# Flags to run code

## Part1

```
g++ -std=c++17 .\test_1.cpp .\Hashing_LinearProbing.h -o .\test_1
.\test_1
```

## Part2

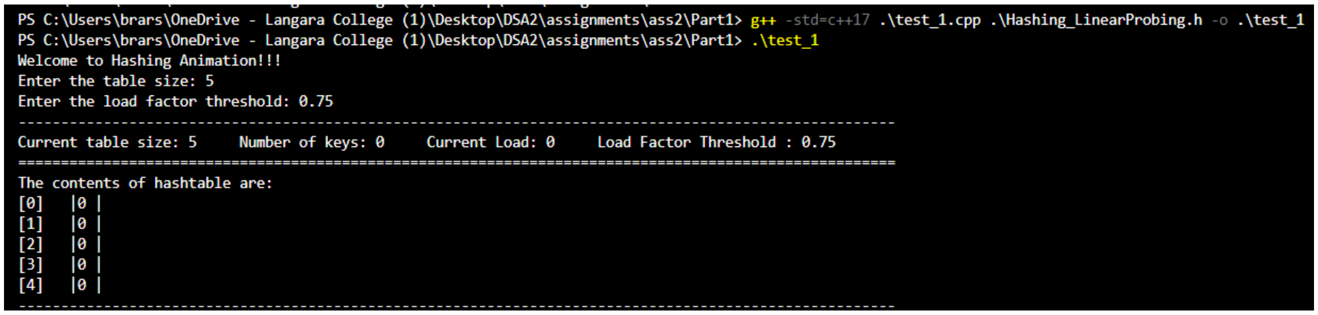
```
g++ -std=c++17 .\test_2.cpp .\Hashing_SeparateChaining.h -o .\test_2
.\test_2
```

## Part2

```
g++ -std=c++17 .\test_3.cpp .\Hashing_TwoSum.cpp -o .\test_3
.\test_3
```

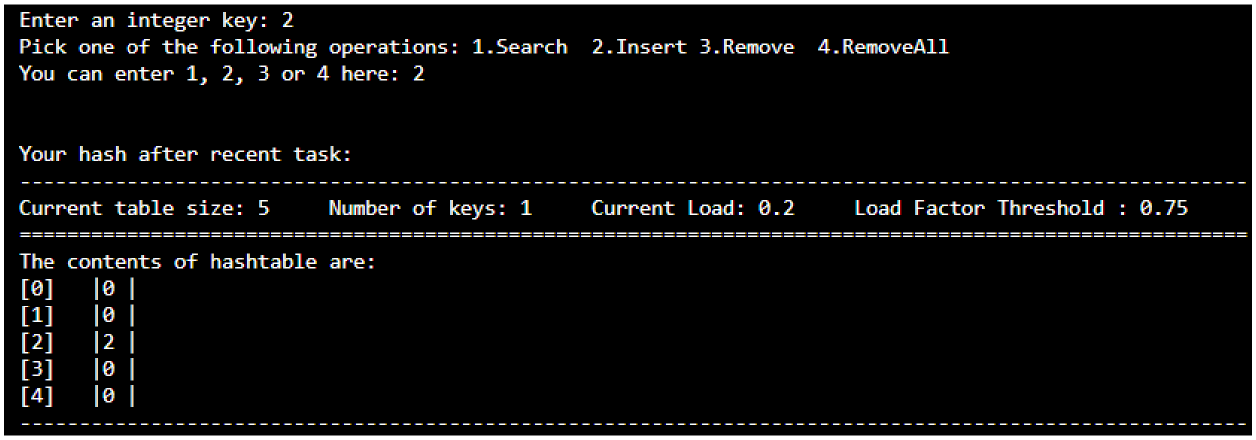
## Sample Runs

## Part 1: Handling collision using linear probing



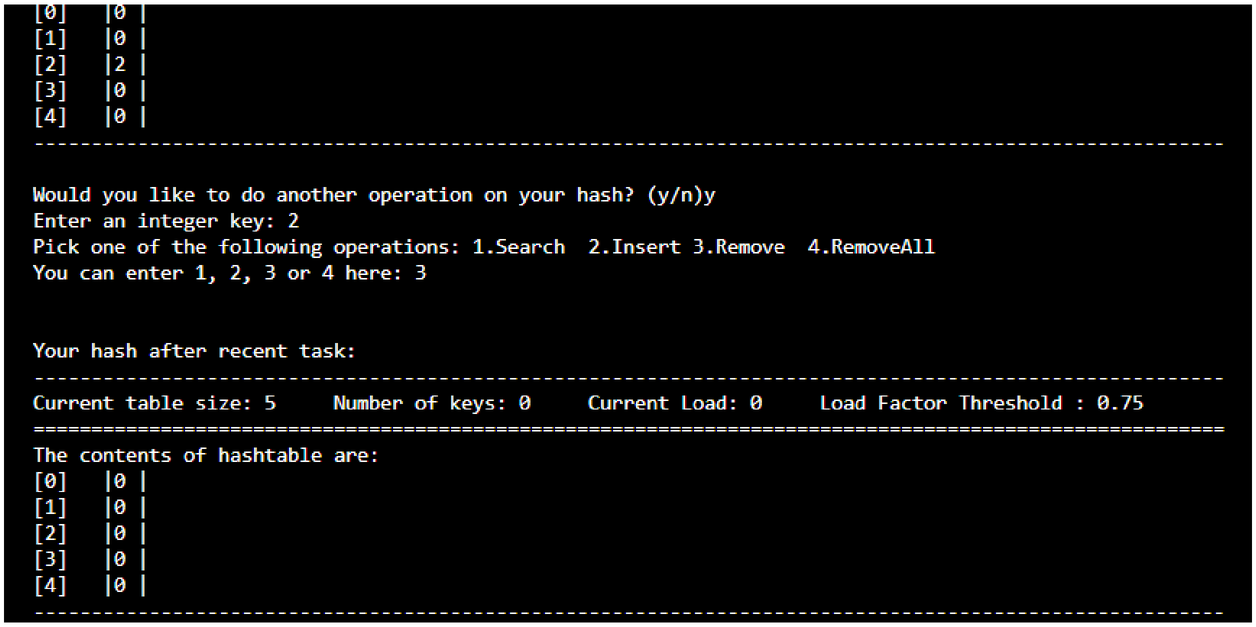
```
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part1> g++ -std=c++17 .\test_1.cpp .\Hashing_LinearProbing.h -o .\test_1
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part1> .\test_1
Welcome to Hashing Animation!!!
Enter the table size: 5
Enter the load factor threshold: 0.75
---------------------------------------------------------------------------------------------
Current table size: 5     Number of keys: 0     Current Load: 0     Load Factor Threshold : 0.75
=============================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |0 |
[3]   |0 |
[4]   |0 |
---------------------------------------------------------------------------------------------
```

