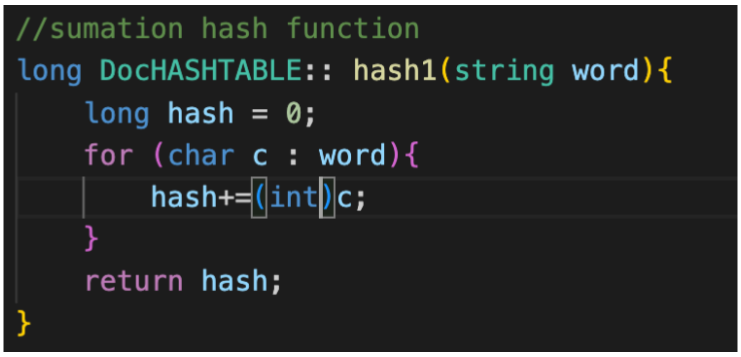
Assignment 3 – Hash Functions Analysis

Hash function 1 – summation

```
**Algorithm** – summation hash function
**Input** – string word
**Output** – hash code

hashcode = 0
for each character c in word do
    hashcode = hashcode + (askii of c)
return hashcode
```

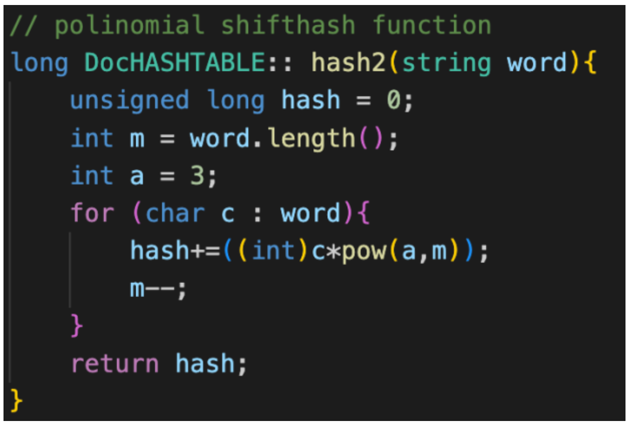


```cpp
//sumation hash function
long DocHASHTABLE:: hash1(string word){
    long hash = 0;
    for (char c : word){
        hash+=(int)c;
    }
    return hash;
}
```

Hash function 2 – polynomial

```
Algorithm – polynomial hash function
Input – word
Output – hashcode

hashcode = 0
length = length of word
constant = 3
For each character c in word do
    hashcode = (askii of c) * (constant^length)
    Length = length -1
Return hashcode
```

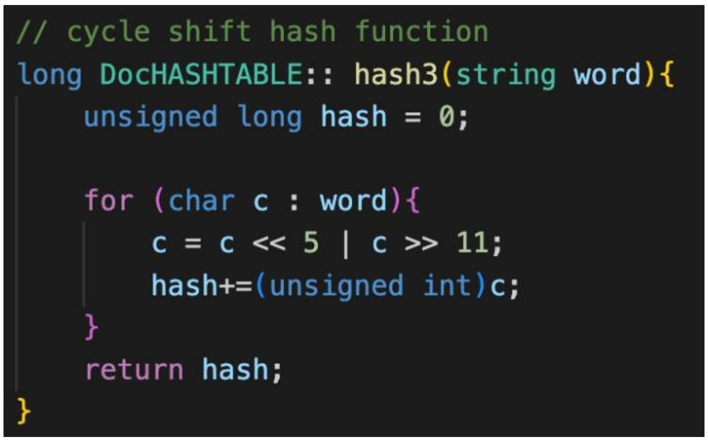


```cpp
// polinomial shifthash function
long DocHASHTABLE:: hash2(string word){
    unsigned long hash = 0;
    int m = word.length();
    int a = 3;
    for (char c : word){
        hash+=((int)c*pow(a,m));
        m--;
    }
    return hash;
}
```

Hash function 3 – standard cycle shift

Algorithm – cycle shift hash function
Input – word
Output – hashcode

```
hashcode = 0
For each character c in word do
        c = Bitwise shift to the left by 5
        c = bitwise shift to the right by 11
        hashcode = hashcode + (updated bit value of c)
return hashcode
```



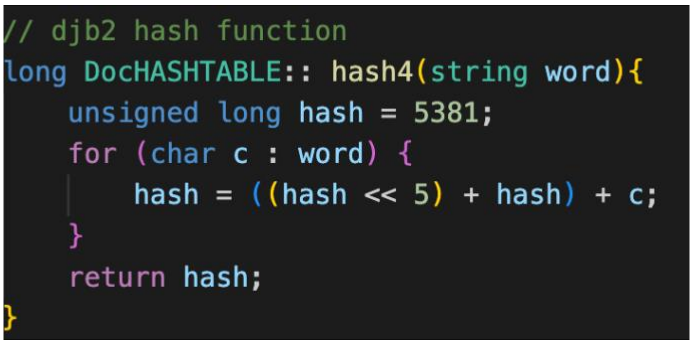
```cpp
// cycle shift hash function
long DocHASHTABLE:: hash3(string word){
    unsigned long hash = 0;

    for (char c : word){
        c = c << 5 | c >> 11;
        hash+=(unsigned int)c;
    }
    return hash;
}
```

