

**BINUS UNIVERSITY**

**Assignment Cover Letter**

**(Individual Work)**

**BINUS INTERNATIONAL**

Student Information: Surname Given Names Student ID Number

1. Jevon Danaristo 2440043591
2. Fadhlan Muhammad Razan 2440047463
3. Rafi Muzakki 2440035614

Course Code : COMP6699

Class : L2AC

Major : Computer Science

Title of Assignment : Coupon Generator

Type of Assignment : Final Project

Submission Pattern

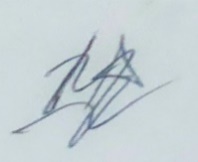
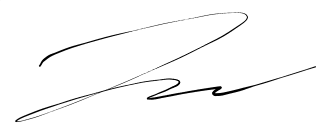
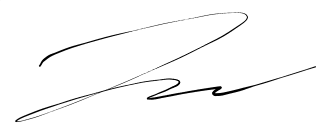
Due Date : 28/06/2021 Submission Date: 27/06/2021

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating** BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality** By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

****Signature of Student: ****Signature of Student: Signature of Student:



(Name of Student)(Name of Student) (Name of Student)

Jevon Danaristo Fadhlan Muhammad Razan Rafi Muzakki

Table of Contents

**I. Problem Description**

**II. Problem Solution**

a. Random Number Generator

b. Hash Function

c. Hash Table

d. Collision Resolution

**III. Implementation**

a. Random Number Generator

b. Hash Function

c. Linear Probe Collision

d. Linked List Collision

c. Bucket Collision

**IV. Program Manual**

a. Executing the Program

b Inserting Coupon

c. Searching Coupon

d. Time Plot

**V. Demo Video**

**VI. GitHub Link**

**Coupon Generator**

1. **Problem Description**

Coupons are essential for introducing customers to a new business, however; it requires creations of many different codes that are hard to guess. Duplicates are also problematic when using coupons since a coupon can only be used once.

1. **Problem Solution**
   1. **Random Number Generator**

Random number generator is a machine process of getting an arbitrary number. Random number commonly generates from a constant such as machine’s clock that is processed to a certain mathematical expression. A good random number generator. Some of the desirable properties a cryptographic random number generator should have are lack of bias, bit independence, unpredictability, and non-repeatability (Tkacik, 2003).

Coupon that dependent to randomness can be automated easily with the use of random number generator. With a modern random number, a random unique coupon can be generated in matter of seconds.

* 1. **Hash Function**

Hash function is an algorithm that generates new value from a key. Hash function provides a way to check an item without directly looking at the value of the actual key. The process of creating the value involves mathematical operation to convert the initial key into a new different value.

Hash function provides security to the process of claiming code, as it is only a one-way process. If for some case data involving the coupon is leaked, the leaked data will not store the actual coupon code.

* 1. **Hash Table**

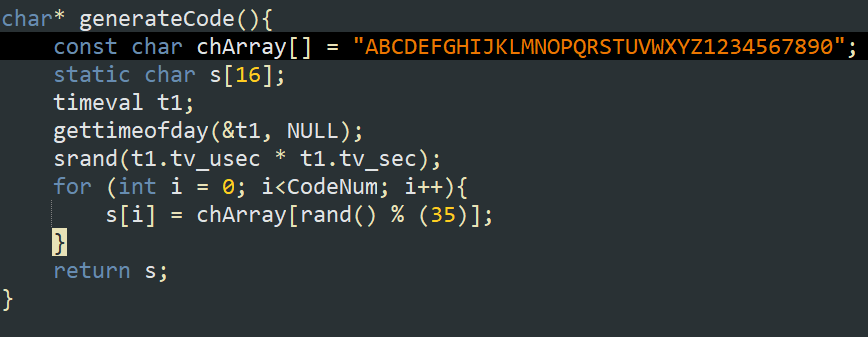
A Hash function stores hash value into a data structure named hash table. The hash table store hash value with the index produced by the hash function. It is possible for a hash function assign two keys into a similar index in the hash table, this causes collisions that require an algorithm to resolve.

The hash table will serve as data structure to coupons hash value.

* 1. **Collision Resolution**

Collision resolution is a method of resolving a collision between to value sharing a same index. Some types of resolution including: separate chaining, linear probe, and linked list resolution.