- Testing Insertion part



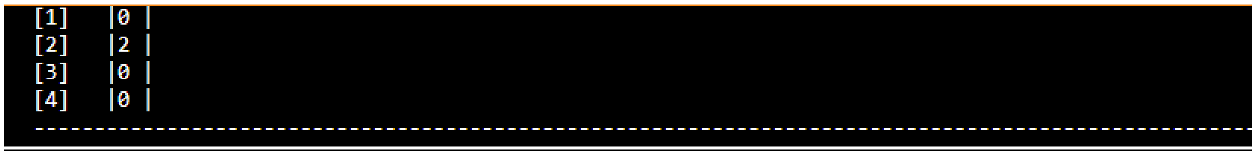
```
Enter an integer key: 2
Pick one of the following operations: 1.Search  2.Insert 3.Remove  4.RemoveAll
You can enter 1, 2, 3 or 4 here: 2


Your hash after recent task:
---------------------------------------------------------------------------------------------
Current table size: 5     Number of keys: 1     Current Load: 0.2     Load Factor Threshold : 0.75
=============================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |2 |
[3]   |0 |
[4]   |0 |
---------------------------------------------------------------------------------------------
```

- Testing Remove part

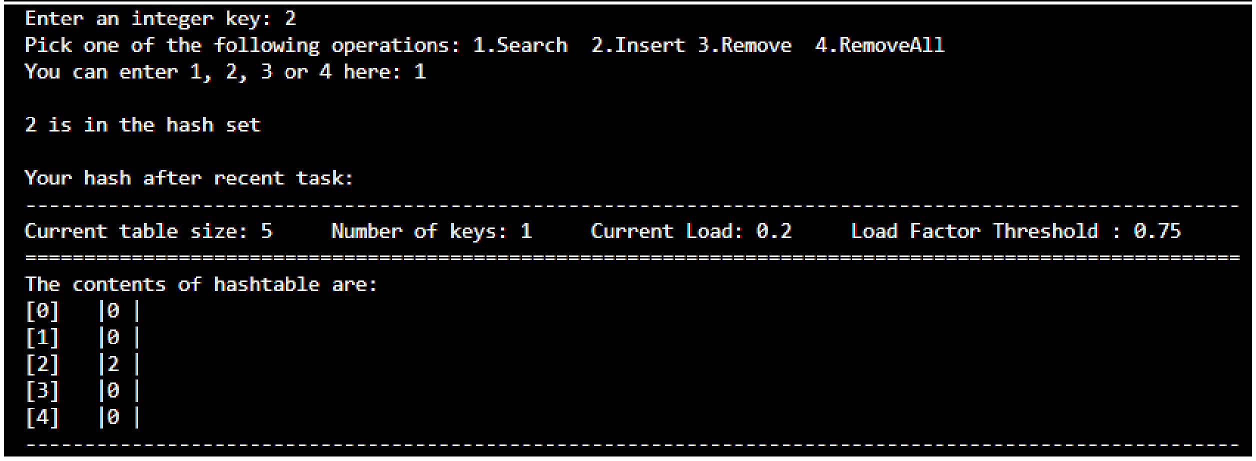


```
[0]   |0 |
[1]   |0 |
[2]   |2 |
[3]   |0 |
[4]   |0 |
---------------------------------------------------------------------------------------------

Would you like to do another operation on your hash? (y/n)y
Enter an integer key: 2
Pick one of the following operations: 1.Search  2.Insert 3.Remove  4.RemoveAll
You can enter 1, 2, 3 or 4 here: 3


Your hash after recent task:
---------------------------------------------------------------------------------------------
Current table size: 5     Number of keys: 0     Current Load: 0     Load Factor Threshold : 0.75
=============================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |0 |
[3]   |0 |
[4]   |0 |
---------------------------------------------------------------------------------------------
```

