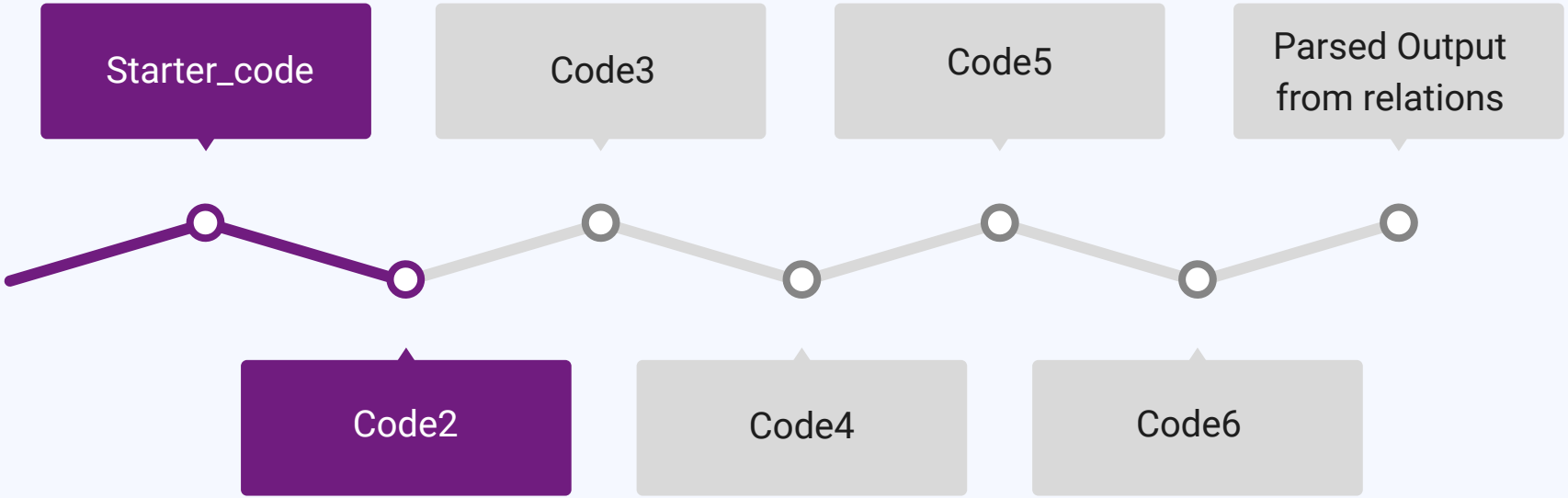
DC Summer Term 2023

Hardware Design Description Analysis

Rahul Barodia (B20CS047)

TimeLine


Starter_code
Code3
Code5
Parsed Output from relations
Code2
Code4
Code6

Task

The parsed output :

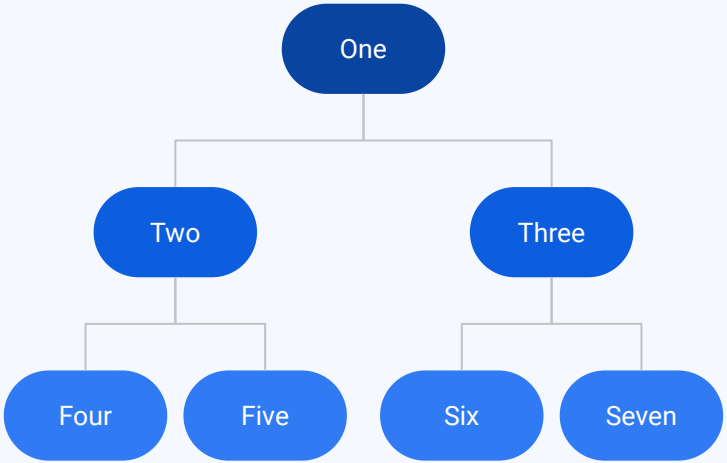
```
one:two,three,four
two:five
three:six,seven
four:eight,nine,ten
```

Backward Traversal :

```
Enter the signal name: ten
Path to top: ten -> four -> one
```

