CA341 - Comparing Procedural and Object-Oriented Programming

Group 40: Benjamin Olojo (19500599) and Przemyslaw Majda (20505049)

Overview
This assignment consisted of implementing a phonebook program in both a procedural and object-oriented language using Binary Trees for the insertion, deletion and searching of contacts inputted by users. We chose C as our procedural language and Java as our object-oriented language.
Within the programs, contacts are stored as nodes within the Binary Trees and consist of names, phone numbers and addresses. Reverse search by phone numbers rather than names is also supported within both programs, with two separate Binary Trees being used for names and phone numbers in each.
The programs also feature a basic command line interface to allow users select commands and enter contact information used within the programs.

C Structs vs Java Classes

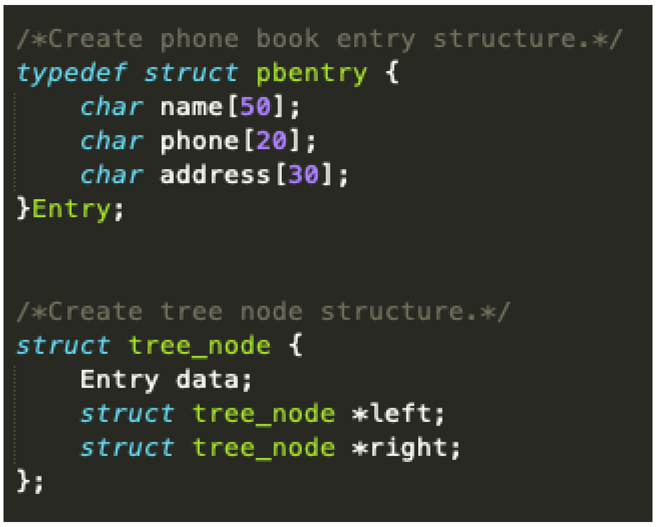
C Structs

C structures offer developers the ability to organise data by grouping variables together under one structure [1]. These variables can be of any type, and can even be structures themselves as illustrated within our C source code in Figure 1. A structure however can only house variables, and any methods that would perform any operations on the struct must be declared explicitly in the program and take a structure as input.

Multiple instances of a struct may be created, each with unique values. Values may be modified either through functions or explicitly written in the program.

In Figure 1, our procedural C code uses structs to define a phone book entry (the name, phone and address of each person), and the nodes that populate the name and number trees.
Functions that perform operations on the structs must be declared at the top of the file along with any regular functions.

Figure 1



```
/*Create phone book entry structure.*/
typedef struct pbentry {
    char name[50];
    char phone[20];
    char address[30];
}Entry;


/*Create tree node structure.*/
struct tree_node {
    Entry data;
    struct tree_node *left;
    struct tree_node *right;
};
```