- Testing Search

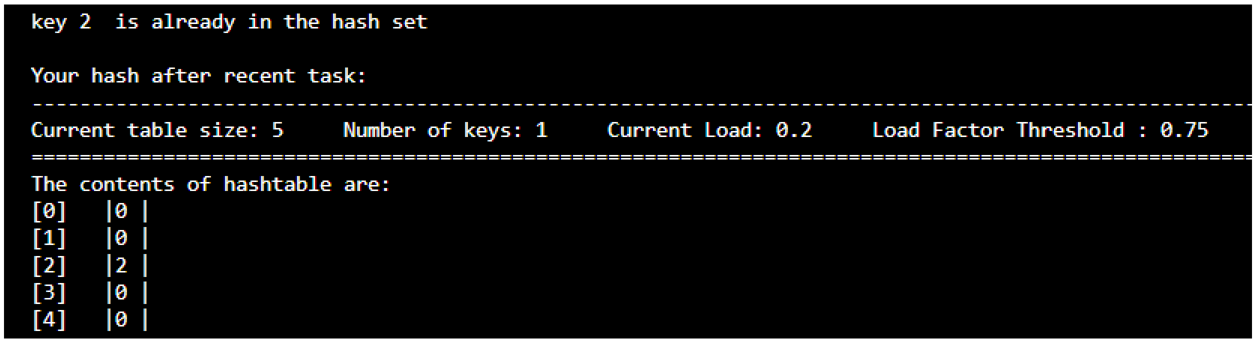


```
 [1]   |0 |
 [2]   |2 |
 [3]   |0 |
 [4]   |0 |
 -----------------------------------------------------------------------------------------
```



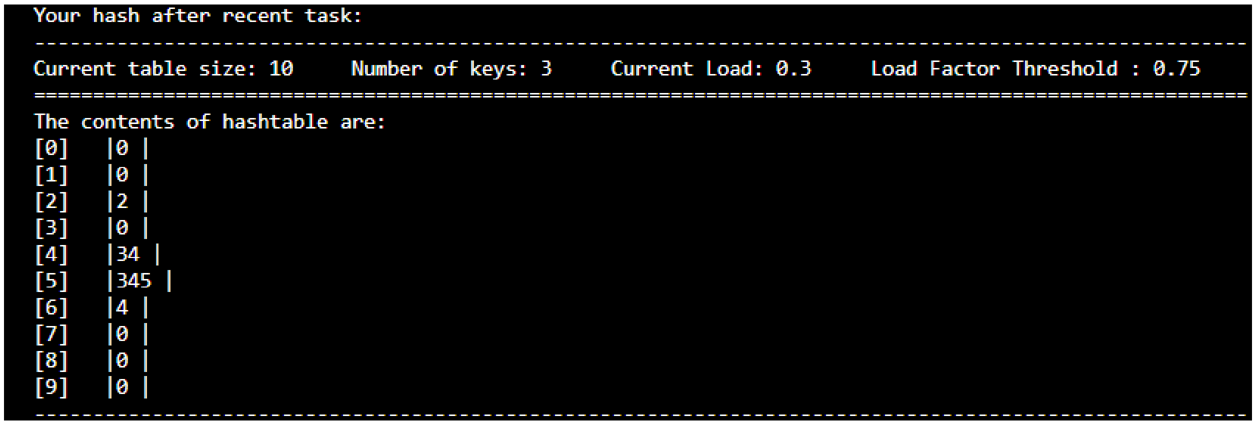
```
Enter an integer key: 2
Pick one of the following operations: 1.Search  2.Insert 3.Remove  4.RemoveAll
You can enter 1, 2, 3 or 4 here: 1

2 is in the hash set

Your hash after recent task:
----------------------------------------------------------------------------------------
Current table size: 5     Number of keys: 1     Current Load: 0.2     Load Factor Threshold : 0.75
========================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |2 |
[3]   |0 |
[4]   |0 |
----------------------------------------------------------------------------------------
```

- Testing Duplicate entry



```
key 2  is already in the hash set

Your hash after recent task:
----------------------------------------------------------------------------------------
Current table size: 5     Number of keys: 1     Current Load: 0.2     Load Factor Threshold : 0.75
========================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |2 |
[3]   |0 |
[4]   |0 |
```

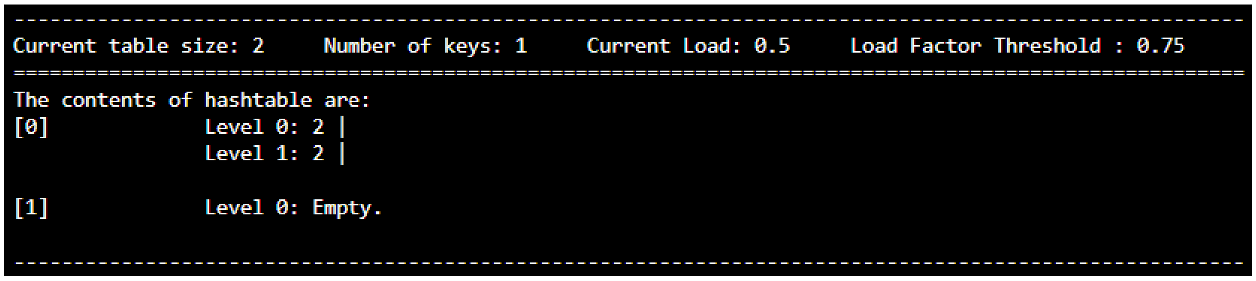
- Testing Rehash part (size doubled)



```
Your hash after recent task:
----------------------------------------------------------------------------------------
Current table size: 10     Number of keys: 3     Current Load: 0.3     Load Factor Threshold : 0.75
========================================================================================
The contents of hashtable are:
[0]   |0 |
[1]   |0 |
[2]   |2 |
[3]   |0 |
[4]   |34 |
[5]   |345 |
[6]   |4 |
[7]   |0 |
[8]   |0 |
[9]   |0 |
----------------------------------------------------------------------------------------
```