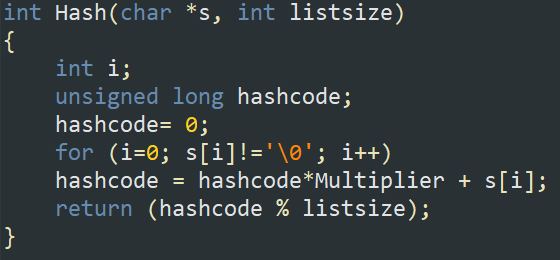
Collision resolution provides a way to handle conflicts between coupons that have similar index.

1. **Implementation**
   1. **Random Number Generator**

To generate a coupon code to be used to then compute an index with our hash function, we created a function that returns an array of characters. To generate it, we create an array chArray that contains alphanumeric characters. We also created the array s to store the generated characters. We then use the rand function to generate a number between 0-35 to get the alphanumeric character from the array chArray. For rand, we need to use a seed everytime we call it to generate a different number because otherwise it will generate a random number once at runtime but then generate the same number again because we havent changed the generation seed. To mitigate that, we get the time of day and use that as the seed for the random generation each time the function is called. After it iterates for a specified amount of time, it will return a character array containing the iteration elements.

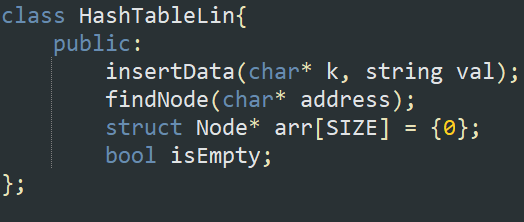
* 1. **Hash Function**

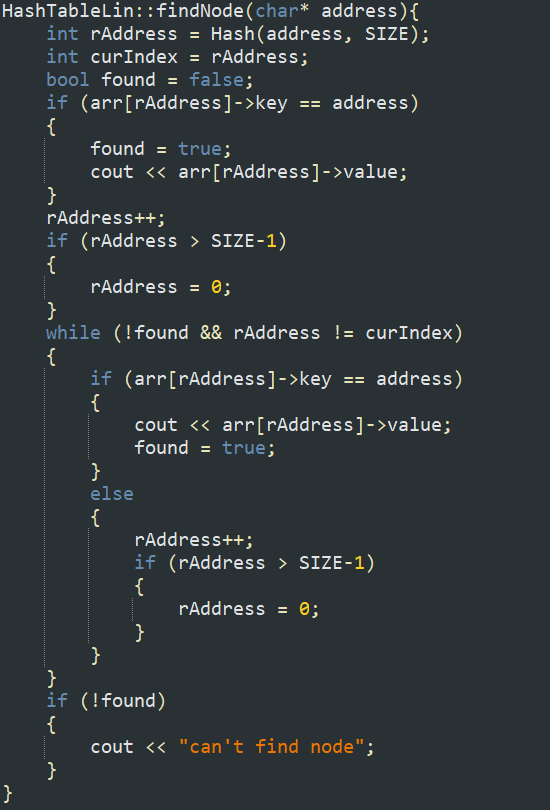
For implementing the hash function, we decided to use the pseudorandom method since we generate a random key everytime for getting the index, if we were to use the rotation method, it will yield results that is hard to control since the key is always random and could be anything without pattern.

