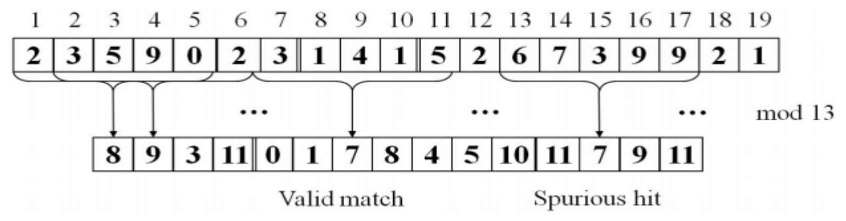
Experiment Number: 02

Problem Statement:-Implementation of Rabin Karp Algorithm.

Aim:-Write a program to implement Rabin Karp Algorithm.

Theory:-
Rabin-Karp string searching algorithm calculates a numerical (hash) value for the pattern p, and for each m-character substring of text t. Then it compares the numerical values instead of comparing the actual symbols. If any match is found, it compares the pattern with the substring by naive approach. Otherwise it shifts to next substring of t to compare with p. We can compute the numerical (hash) values using Horner's rule. Lets assume, h0 = k h1 = d k − p [1] .dm−1 + p [m + 1] Suppose, we have given a text t = [3, 1, 4, 1, 5, 2] and m = 5, q = 13; t0 = 31415 So t1 = 10(31415 - 105−1 .t[1]) + t[5+1] = 10(31415 − 104 .3) + 2 = 10(1415) + 2 = 14152 Here p and substring ti may be too large to work with conveniently. The simple solution is, we can compute p and the ti modulo a suitable modulus q. So for each i, hi+1 = (d hi − t[i + 1] .dm−1 + t[m + i + 1]) mod q The modulus q is typically chosen as a prime such that d.q fits within one computer word.



This algorithm has a significant improvement in average-case running time over naive approach.

Algorithm:

```
Compute hp (for pattern p)
 Compute ht (for the first substring of t with m length)
 For i = 1 to n − m
If hp = ht
Match t[i . . . .i + m] with p, if matched return 1
Else
ht = (d ht − t[i + 1] .dm−1  + t[m + i + 1]) mod q
End
```

Suppose, t= 2359023141526739921 and p = 31415,
Now, hp = 7 (31415 = 7 (mod 13)) substring beginning at position 7 = valid match

NAME:HARSHBHAI SOLANKI
ROLL.N0:62
SE-4-D

EXPERIMENT N0-2

AIM: Implementation of Rabin Karp algorithm.

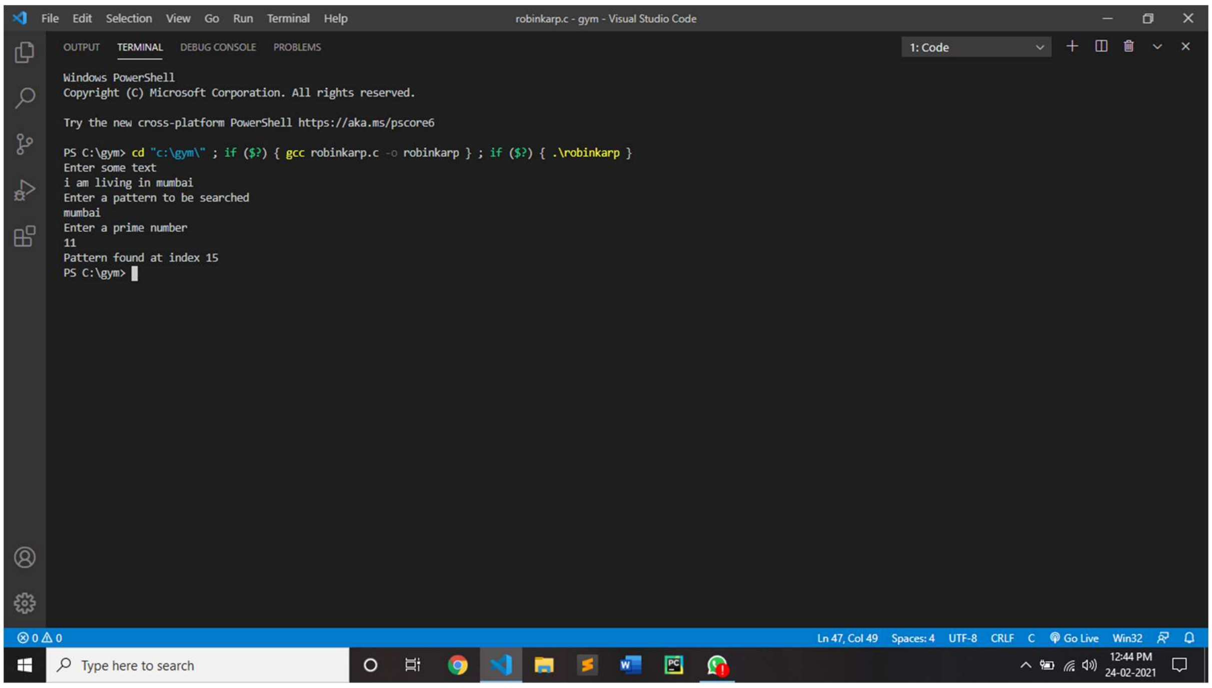
CODE:

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

void search(char pat[], char txt[], int q)
{
   int M = strlen(pat);
   int N = strlen(txt);
   int i, j;
   int p = 0;
   int t = 0;
   int h = 1;
   int d=256;
   for (i = 0; i < M-1; i++)
      h = (h*d)%q;
   for (i = 0; i < M; i++)
   {
      p = (d*p + pat[i])%q;
      t = (d*t + txt[i])%q;
   }
   for (i = 0; i <= N - M; i++)
   {
      if ( p == t )
      {
         for (j = 0; j < M; j++)
```

```
            {
               if (txt[i+j] != pat[j])
                   break;
            }
            if (j == M)
               printf("Pattern found at index %d \n", i);
        }
        if ( i < N-M )
        {
            t = (d*(t - txt[i]*h) + txt[i+M])%q;
            if (t < 0)
            t = (t + q);
        }
    }
}
int main()
{   char txt[80],pat[80];
    int q;
    printf("Enter some text \n");
    gets(txt);
    printf("Enter a pattern to be searched \n");
    gets(pat);
    printf("Enter a prime number \n");
    scanf("%d",&q);
    search(pat, txt, q);
    return 0;
}
```

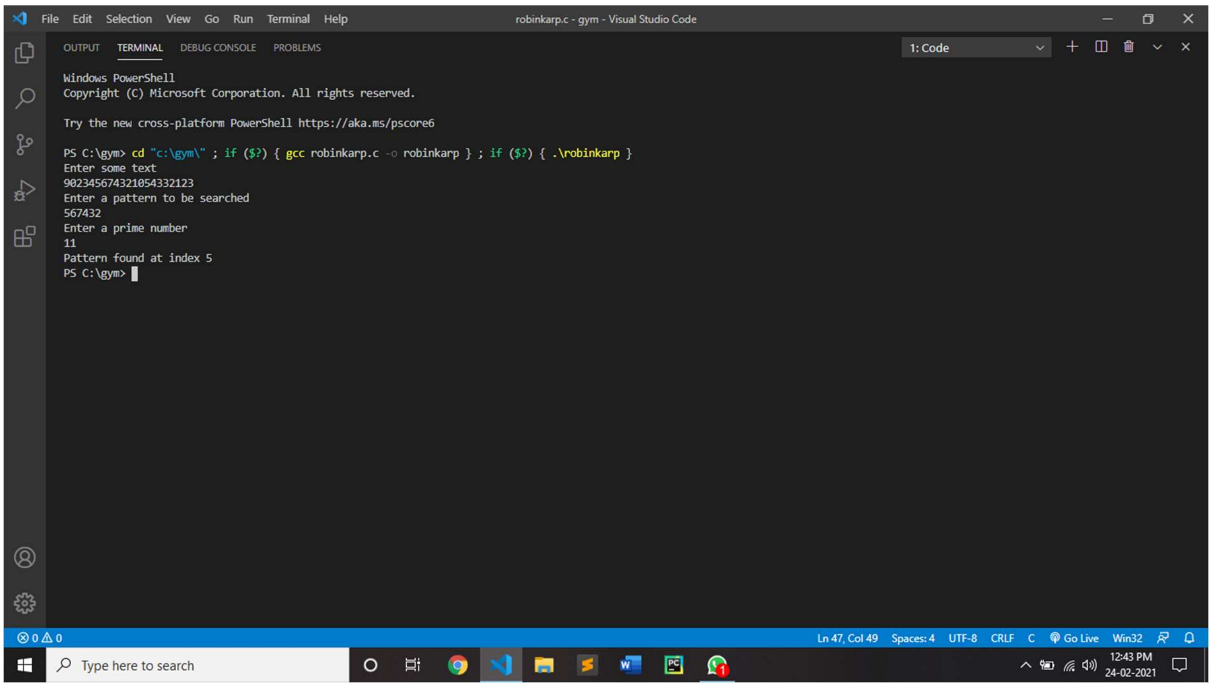
OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\gym> cd "c:\gym\" ; if ($?) { gcc robinkarp.c -o robinkarp } ; if ($?) { .\robinkarp }
Enter some text
i am living in mumbai
Enter a pattern to be searched
mumbai
Enter a prime number
11
Pattern found at index 15
PS C:\gym>
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\gym> cd "c:\gym\" ; if ($?) { gcc robinkarp.c -o robinkarp } ; if ($?) { .\robinkarp }
Enter some text
90234567432105433212З
Enter a pattern to be searched
567432
Enter a prime number
11
Pattern found at index 5
PS C:\gym>
```

Conclusion:

By performing above algorithm we can concluded that,The normal and best-case running season of the Rabin-Karp calculation is O(n+m), however its most pessimistic scenario time is O(nm). Worst case of Rabin-Karp calculation happens when all characters of example and text are same as the hash estimations of the multitude of substrings of txt[] coordinate with hash estimation of pat[]. For instance pat[] = "AAA" and txt[] = "AAAAAAA".