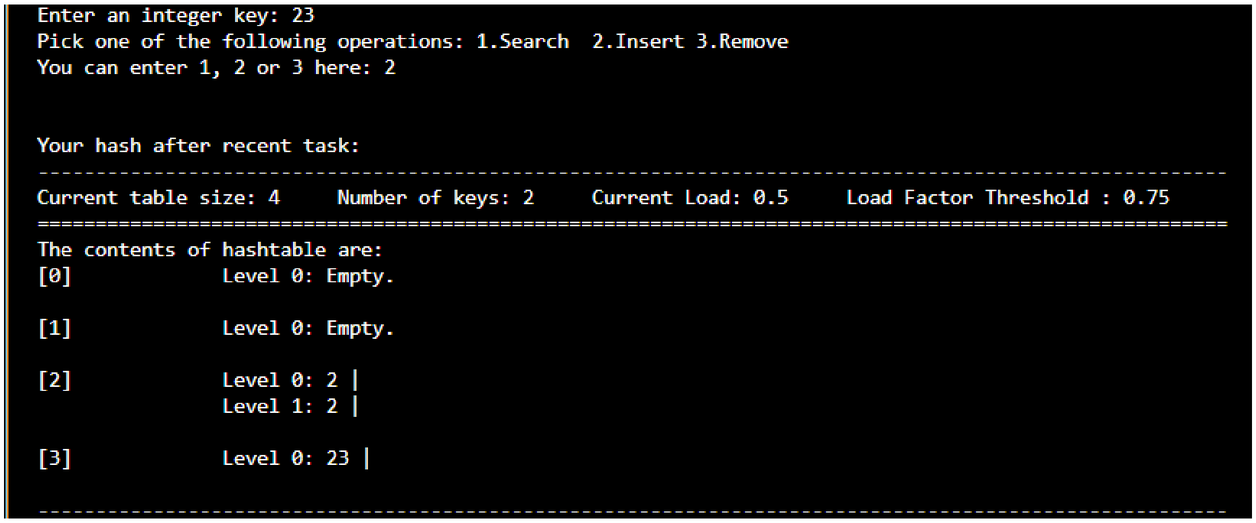
## Part 2: Handling collision using separate chaining (skip- list)

- Testing Rehash part



```
-----------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 1     Current Load: 0.5     Load Factor Threshold : 0.75
===============================================================================================
The contents of hashtable are:
[0]             Level 0: 2 |
                Level 1: 2 |

[1]             Level 0: Empty.

-----------------------------------------------------------------------------------------------
```

Size doubled:



```
Enter an integer key: 23
Pick one of the following operations: 1.Search  2.Insert 3.Remove
You can enter 1, 2 or 3 here: 2


Your hash after recent task:
-----------------------------------------------------------------------------------------------
Current table size: 4     Number of keys: 2     Current Load: 0.5     Load Factor Threshold : 0.75
===============================================================================================
The contents of hashtable are:
[0]             Level 0: Empty.

[1]             Level 0: Empty.

[2]             Level 0: 2 |
                Level 1: 2 |

[3]             Level 0: 23 |

-----------------------------------------------------------------------------------------------
```

- Testing Rescale part

  ***(as to maintain big oh of O(log n), it is important that total levels should be log(n))***

**In my code, you will find MAX_ALLOWED_LEVEL_INDEX = 1 in beginning, as default inside constructor of hash-table.**

Which means, 2^1 = 2 elements can be inserted into all the skip lists at every index of our table, and as soon as 3rd element tries to come in (i.e., size (of skip list) +1 > 2^1), then level count should be increased by 1 (which will accommodate 2^2 = 4 elements before the need of rescaling up again).

