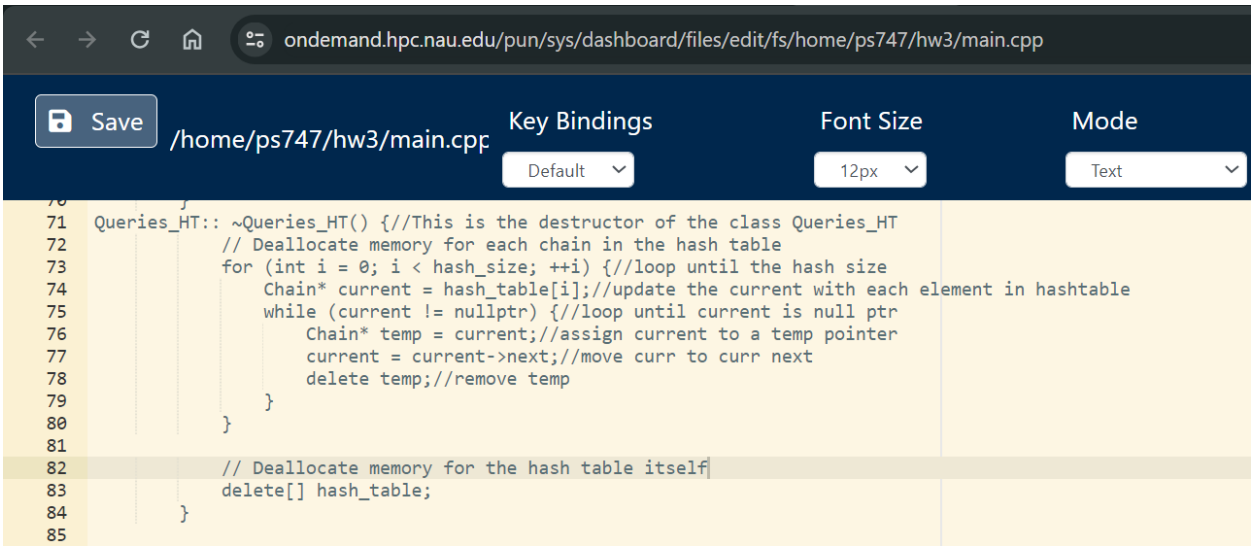
Homework #3

Name: Purnabhishek Sripathi
User id: 6274051
Email: ps747@nau.edu

Problem #1 (of 1) => A constructor

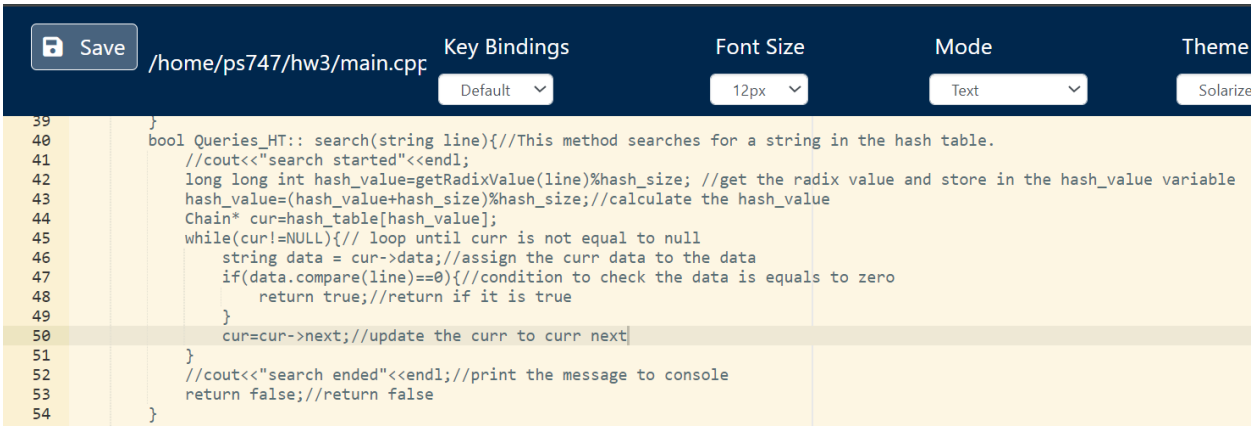

```
4   // This is the constructor of the class Queries_HT which initializes the hash table
5   Queries_HT::Queries_HT(int size, const std::string& file_path) : hash_size(size), hash_table(new Chain*[size]) {
6           // Initialize all buckets with nullptr
7                   for (int i = 0; i < hash_size; ++i) {//This loop initializes each bucket in the hash table to nullptr.
8                       hash_table[i] = nullptr;
9                   }
10
11                  // Open the file
12                  ifstream file(file_path);
13                  if (!file.is_open()) {//condition to check the file is open or not
14                      cerr << "Error opening file " << file_path << std::endl;//show error in opening file
15                      return;//return
16                  }
17
18                  // Read the query file and insert elements into the hash table
19                  string line;//initialize the string variable line
20                  while (getline(file, line)) { // loop to get the each line from the file
21                      if(line[0]!='>'){
22                      insertion(line);
23                      }
24                  }
25
26                file.close();//close the file
27          }
```

