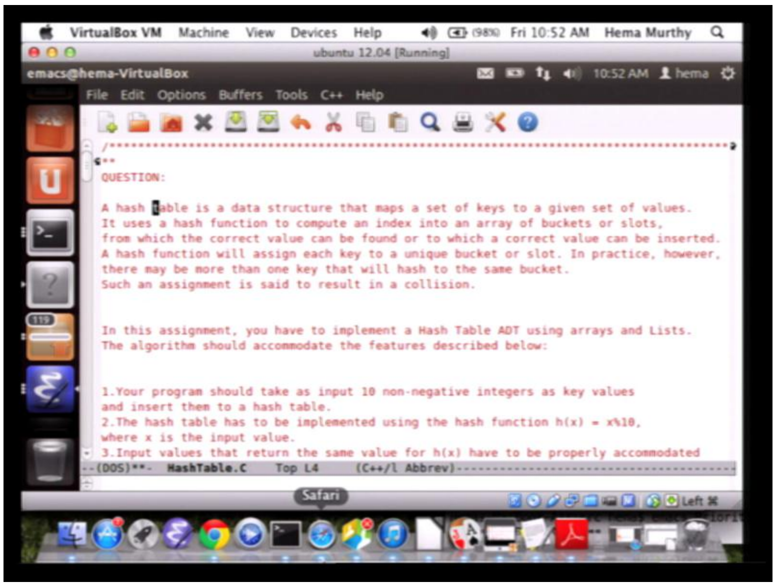
Programming, Data Structures and Algorithms

Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute Technology, Madras

Lecture – 45
Assignment on Data Structures

So, now, what you going to see is to see the solution to the hash table question.

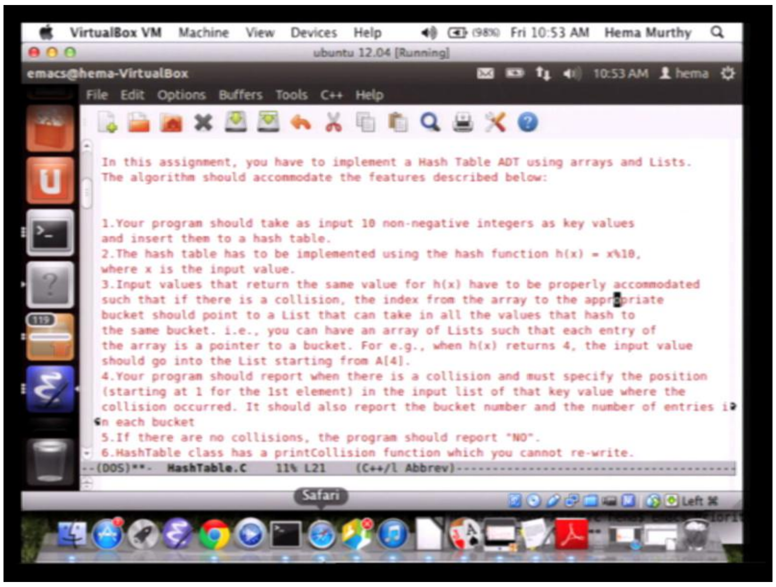
(Refer Slide Time: 00:19)



```
/***************************************************************************
**
QUESTION:

A hash table is a data structure that maps a set of keys to a given set of values.
It uses a hash function to compute an index into an array of buckets or slots,
from which the correct value can be found or to which a correct value can be inserted.
A hash function will assign each key to a unique bucket or slot. In practice, however,
there may be more than one key that will hash to the same bucket.
Such an assignment is said to result in a collision.


In this assignment, you have to implement a Hash Table ADT using arrays and Lists.
The algorithm should accommodate the features described below:


1.Your program should take as input 10 non-negative integers as key values
and insert them to a hash table.
2.The hash table has to be implemented using the hash function h(x) = x%10,
where x is the input value.
3.Input values that return the same value for h(x) have to be properly accommodated
--(DOS)**-  HashTable.C    Top L4    (C++/l Abbrev)-----------------------------
```

So, let us go over this question once again, what is it say? A hash table is a data structure that maps a set of keys to a given set of values. It uses a hash function to compute an index into an array of buckets or slots, from which the correct value can be found or to which a correct value can be inserted. A hash function will assign each key to a unique bucket or slot. In practice; however, there are many more than 1 key that will hash to the same bucket such an assignment is set to result in a collision. In this assignment you have to implement a hash table using arrays and lists.

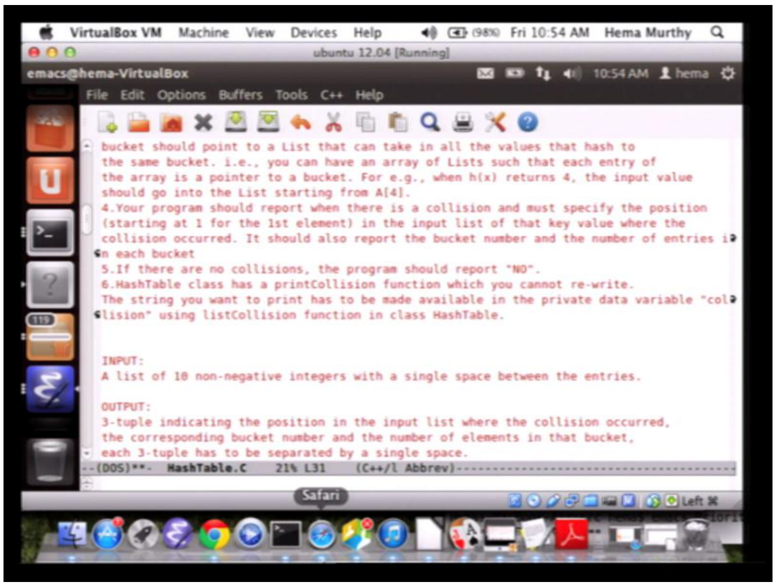
(Refer Slide time: 00:58)



```
In this assignment, you have to implement a Hash Table ADT using arrays and Lists.
The algorithm should accommodate the features described below:


1.Your program should take as input 10 non-negative integers as key values
and insert them to a hash table.
2.The hash table has to be implemented using the hash function h(x) = x%10,
where x is the input value.
3.Input values that return the same value for h(x) have to be properly accommodated
such that if there is a collision, the index from the array to the appropriate
bucket should point to a List that can take in all the values that hash to
the same bucket. i.e., you can have an array of Lists such that each entry of
the array is a pointer to a bucket. For e.g., when h(x) returns 4, the input value
should go into the List starting from A[4].
4.Your program should report when there is a collision and must specify the position
(starting at 1 for the 1st element) in the input list of that key value where the
collision occurred. It should also report the bucket number and the number of entries i
n each bucket
5.If there are no collisions, the program should report "NO".
6.HashTable class has a printCollision function which you cannot re-write.
--(DOS)**-  HashTable.C    11% L21    (C++/l Abbrev)-----------------------------------
```

The algorithm should accommodate the features described. You should take as input a set of 10 non negative integers and insert them into a hash table using this hash function where h of x equal to x for set 10. Then, input values that returns the same value for h of x have to be properly accommodated such that if there is a collision the index from the array to the appropriate bucket should point to a list that can take in all the values that hash to the same bucket. So, basically what you need is you need an array of list such that each entry of the array is a pointer to a bucket. For example, when h of x returns 4 then input value should go in to the list starting from a 4 if a is your hash table.