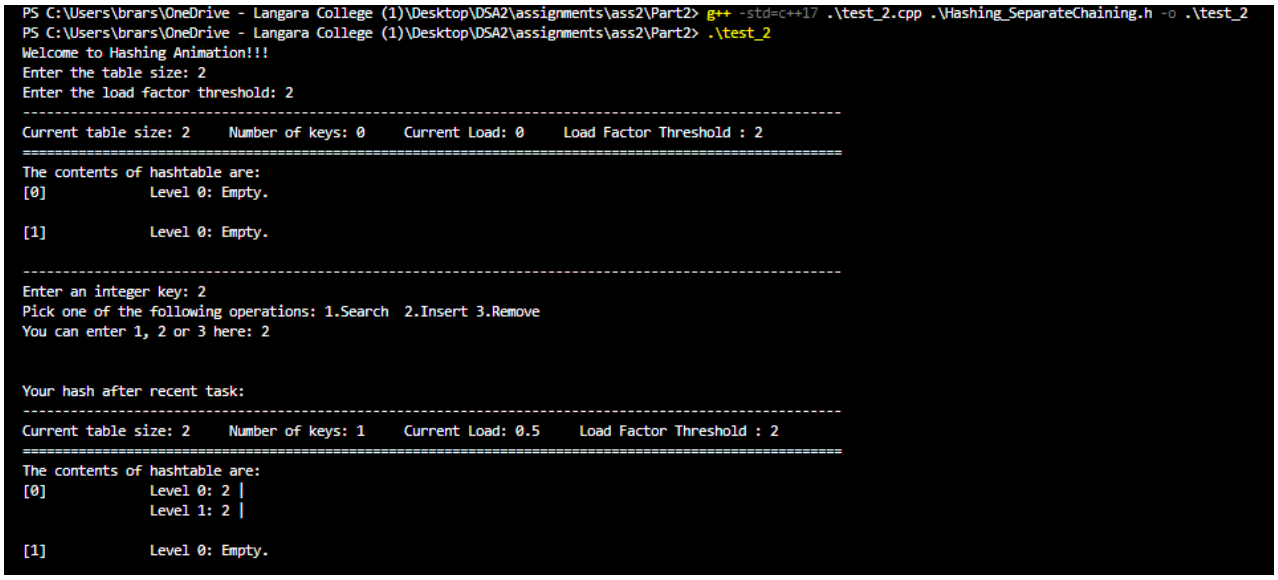
Setting table size = 2, threshold =2 : [so that rehash doesn't happen, as we want to focus on rescale]

**||TESTING REMOVE & skip-list internal rescale down**

Rescaling down:

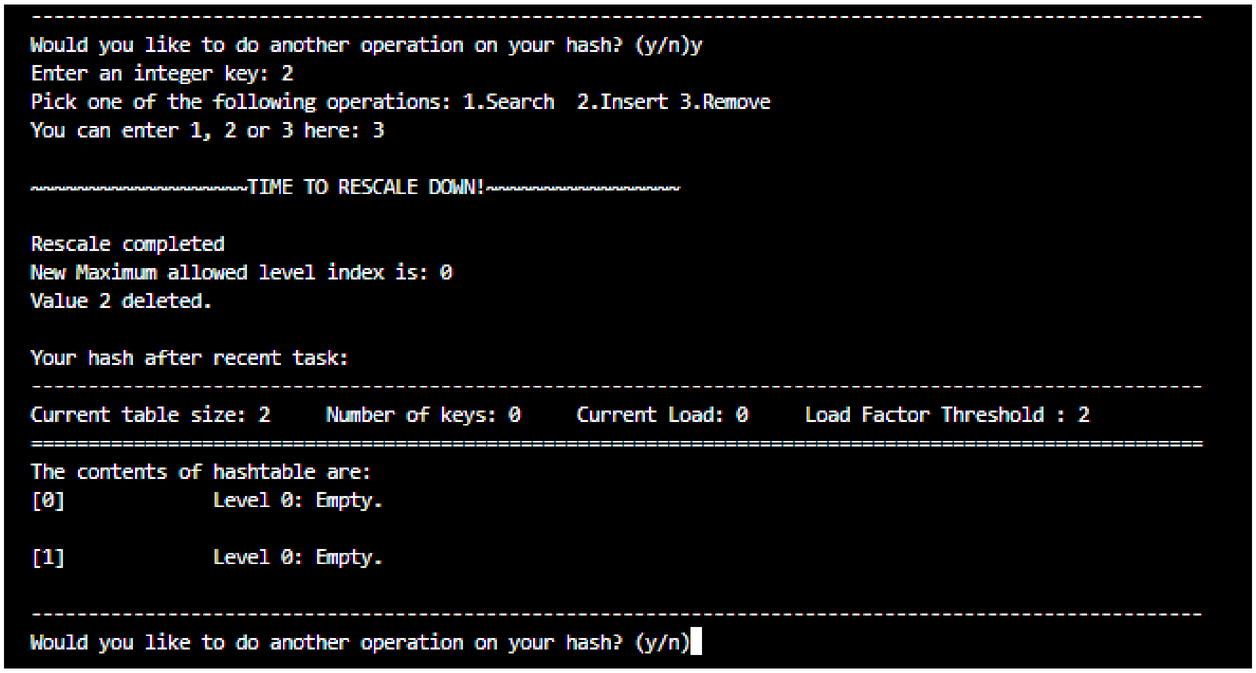
- We inserted 1 element, but later deleted it, hence 0 elements, yet MAX_ALLOWED_LEVEL_INDEX = 1. Although 0 elements can be handled by MAX_ALLOWED_LEVEL_INDEX = 0 (2^0 = 1 capacity), **so our code does rescale down**.

