NAME:	Eshan Bhuse
UID:	2021300013
SUBJECT	Design and analysis of algorithm
EXPERIMEN TNO:	10
AIM:	Experiment 10 on string matching algorithms (Rabin-Karp and Naive).

ALGORITHM:

Rabin Karp Algorithm:

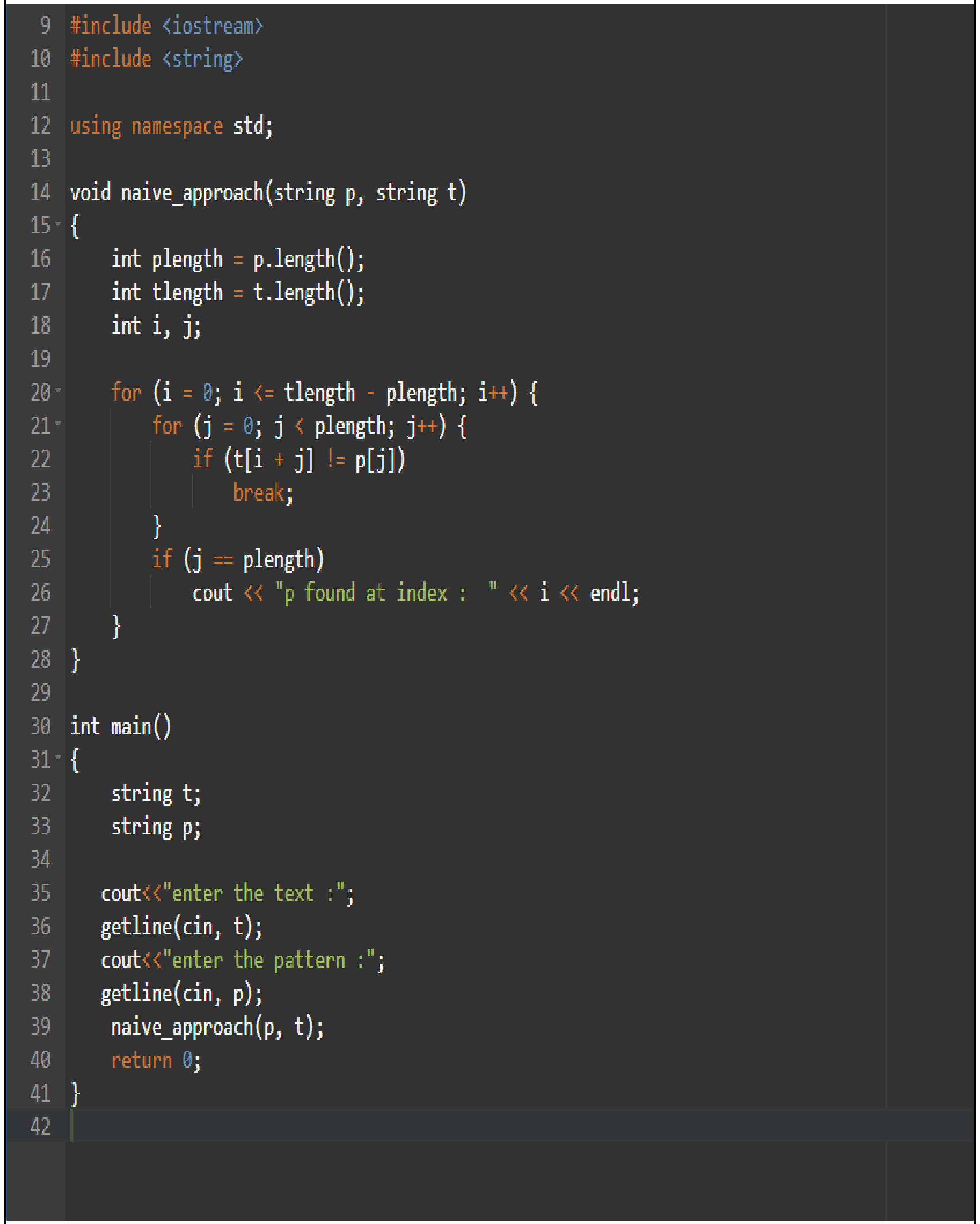
- $n \leftarrow$ length [T]
- $m \leftarrow$ length [P]
- $h \leftarrow d^{m-1}$ mod q
- $p \leftarrow 0$
- $t_0 \leftarrow 0$
- for $i \leftarrow 1$ to m
- do $p \leftarrow (dp + P[i])$ mod q
- $t_0 \leftarrow (dt_0+T[i])$ mod q
- for $s \leftarrow 0$ to n-m
- do if $p = t_s$
- then if P [1.....m] = T [s+1.....s + m]
- then "Pattern occurs with shift" s
- If s < n-m
- $t_{s+1} \leftarrow (d(t_s-T[s+1]h)+T[s+m+1])$mod q

