CA341 Competitive Programming Languages

Project 1

Jamie Clarke

18398783

14/11/2021

Introduction

Procedural :
For this project I chose the programming language C to do this PhoneBook. The reason I chose this language was because of my familiarity with it. Also C has great portability as well as being very simple to understand. There are a lot of guides online for C so when I encountered problems it was easy to find a solution.

Object-Oriented :
For this project I chose java for my object oriented language. The reason I chose this for my OO language is because of its use in today's world. Also it is a great language for binary search trees.

Comparative Analysis

Java

In my java code I began by creating 2 separate files, one main file, which stored, removed, added and displayed all contacts, one for my node object , (which I got help from last year's computer programming 4 module)

MENU

Firstly I created a user menu so that the user could enter a number and be directed to what they chose. I did this by simply using some print statements to give the user options for the phonebook.

```
public static void menu()
{
// create menu for users.//
    System.out.println("Welcome to the menu\n");
    System.out.println("Press 1 to Show contacts. \n");
    System.out.println("Press 2 to remove a contact\n");
    System.out.println("Press 3 to Search for a contact \n");
    System.out.println("Press 4 to quit\n");
}
```

Main Function

Next I went and created 4 different cases, depending on what the user selected. This then calls my function depending on their selection. E.g if the user pressed 1, then It would call my display object which printed out the entire phone book.

```
public class PhoneBook
{
    public static void main(String[] args)
    {
        phoneBook1 = new phoneBook1();
        while(!exit)
        {
            Scanner scan = new Scanner(System.in);
            PhoneBook.menu();

            switch(scan.nextInt())
            {

                case1:
                    PhoneBook.Display();
                    break;

                case2:
                    PhoneBook.add();
                    break;

                case3:
                    PhoneBook.remove();
                    break;

                case4:
                    PhoneBook.Search();
                    break;

                case4:
                    PhoneBook.quit();
                    break;
```

ADD

Next I created my add object so that the user could add a contact to the Phonebook. It had 3 parameters so the user would have to enter a name, address and phone number for it to be added to the Phonebook.

```java
//add person to PhoneBook
public Node add(Node node, String name, String add, int num)
{
    if (node = null)
    {
        Node person = new Node(name, add, num);
        return person;
    }
    if(num < node.number)
    {
        node.left = add(node.left, name, add, num);
    }
    else if(num > node.number)
    {
        node.right = add(node.right, name, add, num);
    }
    return node;
}
```

DISPLAY

For my display object I tried to return the phone book. I was unable to get this working in my code.

```java
public Node Display()
{
    return phoneBook1;
}
```

REMOVE

For my remove object, I had to check if the contact was being stored on the left or right side of the search tree and remove it from there. I used an online video for help with this object which I feel was very useful . This object works so that if the user wishes to remove a contact from their Phonebook, they are able to with ease by just entering the user's details.

```
public Node remove(Node node int num)/* Used this video t
{
    if(node == null)
    {
        return null;
    }
    if(num < node.number)
    {
        node.left = remove(node.right, num);
    }
    else if(num > node.number)
    {
        node.right = remove(node.right, num);
    }
    else
    {
        if(node.left == null || node.right == null)
        {
            Node temp = null;
            if(node.left == null)
            {
                temp = node.left;
            }
            else
            {
                temp = node.right;
            }

            if(temp == null)
            {
                return null;
```

SEARCH

Finally in my main code, I created a search object which searches for users by their name. It checked their name against the users stored in the tree to see if they existed in the tree. If not an error message was returned informing the user of this. This allowed the user to find contacts quicker.

```
private Person Search(Person person, String name);
{
    if (Person == null)
    {
        return null;
        System.out.println("error, user not found");
    }

    else if (name.equals(name.getname()))
    {
        return name;
    }
    else
    {
        if(person.left() != null)
        {
            return Search(person.left(), name);
        }
        else
        {
            return Search(person.right(), name);
        }
    }
}
```

NODE CLASS

Next in my node class I used a similar method to my C code, implementing name, address, Phone number and a left and right node. I also used a hashmap to store keys and values for the phonebook. Finally I created a List for contacts to be stored in the form of an Array list.

```
public class Node
{
    public String name;
    public String address;
    public int PhoneNumber;
    public node left;
    public node right;
    Map<Integer, List<String>> map = new HashMap<Integer, List<String>>();
    List<String> contact = new ArrayList<String>();
```

The next part was not a lot of new things, I just used .put to add the number and contact name to the map.

```
    public Node(String n, String add, int num)
    {
        name = n;
        address = add;
        PhoneNumber = num;
        left = null;
        right = null;
        contact.add(name);
        contact.add(address);
        map.put(number, contact);
    }
}
```

ADD

Next in my binary search tree code I created an add object for adding users to the phonebook. In my C code I was unable to create this

```
public Node add(Node node, String name, String add, int num)
{
    if (node = null)
    {
        Node person = new Node(name, add, num);
        return person;
    }
    if(num < node.number)
    {
        node.left = add(node.left, name, add, num);
    }
    else if(num > node.number)
    {
        node.right = add(node.right, name, add, num);
    }
    return node;
}
```

C

STRUCT ADDRESSBOOK

In my C code I began with just one main file, rather than the three I went for in the object oriented program. I started by initializing the main components of a

phonebook, including the person's name, address and phone number. I also called in a left and right node for storing memory for the phone book.

```
// iniatlising main components of a phonebook //
struct AddressBook
{
    char name[50];
    char address[50];
    int phoneNumber;
    struct node right;
    struct node left;
};
```