  - Inserted 2:



```
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part2> g++ -std=c++17 .\test_2.cpp .\Hashing_SeparateChaining.h -o .\test_2
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part2> .\test_2
Welcome to Hashing Animation!!!
Enter the table size: 2
Enter the load factor threshold: 2
---------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 0     Current Load: 0     Load Factor Threshold : 2
=============================================================================================
The contents of hashtable are:
[0]             Level 0: Empty.

[1]             Level 0: Empty.

---------------------------------------------------------------------------------------------
Enter an integer key: 2
Pick one of the following operations: 1.Search  2.Insert 3.Remove
You can enter 1, 2 or 3 here: 2


Your hash after recent task:
---------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 1     Current Load: 0.5     Load Factor Threshold : 2
=============================================================================================
The contents of hashtable are:
[0]             Level 0: 2 |
                Level 1: 2 |

[1]             Level 0: Empty.
```

  - Deleted 2:



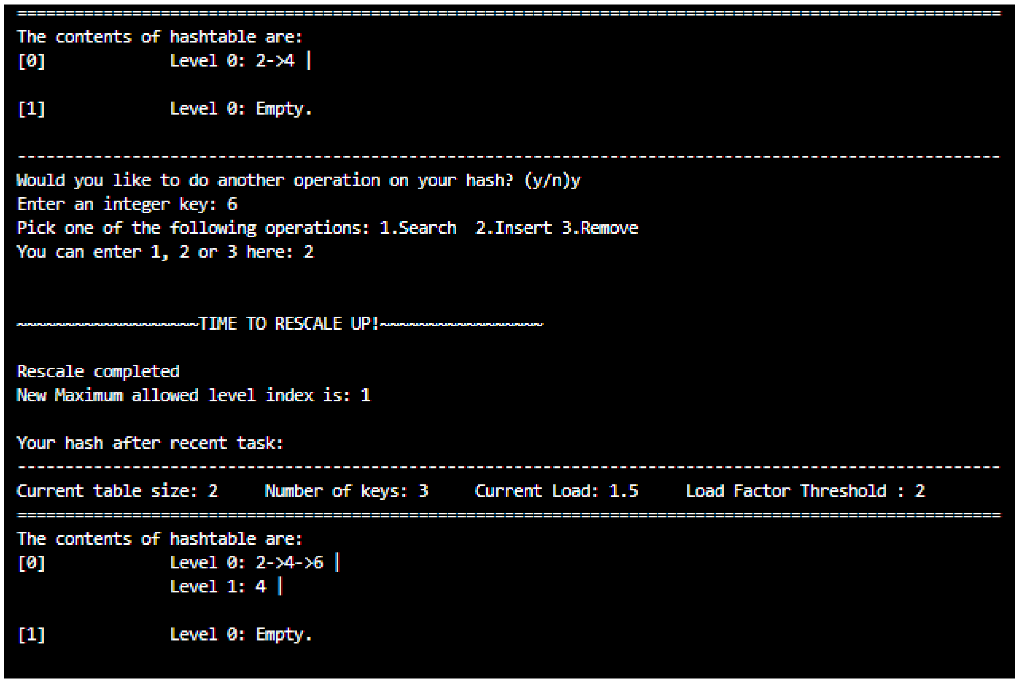
```
---------------------------------------------------------------------------------------------
Would you like to do another operation on your hash? (y/n)y
Enter an integer key: 2
Pick one of the following operations: 1.Search  2.Insert 3.Remove
You can enter 1, 2 or 3 here: 3

~~~~~~~~~~~~~~~~~~~~~~~TIME TO RESCALE DOWN!~~~~~~~~~~~~~~~~~~~~~

Rescale completed
New Maximum allowed level index is: 0
Value 2 deleted.

Your hash after recent task:
---------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 0     Current Load: 0     Load Factor Threshold : 2
=============================================================================================
The contents of hashtable are:
[0]             Level 0: Empty.

[1]             Level 0: Empty.