Naive-String-Matcher (T, P):

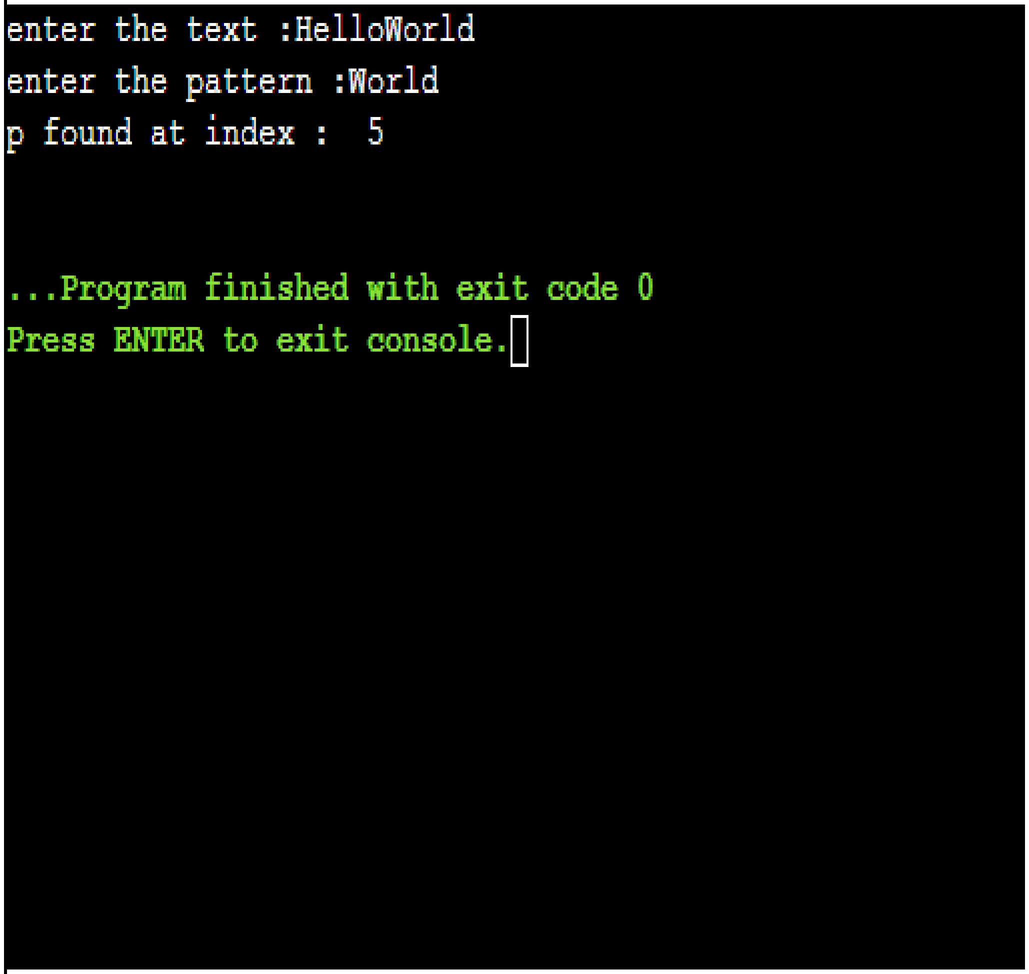
- $n \leftarrow$ length [T]
- $m \leftarrow$ length [P]
- for $s \leftarrow 0$ to n -m
- do if P [1.....m] = T [s + 1....s + m]
- then print "Pattern occurs with shift" s

PROGRAM:

Naive Program

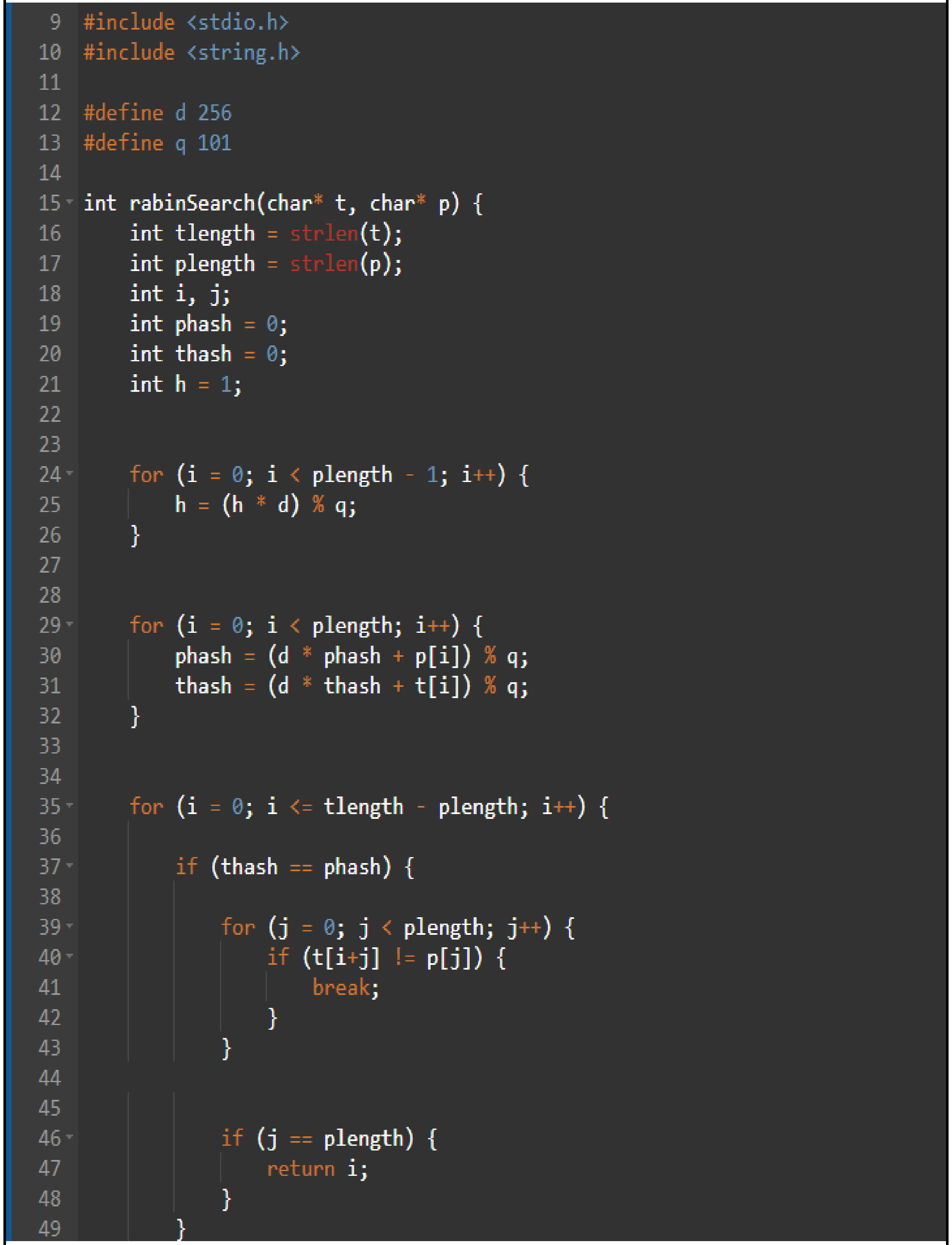


```cpp
#include <iostream>
#include <string>

using namespace std;

void naive_approach(string p, string t)
{
    int plength = p.length();
    int tlength = t.length();
    int i, j;

    for (i = 0; i <= tlength - plength; i++) {
        for (j = 0; j < plength; j++) {
            if (t[i + j] != p[j])
                break;
        }
        if (j == plength)
            cout << "p found at index :  " << i << endl;
    }
}

int main()
{
    string t;
    string p;

   cout<<"enter the text :";
   getline(cin, t);
   cout<<"enter the pattern :";
   getline(cin, p);
    naive_approach(p, t);
    return 0;
}
```