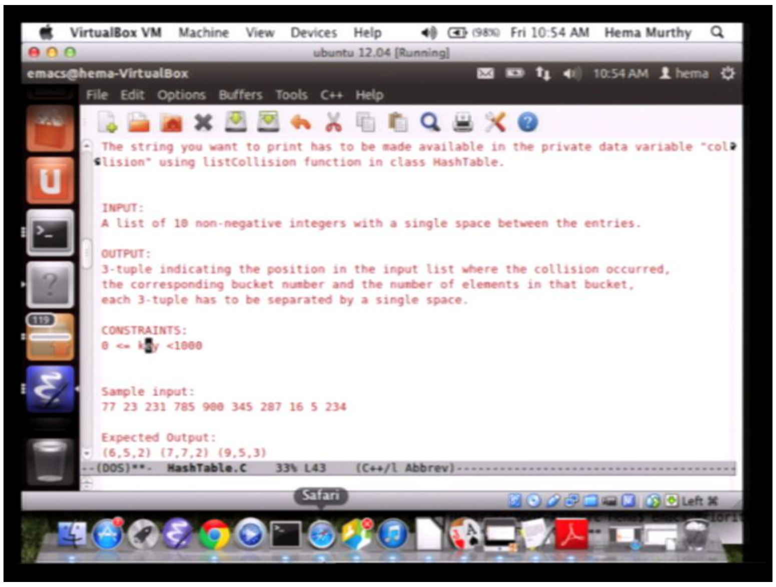
(Refer Slide Time: 01:45)



```
bucket should point to a List that can take in all the values that hash to
the same bucket. i.e., you can have an array of Lists such that each entry of
the array is a pointer to a bucket. For e.g., when h(x) returns 4, the input value
should go into the List starting from A[4].
4.Your program should report when there is a collision and must specify the position
(starting at 1 for the 1st element) in the input list of that key value where the
collision occurred. It should also report the bucket number and the number of entries i
n each bucket
5.If there are no collisions, the program should report "NO".
6.HashTable class has a printCollision function which you cannot re-write.
The string you want to print has to be made available in the private data variable "col
lision" using listCollision function in class HashTable.


INPUT:
A list of 10 non-negative integers with a single space between the entries.

OUTPUT:
3-tuple indicating the position in the input list where the collision occurred,
the corresponding bucket number and the number of elements in that bucket,
each 3-tuple has to be separated by a single space.
--(DOS)**-  HashTable.C    21% L31    (C++/l Abbrev)-----------------------------------
```

Your program should report when there is a collision and must specify the position starting at one (Refer Time: 01.53) the list element in the input list of that key value where the collision occurred. It should also report the bucket number and the number of entries in the bucket. And if there are no collisions, the program should report no collision. The hash table class has been provided with the print collision function which you cannot re-write. What you have to do is the string that you want to print has to be made available in the private data variable collision using list collision function in the class hash table.

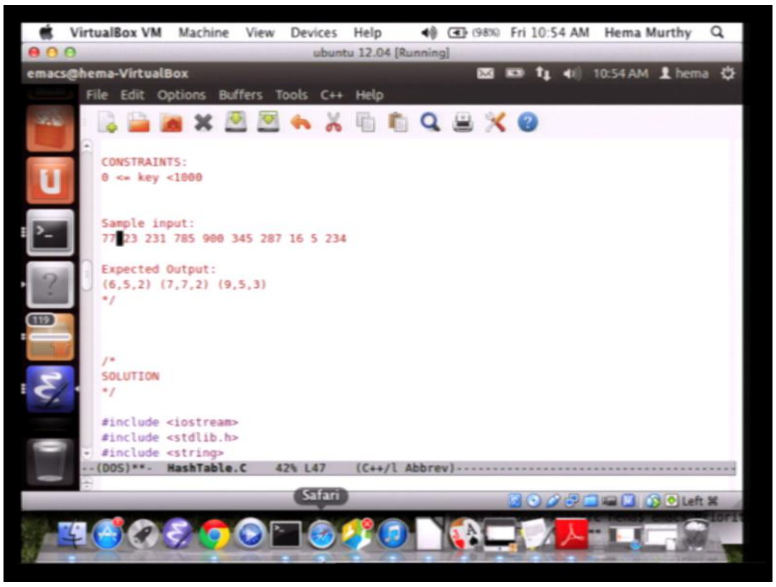
(Refer Slide Time: 02:25)



```
The string you want to print has to be made available in the private data variable "col
lision" using listCollision function in class HashTable.


INPUT:
A list of 10 non-negative integers with a single space between the entries.

OUTPUT:
3-tuple indicating the position in the input list where the collision occurred,
the corresponding bucket number and the number of elements in that bucket,
each 3-tuple has to be separated by a single space.

CONSTRAINTS:
0 <= key <1000


Sample input:
77 23 231 785 900 345 287 16 5 234

Expected Output:
(6,5,2) (7,7,2) (9,5,3)
--(DOS)**-  HashTable.C    33% L43    (C++/l Abbrev)----------------------------
```

So, let us see how this can be implemented so, basically the constraints on the keys are it can be a value between 0 and 1000.

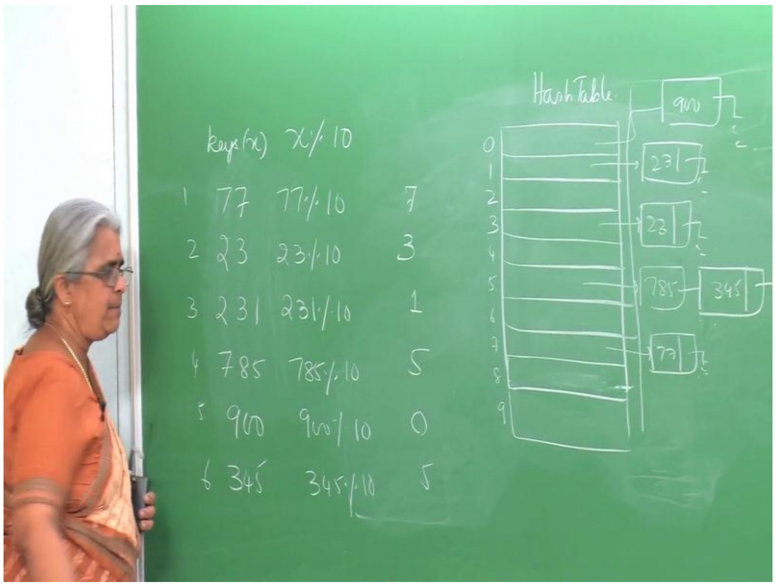
(Refer Slide Time: 02:33)



```
CONSTRAINTS:
0 <= key <1000


Sample input:
77 23 231 785 900 345 287 16 5 234

Expected Output:
(6,5,2) (7,7,2) (9,5,3)
*/




/*
SOLUTION
*/

#include <iostream>
#include <stdlib.h>
#include <string>
--(DOS)**-  HashTable.C    42% L47    (C++/l Abbrev)----------------------------
```

And if you take a sample input like this what does it that we are say () 77 23 231 785 900 345 287 16 5 and 24. So, when you look at this 77 for example, when you take percent 10 it goes into the seventh bucket is the first element there is no collision. Let the go to the board and give this example again for you are ready reflects.

(Refer slide Time: 03:14)



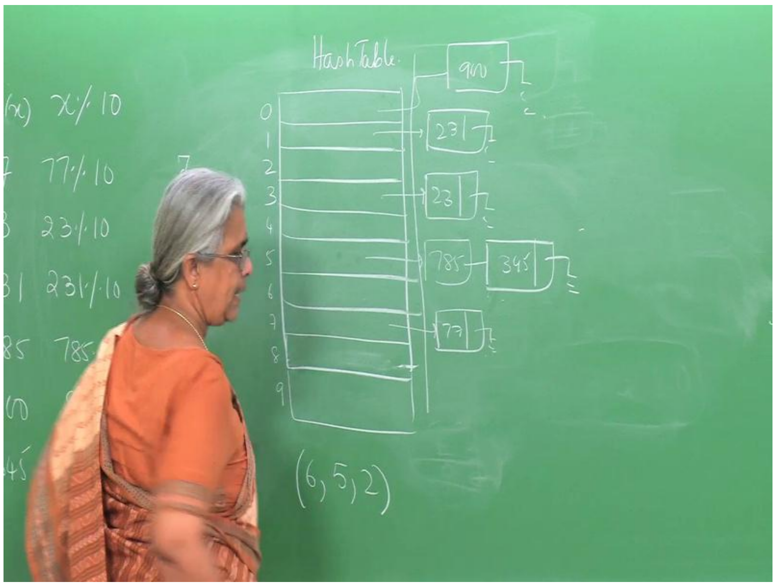
So, what you have here? We have 77 and let me retrieve the numbers appears 77 23 231 785 900 and then I just go this first example 345. So, this is the key. So, what happens? When you take this 77 percent 10 this gives you a value 7. This 23 percent 10 will give you a value of 3, 231 percent 10 gives you a value of 1 and 785 percent 10 gives you a value 5, 900 percent 10 gives you a value of 0, 345 percent 10 gives you a value 5. So, now I am stopping with this, because now what is happen? So, what did you see now? 77 hashes to an index of 7, 23 will hash to an index of 3, 231 will hash to an index of 1, 785 index hashes to an index of 5, 900 to 0, 345 to 5.