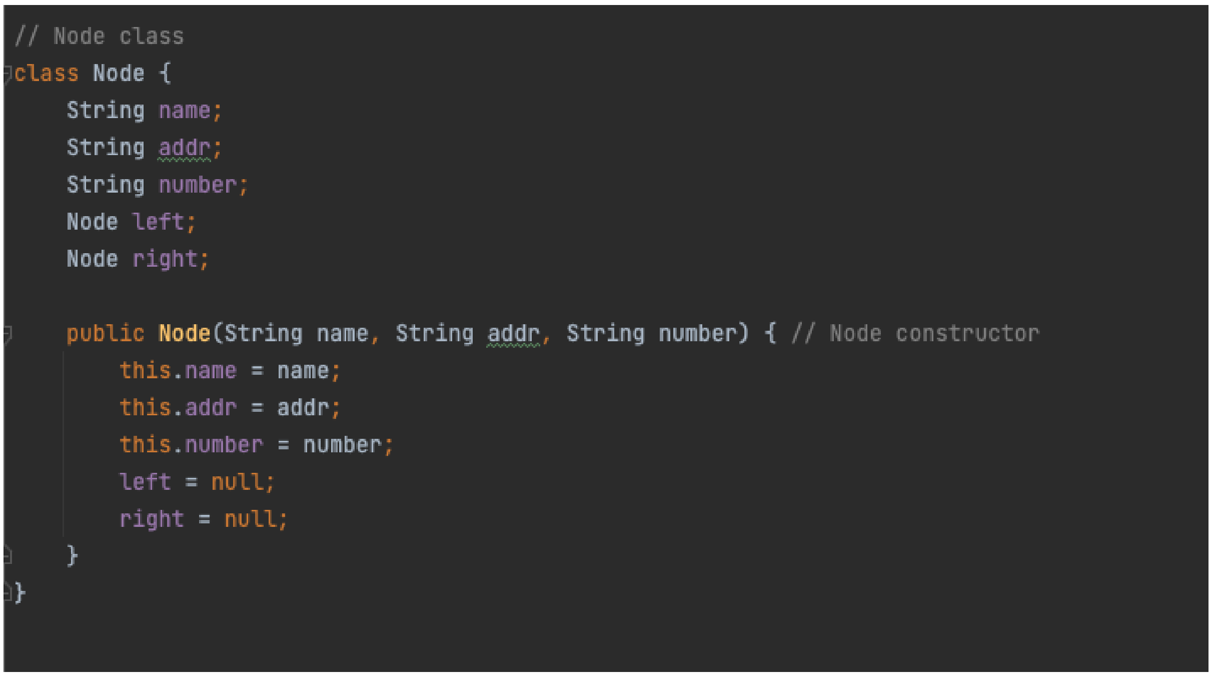
Java Classes

Java classes are similar to structs in that they allow for easy grouping of variables under one class, but there are some major differences. Java classes can have methods written directly inside the class which can perform operations on the data contained within the class. In C one has to explicitly provide the struct as a parameter to a function, whereas in Java this can be done implicitly using the *object.method()* syntax [2].

Java classes also have the option of using a constructor method which is called when an instance of the class is created. This gives the user the option for a class object to be initialised with specific values, or default to preset ones should no value be given.

In our object-oriented Java code shown in Figure 2, we use classes for the tree nodes in the same way we used structs in the C implementation, however in Java we can use a constructor method to initialise the node with input data.

Figure 2



```java
// Node class
class Node {
    String name;
    String addr;
    String number;
    Node left;
    Node right;

    public Node(String name, String addr, String number) { // Node constructor
        this.name = name;
        this.addr = addr;
        this.number = number;
        left = null;
        right = null;
    }
}
```

<u>Memory Management & Garbage Collection</u>

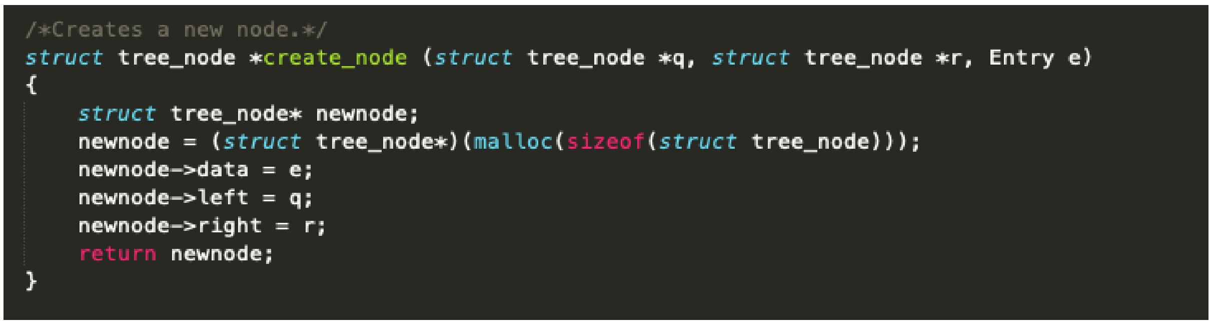
Memory management in C

In C, the job of memory management is left almost entirely to the developer. Static memory is calculated at compile time, and automatically allocated when the program starts [3]. Memory that is statically allocated is automatically freed based on its scope, as soon as the scope is over, the memory will be freed. This memory is allocated from the stack.