---------------------------------------------------------------------------------------------
Would you like to do another operation on your hash? (y/n)
```

Rescaling up:

- Now, we try to insert 4 elements to an empty hash-table, where MAX_LEVEL is 0, **hence rescaling up should be done twice (2^1 = 2 , 2^2 = 4) to accommodate them.**

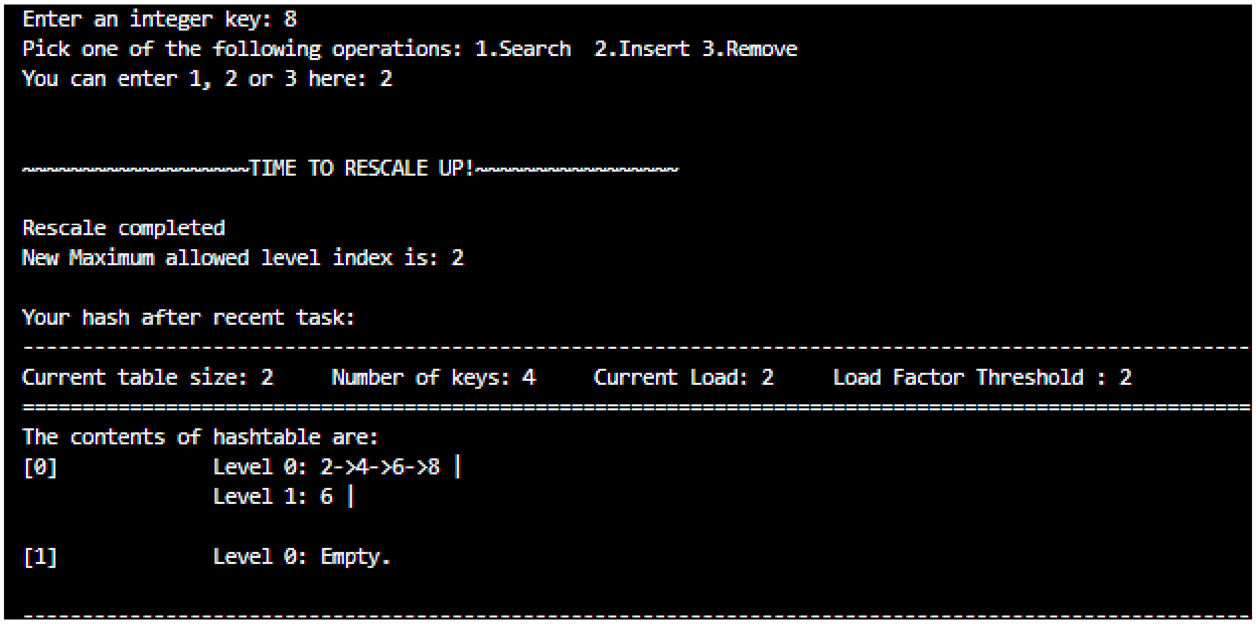
**||TESTING INSERT & skip-list internal rescale up**

Entered 2,4,6 (rescale once):



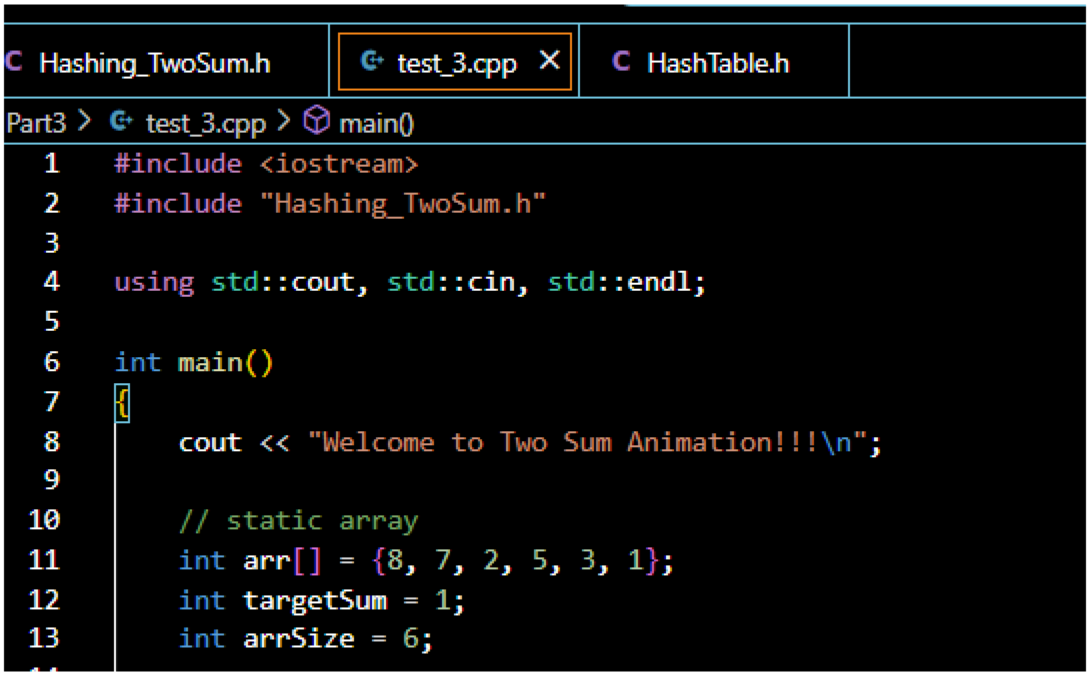
```
==============================================================================================
The contents of hashtable are:
[0]            Level 0: 2->4 |

[1]            Level 0: Empty.

----------------------------------------------------------------------------------------------
Would you like to do another operation on your hash? (y/n)y
Enter an integer key: 6
Pick one of the following operations: 1.Search  2.Insert 3.Remove
You can enter 1, 2 or 3 here: 2


~~~~~~~~~~~~~~~~~~~~TIME TO RESCALE UP!~~~~~~~~~~~~~~~~~~

Rescale completed
New Maximum allowed level index is: 1

Your hash after recent task:
----------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 3     Current Load: 1.5     Load Factor Threshold : 2
==============================================================================================
The contents of hashtable are:
[0]            Level 0: 2->4->6 |
               Level 1: 4 |

[1]            Level 0: Empty.
```

Again rescale:



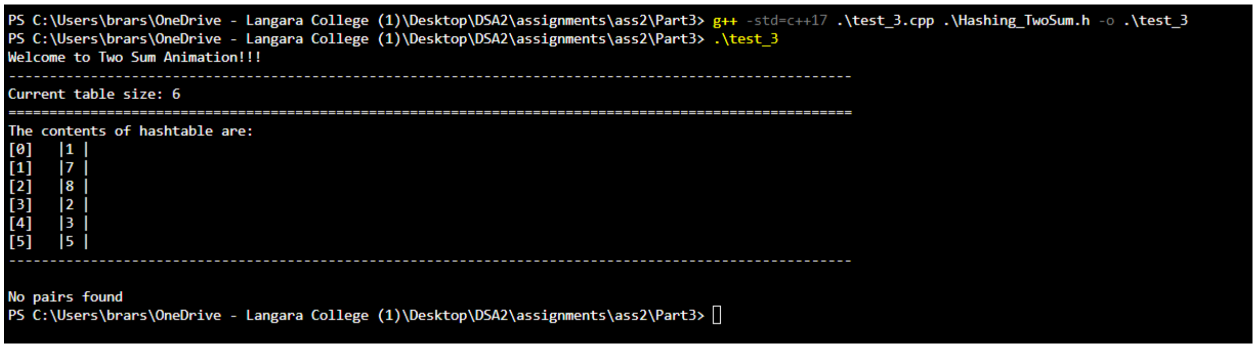
```
Enter an integer key: 8
Pick one of the following operations: 1.Search  2.Insert 3.Remove
You can enter 1, 2 or 3 here: 2