A destructor



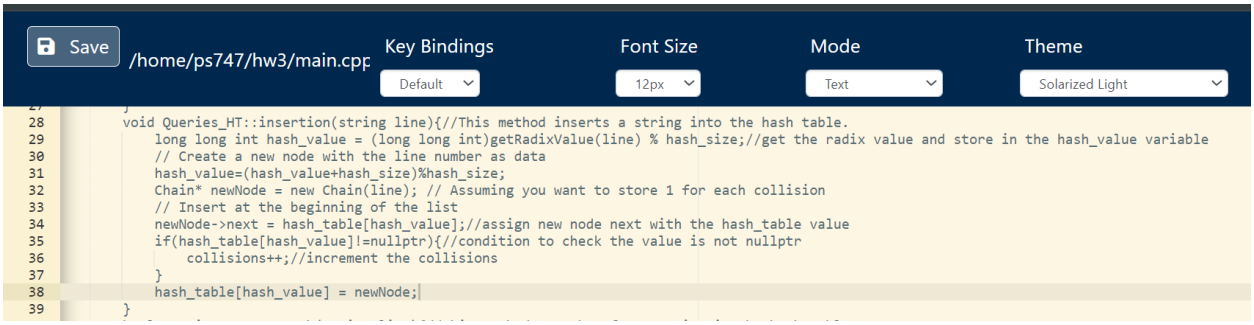

```
71  Queries_HT:: ~Queries_HT() {//This is the destructor of the class Queries_HT
72              // Deallocate memory for each chain in the hash table
73              for (int i = 0; i < hash_size; ++i) {//loop until the hash size
74                  Chain* current = hash_table[i];//update the current with each element in hashtable
75                  while (current != nullptr) {//loop until current is null ptr
76                      Chain* temp = current;//assign current to a temp pointer
77                      current = current->next;//move curr to curr next
78                      delete temp;//remove temp
79                  }
80              }
81
82              // Deallocate memory for the hash table itself
83              delete[] hash_table;
84          }
85
```

A function to search the hash table for a given n-mer sequence (returning a presence/absence Boolean value should be sufficient)



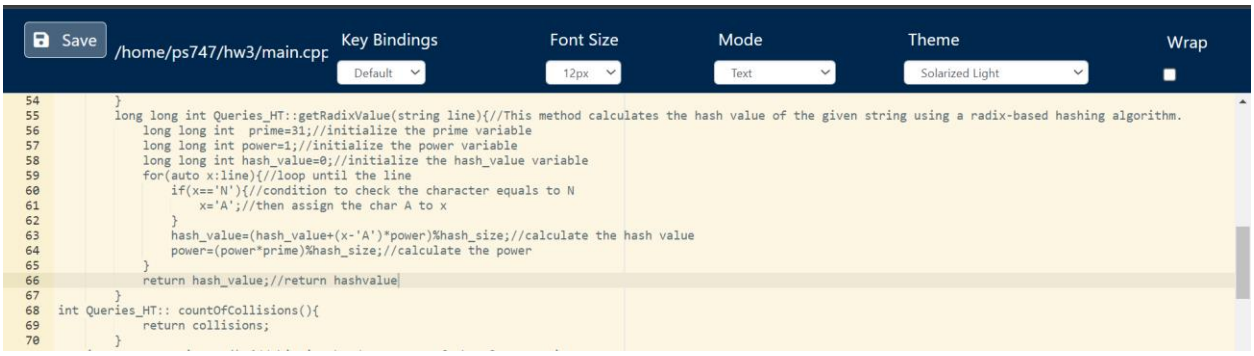

```
39          }
40          bool Queries_HT:: search(string line){//This method searches for a string in the hash table.
41              //cout<<"search started"<<endl;
42              long long int hash_value=getRadixValue(line)%hash_size; //get the radix value and store in the hash_value variable
43              hash_value=(hash_value+hash_size)%hash_size;//calculate the hash_value
44              Chain* cur=hash_table[hash_value];
45              while(cur!=NULL){// loop until curr is not equal to null
46                  string data = cur->data;//assign the curr data to the data
47                  if(data.compare(line)==0){//condition to check the data is equals to zero
48                      return true;//return if it is true
49                  }
50                  cur=cur->next;//update the curr to curr next
51              }
52              //cout<<"search ended"<<endl;//print the message to console
53              return false;//return false
54          }
```