Starter Code

Trees



Enter the signal name : Seven

Output : Seven -> Three -> One

Tree Data structure

```cpp
#include<vector>
using namespace std;

template <typename T>

class TreeNode{
    public:
    T data;
    vector<TreeNode<T>*> children;

    TreeNode(T data){
        this->data=data;
    }

    ~TreeNode() {
        for (int i = 0; i < children.size(); i++) {
            delete children[i];
        }
    }

};
```

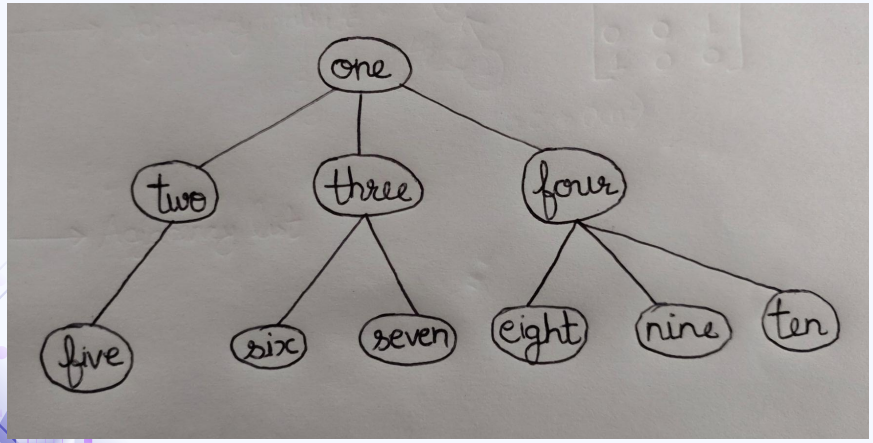
```cpp
TreeNode<string>* takeInputLevelWise() {
    string rootData;
    cout << "Enter root data: ";
    cin >> rootData;
    TreeNode<string>* root = new TreeNode<string>(rootData);

    queue<TreeNode<string>*> pendingNodes;

    pendingNodes.push(root);
    while (!pendingNodes.empty()) {
        TreeNode<string>* front = pendingNodes.front();
        pendingNodes.pop();
        int numChild;
        cout << "Enter number of children of " << front->data << ": ";
        cin >> numChild;
        for (int i = 0; i < numChild; i++) {
            string childData;
            cout << "Enter " << i+1 << "th child of " << front->data << ": ";
            cin >> childData;
            TreeNode<string>* child = new TreeNode<string>(childData);
            front->children.push_back(child);
            pendingNodes.push(child);
        }
    }
    return root;
}
```

```cpp
void printTree(TreeNode<string>* root) {
    if (root == NULL) {
        return;
    }

    cout << root->data << ":";
    for (int i = 0; i < root->children.size(); i++) {
        cout << root->children[i]->data << ",";
    }
    cout << endl;
    for (int i = 0; i < root->children.size(); i++) {
        printTree(root->children[i]);
    }
}

void printPathToTop(TreeNode<string>* node, vector<string>& path) {
    if (node == nullptr) {
        return;
    }

    path.push_back(node->data);  // Add the current node to the path

    // If the node is the top-most node, print the path
    if (node->children.empty()) {
        for (int i = path.size() - 1; i >= 0; i--) {
            cout << path[i];
            if (i != 0) {
                cout << " -> ";
            }
        }
        cout << endl;
    }
```

https://github.com/Rahul28428/Backward-Traversal/blob/main/Starter_code.cpp

The parsed output :

```
one:two,three,four
two:five
three:six,seven
four:eight,nine,ten
```




```
Enter root data: one
Enter number of children of one: 3
Enter 1th child of one: two
Enter 2th child of one: three
Enter 3th child of one: four
Enter number of children of two: 1
Enter 1th child of two: five
Enter number of children of three: 2
Enter 1th child of three: six
Enter 2th child of three: seven
Enter number of children of four: 3
Enter 1th child of four: eight
Enter 2th child of four: nine
Enter 3th child of four: ten
Enter number of children of five: 0
Enter number of children of six: 0
Enter number of children of seven: 0
Enter number of children of eight: 0
Enter number of children of nine: 0
Enter number of children of ten: 0
```

```
one:two,three,four,
two:five,
five:
three:six,seven,
six:
seven:
four:eight,nine,ten,
eight:
nine:
ten:
Enter the signal name to find its path: ten
Path to ten: ten -> four -> one
```