So, what did you have now? We have an array of first we have to define array. Now, in this array what are the how many elements to be have? We have 10 elements in the array 2 3 4 5 6 7 8 and we call this is a bucket, each one of the indices of this array is called a bucket, because it is actually holds more than one value. So, now, what are we going to do? We said due to store these case in this hash table so, this is the hash table. Now, the problem is because, the, we have only 10 entries in the hash table. Clearly, because you are talking about 0 through 9 9 9 as a number set are possible you have you have more than 10 values and therefore, main numbers will hash to the same bucket..

So, that is the meaning of a collision has been already solved. So, now let see where we will put 77 we will put 77 over here. So, we create a list at this point and put 77 over here 23 will go in to the third bucket we put it over here. Then 231 will go in to the first bucket. So, far so, good there are no collisions. 785 will go in to the fifth bucket, this is 785 () here then 900 will go in to the zeroth bucket where remainder is zero and we have still not seen any collision. Now, 345 comes alone and we have a collision so, what is the collision now?

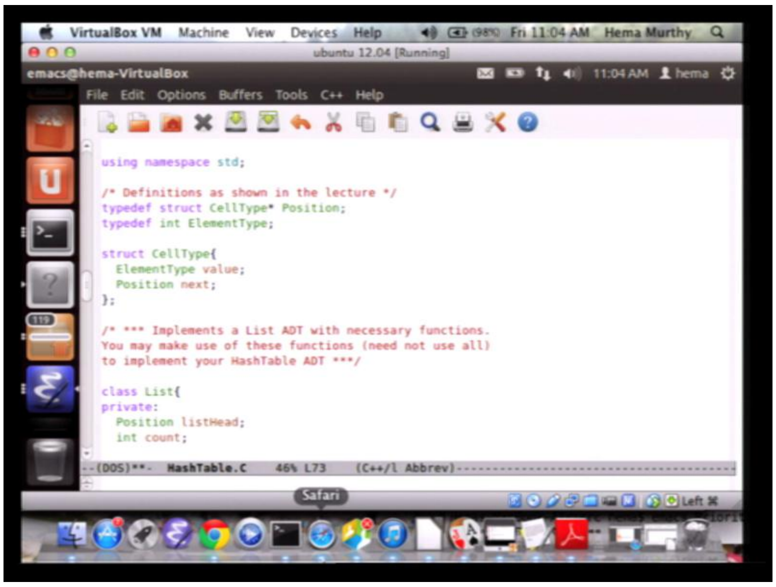
So, now, at 340 what is that? It remainder is 5 therefore, would goes into the fifth bucket we already have a 1 element. So, we append this to the dust. So, what is this now here? Notice that each element of the bucket is a list this, implementation of a list can be done either using an array or a link list the list a d t is what you need to use over there. Now, what do you supposed to report? You are suppose to report the index in the element in the input list. So, the input list again 77 23 231 785 900 and 345. So, if you look at the numbers over here. So, this is the sixth element in the list where the collision is occurred and what is the bucket index? The index is 5 at how many elements are there in that particular bucket? There are 2 elements.

(Refer Slide Time: 07:34)



So, now, and you are supposed to report the result like this 6 5 comma 2 and that is what is given in the sample output and have more examples there are given in this in the question that is being given ().

(Refer Slide Time: 07:55)

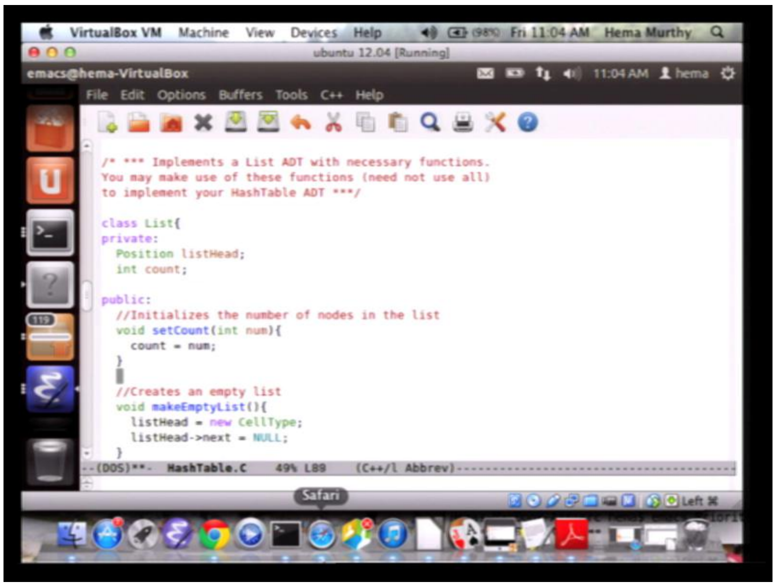


```
using namespace std;

/* Definitions as shown in the lecture */
typedef struct CellType* Position;
typedef int ElementType;

struct CellType{
  ElementType value;
  Position next;
};

/* *** Implements a List ADT with necessary functions.
You may make use of these functions (need not use all)
to implement your HashTable ADT ***/

class List{
private:
  Position listHead;
  int count;
```

```
--(DOS)**-  HashTable.C    46% L73    (C++/l Abbrev)----------------------------
```

So, let us look at a solution to this particular program. So, this is kind of similar to the ADT that was done in the lectures on the list. So, basically you have a link list representation here.

(Refer Slide Time: 08:07)

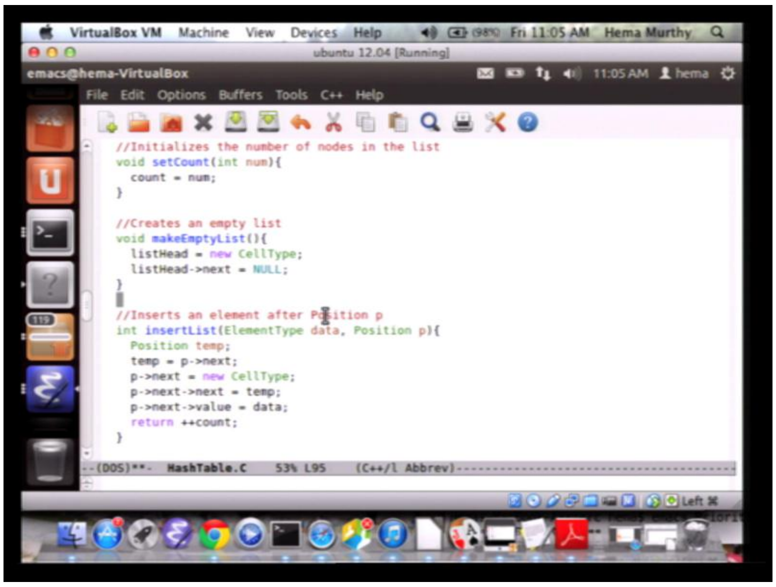


```
/* *** Implements a List ADT with necessary functions.
You may make use of these functions (need not use all)
to implement your HashTable ADT ***/

class List{
private:
  Position listHead;
  int count;

public:
  //Initializes the number of nodes in the list
  void setCount(int num){
    count = num;
  }

  //Creates an empty list
  void makeEmptyList(){
    listHead = new CellType;
    listHead->next = NULL;
  }
```

```
--(DOS)**-  HashTable.C    49% L89    (C++/l Abbrev)----------------------------
```

And position point something point should list 8. In addition to that a new variable has been defined in the, for count in the list.