Hash function 4 – djb2 Hash Function

Algorithm – djb2 hash function
Input – word
Output – hashcode

```
hashcode = 5381
For each character c in word do
        Hashcode = hashcode + (bitwise left-shift by 5 on hashcode) + character c
return hashcode
```

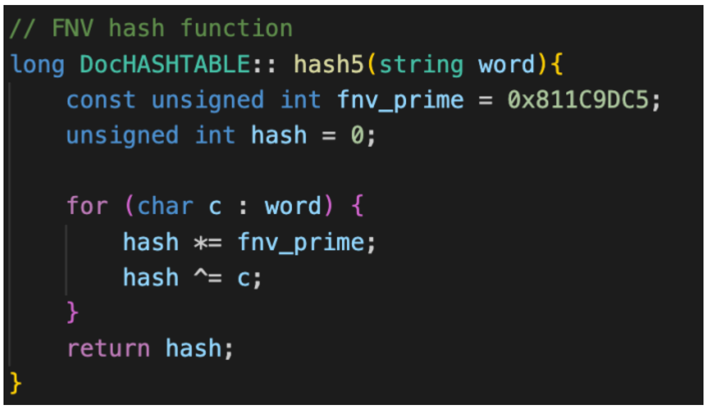


```
// djb2 hash function
long DocHASHTABLE:: hash4(string word){
    unsigned long hash = 5381;
    for (char c : word) {
        hash = ((hash << 5) + hash) + c;
    }
    return hash;
}
```

Hash function 5 – FNV hash function

Algorithm – FNV hash
Input – word
Output – hashcode

```
Constant int fnv_prime = 0x811C9DC5
Hashcode = 0
For each character c in word do
        Hashcode = Hashcode * fnv_prime
        Hashcode = Hashcode ^ c
Return Hashcode
```



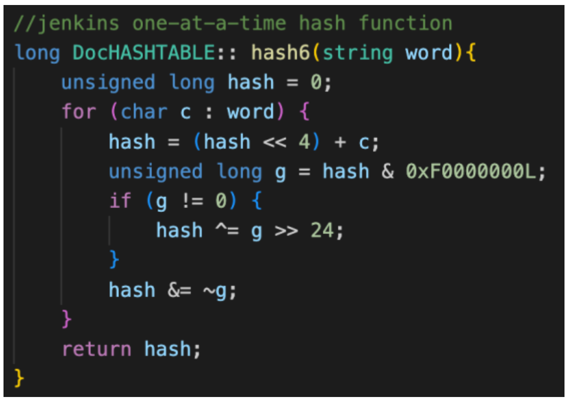
```
// FNV hash function
long DocHASHTABLE:: hash5(string word){
    const unsigned int fnv_prime = 0x811C9DC5;
    unsigned int hash = 0;

    for (char c : word) {
        hash *= fnv_prime;
        hash ^= c;
    }
    return hash;
}
```

Hash function 6 – Jenkins one-at-a-time Hash Function

Algorithm – Jenkins hash
Input – word
Output – hashcode

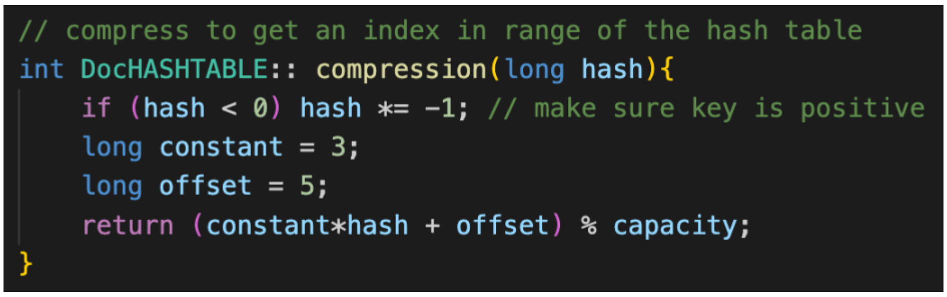
```
Hashcode = 0
For each character c in word do
        Hashcode = (bitwise left shift of 4 on Hashcode) + c
        Second_shift = bitwise ‘and’ operation on Hashcode and ‘0xF0000000L’
        If Second_shift != 0 then:
                Hashcode = Hashcode ^ (bitwise right shift of 24 on Hashcode)
        Hashcode = bitwise ‘and’ operation on Hashcode and ~Second_shift
Return Hashcode
```



```
//jenkins one-at-a-time hash function
long DocHASHTABLE:: hash6(string word){
    unsigned long hash = 0;
    for (char c : word) {
        hash = (hash << 4) + c;
        unsigned long g = hash & 0xF0000000L;
        if (g != 0) {
            hash ^= g >> 24;
        }
        hash &= ~g;
    }
    return hash;
}
```