Code 2

Inserting parsed output directly into the code in hardcoded form

```
struct TreeNode
{
    string value;
    vector<TreeNode *> children;

    TreeNode(const string &val) : value(val) {}
};

map<string, TreeNode *> nodeMap;
```

```
void constructTree(const string &input)
{
```

```
void printTree(TreeNode *node, const string &prefix = "")
```

```
bool printPathToTop(TreeNode *node, const string &targetNode)
```

Logic of constructTree

```
void constructTree(const string &input)
{
    size_t colonPos = input.find(':');
    string parent = input.substr(0, colonPos);
    string childrenStr = input.substr(colonPos + 1); // Skip colon

    TreeNode *parentNode;
    if (nodeMap.find(parent) != nodeMap.end())
    {
        parentNode = nodeMap[parent];
    }
    else
    {
        parentNode = new TreeNode(parent);
        nodeMap[parent] = parentNode;
    }

    size_t commaPos = 0;
    while (commaPos != string::npos)
    {
        size_t nextCommaPos = childrenStr.find(',', commaPos);
        string child;
        if (nextCommaPos != string::npos)
        {
            child = childrenStr.substr(commaPos, nextCommaPos - commaPos);
            commaPos = nextCommaPos + 1; // Skip comma
        }
        else
        {
```

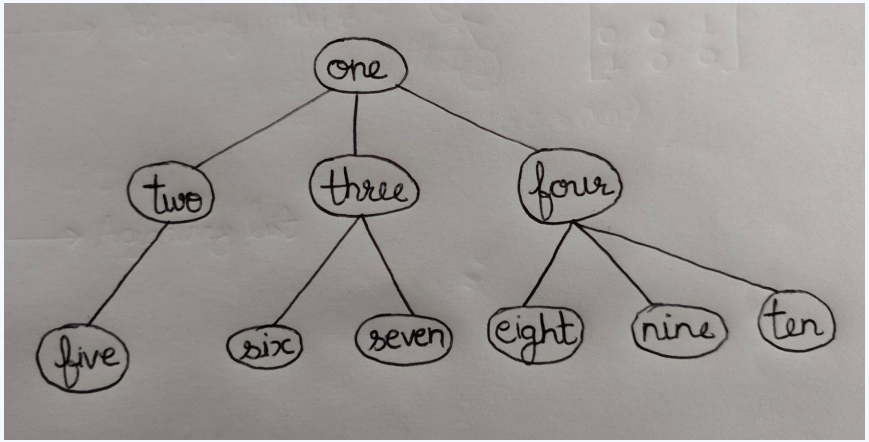
The constructTree function takes an input string representing a parent-child relationship and constructs a tree structure using TreeNode objects.

It finds the position of the colon character in the input string to separate the parent and children parts. It then creates or retrieves the parent TreeNode object from the nodeMap based on the parent string.

Next, it iterates through the children part, extracting each child string delimited by commas. For each child, it creates or retrieves the corresponding TreeNode object and adds it to the parent's children vector.

Overall, the function builds a tree structure by creating TreeNode objects for parents and their children, connecting them appropriately based on the input string.

```cpp
int main()
{
    vector<string> inputLines;
    inputLines.push_back("one:two,three,four");
    inputLines.push_back("two:five");
    inputLines.push_back("three:six,seven");
    inputLines.push_back("four:eight,nine,ten");
```




```
Tree Structure:
one : two, three, four
two : five
five:
three : six, seven
six:
seven:
four : eight, nine, ten
eight:
nine:
ten:
Enter the signal name: ten
Path to top: ten -> four -> one
```