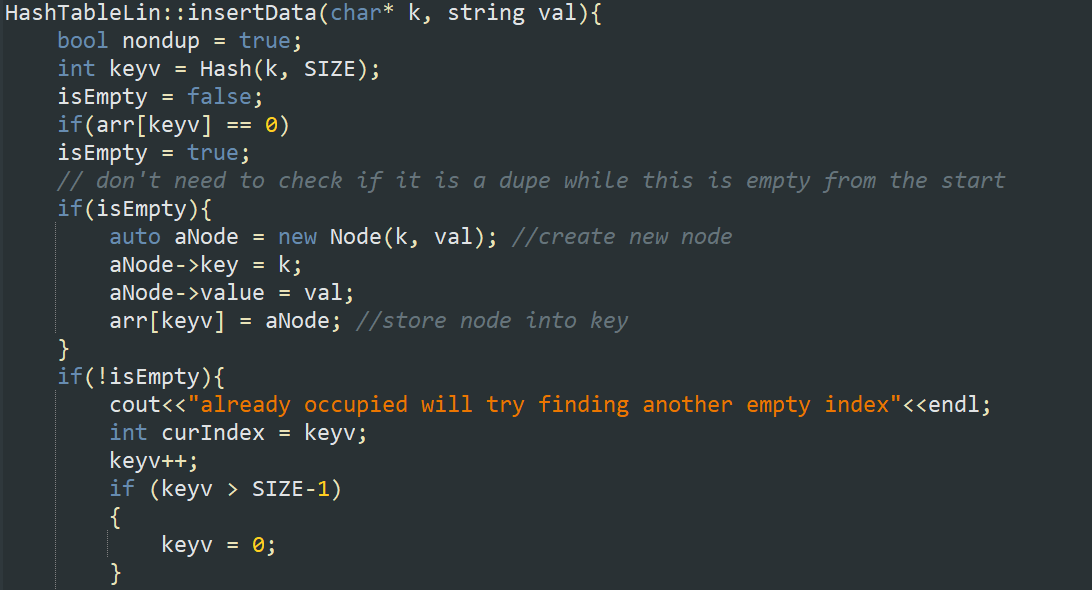
For the formula of our hash function, it is y = y\*m +c for the length of the key (coupon code), So if the key length is 16, the y = y\*m + c is going to be iterated 16 times and then we return that value to get the index with m being the magic number which in the programs case we use 12 and the c is the value of the character of the current key character by in the iteration index.

* 1. **Linear Probe Collision**

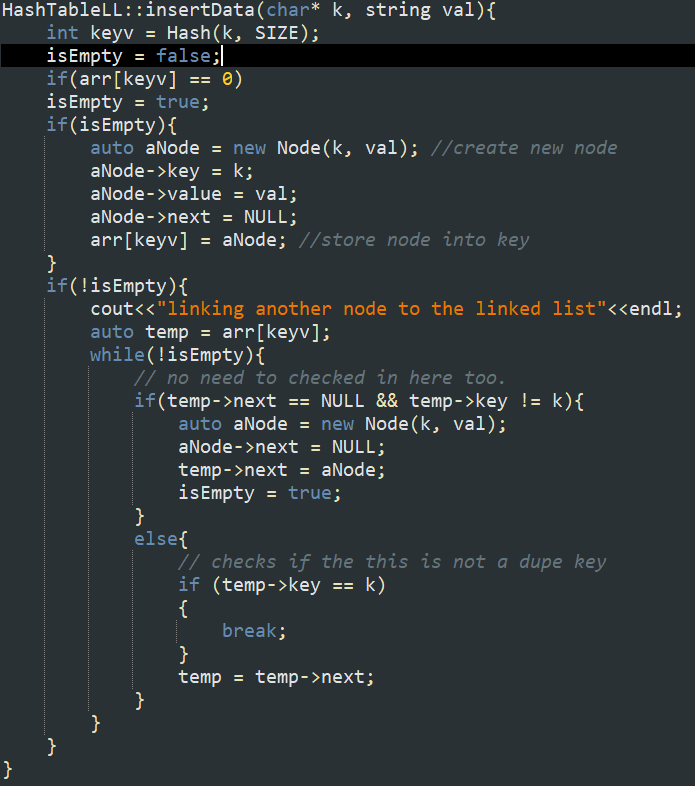
For our hashtable that uses the linear probe collision method, what it will do is essentially just filling the element index specified by our hash function and then will check one by one until it checked all of the indexes if there is no emty space in which case the value is not going to be inserted.

For the hashtable it has the insertData method and the findNode method to process when we want to insert or search a coupon. It also has the attribute arr of size 100 that initially all the elements are 0’s.

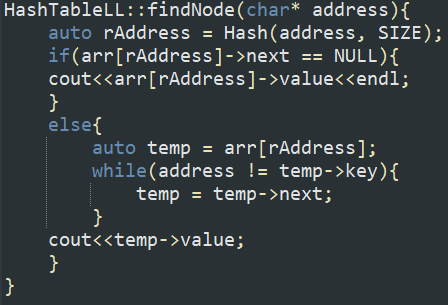
For the insert node Method, it will check if the index is empty and then insert the node if it is empty but if its not, it will one by one check all of the index until it finds an empty element or it reaches back to its original index in which case it will finish and not insert the node since it means that the array is full. When it finds that the computed index is occupied, it will take note of that index and then adding one to it and check if the array with the index is full or not. If the index gotten out of the range of the array length, it will then change immediately to zero to avoid an error when checking the arr attribute of the hashtable. It will then stop when it finds an empty index or when it reaches the original computed index in which case the array is full and that the node is not going to be inserted. The findNode method also work in almost the same way the only difference is that it is checking if the value matches the key attribute of the node which is the coupon code that we searched. The checking algorithm goes roughly the same with the insertNode when the index is occupied.