Compression Function

Algorithm – compression
Input – Hashcode
Output – index in hashtable
constant = 3
offset = 5
Return (constant*Hashcode + offset) % (capacity of Hashtable)



```
// compress to get an index in range of the hash table
int DocHASHTABLE:: compression(long hash){
    if (hash < 0) hash *= -1; // make sure key is positive
    long constant = 3;
    long offset = 5;
    return (constant*hash + offset) % capacity;
}
```

Capacity

Algorithm – getCapacity
Input – file path
Output – optimal capacity for table

```
DocHASHTABLE hashTable
hashTable.import(path)
capacity = (number of unique words in hashtable) *125%

while (is_not_prime(capacity)) do
        capacity = capacity +1
return capacity
```

Default Hash Function

Hash Function	Average Collisions	No. of files (with min collisions)
Summation - 1	5942	0
Polinomial Shift - 2	2474	3
Cyclic shift -3	6745	0
djb2 - 4	2196	30
FNV - 5	2189	40
Jenkins - 6	2198	28

- As seen in the above table, the FNV hash function has the lowest average collisions of all the hash functions. However, it must be noted that the djb2 and Jenkins hash functions are very close.
- Additionally, it also results in the lowest collisions for 40 out of 101 files. This is 10 more than the djb2 hash function which is 2nd.
- Nevertheless, due to the above results, **the FNV hash function (Hash 5) will be the default hash function** in the system.

Data for each file

Filename	Unique Words	Table Capacity	Collisions for hash functions for a given file					
			Hash 1	Hash 2	Hash 3	Hash 4	Hash 5	Hash 6
23210-0.txt	2549	3187	1614	843	2208	818	804	776
3254.txt	69243	86561	67148	26866	68695	21559	21589	21468
30044.txt	1396	1747	731	441	1216	417	434	415
39706.txt	3235	4049	2348	1167	3036	986	977	1017
51493.txt	1850	2333	1098	571	1647	594	591	593
32040.txt	2440	3061	1547	806	2234	743	755	771
51268.txt	2630	3299	1789	849	2423	827	827	792
51687.txt	2315	2897	1448	777	2114	739	729	707
29503.txt	1756	2203	1012	563	1556	537	536	552
51296.txt	1978	2473	1210	637	1788	605	615	586
5592.txt	10715	13397	9389	3772	10425	3287	3333	3353
32735.txt	2500	3137	1631	822	2290	764	757	759
32133.txt	1933	2417	1187	607	1739	591	596	605
9790-8.txt	15762	19709	14159	5695	15373	4912	4880	4878
57040-0.txt	12224	15287	10765	3961	11783	3790	3751	3849
59368.txt	1732	2179	968	550	1536	540	556	561
2781-0.txt	4487	5623	3259	1531	4017	1359	1391	1400
32078.txt	2370	2963	1535	780	2167	739	746	743
40745-8.txt	9208	11519	7877	3175	8894	2839	2901	2885
34766-0.txt	14819	18523	13278	5360	14143	4618	4692	4578
32046-8.txt	17405	21757	15887	6079	17010	5350	5429	5447
9629-8.txt	7038	8803	5771	2439	6761	2193	2159	2209
1982-0.txt	965	1213	383	377	795	278	283	301
6120-0.txt	13011	16267	11443	4597	12537	3984	4034	4090
10947-8.txt	15346	19183	13787	5396	14943	4784	4756	4774
42664.txt	1766	2207	1067	558	1590	562	543	548
31217-8.txt	15943	19937	14389	5684	15559	5099	4924	4981
17669-8.txt	15356	19207	13883	5399	14976	4779	4780	4807
23099.txt	1321	1657	656	421	1143	413	418	398
58991.txt	2369	2963	1526	803	2173	727	708	735
2327-8.txt	6701	8377	5405	2271	6407	2110	2034	2074
2334-0.txt	27913	34897	26038	10343	27120	8786	8647	8754
32845-8.txt	15796	19751	14290	5703	15435	5017	4916	4928
14744-8.txt	8525	10657	7243	2912	8229	2647	2669	2680
23942-8.txt	2008	2521	1219	646	1829	651	631	617
51008.txt	1544	1931	845	472	1367	481	477	502
51752.txt	2318	2897	1484	790	2115	742	743	732