https://github.com/Rahul28428/Backward-Traversal/blob/main/Code2.cpp

Code 3

Extracting parsed output from a text file and then feeding it to the code

```cpp
ifstream fileIn("input.txt");
if (!fileIn.is_open())
{
    cout << "Failed to open file!\n";
    return 0;
}

string line;
vector<string> inputLines;

while (getline(fileIn, line))
{
    inputLines.push_back(line);
}

for (size_t i = 0; i < inputLines.size(); ++i)
{
    constructTree(inputLines[i]);
}
```

```cpp
void constructTree(const string &input)
{
    size_t colonPos = input.find(':');
    string parent = input.substr(0, colonPos);
    string childrenStr = input.substr(colonPos + 1); // Skip colon

    // Trim leading and trailing whitespace from parent and children strings
    parent = parent.substr(parent.find_first_not_of(" \t\r\n"), parent.find_last_not_of(" \t\r\n") + 1);
    childrenStr = childrenStr.substr(childrenStr.find_first_not_of(" \t\r\n"), childrenStr.find_last_not_of("
```

```cpp
bool isRootSignal(TreeNode *node)
{
    for (map<string, TreeNode *>::const_iterator it = nodeMap.begin(); it != nodeMap.end(); ++it)
    {
        if (it->first != node->value)
        {
            for (vector<TreeNode *>::const_iterator child = it->second->children.begin(); child != it->second->
            {
                if (*child == node)
                    return false;
            }
        }
    }
    return true;
}
```

Input.txt

```
one:two,three,four
two:five
three:six,seven
four:eight,nine,ten
```

```
Tree Structure:
one : two, three, four
    two : five
        five:
    three : six, seven
        six:
        seven:
    four : eight, nine, ten
        eight:
        nine:
        ten:
Enter the signal name: nine
Path to top: one -> four -> nine
```

Input2.txt

```
dog:cat,mat,rat
mat:fat,bat
rat:net
```

```
Tree Structure:
dog : cat, mat, rat
    cat:
    mat : fat, bat
        fat:
        bat:
    rat : net
        net:
Enter the signal name: net
Path to top: dog -> rat -> net
```

Code3

Code 4

Optimizing Code 3

Use const references for function parameters: In some functions like printTree and printPathToTop, pass the tree nodes as const TreeNode* or const TreeNode& to avoid unnecessary copies of the nodes.

Improve data structures and algorithms: Depending on the specific requirements of your application, you may consider using more efficient data structures and algorithms. For example, using a hash-based data structure like std::unordered_map instead of std::map can provide faster lookup times.

```
TreeNode *root = nullptr;
for (unordered_map<string, TreeNode *>::const_iterator it = nodeMap.begin(); it != nodeMap.end(); ++it)
{
    if (isRootSignal(it->second))
    {
        root = it->second;
        break;
    }
}
```

```
// Clean up memory
for (unordered_map<string, TreeNode *>::const_iterator it = nodeMap.begin(); it != nodeMap.end(); ++it)
{
    delete it->second;
}
```

```
unordered_map<string, TreeNode *> nodeMap;
```

```
unordered_map<string, TreeNode *>::iter
if (it != nodeMap.end())
{
    parentNode = it->second;
}
else
{
    parentNode = new TreeNode(parent);
    nodeMap[parent] = parentNode;
}
```

Code4.cpp

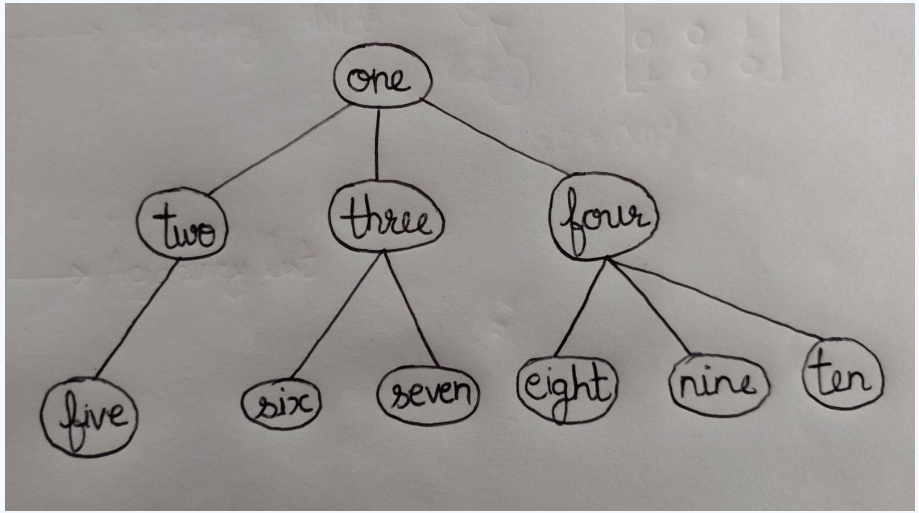
Code 5

Using unordered_map inplace of Tree

```
struct TreeNode
{
    string value;
    vector<TreeNode *> children;

    TreeNode(const string &val) : value(val) {}
};
```

```
unordered_map<string, pair<string, vector<string> > > nodeMap;
```

