well as the left and right child nodes (shown in Figure 1.1). The tree is sorted by the phone number and if we need the name and address which is linked, for instance, the search function, this data can be extracted from the node. The next step was to implement the requirements of the phonebook which are: to search for a contact, add a new contact to the phonebook and delete a contact.

Figure 1.1 - Node class 

* Search(): Given a number as input search for the corresponding name and address linked to the phone number. If the search is not the root, the search the function will go through all nodes within the BST until found.
* New(): To add a new contact, the user will input details which then be inserted into the BST to the number in order. Not only will it be imported to the BST, but it will also be inserted into the .txt file with all of the other contact information.
* Delete(): Deleted the node from the BST. For this function, we need to assign a minimum and temporary value, so that if Node has a left and right child it can replace the value and remove the minimum from the subtree (shown in Figure 1.2)

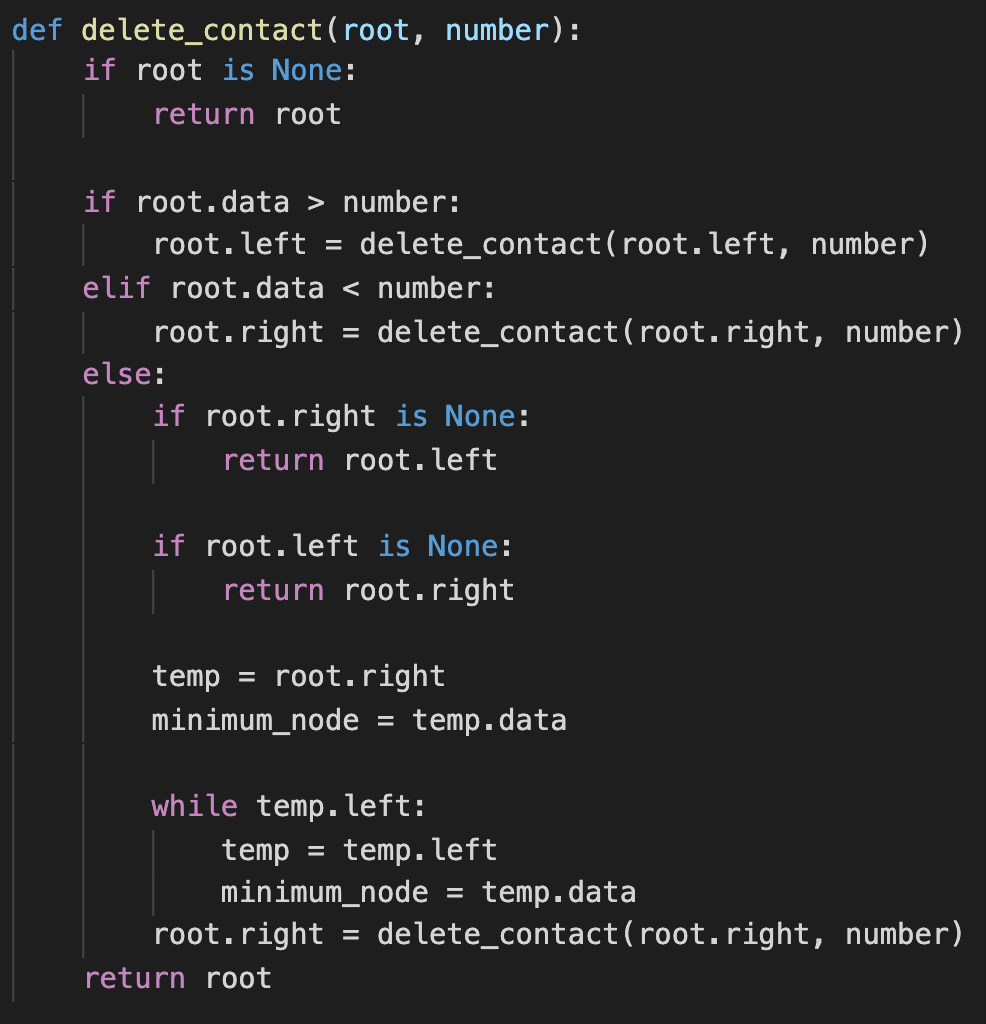
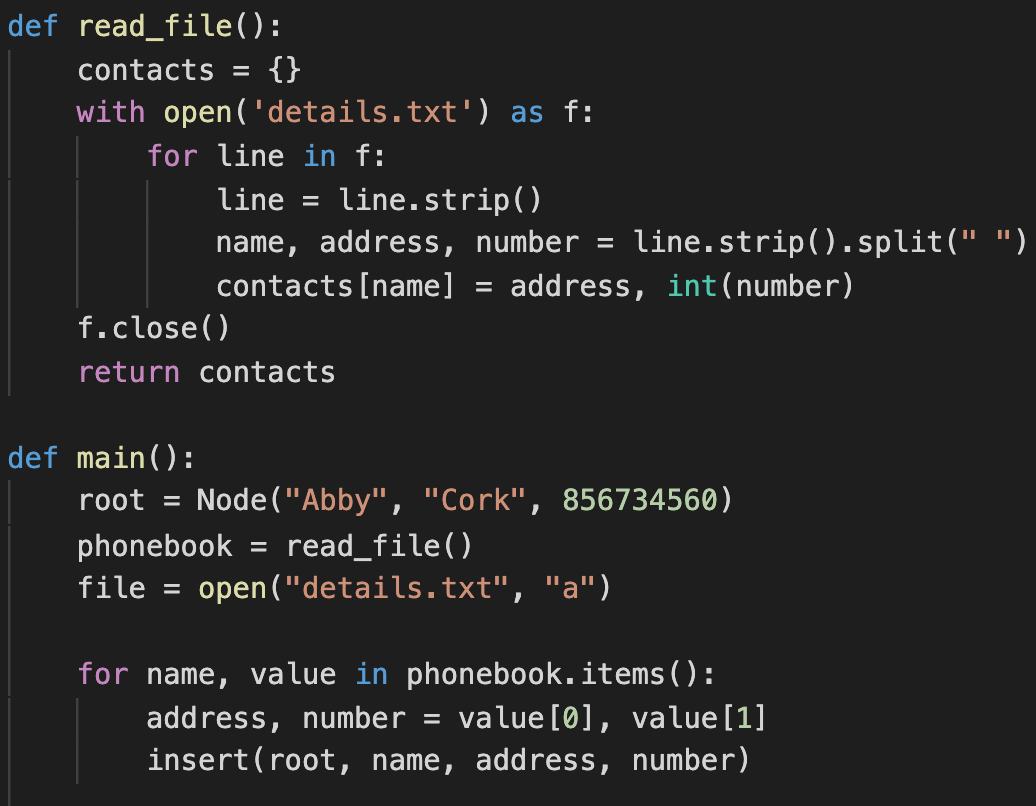