30029-8.txt	1875	2347	1071	565	1670	553	585	574
1944-0.txt	8806	11027	7414	3184	8296	2697	2704	2746
38531-8.txt	15496	19373	13911	5406	15088	4808	4799	4779
31840.txt	1898	2377	1115	578	1709	580	572	641
57006-0.txt	7490	9371	6145	2677	7058	2308	2263	2356
51193.txt	2482	3109	1630	839	2278	763	743	757
56870-8.txt	9246	11579	7896	3324	8944	2875	2905	2877
32347.txt	1291	1613	640	399	1230	381	392	398
13799.txt	11837	14797	10411	4200	11527	3699	3698	3765
24878-8.txt	11550	14437	10177	4113	11234	3609	3596	3598
8933-0.txt	12154	15193	10730	4283	11553	3789	3785	3744
59255.txt	2548	3187	1684	822	2346	797	779	745
28650.txt	1556	1949	867	505	1370	480	449	484
2518.txt	7226	9041	6049	2709	6973	2183	2208	2285
41562.txt	2199	2749	1406	758	2086	681	640	666
50877.txt	2028	2539	1232	630	1832	653	645	631
38172-8.txt	13620	17027	12130	4812	13252	4271	4232	4299
34313-8.txt	7743	9679	6371	2559	7480	2382	2437	2389
18776-8.txt	10308	12889	8921	3776	9992	3215	3208	3157
22897-8.txt	2793	3491	1861	885	2567	883	851	865
2550-0.txt	10139	12689	8623	3499	9669	3178	3187	3168
1626-0.txt	12695	15877	11049	4308	12083	3891	3961	4008
3181-0.txt	2582	3229	1670	857	2327	825	804	826
58743.txt	1927	2411	1208	601	1739	582	591	618
25035.txt	2466	3083	1608	847	2263	755	781	774
28698.txt	2468	3089	1640	854	2277	775	776	789
51603.txt	1666	2083	919	562	1485	545	511	509
28726-8.txt	14293	17881	12832	5236	13961	4455	4439	4450
55514-0.txt	14405	18013	12877	4890	13739	4481	4455	4461
2305-0.txt	11360	14207	9703	3998	10945	3524	3571	3592
373-0.txt	12554	15727	11073	4318	11965	3923	3933	3930
29618.txt	1630	2039	892	549	1445	506	531	500
5737-0.txt	11307	14143	9794	4043	10911	3495	3521	3553
28062.txt	1902	2377	1160	584	1710	571	560	587
29750.txt	1704	2131	953	546	1502	528	505	532
6168.txt	4510	5639	3546	1550	4296	1415	1405	1414
49598-8.txt	7594	9497	6321	2709	7316	2430	2367	2408
58995-8.txt	1660	2081	968	529	1485	532	503	507
51699.txt	2133	2671	1345	703	1934	672	660	679
51129.txt	2164	2707	1361	712	1964	692	677	671
6073-0.txt	9370	11717	7947	3309	8868	2970	2904	2884
26772.txt	2898	3623	1983	950	2677	888	910	878
58735.txt	1847	2309	1056	563	1647	557	579	582

32077.txt	2062	2579	1253	661	1870	634	636	619
51498.txt	1641	2053	905	520	1452	541	520	486
24313-8.txt	10608	13267	9210	3702	10276	3332	3344	3305
22662-8.txt	2229	2789	1422	736	2024	711	718	734
8129-8.txt	5214	6521	4204	1782	4982	1622	1607	1639
55865-0.txt	5248	6563	4043	1819	5037	1620	1634	1671
22522-8.txt	5521	6907	4411	1847	5248	1760	1716	1739
54183-0.txt	9005	11257	7602	2932	8398	2840	2787	2863
877-0.txt	2320	2903	1511	760	2053	726	716	713
22426-8.txt	4376	5471	3356	1413	4153	1367	1366	1369
58341-0.txt	11711	14639	10242	3819	11144	3660	3648	3615
15717-8.txt	9894	12373	8442	3682	9515	3113	3110	3073
32067.txt	2501	3137	1612	811	2296	780	769	779
21782.txt	2352	2953	1490	767	2142	765	739	747
6040.txt	7755	9697	6514	2717	7492	2420	2377	2407
6696-8.txt	15363	19207	13725	5130	14696	4843	4805	4792
9205.txt	1725	2161	1030	548	1537	518	550	546
24558.txt	2816	3527	1881	899	2596	896	887	901
pg4081.txt	10649	13313	9315	3949	10376	3309	3290	3360
32104.txt	2195	2749	1366	743	2075	696	655	674
2429-0.txt	6335	7919	5092	2259	5877	1949	2012	2039
Average values	**7044**	**8810**	**5942**	**2474**	**6745**	**2196**	**2189**	**2198**