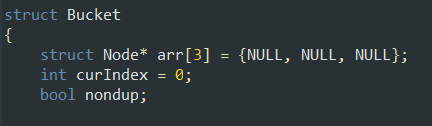
* 1. **Linked List Collision Resolution**

The hashtable structure itself is similar to the structure of the linear probe collision resolutin method hashtable. With this hashtable, the difference is in the insertNode and findNode function since it uses a linked list rather than a linear array.

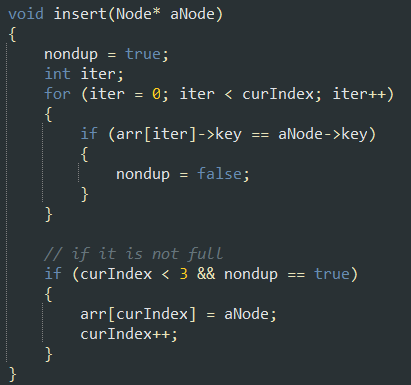
For the insertData method of the linked list collision resolution, it will check at first if the index from the hashtable is empty in which case it will assign it with the node (coupon code) that has the next value of NULL. But if it finds that it is not empty, it will then just attach the node to the chain of the linked lists in each index. And as for the findNode method of the linked list collision resolution method, It also works kinda the same way with the insertData method with the difference that in the findData method, rather then checking if it is empty it will check if the key value matches the search value from the linked list of the index computed using the hashFunction.



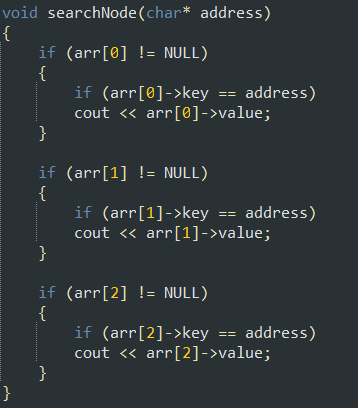
* 1. **Bucket Collision Resolution**

As for the hashtable that uses the bucket collision resolution, we created a data structure of bucket that contains an array of 3 elements that will store the nodes.

It has curIndex to indicate which index is empty right now and then nondup which will going to be used to prevent any duplicate code to be inserted in the bucket. As for the method of inserting and searching it is going to be implemented inside the bucket structure rather then the hashTable. In the hashTable we are going to only call the method of the bucket rather than having the method itself compiled there.



For the insert method, it will first check the arr in the bucket for any duplicate. If it didn’t find any it will insert the node to the index of the curIndex if it is smaller than 3 in which case it is not full yet.



And as for the search method, it will check the arr values one by one to try and check whether it matches the search value or not. If it is the same it will print out the value of the coupon that has the same key.

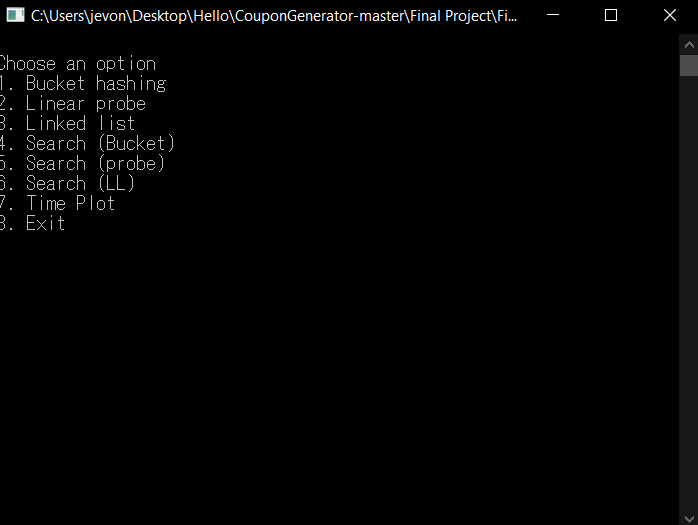
1. **Program Manual**
   1. **Executing the program**
2. Double click the FinalProject.exe in the “Final Project” folder