(Refer Slide Time: 08:25)

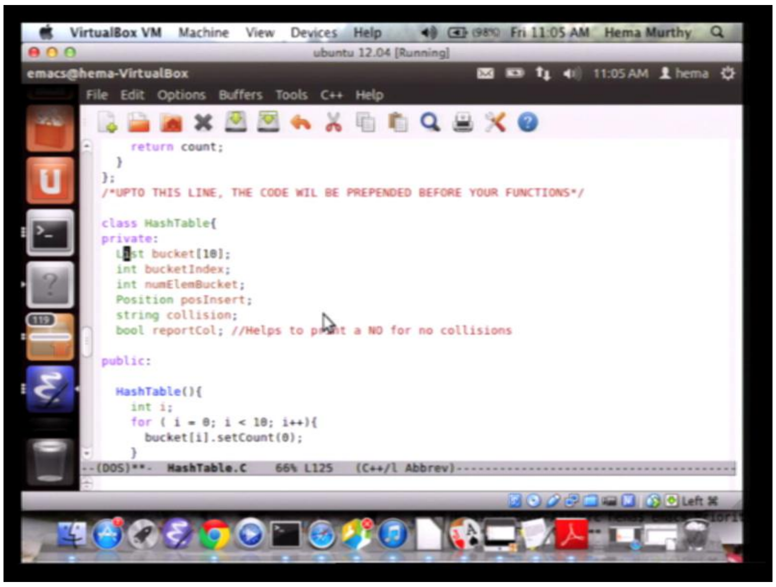


```
//Initializes the number of nodes in the list
void setCount(int num){
  count = num;
}

//Creates an empty list
void makeEmptyList(){
  listHead = new CellType;
  listHead->next = NULL;
}

//Inserts an element after Position p
int insertList(ElementType data, Position p){
  Position temp;
  temp = p->next;
  p->next = new CellType;
  p->next->next = temp;
  p->next->value = data;
  return ++count;
}
```

```
--(DOS)**-  HashTable.C    53% L95    (C++/l Abbrev)----------------------------------------
```

So, it is gives you the number of elements in the list. Suppose count was not there what would you do? You would write a function to count the number of elements in the list by going traversing in the list from the beginning to the end. Then make an empty list creation null node and insert elements in to the list find the end of the list all of them are the same get count returns the number of elements in a given list.

(Refer Slide Time: 08:42)

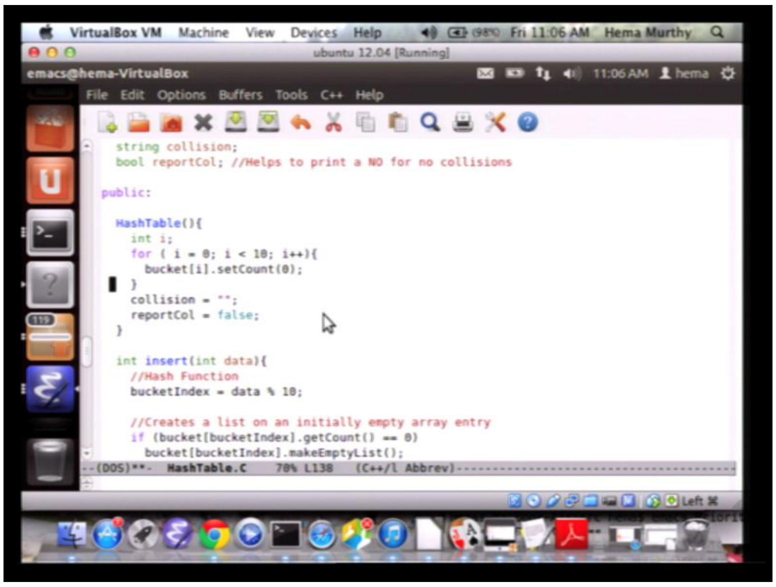


```
    return count;
  }
};
/*UPTO THIS LINE, THE CODE WIL BE PREPENDED BEFORE YOUR FUNCTIONS*/

class HashTable{
private:
  List bucket[10];
  int bucketIndex;
  int numElemBucket;
  Position posInsert;
  string collision;
  bool reportCol; //Helps to print a NO for no collisions

public:

  HashTable(){
    int i;
    for ( i = 0; i < 10; i++){
      bucket[i].setCount(0);
    }
```

```
--(DOS)**-  HashTable.C    66% L125    (C++/l Abbrev)----------------------------------------
```

So, let us look at a class hash table now, what are the class hash table contain now? Notice this variable here list bucket of 10; that means, it is an array of list then it gives you the bucket index, number of element bucket and position in the where you want to insert in the particular bucket. And a strain a collision which has suppose to fill corresponding to the collision which has suppose to the fill up and report called this is required. Because they also say would print () whereas, be and no collision.

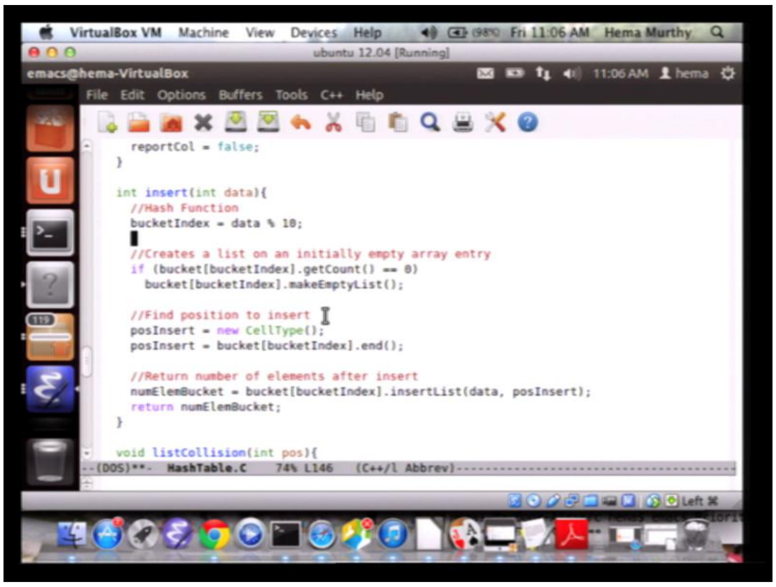
(Refer Slide Time: 09:15)



```
  string collision;
  bool reportCol; //Helps to print a NO for no collisions

public:

  HashTable(){
    int i;
    for ( i = 0; i < 10; i++){
      bucket[i].setCount(0);
    }
    collision = "";
    reportCol = false;
  }

  int insert(int data){
    //Hash Function
    bucketIndex = data % 10;

    //Creates a list on an initially empty array entry
    if (bucket[bucketIndex].getCount() == 0)
      bucket[bucketIndex].makeEmptyList();
--(DOS)**-  HashTable.C    70% L138    (C++/l Abbrev)-----------------------------
```

So, what are we doing now? Initially, because they the hash table is empty we said the counter in the hash table in the count for example, to zero for every one of the list we have an array of list how many do you have? We have 10 elements in the array and the counter in every element in the list every element of the bucket hash table corresponding to the bucket is said to 0..

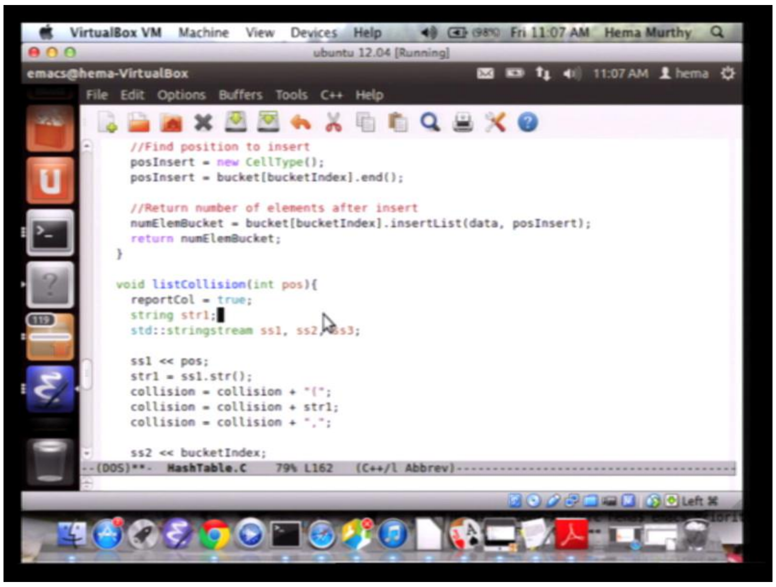
(Refer Slide Time: 09:45)



```
    reportCol = false;
  }

  int insert(int data){
    //Hash Function
    bucketIndex = data % 10;

    //Creates a list on an initially empty array entry
    if (bucket[bucketIndex].getCount() == 0)
      bucket[bucketIndex].makeEmptyList();

    //Find position to insert
    posInsert = new CellType();
    posInsert = bucket[bucketIndex].end();

    //Return number of elements after insert
    numElemBucket = bucket[bucketIndex].insertList(data, posInsert);
    return numElemBucket;
  }

  void listCollision(int pos){
--(DOS)**-  HashTable.C    74% L146    (C++/l Abbrev)-----------------------------
```

