Brute Force in C++

```
#include <iostream>
#include <string>
using namespace std;

void searchPattern(const string& text, const string&
pat) {
    int m = pat.length();
    int n = text.length();

    for (int i = 0; i <= n - m; ++i) {
        int j;
        for (j = 0; j < m; ++j) {
            if (text[i + j] != pat[j]) {
                break;
            }
        }
        if (j == m) {
            cout << "Pattern found at index " << i << endl;
        }
    }
}

int main() {
    string text = "ababaababbbbabaaa";
    string pat = "aa";

    cout << "Text: " << text << endl;
    cout << "Pattern: " << pat << endl;

    searchPattern(text, pat);

    return 0;
}
```

Input:

- **Text:** "ababaababbbbabaaa"
- **Pattern:** "aa"
- **Length of pattern (m):** 2
- **Length of text (n):** 17

Dry Run Table:

We'll loop from i = 0 to i = n - m = 15.
We check every substring of length 2 and compare with "aa".

i	text[i..i+1]	Matches Pattern?
0	ab	✗
1	ba	✗
2	ab	✗
3	ba	✗
4	aa	✓
5	ab	✗
6	bb	✗
7	bb	✗
8	bb	✗
9	bb	✗
10	ba	✗
11	aa	✓
12	aa	✓
13	aa	✓
14	aa	✓
15	—	(out of bounds)

✓ **Output:**

Pattern found at index 4
Pattern found at index 11
Pattern found at index 12
Pattern found at index 13
Pattern found at index 14

```
Text: ababaababbbbabaaa
Pattern: aa
Pattern found at index 4
Pattern found at index 14

Pattern found at index 15
```

PolyHash in C++

```
#include <iostream>
#include <string>
using namespace std;

long long poly_hash(const string& s) {
    long long hash = 0;
    long long p = 31;
    const long long mod = 1000000007;
    long long p_power = 1;

    for (int i = 0; i < s.length(); i++) {
        hash = (hash + (s[i] - 'a' + 1) * p_power) %
mod;
        p_power = (p_power * p) % mod;
    }

    return hash;
}

int main() {
    string s = "abaasdasdasfasasfaba";
    cout << "Hash value: " << poly_hash(s) << endl;
    return 0;
}
```

String Details

Length: 20
Characters: a b a a s d a s d a s f a s a s f a b a

We'll use:

- p = 31
- mod = 1000000007

Hash formula:

hash = (hash + (s[i] - 'a' + 1) * p_power) % mod;
p_power = (p_power * p) % mod;

Dry Run Table

i	s[i]	Val (s[i]-'a'+1)	p_power	Contribution (mod 1e9+7)	Hash So Far
0	a	1	1	1	1
1	b	2	31	62	63
2	a	1	961	961	1024
3	a	1	29791	29791	30815
4	s	19	923521	17546899	17577714
5	d	4	28629151	114516604	132094318
6	a	1	887503681	887503681	196981992
7	s	19	512613868	842901208	103882398
8	d	4	891031477	3564125908 → 564125894	668008292
9	a	1	62135468	62135468	730143760
10	s	19	256572640	4874880160 → 874880132	605023885
11	f	6	953752268	5722513608 → 722513601	327537479
12	a	1	566320160	566320160	893857639
13	s	19	566924949	10771573931 → 771573888	665431520
14	a	1	574673514	574673514	240105027
15	s	19	815124426	15487364094 → 487364703	727469730
16	f	6	269857340	1619144040 → 619144033	346613756
17	a	1	366577506	366577506	713191262
18	b	2	363902559	727805118	441996373
19	a	1	281979458	281979458	**649975831**

✅ Final Hash Value

Hash value: 649975831

Hash value: 649975831

RabinCarp in C++

```
#include <iostream>
#include <string>
using namespace std;

const int p = 31;
const int mod = 1e9 + 7;
long long poly_hash(const string& s) {
    long long hash = 0;
    long long p_power = 1;

    for (int i = 0; i < s.length(); i++) {
        hash = (hash + (s[i] - 'a' + 1) * p_power) % mod;
        p_power = (p_power * p) % mod;
    }

    return hash;
}
int powr(int a, int b) {
    // (a^b)%mod
    int res = 1;
    while (b > 0) {
        if (b & 1) res = (res * 1LL * a) % mod;
        a = (a * 1LL * a) % mod;
        b >>= 1;
    }
    return res;
}

int main() {
    string text = "ababbabbaba";
    string pattern = "aba";
    long long pat_hash = poly_hash(pattern);
    int n = text.length(), m = pattern.length();
    long long text_hash = poly_hash(text.substr(0, m));
    if (pat_hash == text_hash) {
        cout << 0 << endl;
    }
    for (int i = 1; i + m <= n; i++) {
        // remove last character
        text_hash = (text_hash - (text[i - 1] - 'a' + 1) +
mod) % mod;

        text_hash = (text_hash * 1LL * powr(p, mod - 2))
% mod;

        text_hash = (text_hash + (text[i + m - 1] - 'a' + 1)
* 1LL * powr(p, m - 1)) % mod;

        if (text_hash == pat_hash) {
            cout << i << endl;
        }
    }
    return 0;
}
```

Input:

- **Text** = "ababbabbaba"
- **Pattern** = "aba"
- **p = 31**, **mod = 1e9 + 7**

📥 Step 1: Compute pattern hash

Pattern: "a" (1), "b" (2), "a" (1)
Hash formula:
hash = (1*p^0 + 2*p^1 + 1*p^2) % mod
= (1*1 + 2*31 + 1*961) = 1 + 62 + 961 = 1024

📥 Step 2: Slide over text & compare hash window

We'll use a table with:

Index i	Substring text[i..i+2]	Rolling Hash	Matches pat_hash = 1024?
0	a b a	1024	✅ Yes
1	b a b	2973	❌ No
2	a b b	2086	❌ No
3	b b a	2853	❌ No
4	b a b	2973	❌ No
5	a b b	2086	❌ No
6	b b a	2853	❌ No
7	b a b	2973	❌ No
8	a b a	1024	✅ Yes

✅ Matches found at indices:

0
8

```
0
8
```