Figure 1.2 - Delete contact function

Once these functions were implemented, the file was read through and data was extracted into a dictionary so we could insert all information together into the binary tree, shown in Figure 1.3. When making the command line, 5 commands were created which are “search”, “add”, “delete”, and “print” used when the user would like to look at all contacts in the phonebook, and “quit”. The contacts are printed in an inorder traversal, which can be seen in figure 1.4. Command line was interpreted by using a while statement that would only break if the user enters “quit”.



## 

## 

## 

## 

## 

## Figure 1.3 - Read file Figure 1.4 - Print Inorder Transversal

## 

## PROCEDURAL:

To implement the phone book using a BST (Binary Search Tree), I first had to look at what I would need to create nodes. In the implementation, structs are used to represent node entries. A struct/structure in C is a composite data type that has a collection of variables in a single group. I would then look at what variables I need within the node struct. These variables include the contactName, contactAddress, and phoneNumber.

| struct node {  char contactName[20];  char contactAddress[20];  char phoneNumber[20];  struct node \*leftChild, \*rightChild; }; |
| --- |

Next step is implementing the basic functions required which are:

* newNode(): Creating a new node based on the details passed to the struct function.

| struct node\* newNode(char \*newNumber, char \*newName, char \*newAddress) {  struct node\* temp = (struct node\*)malloc(sizeof(struct node));  strncpy(temp->phoneNumber, newNumber, 20);  strncpy(temp->contactAddress, newAddress, 20);  strncpy(temp->contactName, newName, 20);  temp->leftChild = temp->rightChild = NULL;  return temp; } |
| --- |

* insert(): Finding a node’s position and inserting it into the BST

| struct node\* insert(struct node\* node, char \*newNumber, char \*newName, char \*newAddress) {  if (node == NULL)  {  return newNode(newNumber, newName, newAddress);  }  if (atoi(newNumber) < atoi(node->phoneNumber))  {  node->leftChild = insert(node->leftChild, newNumber, newName, newAddress);  }  else if (atoi(newNumber) > atoi(node->phoneNumber))  {  node->rightChild = insert(node->rightChild, newNumber, newName, newAddress);  }  return node; |
| --- |

* delete(): Deleting a node from the BST

minValue: This is needed when deleting a node with two children. Gets the deleted node’s inorder successor and replaces its position in the BST.