Then what are we doing? To insert an element we find the bucket index that is as define in the private variable. Once we get the bucket index and what we do? With this we create an empty list of the count is zero;; that means, there is no list of there then we find the position to insert and insert it at the end of the list. Then what are we doing? We get the number of number elements in the bucket basically at what position it has to be inserted in the given bucket and you return the number of elements on the given bucket.

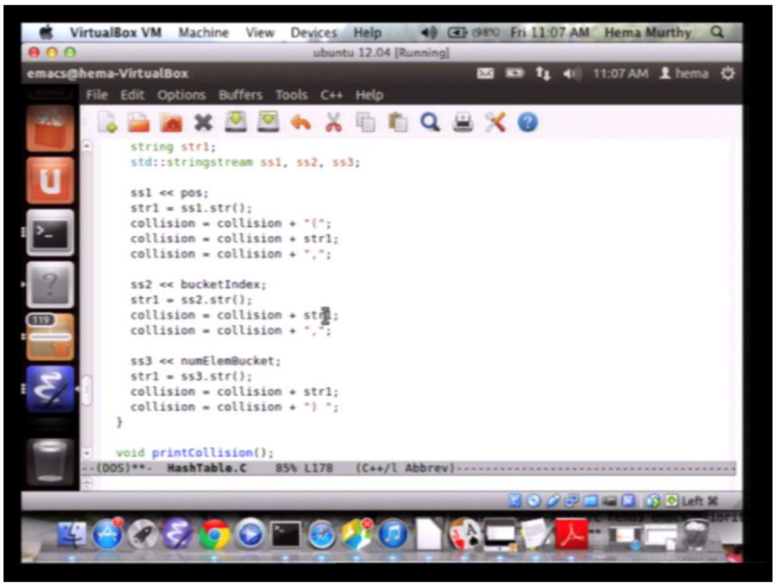
(Refer Slide Time: 10:23)



```
    //Find position to insert
    posInsert = new CellType();
    posInsert = bucket[bucketIndex].end();

    //Return number of elements after insert
    numElemBucket = bucket[bucketIndex].insertList(data, posInsert);
    return numElemBucket;
  }

  void listCollision(int pos){
    reportCol = true;
    string str1;
    std::stringstream ss1, ss2, ss3;

    ss1 << pos;
    str1 = ss1.str();
    collision = collision + "(";
    collision = collision + str1;
    collision = collision + ",";

    ss2 << bucketIndex;
--(DOS)**-  HashTable.C     79% L162    (C++/l Abbrev)--------------------------------
```

That is what we insert function does slight variant of the insert that was done. And once you have this they are all done then list collision what are you doing? You are giving the position where the collision occurred.

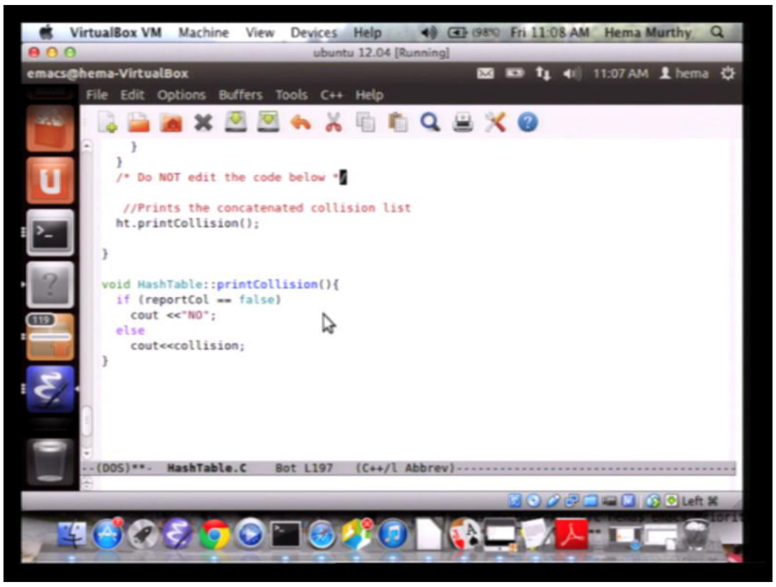
(Refer slide Time: 10:39)



```
    string str1;
    std::stringstream ss1, ss2, ss3;

    ss1 << pos;
    str1 = ss1.str();
    collision = collision + "(";
    collision = collision + str1;
    collision = collision + ",";

    ss2 << bucketIndex;
    str1 = ss2.str();
    collision = collision + str1;
    collision = collision + ",";

    ss3 << numElemBucket;
    str1 = ss3.str();
    collision = collision + str1;
    collision = collision + ") ";
  }

  void printCollision();
--(DOS)**-  HashTable.C     85% L178    (C++/l Abbrev)--------------------------------
```

The bucket index and the number of the number elements in the bucket that is all you need to do. So, basically this is what you have to fill up this () no big deal over here this is string to which your basically concatenating the position and the bucket index and number of elements in the index.

(Refer Slide Time: 10:55)

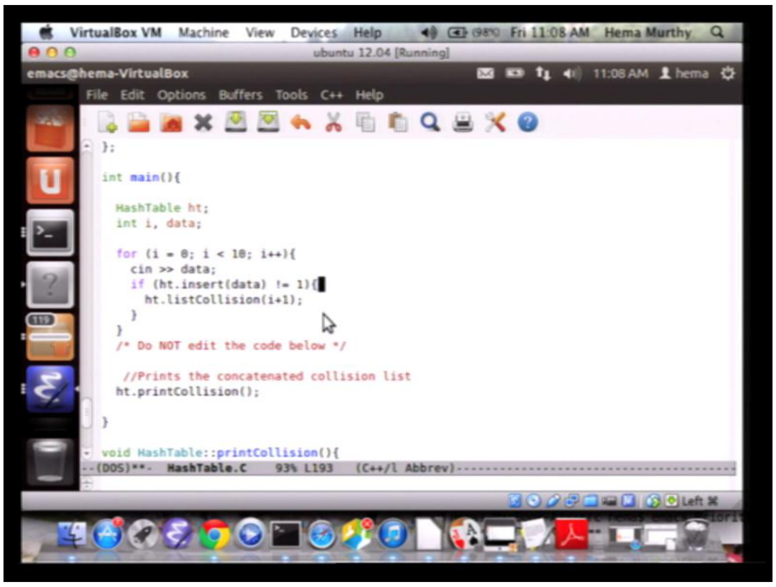


```
    }
  }
  /* Do NOT edit the code below */

   //Prints the concatenated collision list
  ht.printCollision();

}

void HashTable::printCollision(){
  if (reportCol == false)
    cout <<"NO";
  else
    cout<<collision;
}
```

```
--(DOS)**-  HashTable.C    Bot L197   (C++/l Abbrev)----
```

And the print collision will take care of it then what is happening here? The print collision function is given and what is a print collision do if it the, if for example, there was the Boolean variable called report call. Basically there is no collision that occurs then it will print on no over there.