However, dynamic memory allocation must be handled by the developer and must be explicitly allocated using the *malloc()* or *calloc()* functions. This is used when the compiler cannot determine how much memory will be needed. This memory is allocated from the heap, a bank of memory that the program has for this exact purpose.

An example of this can be seen in Figure 3, where memory is dynamically allocated for creating a new tree node. It must be allocated dynamically as we give the user the option to input many nodes, and do not know how much memory will be needed beforehand.

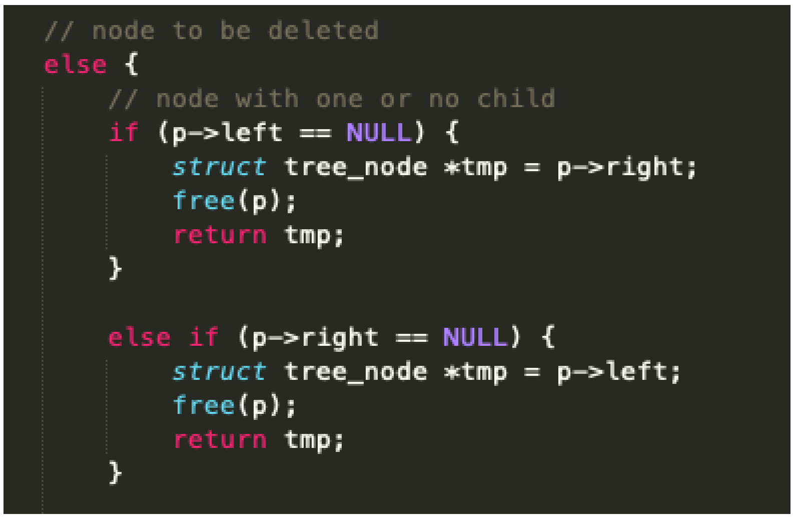
Figure 3



```c
/*Creates a new node.*/
struct tree_node *create_node (struct tree_node *q, struct tree_node *r, Entry e)
{
    struct tree_node* newnode;
    newnode = (struct tree_node*)(malloc(sizeof(struct tree_node)));
    newnode->data = e;
    newnode->left = q;
    newnode->right = r;
    return newnode;
}
```

C also does not have any inbuilt garbage collection, so memory must be freed manually when it is no longer being used. An example of this can be seen in Figure 4 when we delete a tree node.

Figure 4



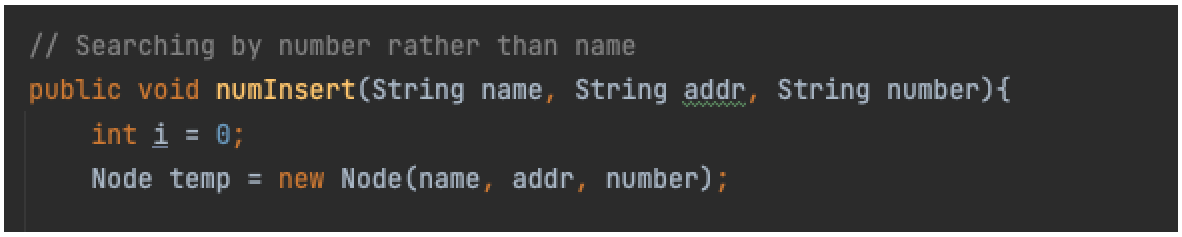
```c
// node to be deleted
else {
    // node with one or no child
    if (p->left == NULL) {
        struct tree_node *tmp = p->right;
        free(p);
        return tmp;
    }

    else if (p->right == NULL) {
        struct tree_node *tmp = p->left;
        free(p);
        return tmp;
    }
```

Memory management in Java

In contrast to C, Java handles the majority of memory management for the developer. Memory does not need to be explicitly allocated by the developer. Instead of a function like *malloc()*, the *new* keyword in java allocates memory for a new class object [4]. Java automatically knows how much memory this object requires and allocates the necessary amount from the heap. This can be seen in our Java shown in Figure 5 when we create a new tree node.