Code5 uses an unordered map nodeMap with string keys and a pair of string and vector<string> as the value type. Whereas Code4 uses an unordered map nodeMap with string keys and TreeNode* as the value type. The tree nodes are represented using a custom TreeNode struct.




```
Tree Structure:
one : two, three, four
    two : five
        five:
    three : six, seven
        six:
        seven:
    four : eight, nine, ten
        eight:
        nine:
        ten:
Enter the signal name: ten
Path to top: one -> four -> ten
```

Code5.cpp

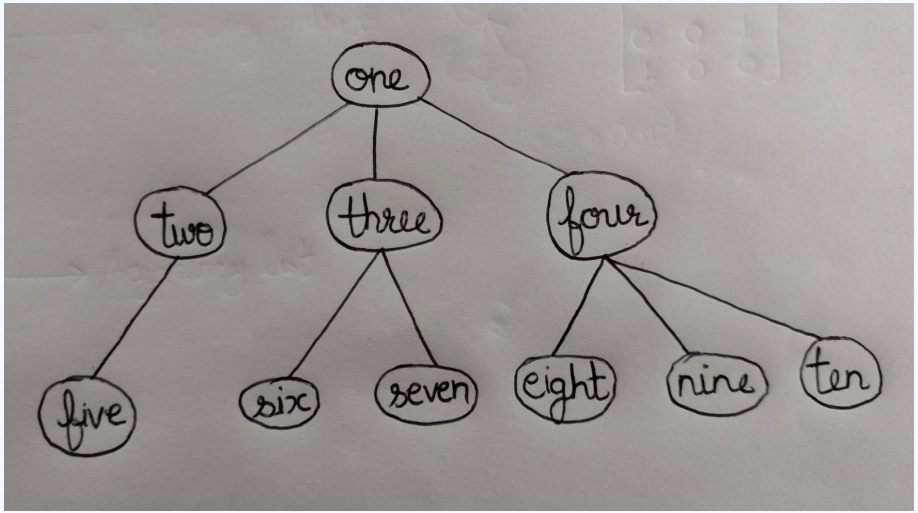
Code 6

Making a new struct

```
struct Node {
    string parent;
    vector<string> children;
};

unordered_map<string, Node> nodeMap;
```

Explanation :

The tree structure is represented using a custom struct called Node.
The nodeMap is an unordered_map with string keys and Node values. It stores the parent node as a string and the children nodes as a vector of strings.
The constructTree function creates and populates the nodeMap by parsing the input string and storing the parent and children nodes in the Node structure.




```
Tree Structure:
one : two, three, four
    two : five
        five:
    three : six, seven
        six:
        seven:
    four : eight, nine, ten
        eight:
        nine:
        ten:
Enter the signal name: ten
Path to top: one -> four -> ten
```

Code6.cpp

Parsed output from relations

Input_pre.txt

```
one<=two&&three&&four
two<=five
three<=six&&seven
four<=eight&&nine&&ten
```

Convert.cpp

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <algorithm>
#include <cctype>
using namespace std;

// Split a string based on a delimiter
vector<string> splitString(const string& str, const string& delimiter) {
    vector<string> tokens;
    size_t start = 0;
    size_t end = str.find(delimiter);

    while (end != string::npos) {
        tokens.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }

    tokens.push_back(str.substr(start, end));
    return tokens;
}

// Trim leading and trailing whitespaces from a string
string trimString(const string& str) {
    size_t start = 0;
    size_t end = str.length();

    while (start < end && isspace(str[start])) {
        ++start;
    }

    while (end > start && isspace(str[end - 1])) {
        --end;
    }
```

Input_post.txt

```
four:eight,nine,ten
one:four,three,two
three:seven,six
two:five
```

Thanks!