(Refer Slide Time: 11:13)

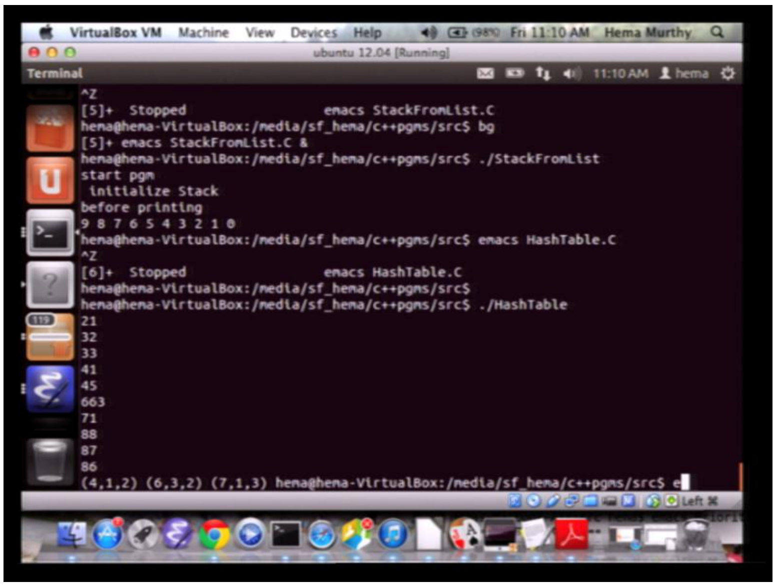


```
};

int main(){

  HashTable ht;
  int i, data;

  for (i = 0; i < 10; i++){
    cin >> data;
    if (ht.insert(data) != 1){
      ht.listCollision(i+1);
    }
  }
  /* Do NOT edit the code below */

   //Prints the concatenated collision list
  ht.printCollision();

}

void HashTable::printCollision(){
```

```
--(DOS)**-  HashTable.C    93% L193   (C++/l Abbrev)----
```

So, this is the essentially what this ADT is all about. So, what are we doing here? Basically are supposed to list the collisions that occur in the given data set as you have seen in the example.

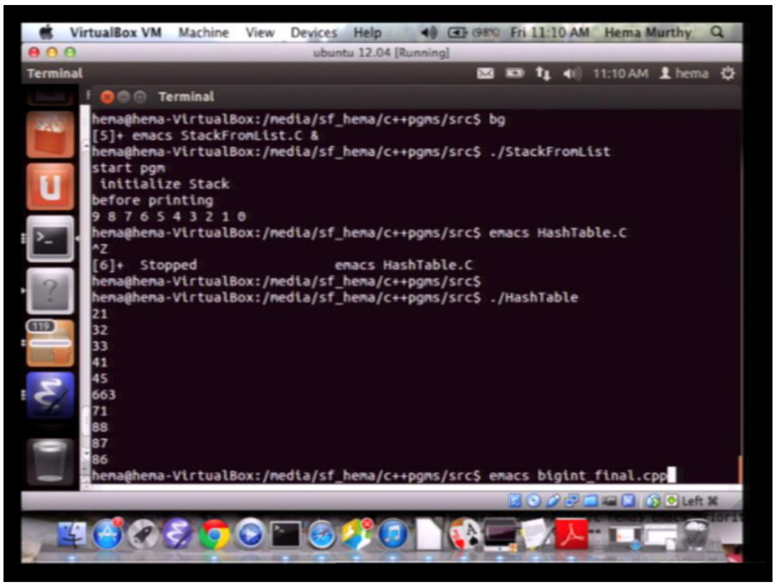
(Refer Slide Time: 11:29)



```
^Z
[5]+  Stopped                 emacs StackFromList.C
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ bg
[5]+ emacs StackFromList.C &
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./StackFromList
start pgm
 initialize Stack
before printing
9 8 7 6 5 4 3 2 1 0
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ emacs HashTable.C
^Z
[6]+  Stopped                 emacs HashTable.C
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./HashTable
21
32
33
41
45
663
71
88
87
86
(4,1,2) (6,3,2) (7,1,3) hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ e
```

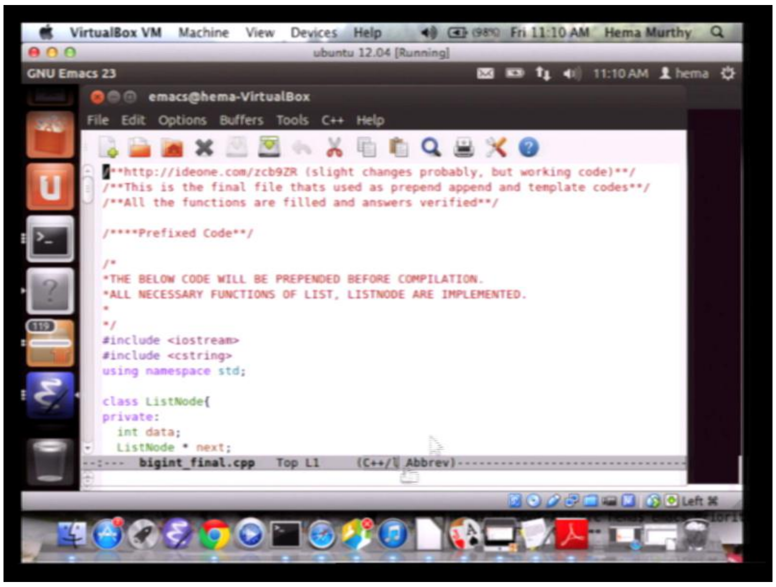
So, let us run this with the given example over here. Table and let us say I gave even ((refer team: 11:52)) 21 32 33 41 45 663 71 88 87 86. So, what is it telling me now, 21 for example, will hash to bucket 1, 32 will hash to bucket 2 and 33 will hash to bucket 3. So, far no collision. But when 41 comes along it hashes to the bucket one again and what does it do? It gives you so, 41 as the 1 2 3 fourth element therefore, the first indexes 4 here. And then what happens you have is the bucket number 1 and the number of element. So, far in that bucket are 21 and 41 which is 2 then 45 again hashes to bucket 5 no collision. Then what happens? 663 hashes to bucket 3 and therefore, there is a collision therefore, this is the sixth element third bucket and now there are 2 elements, because 33 and 663 next what is happening? You have 71, now 71 will hash to one again. So, now, this is the seventh element in the list hashes to the first bucket, but, now 21 41 71 there are 3 elements in the list. So, this completes the solution for the hash table problem. So, next we will look at the solution for the big int problem let us look at the big int.

(Refer slide Time: 13:42)



```
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ bg
[5]+ emacs StackFromList.C &
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./StackFromList
start pgm
 initialize Stack
before printing
9 8 7 6 5 4 3 2 1 0
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ emacs HashTable.C
^Z
[6]+  Stopped                 emacs HashTable.C
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./HashTable
21
32
33
41
45
663
71
88
87
86
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ emacs bigint_final.cpp
```

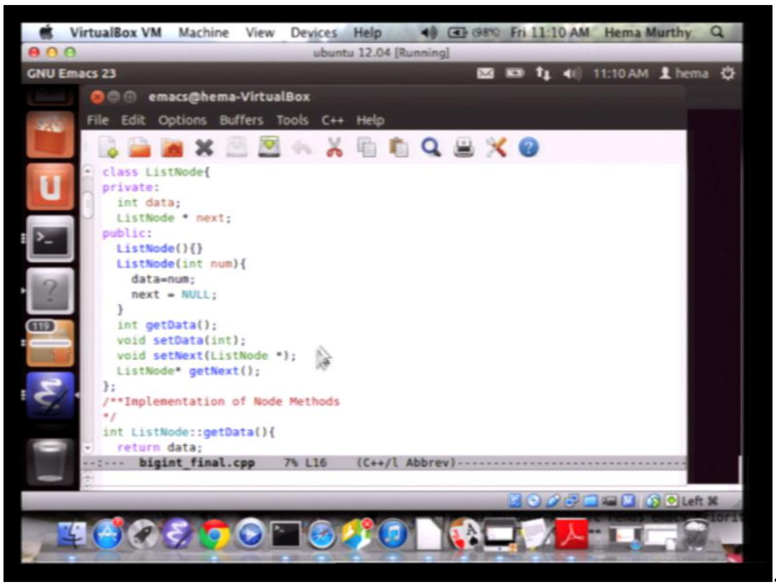
(Refer Slide Time: 13:43)



```
/**http://ideone.com/zcb9ZR (slight changes probably, but working code)**/
/**This is the final file thats used as prepend append and template codes**/
/**All the functions are filled and answers verified**/

/****Prefixed Code**/

/*
*THE BELOW CODE WILL BE PREPENDED BEFORE COMPILATION.
*ALL NECESSARY FUNCTIONS OF LIST, LISTNODE ARE IMPLEMENTED.
*
*/
#include <iostream>
#include <cstring>
using namespace std;

class ListNode{
private:
  int data;
  ListNode * next;
--:---  bigint_final.cpp    Top L1     (C++/l Abbrev)-----------------------------
```

And we will guess see how this is getting done.