~~~~~~~~~~~~~~~~~~~~~~~TIME TO RESCALE UP!~~~~~~~~~~~~~~~~~~~~~

Rescale completed
New Maximum allowed level index is: 2

Your hash after recent task:
----------------------------------------------------------------------------------------------
Current table size: 2     Number of keys: 4     Current Load: 2     Load Factor Threshold : 2
==============================================================================================
The contents of hashtable are:
[0]            Level 0: 2->4->6->8 |
               Level 1: 6 |

[1]            Level 0: Empty.

----------------------------------------------------------------------------------------------
```

## TEST CASES: Part 3



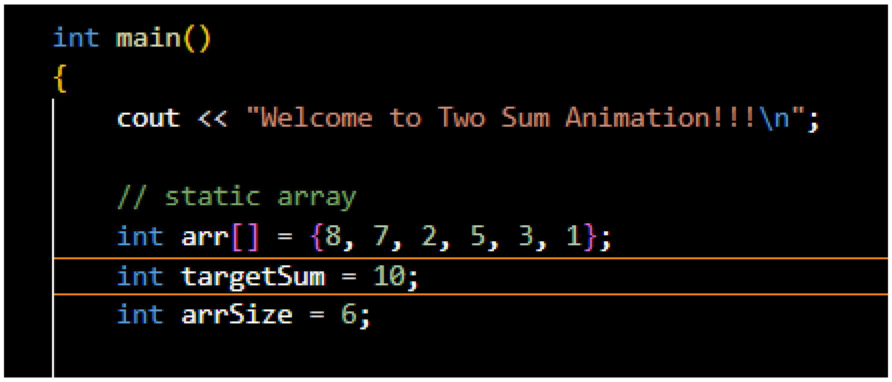
```cpp
#include <iostream>
#include "Hashing_TwoSum.h"

using std::cout, std::cin, std::endl;

int main()
{
    cout << "Welcome to Two Sum Animation!!!\n";

    // static array
    int arr[] = {8, 7, 2, 5, 3, 1};
    int targetSum = 1;
    int arrSize = 6;
```

-------------------------



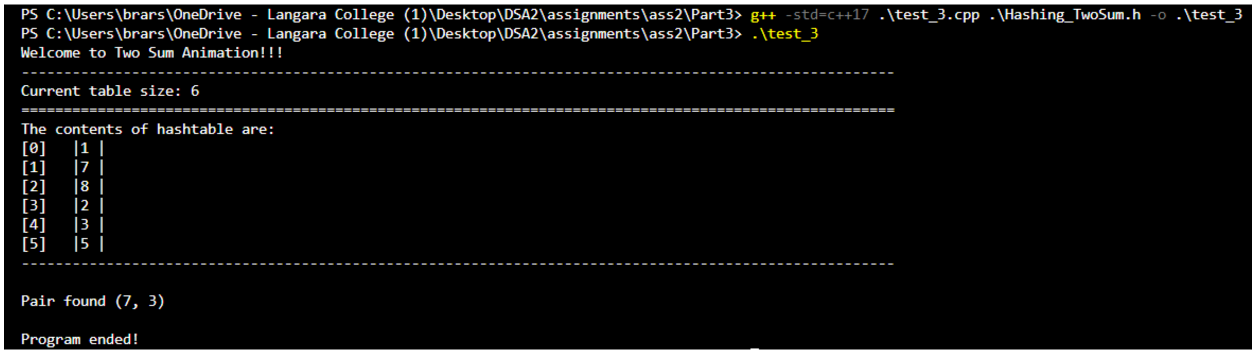
```
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part3> g++ -std=c++17 .\test_3.cpp .\Hashing_TwoSum.h -o .\test_3
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part3> .\test_3
Welcome to Two Sum Animation!!!
------------------------------------------------------------------------------------------------------
Current table size: 6
======================================================================================================
The contents of hashtable are:
[0]   |1 |
[1]   |7 |
[2]   |8 |
[3]   |2 |
[4]   |3 |
[5]   |5 |
------------------------------------------------------------------------------------------------------

No pairs found
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part3> 
```

When pair exist:



```cpp
int main()
{
    cout << "Welcome to Two Sum Animation!!!\n";

    // static array
    int arr[] = {8, 7, 2, 5, 3, 1};
    int targetSum = 10;
    int arrSize = 6;
```

-------------------



```
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part3> g++ -std=c++17 .\test_3.cpp .\Hashing_TwoSum.h -o .\test_3
PS C:\Users\brars\OneDrive - Langara College (1)\Desktop\DSA2\assignments\ass2\Part3> .\test_3
Welcome to Two Sum Animation!!!
----------------------------------------------------------------------------------------------------------
Current table size: 6
==========================================================================================================
The contents of hashtable are:
[0]   |1 |
[1]   |7 |
[2]   |8 |
[3]   |2 |
[4]   |3 |
[5]   |5 |
----------------------------------------------------------------------------------------------------------

Pair found (7, 3)

Program ended!
```

NOTE: The hash-table contents are displayed only for displaying that hashing is working properly, it is not printed in the actual code (you may, print it by calling display)