****

* 1. **Inserting Coupon**

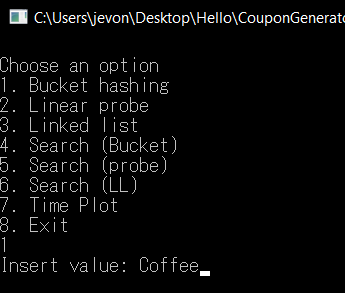
1. Select the desired collision resolution

In the example below, bucket hashing is chosen as the collision resolution method



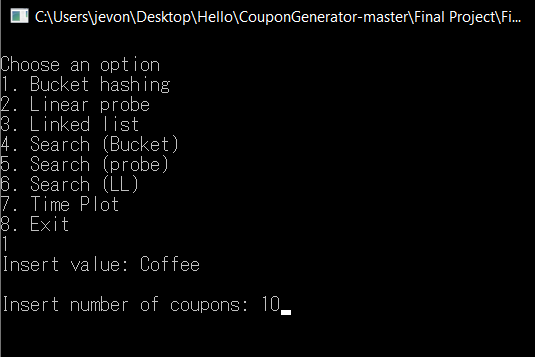
1. Insert value of the coupons (preferably less than 100 item)

In the example below, coffee is assigned as the value of the coupon

****

1. Insert the number of coupons

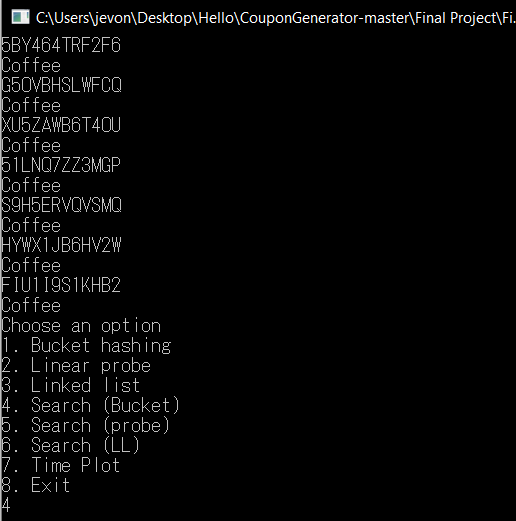
In the example below 10 coupons were generated and stored into the hash tables

****

* 1. **Searching Coupon**

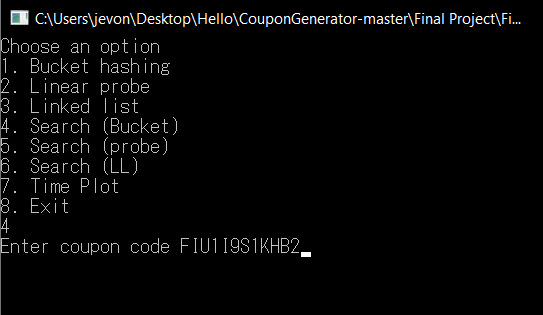
1. Select search method that correspond to the previous selected collision method

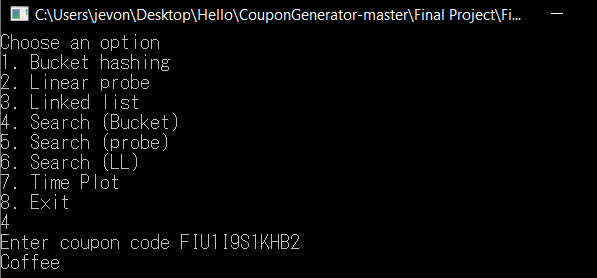
In the example “Search (Bucket)” is chosen as previously bucket hashing was chosen as the collision resolution method