(Refer slide Time: 13:52)



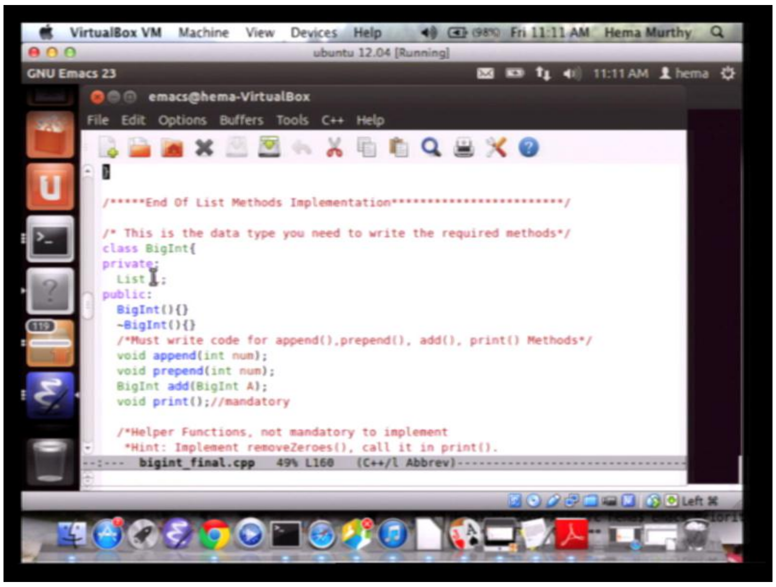
```
class ListNode{
private:
  int data;
  ListNode * next;
public:
  ListNode(){}
  ListNode(int num){
    data=num;
    next = NULL;
  }
  int getData();
  void setData(int);
  void setNext(ListNode *);
  ListNode* getNext();
};
/**Implementation of Node Methods
*/
int ListNode::getData(){
  return data;
```

```
--:---  bigint_final.cpp    7% L16    (C++/l Abbrev)-------
```

.

So, what this is big int now, you are ask to find the sum of 2 large numbers here again and other implementation of list is given a different implementation.

(Refer Slide Time: 14:01)

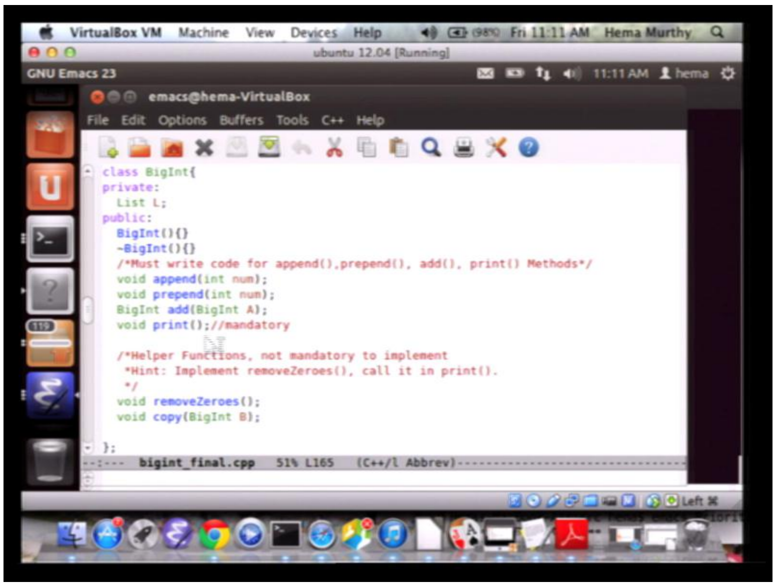


```
}

/*****End Of List Methods Implementation************************/

/* This is the data type you need to write the required methods*/
class BigInt{
private:
  List ;
public:
  BigInt(){}
  ~BigInt(){}
  /*Must write code for append(),prepend(), add(), print() Methods*/
  void append(int num);
  void prepend(int num);
  BigInt add(BigInt A);
  void print();//mandatory

  /*Helper Functions, not mandatory to implement
   *Hint: Implement removeZeroes(), call it in print().
```

```
--:---  bigint_final.cpp   49% L160   (C++/l Abbrev)-------
```

And you are suppose to use this implementation of list to perform big int arithmetic. So, what is being done here let us see. So, what are we doing? To perform big int arithmetic what us suggested is you have to implement at last call big int now, and with in this class you using a private variable list just like we did this stack from the list.

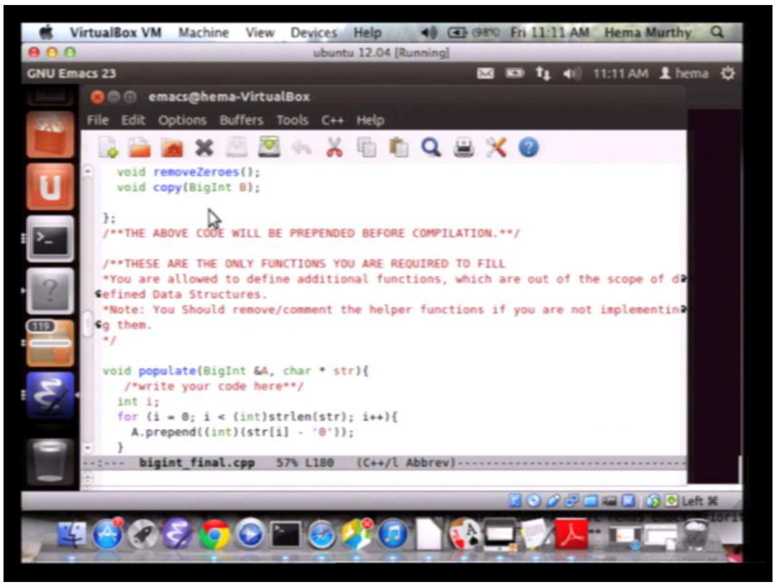
(Refer Slide Time: 14:21)



```
class BigInt{
private:
  List L;
public:
  BigInt(){}
  ~BigInt(){}
  /*Must write code for append(),prepend(), add(), print() Methods*/
  void append(int num);
  void prepend(int num);
  BigInt add(BigInt A);
  void print();//mandatory

  /*Helper Functions, not mandatory to implement
   *Hint: Implement removeZeroes(), call it in print().
   */
  void removeZeroes();
  void copy(BigInt B);

};
--:---  bigint_final.cpp   51% L165   (C++/l Abbrev)----------------------------
```

And what are you doing? You are appending a given number prefixing the big int number and you have to perform is add operations and print.

