****

**Innovative Assignment**

**Group Members:**

1. Darshil Ladani (21BCE132)

2. Manan Ladani (21BCE133)

3. Zemi Lakkad (21BCE134)

**Project Title: Basic Implementation of DNS**

**What is DNS ?**

* The Domain Name System is the phonebook of the Internet. DNS lets users

connect to websites using domain names instead of IP addresses. A Domain Name System (DNS) turns domain names into IP addresses, which allow browsers to get to websites and other internet resources. All devices connected to the Internet have a unique IP address. An example of a DNS is that which is provided by Google. The address of Google's primary DNS is 8.8.8.8.

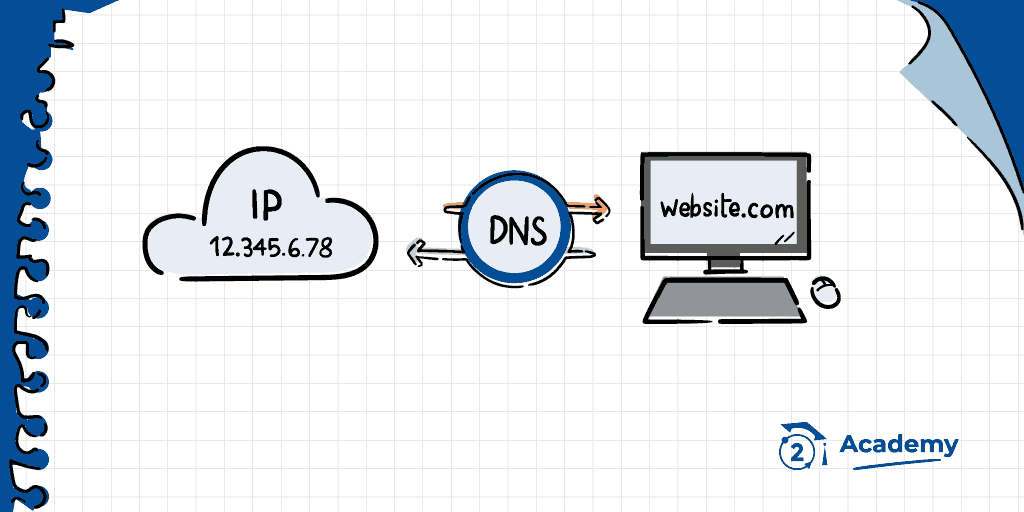
**How does DNS work?**

* The process of DNS resolution involves converting a hostname (such as

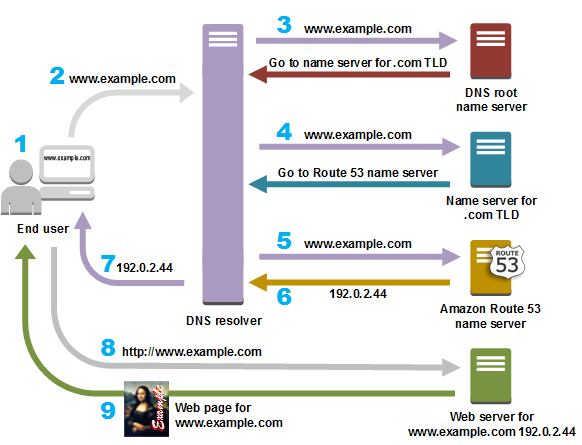
www.example.com) into a computer-friendly IP address (such as 192.168.1.1).

DNS works in 9 steps:

1. A user opens a web browser, enters www.example.com in the address bar.
2. The request for www.example.com is routed to a DNS resolver, which is typically managed by the user's Internet service provider (ISP).
3. The DNS resolver for the ISP forwards the request for www.example.com to a DNS root name server.
4. The DNS resolver for the ISP forwards the request for www.example.com again, this time to one of the TLD name servers for .com domains. The name server for .com domains responds to the request with the names of the four Amazon Route 53 name servers that are associated with the example.com domain.
5. The DNS resolver for the ISP chooses an Amazon Route 53 name server and forwards the request for www.example.com to that name server.
6. The Amazon Route 53 name server looks in the example.com hosted zone for the www.example.com record, gets the associated value, such as the IP address for a web server, 192.0.2.44, and returns the IP address to the DNS resolver.
7. The DNS resolver for the ISP finally has the IP address that the user needs. The resolver returns that value to the web browser. The DNS resolver also caches (stores) the IP address for example.com for an amount of time that you specify so that it can respond more quickly the next time someone browses to example.com. For more information, see time to live (TTL).
8. The web browser sends a request for www.example.com to the IP address that it got from the DNS resolver. This is where your content is, for example, a web server running on an Amazon EC2 instance or an Amazon S3 bucket that's configured as a website endpoint.
9. The web server or other resource at 192.0.2.44 returns the web page for www.example.com to the web browser, and the web browser displays the page.



**Flow Diagram:**



**Why is DNS Important?**

* There should be an organised system to translate what users are searching for.

Without DNS services, we have to remember the website's IP address in order to visit them.