MENU

These were all stored in a struct called AddressBook. The next step was my main function which I used to call all other functions. Next I set up a basic command line menu so the user could enter a number to either Display the phonebook, Search a user, remove a user or quit. I just used simple print statements followed by a scan to see the users input. I also called an integer “choice” so that we could see what the user had entered. I used “%i” rather than “%d” as the only options for the user to enter were 1-4, therefore not requiring any decimal places.

```
        printf("\n Welcome to the Phone book");
        printf("\n Show contacts  [1]");
        printf("\n Remove Contact  [2]");
        printf("\n Search contact  [3]");
        printf("\n Exit [4]");
        printf("Enter your choice");
        scanf("%i", &choice);
```

CASES

Next I used a switch function with the parameter of the previously mentioned integer “choice”, so when the user entered their choice, the program sent them to case 1, 2, 3 or 4 depending on their input. As you can see below this is where I called the functions I wrote for each scenario depending on user input.

```
switch(choice)
{
case 1:
     Display();
break;

case 2:
     Remove();
break;

case 3:
     Search();
break;

case 4:
     Exit();
break;
```

DISPLAY

Next we will look at each of the functions I wrote, starting with the display function. This function would be called if the user entered 1. I used an external website for help on this function. [1] This display function prints out the entire phone book, in the order name, address, Phone number. This function simply checked to see if tree_node was empty, if not it returned p (phonebook). Else if it was empty the user received an error message saying that the phonebook was empty.

```
void Display(struct tree_node *p)
{
    char choice1;
    if (p != NULL) {
        Display(p->left);
        printf("%s, %s, %s \n", p->data.name, p->data.address, p->data.phonenumber);
        Display(p->right);
    }
    else
    {
        printf("Phonebook is empty");
        break;
    }
}
```

[1] https://cboard.cprogramming.com/c-programming/68409-binary-search-tree.html

REMOVE

Next we moved onto the remove function so that the user could remove an entry from the phonebook if they no longer required it. The remove function checks if the root is smaller than the number, and if so it checks either the left or right of the tree to delete it. We used recursion of the Remove function in order to delete and user not wanted. For this function I used an online source for help in understanding how to delete from a binary tree.[2]

```
void Remove(struct node *root, int number)
{
    if(root == NULL)
        return root;
    // if number is smaller then root, it is found in left subtre //
    if(number < root -> number)
    {
        root -> left = Remove(root->left, number);
    }

    //if the number is bigger then the root, it will be in the right subtree //
    else if(number > root-> number)
    {
        root -> right = Remove(root -> right, number);
    }
    // if root is same as number //
    else
    {
        if(root -> left == NULL)
        struct node* temp = root -> right;
        free(root);
        return temp;
    }

    else if(root -> right == NULL)
    {
        struct node* temp = root-> left;
        free(root);
        return temp;
    }
}
```

SEARCH

Next we had the search function which we used to check if a user was in the Phonebook. This function searched by name to find the user you were looking for. I then tried to match the address to the user to confirm it is them.If the user was confirmed we printed out their name, address and phone number, if not then an error message was displayed.

[2] https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/

```c
/*Search by first name */
void Search(struct tree_node *p, char name[], char address[])
{
    char choice3;
    if (strcmp(l, p->data.name) < 0) {
        Search(p->left, name);
    }
    else if(strcmp(add, p-> data.address) < 0)
        {
            Search(p -> left, address)
        }
        if (strcmp(f, p->data.name) > 0) {
            Search(p->right, name);
        }
    }
    else if (strcmp(l, p->data.name) == 0)
    {
        printf("%s\n, %s\n, %s\n", p->data.name, p->data.address, p->data.phone);
    }
    else {
        printf("Contact could not be found.\n");
    }
}
```

Comparison of implementation:

I tried following a similar format when creating both pieces of code. I used a case system to see what the user had entered and move from there. Also in both codes I called functions to my main function to try to keep it tidier. When it comes to memory management in both languages there are some key differences. In Java I was not able to use pointers as java doesn't support them, whereas in C I was able to use pointers[3]. Also if I tried memory allocation, In C it was as simple as using malloc, whereas in Java, I had to use the new keyword. In Java I used a lot of objects whereas in C it was functions being used in nearly all my code.

With Java being high level I found it easier to write the code and remember what I've done. In C though, with it being low level I found certain functions harder to implement. Also within my main function I used threading, which is not an option for C. I was able to call by reference in C, but had to only call by value in Java which was not too bad but made life with C a slight bit easier. I also had to manually manage memory in my C code, but for Java this was not the case as Java internally manages it's memory.[4] Finally in C my functions were as default, public and if I wanted them to be private I had to specifically

---

[3] Week 3 - Pointers and Memory management, David Sinclair

[4] Week 3 - Pointers and Memory management, David Sinclair

declare this[5]. Whereas in Java it was the opposite, meaning I had to declare my objects public, as you can see in the example below:

Java

```
public Node remove(Node node int num)/* Us
{
```

C

```
void Remove(struct node *root, int number)
```

As you can see I specifically declare the node object as public in my Java file.[6]

References

- C Programming - A modern Approach, Second edition, K.N Knight, Chapter 11 -Pointers

- https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming

- Effective Java - 3rd edition, Chapter 4: Classes and Interfaces

- https://dzone.com/articles/java-memory-management
- Week 3 - Pointers and Memory management, David Sinclair
- https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/
- https://cboard.cprogramming.com/c-programming/68409-binary-search-tree.html
-

Appendices

Refer to code ..

[5] Week 4 - Data Types and scopes - David Sinclair
[6] https://www.geeksforgeeks.org/difference-between-java-and-c-language/