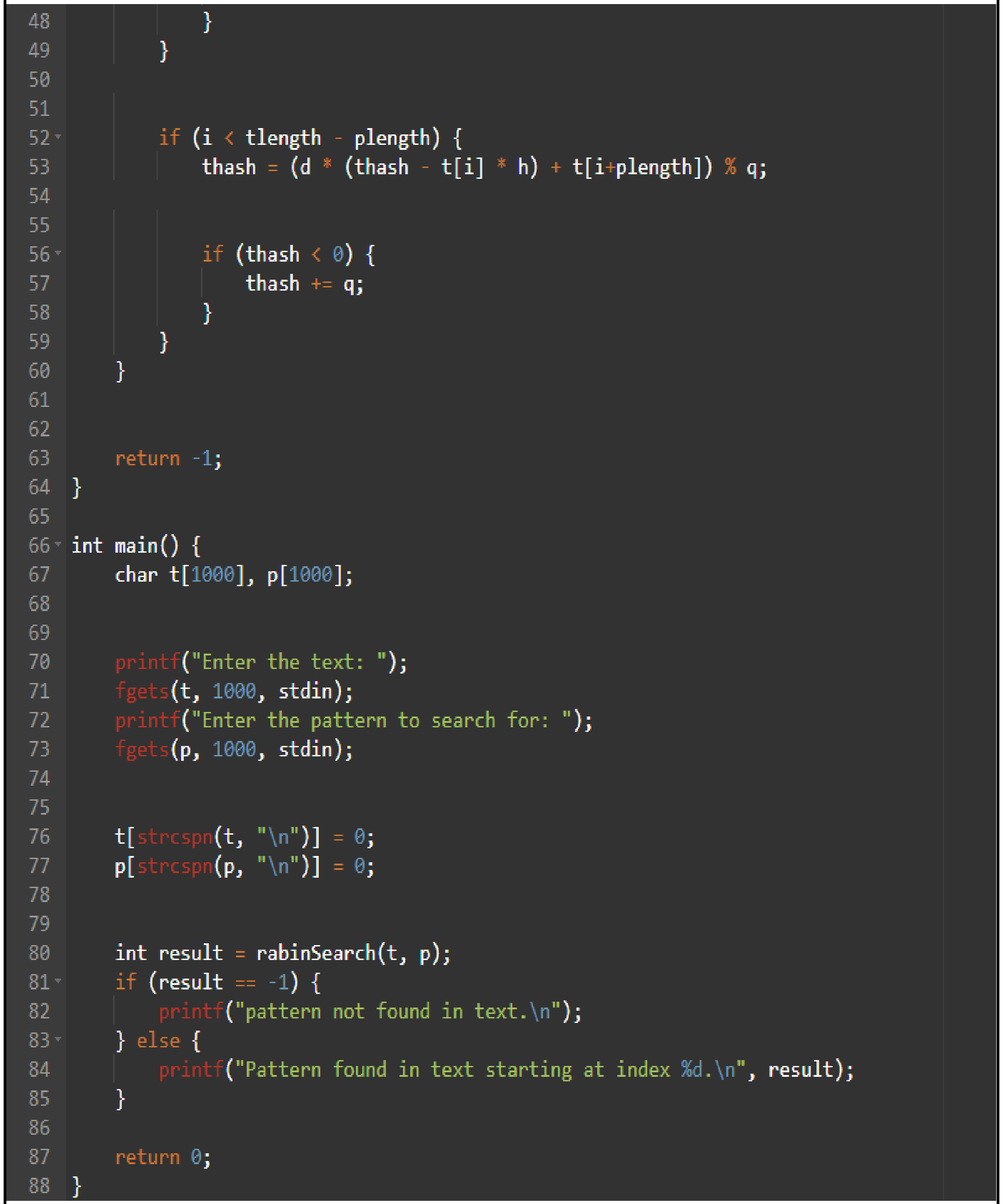


```
enter the text :HelloWorld
enter the pattern :World
p found at index :  5


...Program finished with exit code 0
Press ENTER to exit console.
```

