B.TECH. PROJECT REPORT

On

Implementation of 0in Hash table using C++ STL

BY

Kailas N Sheregar

150001031

DISCIPLINE OF COMPUTER SCIENCE and ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2018

Implementation of 0in Hash table using C++ STL

A PROJECT REPORT
Submitted in partial fulfillment of the
requirements for the award of the degrees
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Kailas N. Sheregar

Guided by:
Dr. Bodhisatwa Mazumdar,
Assistant Professor,
Discipline of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December, 2018

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Implementation of 0in Hash table using C++ STL "** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Bodhisatwa Mazumdar, Assistant Professor, Discipline of Computer Science and Engineering, IIT Indore** and **Mr. Chakra Agarwal, Lead Member Technical Staff, Mentor Graphics** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Kailas N Sheregar

CERTIFICATE BY BTP GUIDE

It is certified that the above statement made by the students is correct to the best of my knowledge.

Dr. Bodhisatwa Mazumdar,
Assistant Professor,
Discipline of CSE,
IIT Indore.

Preface

This report on **"Implementation of 0in Hash table using C++ STL"** is prepared under the guidance of my mentor at Mentor Graphics, Mr. Chakra Agarwal and BTP supervisor at IIT Indore, Dr. Bodhisatwa Mazumdar.

Through this report I have tried to give a description of how the project was completed in the course of 6 months in the company and the various technologies used for the purpose.

I have tried to the best of my abilities and knowledge to explain the content in a lucid manner.

Kailas N Sheregar

B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgement

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to my BTP project supervisor, Dr. Bodhisatwa Mazumdar, Assistant Professor ,Discipline of Computer Science and Engineering, IIT Indore.

Furthermore I would also like to acknowledge with much appreciation the crucial role of Mentor Graphics , who gave me the opportunity to work for the company. Last but not least, many thanks go to the manager of the project, Mr. Sanjeev Aggarwal, and my mentor Mr. Chakra Agarwal who have invested their full effort in guiding me in achieving our goal. I have to appreciate the guidance given by other supervisors as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

Kailas N Sheregar

B.Tech. IV Year,

Discipline of Computer Science and Engineering,

IIT Indore

Contents

Important terms :-

1. 0in :- The company which originally implemented the old hash table for Mentor Graphics.
2. Hash table :- Hash Table is a data structure which stores data in an associative manner. This makes data access and search very fast.
3. C++ STL :- It is short for C++ Standard Template Library. It is a powerful library that implements many popular and commonly used algorithms and data structures like vectors, lists, queues, and stacks.

Motivation behind project :

The main purpose behind the project to make the hash table more in line with today's needs. The original hash table was written in C back in the 1990s and was implemented from scratch. The idea was to reimplement the widely used hash table in such a manner that it now used C++ STL. The advantages are many:

1. C++ STL implements low level optimization which improves operations like access time.
2. In case there are any upgrades to STL internally, simply recompiling will exploit them, making them much easier to keep up to date and maintain.
3. C is quite outdated, majorly due the absence of powerful features like OOPs and standard libraries. It was felt that the hash table should upgraded as well to make it easier to use for developers more familiar in C++.
4. To check whether certain regressions could possibly run on STL hash table.

Goals of the Project :

As mentioned the main aim for the project was to port all the hash table APIs to ones using C++ STL internally. Other than that, the focus should should be on minimizing external changes to APIs, which would warrant developers and clients who have used these APIs to modify their code, and would turn out to be very cumbersome. Finally, even though the largest effort would be put into minimizing these said external changes, many functionalities inevitably require an external change. In such a case it would be necessary to preserve all original functionalities along with clear instructions and explanations as to how to use them in case they are needed.

Major challenges for project :