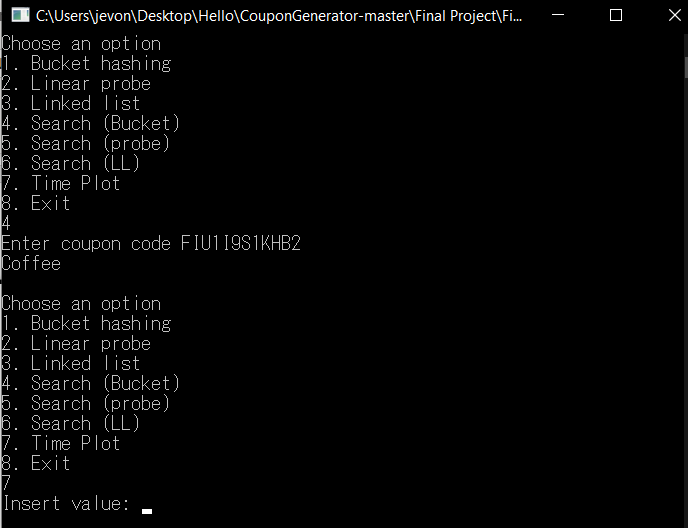
1. Enter coupon code that have been generated

In the example below “FIU1I9S1KHB2” code is used

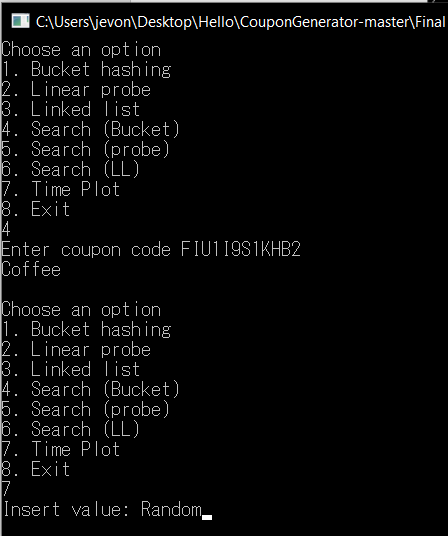


1. The Coffee value is displayed as the coupon has the value of Coffee
   1. **Time plot**

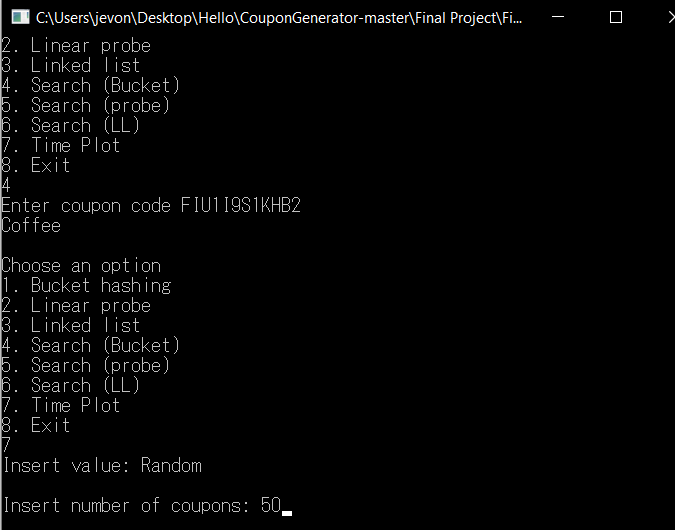
Time plot is a method to analyze the speed of searching of each collision method.

1. Choose Time Plot in the menu 

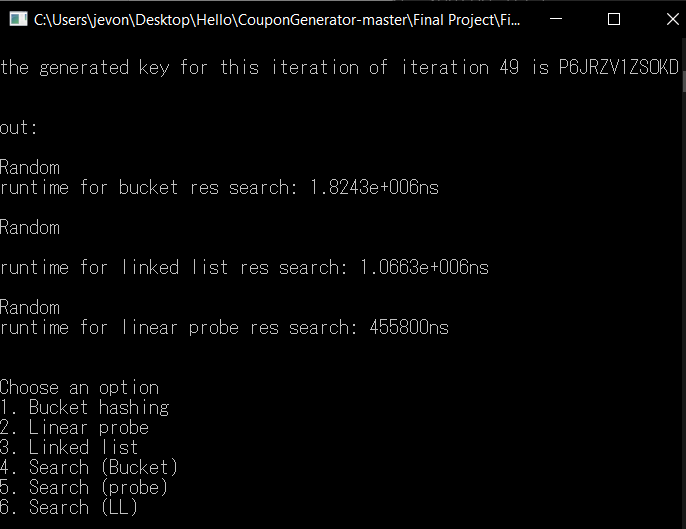
2. Insert any value



1. Insert number of coupons generated (preferably less than 100 item)



1. The time plot result will be displayed along with the found item for each method



1. **Demo Video**

****

1. **GitHub Link**

**https://github.com/JVuns/CouponGenerator**