**DOCUMENTATION**

**Machine**

**Attributes:**

We have a total of **five** attributes. The **std::string hash** is the hash of the machine that the user will enter or it will auto-generated depending on the choice of the user. Then we have **the std::string name** of the machine that the user will enter. Then we have a **Machine \*next.** The purpose of this is to create the circular link list of the machine. The next attribute is **Files \*BTree**. This attribute is used to assign a B-Tree to the machine in which the files will be stored based on the hash of the files. The hash is calculated using SHA-1. The last but not the least attribute is RT\* head\_fingertable. As the name suggests This is the head to the fingertable that is used to jump to the next machine depending on the hash of the file.

**Functions:**

**Constructor:**

* Machine(std::string n, int m)

This constructor is used to auto-generate the hash of the machine using sha-1. The rest of the pointers are equal to nullptr. The name is set to n the user will tell and the order of the b-tree is equal to m.

* Machine(int m, int Bits\_n)

In this constructor, the order of the B-Tree is set, and the head\_fingertable and next are set to nullptr. The n is set to bits. Then for the size of the routing table n2 is set that takes the log to the base 2 of bit-space. Then the user enters the hash of the machine and the name of his own choice. A loop is used which validates that the hash has the correct length. If the length of the hash is less than the bits then a function fixlen() is called that appends zeros at the start of the hash to make its correct size.

**Other Functions:**

* void PrintBtree()

This function as the name suggests is used to print the B-Tree. It calls the function PrintTree() which is a function of the BTree class end that prints the B-Tree;

* void BitExtraction(string binary, int Bits, string& newHash)

In this function a specified number of bits from the end of binary are extracted that are saved in newHash and then returned. Its basic purpose is make sure that a proper size of binary is made so that no further problems arise.

* string hexaToBinary(const string& hash, int char\_no)

As the name suggests this function uses a switch statement that converts a hexadecimal number into binary.

* string binaryToHex(const string& binaryString)

As the name suggests this function uses a if-else-if to converts a binary number into hexadecimal number.

* string hash\_bits(int n)

The provided code is used to convert a big hash that comes from sha-1 into reduced hexadecimal then perfroms certain operation and return newHash. The explanation of those functions is given above.

* void fixlen(string& hash, int bit)

This function makes sure that the length of the string and the bits are of the same length. This is done by appending zeros at the start of the hash that is the reduced one.

* void add\_a\_file( string hash\_of\_file )

The functions add a file to a certain machine and inserts it in B-Tree as well.

**Routing Table**

* First, we have a **Node** class which consists of a machine node, and a previous and next pointer to make routing table a doubly linked list. Then, in the RT class we have a head pointer of Node type which stores the starting of the routing table of a particular machine. And we have two integer values one is the log of bit space (based of this value out routing table is generated) and a cur\_bit which works as a counter.
* Our main creation of routing table happens in updatefingerTable function where using the formula mention in project file we add two machine hash (using the function addHexString), then using the findwithHash function we decide in which machine we are inserting the file. Depending on the value of cur\_bit and bits we create a routingTable for one machine.
* The function addHexString takes two strings which are hexadecimal values, and these are added and returned. Furthermore, the function hashIT makes the hash of correct size for e.g. if a user enters a smaller hexadecimal, it transforms it into the correct size. Similarly, there is a function the subtracts two string hexadecimal values.

**RingDHT**

This is the main functioning class through which other classes are accessed, here using bits as identifier space we create a circular linked list of machines based on priority (depends on hashes). We further store the tail of the linked list i.e. the last value of linked list. Here we have two ways of insertion of machines one is manually where user inputs a hash value and the other way is automatic where user just inputs the name of the machine and the hash is calculated automatically using the SHA-1 functionality and the other is manual method where the user inputs a hash value as well as the name of the machine these machines are then inserted into the circular linked list.

* We further have two functions one in which hexadecimal value is converted to decimal (hexadecimaltoDecimal) and another in which decimal value is converted to hexadecimal value (decToHexa). Then there is a function to print the Machine names as well as their routing tables.
* The routing table is created using the function createRT which in a loop create routing table of all the machines using a function called updatefingertable from RT(routing table) class .
* Then we have three printing functions one is print\_machines prints hashes and name of the machine and the other is print\_machines\_RT() print machine names and hashes along with their routing tables and PrintBtree\_of\_a\_machine() which prints the BTree of machines containing files
* A parameterized constructors that sets the head and tail to nullptr and initializes the value of bitsVar1 and bitsVar2 and order\_m which is the order of the m-ary tree.
* A search function called Search\_a\_hash() which takes a string i.e. hash of the file and searches it in appropriate B-tree.

**FileTree**

**Attributes:**

It has a root pointer that is used to point to the head of files. Then we have m that determines the order of the B-Tree. Then we have the max and min double variable that saves the max and min child of a B-Tree.

**Functions:**

* Files( int m = 5 )

This is a parametrized constructor that takes the order of B-Tree as a parameter and assigns it to m. Then by using the formula the max and min are calculated for later use.

* void insert\_a\_file ( string hash , string content)

This function calls the insert function from the BTNode struct. For inserting the file a hash and content are taken from the main. cpp and sent it to be inserted in the B-Tree.

* void PrintTree()

As the name suggests, this function is used to print the B-Tree. It calls the display function from the BTNode.

* void Delete\_a\_file ( string hash )

For deleting the file the function takes the hash of the file in input to know which file is to be deleted and then calls del function from BTNode to delete it and then returns the deleted node from the function.

* void Searching (string hash)

As the name suggests this function takes the hash of the string to find if the file exists or not in the BTree and displays the corresponding statement.

**Filehandling**

**Functions:**

* void createTextFile(const string & str)

In this function, by using the ofstream header file we create a new file in our directory and then the file is closed in the second line.

* string ReadTextFile (const string& str)

In this function, an empty string toReturn is created in which all the content of the file is saved. In this function, a while loop is used which is used to store endl in the file in a string. In the end, the file is closed and the string is returned with the content of the file.

* void writeFile(const string& str , const string & content)

In this, the str is the path of the file, and content is passed as a parameter. Then for the file, the content is replaced and then the file is closed.

**Filehandling**

**Functions:**

* bool createDirectory(const std::string& folderName)

In this function, as the name suggests a directory is created on runtime and bool tells us whether a file is created or it failed to create a file.

* bool deleteFolder(const std::string& folderName)

In this function as the name suggests a directory at the run time is deleted and then the function is returned.