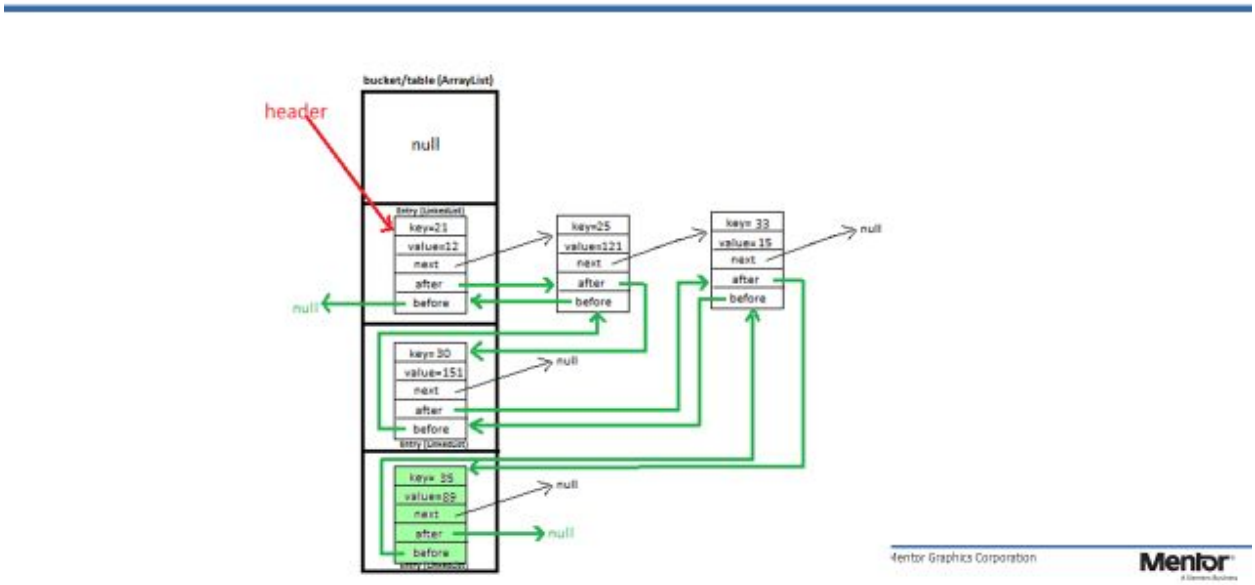
1. In the existing library, data types and related functions like hasher functions and comparison functions are decided at run time while in C++ STL these have to be specified at compile time . Basically the C++ STL hash table namely unordered_map takes parameters in its template regarding the data types of the key type, value type and other functions involved. However, templates are expanded at compile time. In the old hash table however, a hash creation function identifies these when it is called over a old hash table object.
2. Iterating macros for traversing the old hash table have been written but they have not been written to handle data type mismatches. Hence this issue needed to be addressed.
3. Unlike traditional hash tables this hash table also keeps track of insertion order of elements which needs to be implemented in the new hash table. Unordered_map does not provide such a facility because of which such a functionality had to implemented over it.

Original Implementation :

Original hash table was implemented through two structures :-

1. Main structure for collection of bins corresponding to every hash value.
2. Structure for each data element node.
3. Each element consists of key value, data value and three pointers:
    a. For next element in bin.
    b. For next element inserted.
    c. For previous element inserted.

Hasher functions, comparison functions, freeing functions and copying functions are passed to hash table at the time of creation. Each key corresponds to a bin where data elements corresponding to these its hash key are stored. Separate chaining method is used to address collisions.



New Implementation :

1. The hash table provided by C++ STL, unordered_map is generally implemented in the same manner as in the 0in hash table. However, it doesn't provide the functionality of keeping track of the order of elements. Hence the new structure consists of unordered map along with list to store ordering.
2. We shall also need a list to store the ordering of the elements along with an iterator to store a link to these. An additional list iterator is stored along with every element.
3. Different types of hash table declared depending on type of values and hash function and copy function required. Mainly four data types will be handled:
    a. Void * key, void * value
    b. Void * key, integer value
    c. Char * key, void * value
    d. Char * key, integer value
4. Since the maintenance of a list results in extra overhead in terms of both time and memory, an option has been given to avoid the use of list in case any client doesn't require the storing elements' order.
5. References to different declarations of hash table placed inside the main structure for new hash table.
6. Extra variable stored to indicate whether the new table is being used or the old table.

APIs which were modified :

Hash creation functions

There are many different hash creation functions which create different types of hash table when invoked with a object of our hash table structure. As mentioned, there are mainly four hash creation functions corresponding to each type mentioned above. Other than these there are also other variations which create hash functions with the same data types but with different accompanying hasher , comparison functions etc.

To implement these functionalities in the new table we used the corresponding pointers to the different types of unordered_map and lists we had defined in the overall struct already.

Other than creating entirely new hash tables, a duplication function also exists which takes a existing hash table object as argument, creates a new table and copies all elements into it using old copying function.

Functions for retrieval of elements

Various functions used to access values in the hash table were reimplemented.

Two basic functions for querying include one to access integer values and one to access generic void * pointer values.

Other variations of querying functions include a method to get the first element of a hash table, method to get the last element of a hash table, method to delete element after its value has been retrieved and one to get unsigned integer values. Separate functions for all of these had to be written as we didn't wish to change the external interface.

A final very important function which falls under the class of querying functions would be the print function.