Figure 5



```java
// Searching by number rather than name
public void numInsert(String name, String addr, String number){
    int i = 0;
    Node temp = new Node(name, addr, number);
```

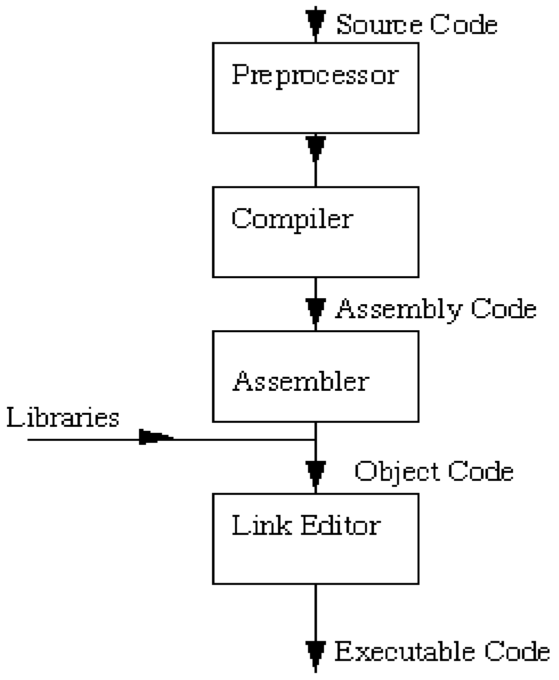
Java also automatically deallocates unreferenced objects using its Garbage Collector. This makes the process simpler for the developer, but it can have a negative effect on the speed of the overall program, as it has to wait for the garbage collector to run before it can continue [5].

C compilation vs Java Interpretation

C Compilation

C source code requires a compilation process before it can be executed on a machine. Compilation involves converting human-readable source code into machine code that can be executed by a computer's CPU [8]. The compilation process also includes checking the syntax and semantics of code to determine any syntax errors or warnings present in the source code [8]. The overall compilation process includes four stages: pre-processing, compilation, assembly and linking.
Since C compilation uses a static compilation or Ahead-Of-Time (AOT) method, our source code needs to be compiled on the machine it is being executed on.

Figure 6

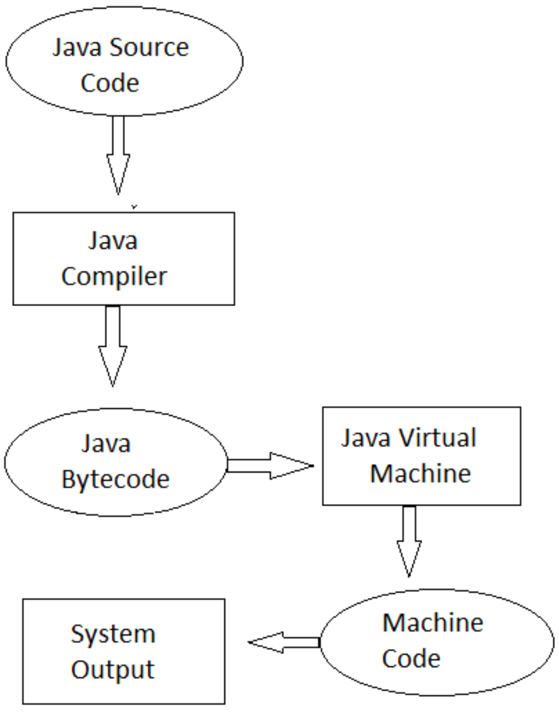


Java Interpretation

There is a two step compilation process for the execution of Java programs. The Java source code is compiled down to bytecode by the java compiler such as javac. This bytecode is then executed by a Java Virtual Machine (JVM) through Just-In-Time (JIT) compilation, also known as dynamic compilation. This execution process involves compiling bytecode to native instructions that are optimised for the running machine's CPU architecture [6].