**Data Structures used in the program:**

1. N-ary Tree
2. Arrays of pointer nodes to store children of parent node

**Other approach:**

* **Trie data structure:** Trie is a sorted and efficient tree-based special data structure that is used to store and retrieve keys in a dataset of strings. It is based on the prefix of a string. Each node of a trie can have as many as 26 pointers/references. These 26 pointers represent the 26 characters of the English language.
* **Its disadvantage:** The main disadvantage of the trie is that it takes a lot of memory to store all the strings. For each node, we have too many node pointers which are equal to the no of characters in the worst case.

**Code:**

#include<bits/stdc++.h>

#define SIZE 5

using namespace std;

struct Node {

string domain;

string data;

Node\* child[SIZE];

};

string search(string domainList[], int index, Node\* node) {

if((node->domain == domainList[index]) && (index == 0))

return node->data;

else {

if(index < 0)

return "";

else

index -= 1;

for(int i = 0; i < SIZE; i++) {

if(node->child[i] != NULL) {

if(node->child[i]->domain == domainList[index])

return search(domainList, index, node->child[i]);

}

}

}

return "";

}

string update(string domainList[], int index, Node\* node, string str) {

if((node->domain == domainList[index]) && (index == 0)) {

node->data = str;

return node->data;

}

else {

if(index < 0)

return "";

else

index -= 1;

for(int i = 0; i < SIZE; i++) {

if(node->child[i] != NULL) {

if(node->child[i]->domain == domainList[index])

return update(domainList, index, node->child[i], str);

}

}

}

return "";

}

int main() {

//drive

Node greatGrandChild = {"drive", "8.5.2.0", NULL};

//google

Node grandChild1 = {"google", "1.2.3.4", {&greatGrandChild}};

//samsung

Node grandChild2 = {"samsung", "5.6.7.8", NULL};

//amazon

Node grandChild3 = {"amazon", "6.5.4.3", NULL};

//.com

Node child1 = {"com", "", {&grandChild1, &grandChild2}};

//.in

Node child2 = {"in", "", {&grandChild3}};

//head

Node root = {"", "", {&child1, &child2}};

int operation;

string query;

struct Node\* temp = NULL;

label: cout << "DNS Operations: \n1.Search \n2.Update \n";

cout << "Enter the number of operation you want to perform(else 0 to end): ";

cin >> operation;

while(operation != 0) {

cout << "Enter DNS domain: ";

cin >> query;

string arr[5];

int ct = -1;

string tmp = "";

for(int i = 0; i < query.size(); i++) {

tmp += query[i];

if(query[i] == '.') {

ct++;

tmp.pop\_back();

arr[ct] = tmp;

tmp = "";

}

}

ct++;

arr[ct] = tmp;

ct++;

arr[ct] += "";

if(operation == 1) {

string result = search(arr, ct, &root);

if(result != "")

cout << "DNS result is: " << result << endl << endl;

else

cout << "Domain not found!" << endl << endl;

goto label;

}

if(operation == 2) {

string newResult;

cout << "Enter new DNS value: ";

cin >> newResult;

string result = update(arr, ct, &root, newResult);

if(result != "")

cout << "New DNS result is: " << result << endl << endl;

else

cout << "Domain not found!" << endl << endl;

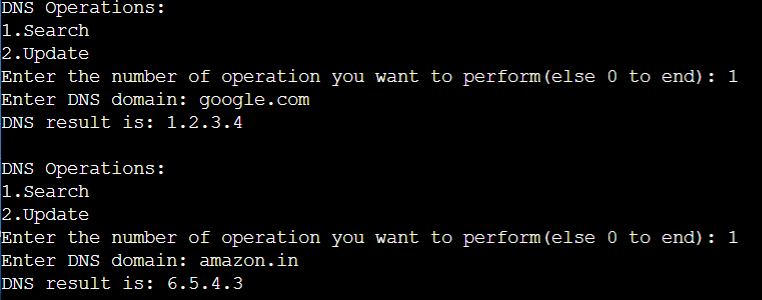
goto label;

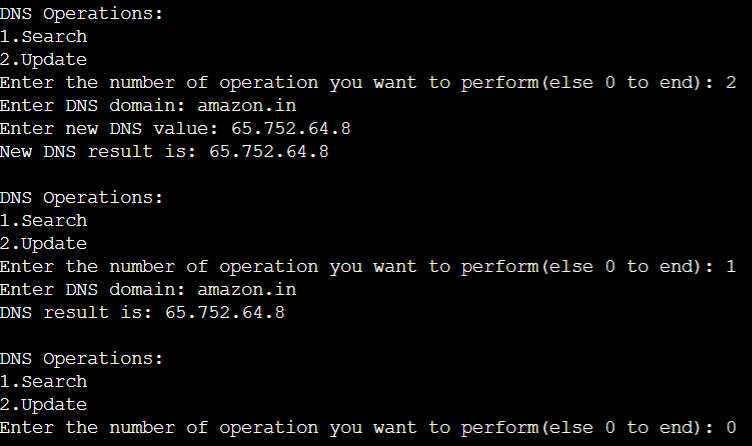
}

}

}

**Output:**

****

****