Updation and insertion functions

Functions capable of inserting elements into the hash table fall into two categories :

1. Normal insertion of data : In this method the key is searched in the hash table, and if it is found, no changes are made to the table. Elements are inserted only if the key doesn't exist previously in the table.
2. Forceful insertion : In this method the key value pair is stored in the table regardless of whether the key is found or not. If the key exists previously, that value will be deleted.

Copying a element is done by using the hash table's copy function. There also exists a API to move the first element to the end position.

Deletion functions

The deletion functions which were rewritten include :

1. Function to delete entries storing void pointers as values.
2. Method to delete integer type values.
3. Method to empty hash table.
4. Method to completely delete table.

Deletion APIs were required to use the free function.

Macros used for iterating through hash table

Various macro APIs had been written to traverse the old hash table including traversing APIs to traverse over all the keys of the table, values in the table and traversing in ordered or unordered manner. Separate APIs were also written for integer value type hash tables.

The main problem with writing them using C++ was that there was a type mismatch which had to be resolved at compile time. To solve this problem,

function templates for carrying out this type conversion was written.

Templates

To deal with the recurring problem of data type mismatch two types of templates functions were written :

1. Templated functions which return an object of the type of unordered_map and list being used.
2. A converting function which typecasts variables into the one passed to it as a template parameter.

Miscellaneous APIs :

Other minor APIs which we had to written, were for :

1. Calculating number of elements of hash table.
2. Creating a special hash table storing string keys but without any copying function.
3. Sorting of hash table entries.

Removing redundancy :

Although every function had to implemented differently for different types, many required similar code which could be avoided if they could be replaced with generic macros. This could again be done with help of function templates we had already written.

Support for VHDL strings

In the later stages of the project a requirement for APIs involving hash tables storing vhdl strings as keys came up. These vhdl strings differ from normal strings only in their comparison function with key type and value type remaining the same. This warranted defining another type of hash table. To make all APIs work on this kind of hash table as well, all functions and macros had to be modified and

additional templated functions had to be written. Additions were also made to the test program.

Testing Procedure :

1. Testing involved three components.
2. First section called all necessary functions to check portability.
3. Second section involved time analysis for the new implementation.
    a. Random data generated for each data type and dumped into large file.
    b. Hash table APIs then insert these values in hash table and then perform query operations generating time results for both.
4. Third section generated time analysis results for old hash table for benchmarking.

Performance analysis :

Performance Table (10M entries)

1. **Insertion benchmarking**

	0in hash table		STL hash table(ordered)		STL hash table (unordered)	
	Time (s)	Memory (kB)	Time in sec (% improve ment)	Memory (% improve ment)	Time in sec(% improve ment)	Memory (% improve ment)
Void * vs void *	6.41	49220	6.19 (3.43)	1076708 (-118.74)	5.44 (15.13)	607876 (-23.49)
char * vs void *	12.6	1586976	8.93 (29.12)	2171548 (-36.83)	8.24 (34.60)	1702852 (-7.30)

Note :- The larger memory requirement, even when list is not used, is possibly due to large block size allocation used by unordered_map to avoid frequent rehashing. This proves beneficial if querying operations are high as compared to insertion and deletion operations, which is generally the case.

2. Query benchmarking

	0in hash table	STL hash table(ordered)		STL hash table (unordered)	
	Time (s)	Time (s)	Time (% improvement)	Time (s)	Time (% improvement)
Void * vs void *	2.33	1.57	32.6	1.43	38.6
char * vs void *	5.9	3.95	33.05	3.9	33.9

Performance Table (50M entries)

1. **Insertion benchmarking:**

	0in hash table		STL hash table(ordered)		STL hash table (unordered)	
	Time (s)	Memory (kB)	Time in sec (% improvement)	Memory (% improvement)	Time in sec(% improvement)	Memory (% improvement)
Void * vs void *	37.34	2414872	33.47 (10.36)	5239016 (-116.94)	28.55 (23.54)	2895256 (-19.89)
char * vs void *	70.99	7910764	48.33 (31.91)	10734860 (-35.69)	46.91 (33.92)	8391104 (-6.07)

2. Query benchmarking:

	0in hash table	STL hash table(ordered)		STL hash table (unordered)	
	Time (s)	Time (s)	Time (% improvement)	Time (s)	Time (% improvement)
Void * vs void *	18.43	10.88	40.96	10.63	42.32
char * vs void *	40.28	28	30.48	25.66	36.29

Code Deployment

1. Code reviews and test for corrections were carried out.
2. New hash table was deployed on a directory of code files.
3. After managing a few minor bugs, code was successfully deployed.
4. This was followed by pushing the new hash table library to main code body.
5. Clients wanting to use the code can compile their code in C++ and include our header file.

References

Modern C++ Design - Andrei Alexandrescu.

cppreference .com

cplusplus.com