Due to the use of the JVM, Java programs are platform independent once compiled to bytecode. This has the advantage of allowing our Java program to be run on any system regardless of the platform it was developed on [7].

Figure 7

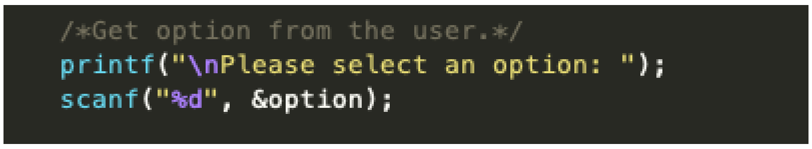


Java References vs C Pointers

C Pointers

A pointer is a variable whose value is the memory address of another variable. Pointer variables point to a variable of the same data type and can be incremented or decremented to point to other memory addresses within contiguous blocks [9]. Within C, a pointer can store the memory address of an existing through the reference operator (&) as depicted in Figure 8. The dereference operator (*) can then be used to get the value at the memory address stored by a pointer.
The manipulation of C pointers provides flexibility to developers through lower-level control over resources such as memory blocks, but can be problematic and error-prone if used incorrectly with user inputs [10].

Figure 8



```
/*Get option from the user.*/
printf("\nPlease select an option: ");
scanf("%d", &option);
```

Java References
In Java, reference variables point to objects or values. Classes, arrays, interfaces, enumerations and annotation are all reference types in Java that can be stored by reference variables. The *new* keyword in Java can be used to create a reference to a particular object defined by a class as shown in Figure 5. Java references are strongly-typed and can also be categorised by how they are managed by the garbage collector of the JVM. These categories include strong, weak, soft and phantom references.
Since references are initialised at the time of their declaration, developers don't need to handle uninitialized wild pointers that point to arbitrary memory addresses like in C [11]. Java references also don't support arithmetic like C pointers, resulting in a more robust security model for programs [10].

References

[1] W3Schools, “C Structures”, https://www.w3schools.com/c/c_structs.php

[2] Stack Overflow, “What is the difference between C structures and Java classes?”, 23 March 2011, https://stackoverflow.com/questions/5413001/what-is-the-difference-between-c-structures-and-java-classes

[3] OpenGenus, “Static memory allocation in C”, https://iq.opengenus.org/static-memory-allocation-c/

[4] S.Trivedi, “New Keyword in Java”, 13 May 2022, https://www.scaler.com/topics/new-keyword-in-java/

[5] A.Altvater, “What is Java Garbage Collection?”, https://stackify.com/what-is-java-garbage-collection/

[6] freeCodeCamp, “Just in Time Compilation Explained”, 1 February 2020, https://www.freecodecamp.org/news/just-in-time-compilation-explained/

[7] R.Vats, “Why is Java Platform Independent Language?”, 8 February 2021, https://www.upgrad.com/blog/why-is-java-platform-independent-language/

[8] A.Chandra, “Why is Java Platform Independent Language?”, 28 February 2022, https://www.scaler.com/topics/c/compilation-process-in-c/

[9] B.Thompson, “Pointers in C: What is Pointer in C Programming? Types”, 25 August 2022, https://www.guru99.com/c-pointers.html

[10] GeeksForGeeks, “C/C++ Pointers vs Java References”, 8 May 2017, https://www.geeksforgeeks.org/is-there-any-concept-of-pointers-in-java/

[11] A. Nandi, “Pointers vs References in C++”, 19 July 2021, https://www.scaler.com/topics/cpp/pointers-vs-references-in-cplusplus/