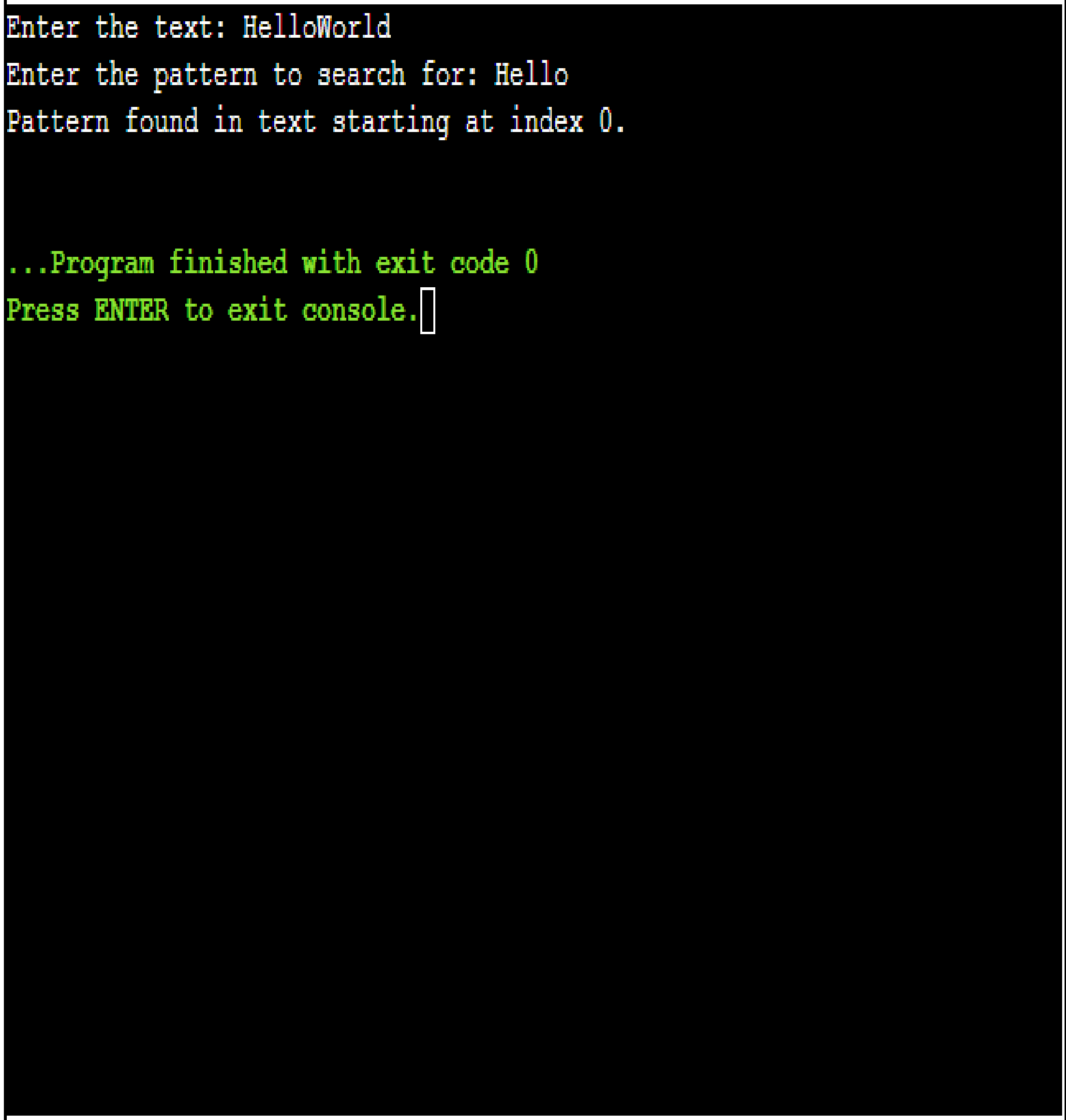
Rabin-Karp Program



```c
#include <stdio.h>
#include <string.h>

#define d 256
#define q 101

int rabinSearch(char* t, char* p) {
    int tlength = strlen(t);
    int plength = strlen(p);
    int i, j;
    int phash = 0;
    int thash = 0;
    int h = 1;


    for (i = 0; i < plength - 1; i++) {
        h = (h * d) % q;
    }


    for (i = 0; i < plength; i++) {
        phash = (d * phash + p[i]) % q;
        thash = (d * thash + t[i]) % q;
    }


    for (i = 0; i <= tlength - plength; i++) {

        if (thash == phash) {

            for (j = 0; j < plength; j++) {
                if (t[i+j] != p[j]) {
                    break;
                }
            }


            if (j == plength) {
                return i;
            }
        }
```



```c
            }
        }


        if (i < tlength - plength) {
            thash = (d * (thash - t[i] * h) + t[i+plength]) % q;


            if (thash < 0) {
                thash += q;
            }
        }
    }


    return -1;
}

int main() {
    char t[1000], p[1000];


    printf("Enter the text: ");
    fgets(t, 1000, stdin);
    printf("Enter the pattern to search for: ");
    fgets(p, 1000, stdin);


    t[strcspn(t, "\n")] = 0;
    p[strcspn(p, "\n")] = 0;


    int result = rabinSearch(t, p);
    if (result == -1) {
        printf("pattern not found in text.\n");
    } else {
        printf("Pattern found in text starting at index %d.\n", result);
    }

    return 0;
}
```

```
Enter the text: HelloWorld
Enter the pattern to search for: Hello
Pattern found in text starting at index 0.


...Program finished with exit code 0
Press ENTER to exit console.
```

OBSERVATION:	Thus, we observe that by using the hash function and modulus we can avoid matching each and every character in the string and the pattern like in the naive method and only compare the result of the mod with the mod of the pattern which reduces the number of computations as well as running time significantly.
CONCLUSION:	Thus, after performing this experiment I understood and implemented Rabin Karp pattern matching algorithm as well as naive method.