| struct node\* minValue(struct node\* node) {  struct node\* currentNode = node;   while (currentNode && (currentNode->leftChild != NULL))  {  currentNode = currentNode->leftChild;  }   return currentNode; }  struct node\* delete(struct node\* root, int deleteNumber) {  if (root == NULL)  {  return root;  }   int rootNumber = atoi(root->phoneNumber);   if (deleteNumber < rootNumber)  {  root->leftChild = delete(root->leftChild, deleteNumber);  } else if (deleteNumber > rootNumber)  {  root->rightChild = delete(root->rightChild, deleteNumber);  } else  {  struct node\* tempNode;  if (root->leftChild == NULL)  {  tempNode = root->rightChild;  free(root);  return tempNode;  } else if (root->rightChild == NULL)  {  tempNode = root->leftChild;  free(root);  return tempNode;  }   tempNode = minValue(root->rightChild);  strncpy(tempNode->phoneNumber, root->phoneNumber, 20);   root->rightChild = delete(root->rightChild, atoi(tempNode->phoneNumber));  }   return root; } |
| --- |

* Inorder(): Printing all values within the BST in an inorder fashion

| void inorder(struct node\* root) {  if (root != NULL)  {  inorder(root->leftChild);  printf("number:%s name:%s address:%s \n", root->phoneNumber, root->contactName, root->contactAddress);  inorder(root->rightChild);  } } |
| --- |

* search(): Searching for the presence of a value in a BST, this value can be either
  + searchName(): Given a name, find its node in the BST. Print the name’s corresponding contact details. In the current implementation, all nodes are searched until the contact is found.
  + searchNumber(): Given a phone number, find its node in the BST. Print the phone number’s corresponding contact details. In this implementation, not all nodes will need to be searched. It iterates through the tree in an inorder fashion.

| void searchName(struct node\* root, char \*value) {  if (root != NULL)  {  searchName(root->leftChild, value);  if (strcmp(root->contactName, value) == 0)  {  printf("Found Contact:\nName: %s\nAddress: %s\nPhone Number: %s\n",  root->contactName, root->contactAddress, root->phoneNumber);  }  searchName(root->rightChild, value);  } }  void searchNumber(struct node\* root, int value) {  if (root != NULL)  {  if (atoi(root->phoneNumber) == value)  {  printf("Found Contact:\nName: %s\nAddress: %s\nPhone Number: %s\n",  root->contactName, root->contactAddress, root->phoneNumber);  }  else if (value < atoi(root->phoneNumber))  {  searchNumber(root->leftChild, value);  }  else  {  searchNumber(root->rightChild, value);  }  } } |
| --- |

After establishing the functions needed, I then constructed how the data is parsed into the tree. There are two options when giving data. The user can initially provide a file containing the data in which the tree is created. If the user chooses to do so, the file is then iterated through character by character. The name, address, and number in the file must be separated by white space, and each contact must be separated by a new line.

Every time the program encounters a new line character the name, address, and number are inserted into the tree. Before this, a check is made to see if the node being inserted is the first, initial node. If so, then the node is set as the root. The name, address, and number arrays are then reset for the next contact details to be inserted.

Finally, I had to implement a way for the user to interact with the program. In this implementation, a while loop is used to read the user’s input and is exited once the user enters “quit”. The current commands for the program are the same provided by the object-oriented command line.

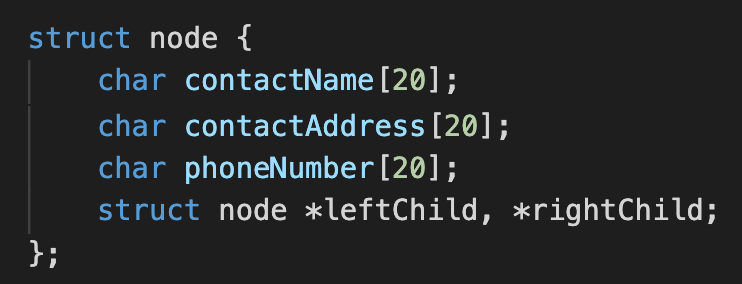
# COMPARATIVE ANALYSIS:

## DATA TYPES AND SCOPES:

For most programming languages, values are grouped into types. The most common types built into any programming language are boolean; Character; Integer and Float. When looking at a boolean expression this is denoted as ‘true’ or ‘false’ in python, whereas in C it is predefined by 0 identifying as ‘False’, and any non-zero number being ‘true’. [1]

A comparison in the implementation was how we assigned the data types. For instance, in Python, we don’t need to declare the type of value that an assignment holds as it holds a dynamic type, unlike in C we need to declare this variable. The Python interpreter reads the variable and the value assigned to it, tracking the variable name and type it holds [2]. This differs in C, the type must be declared by the developer in order to assign a variable. The type determines the size and layout of the memory and the operations which can be applied to the variable that is being told to the compiler. [3]

Another feature of the data types is known as mapping. The concept of mapping from one set to another is vital when it comes to programming, as it helps declare attributes linked to particular instances. In an object-oriented programming language, such as Python, one mapping type used in our source code is dictionaries, as it matches the key types to the objects. [4]

Figure 2.1 - Structure in C

A structure is used to map multiple data attributes as one member. Declaring a structure in C is different, as it is necessary for the struct to be declared separate, similar to assigning basic variables [5], as shown in figure 2. In order for the memory to be allocated for the structure the variables should already be created and assigned.