A function to insert a given n-mer sequence into the hash table.




```
28          void Queries_HT::insertion(string line){//This method inserts a string into the hash table.
29              long long int hash_value = (long long int)getRadixValue(line) % hash_size;//get the radix value and store in the hash_value variable
30              // Create a new node with the line number as data
31              hash_value=(hash_value+hash_size)%hash_size;
32              Chain* newNode = new Chain(line); // Assuming you want to store 1 for each collision
33              // Insert at the beginning of the list
34              newNode->next = hash_table[hash_value];//assign new node next with the hash_table value
35              if(hash_table[hash_value]!=nullptr){//condition to check the value is not nullptr
36                  collisions++;//increment the collisions
37              }
38              hash_table[hash_value] = newNode;
39          }
```

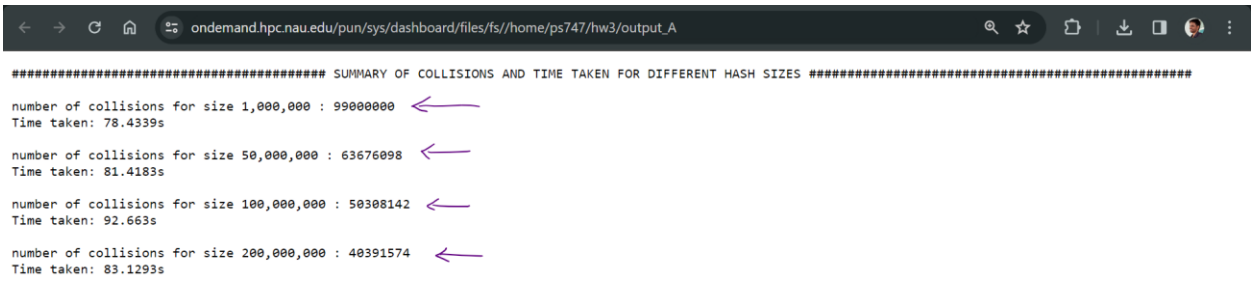
A function to convert a given sequence to a Radix notation (use *double* or *unsigned int* data type to store the radix value)



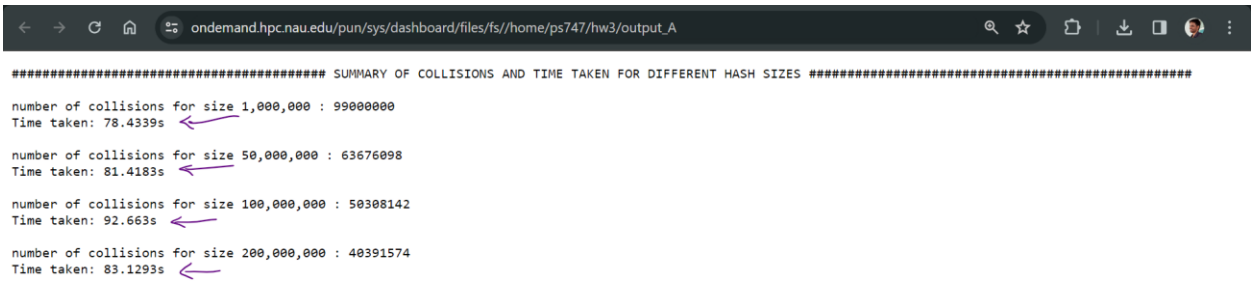

```
54          }
55          long long int Queries_HT::getRadixValue(string line){//This method calculates the hash value of the given string using a radix-based hashing algorithm.
56              long long int  prime=31;//initialize the prime variable
57              long long int power=1;//initialize the power variable
58              long long int hash_value=0;//initialize the hash_value variable
59              for(auto x:line){//loop until the line
60                  if(x=='N'){//condition to check the character equals to N
61                      x='A';//then assign the char A to x
62                  }
63                  hash_value=(hash_value+(x-'A')*power)%hash_size;//calculate the hash value
64                  power=(power*prime)%hash_size;//calculate the power
65              }
66              return hash_value;//return hashvalue
67          }
68      int Queries_HT:: countOfCollisions(){
69              return collisions;
70          }
```

A. (30 pts) Assess the impact of the hash table size:

1. For each of your 4 hash table sizes, how many collisions did you observe while populating the hash?



2. For each of your 4 hash table sizes, how long did it take you to populate the hash table? Do the timing results make sense (provide big O notation)? Explain.



When we doubled the hash size, the running time also doubled. **So, the time complexity is O(N).** Here the running time is dependent on the hash size because we are initializing the hash table with the nullptr, we are doing this because we are using the dynamic hash table.

B. (30 pts) Searching speed:

Q: How long did it take to search for every possible 16-character-long fragment of the *subject dataset* within the *query dataset*?

A: The time taken to search for every possible 16-character is: 2168.87s
Approximately 36.148 **minutes.**

Q: How many such fragments did you find?

A: I found **848542890** fragments.

Q: Print the first 10 fragments of the *subject dataset* that you found within the Query_HT.