## MEMORY MANAGEMENT:

In Python variables, and memory management is controlled by an automated heap. Think of the heap as an area of memory. When a new object is created in Python, a sufficient amount of memory from the heap is allocated to the new object. The Python automated heap works by deleting unwanted objects, therefore freeing the memory space. These operations of allocating and deallocating memory do not need to be controlled by the programmer but are done by the Python Memory Manager.

In comparison to C, the deallocation, reallocation, and manipulation of memory are controlled by the programmer. When requesting a block of memory the run-time system allocates a block of memory from the memory pool (similar to the heap in Python) and returns a pointer to the allocated memory block. When releasing memory back to the memory pool in C it is controlled by the programmer. If this memory is not released correctly it can leak and become lost. These memory operations are achieved using library functions, an example of this in the imperative implementation this:

The declaration of the memset() function is *void \*memset(void \*str, int c, size\_t n)*. This is a case of memory manipulation where:

* str - is a pointer to a block of memory to fill
* c - is the set value, passed as it, but the function fills the block of memory with the unsigned char conversion of its value
* n - is the number of bytes to be set to the value

## ABSTRACTION:

Abstraction allows the programmer to separate what the programmer does and how the program completes it. Object Oriented programming can be seen as part of data abstraction, as objects can be seen as a bundle of multiple attributes. For our implementation, we use classes to define the Node for the binary tree [7]. The attributes set to our class were the name, address, and phone number of the contact as well as the left and right child nodes.

As classes are an object-oriented programming feature, for procedural languages such as C, the attributes are inherited from structures, where the data is retrieved from. The structures were described above in “Data Types and Scopes”.

# ADVANTAGED AND DISADVANTAGES:

|  | Advantages | Disadvantages |
| --- | --- | --- |
| Python | * Dynamic Types: Data is automatically assigned during execution so the programmer does not need to assign a type. [8] * As python is an Object-Oriented language, features such as abstraction can be used which is how a programmer can interact with the code. | * Because Python is a dynamically typed language, it can tend to be executed at a slower speed. |
| C | * C code can be executed at a much more sufficient speed as the C compiler can help with faster execution. [9] * Memory is able to grow or shrink according to the size necessary for the program. | * Memory is either static or dynamic so the programmer must assign the type linked to the variable they are providing. |

# 

# CONCLUSION

In conclusion, both programming languages are implemented differently based on the assignment of variables, how memory is stored and how the program can be constructed. Each language has its own advantages and disadvantages when comparing them for constructing a phonebook with a Binary tree and this assignment shows how in-depth these two can be evaluated and compared.

# REFERENCES

* [1] “Week 2 - Data Types and Scope” CA341 Dublin City University Dr. Brain Davis: Type and Operations Slide 8
* [2] Halperin, N. (2018) *Nuri Halperin*, *Pluralsight*t: https://www.pluralsight.com/guides/python-basics-variables-assignment (Accessed: October 28, 2022): <https://www.pluralsight.com/guides/python-basics-variables-assignment>
* *[3] C - variables* (no date) *Tutorials Point*. Available at: <https://www.tutorialspoint.com/cprogramming/c_variables.htm>(Accessed: October 28, 2022).
* [4] (no date) *Mapping types -- python library reference*. Available at: http://insti.physics.sunysb.edu/itp/computing/doc/python/python-texinfo/mapping\_types.html (Accessed: October 28, 2022).
* [5] “Structures in C” GeeksForGeeks: <https://www.geeksforgeeks.org/structures-c/>
* [6] “Pointers and Memory Management” CA341 Dublin City University Dr.[David Sinclair](mailto:david.sinclair@dcu.ie): <https://loop.dcu.ie/pluginfile.php/4621330/mod_resource/content/2/CA341_Pointers_and_Memory_Management.pdf>
* [7] “Object Oriented Programming Paradigm” CA341 Dublin City University Dr. [David Sinclair](mailto:david.sinclair@dcu.ie): <https://loop.dcu.ie/pluginfile.php/4621341/mod_resource/content/2/CA341_Object-Oriented_Programming_Paradigm.pdf>
* [8] Team, D.F. (2021) *Advantages and disadvantages of python - how it is dominating Programming World*, *DataFlair*. Available at: <https://data-flair.training/blogs/advantages-and-disadvantages-of-python/>(Accessed: November 6, 2022).
* [9] *Advantages and disadvantages of C programming* (no date) *Unstop*. Available at: <https://unstop.com/blog/advantages-and-disadvantages-of-c-programming-language> (Accessed: November 6, 2022).

## 

## 