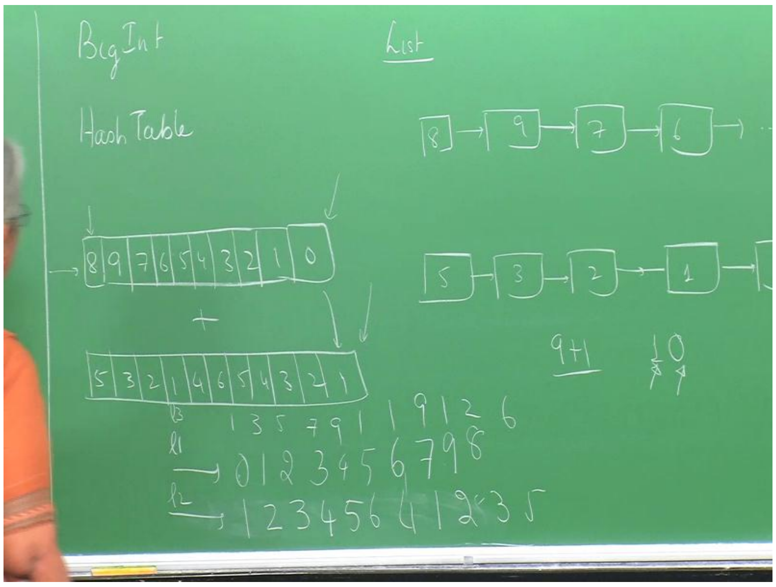
(Refer Slide Time: 14:30)



```
  void removeZeroes();
  void copy(BigInt B);

};
/**THE ABOVE CODE WILL BE PREPENDED BEFORE COMPILATION.**/

/**THESE ARE THE ONLY FUNCTIONS YOU ARE REQUIRED TO FILL
*You are allowed to define additional functions, which are out of the scope of d
efined Data Structures.
*Note: You Should remove/comment the helper functions if you are not implementin
g them.
*/

void populate(BigInt &A, char * str){
   /*write your code here**/
  int i;
  for (i = 0; i < (int)strlen(str); i++){
    A.prepend((int)(str[i] - '0'));
  }
--:---  bigint_final.cpp   57% L180   (C++/l Abbrev)----------------------------
```

If you want you can have suppose there are leading is () 0 you can removed are whatever. So, what are we doing now? The first function that is given to you it is populate and it is suppose to write you code here.

(Refer Slide Time: 14:52)



So, what are we doing? As I am mentioned previously we read the big integers this is the big int 1 and this is big int 2 we read them as strings and what is being done in this particular program subbing clever is being done what is done is () what is the big int using? It is using a list. So, what we doing is we are inserting every times so, I am reading from left to right as well as you read this elements approved in the first position next 9 is put in the first position. So, actually what is represented in the big int is I will tell you why it is being done list 6 5 4 3 2 1 and 0 it written text.

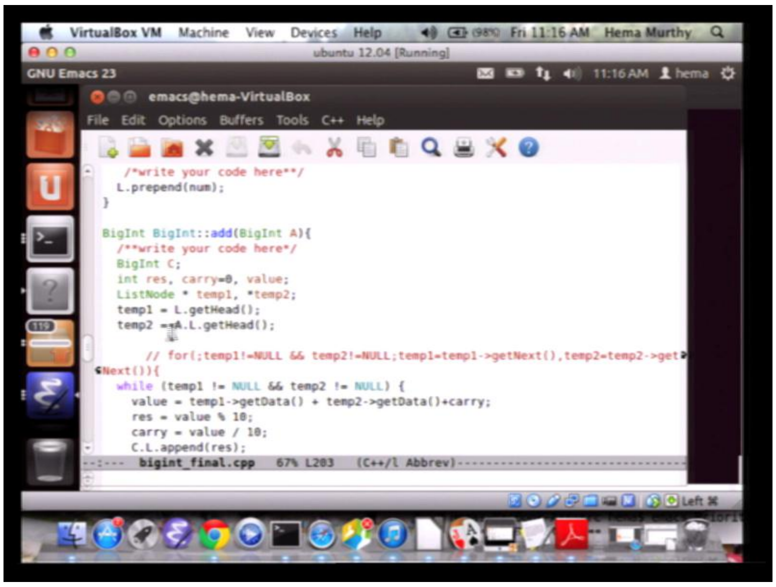
And similarly, this other number here is written as 1 2 is inserted into the () list 1 2 3 4 5 6 4 1 2 and 3 and 5. The reason for this is both of these let me rewrite this again what we are doing is in fact, this is the starting point in list one this let me call this list 1 are the big int 1 and this is 8 2. So, what is being done is I store 1 2 3 4 5 6 4 1 2 3 and 5. The reason for being this is that when we add 2 numbers remember we have to add from the least significant number position. So, what happens even these 2 are the beginning you can add these and propagate the carry..

Once you propagate the carry what you will do now? See if you add these 2 numbers this gives you 1 there is no carry 2 plus 1 for put it back out here this is put create a new list lecture 3 1 3 5 7 9. Now, this is an 11 here. So, now, this 1 should be added to this 2 elements 6 plus 4 10, 10 plus 1 11 and this one more here 9 here 9 plus 2 11 and this becomes 12 and then finally, this 5 has to be added to the carry and should give you 6. So, this is what the 2 numbers should be. So, what are you saying? 0 plus 1 is 1 2 plus 1 is 3 3 plus 2 is 5 4 plus 3 is 7 9 11 therefore, this becomes 6 plus 5 11 over here and 1 is carry.

So, 6 plus 2 I wrote the number from over here 5 6 7 9 and 8. So, what would you do here? 0 6 plus 5 11 well, we are here now, so, basically 0 and 1 1 and 2 2 and 3 5 7 4 and 5 9 6 plus 5 11. This is what is happening here then after the 6 and 5 have been add I get a 11. Therefore, 1 is the carry then after I have carried the 1 here I have here 6 plus 1 7, 7 plus 4 11 again here then again there is one more carry 7 plus 1 8 gives you a 9. Because the carry there and then 9 plus 2 11 and whether carry there and then we have 8 plus 3 11 plus 1 carry () said 12. And finally, this number which are longer than the second number longer than the first number you add the carry to it.

So, essentially you should get this particular result, but, the result is in the reverse order. So, basically finally, 5 plus 1 6 is what you are suppose to get. So, this should be the result of the number, but, it is in the reverse order. So, what is done in this particular program if you look at it is to make this convenient, because you can start from the beginning of the list the population of the integer is done such that you print them put in the beginning..

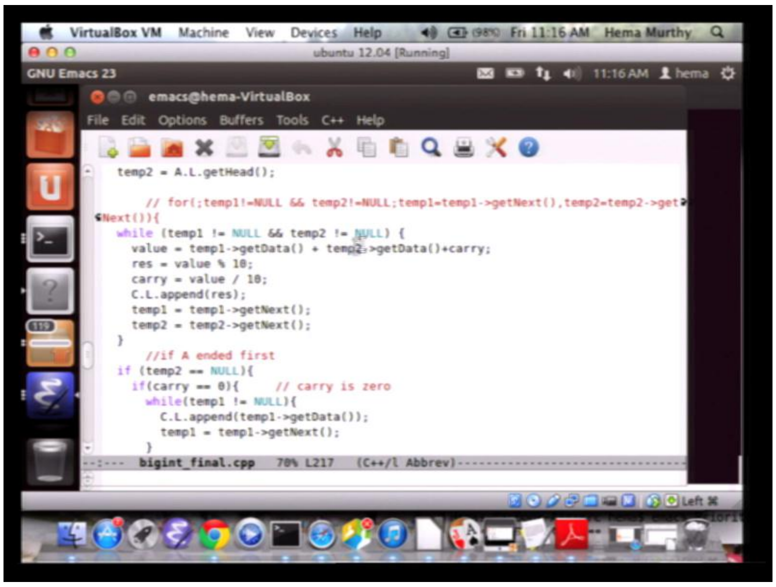
(Refer slide Time: 19:07)



```
  /*write your code here**/
 L.prepend(num);
}

BigInt BigInt::add(BigInt A){
 /**write your code here*/
 BigInt C;
 int res, carry=0, value;
 ListNode * temp1, *temp2;
 temp1 = L.getHead();
 temp2 = A.L.getHead();

    // for(;temp1!=NULL && temp2!=NULL;temp1=temp1->getNext(),temp2=temp2->get
Next()){
 while (temp1 != NULL && temp2 != NULL) {
   value = temp1->getData() + temp2->getData()+carry;
   res = value % 10;
   carry = value / 10;
   C.L.append(res);
--:---  bigint_final.cpp   67% L203   (C++/l Abbrev)-----------------------
```

Then there is the basically there are two functions append and prepend which you give here. Now, how you are suppose to write it when you add so, what you need now when you adding now, and yet, because you are adding two elements at a time. They will be a reminder and a carry which is represented by this and what are temp 1 and temp 2? Temp 1 temp 2 correspond to the 2 list which represent the 2 numbers.

(Refer Slide Time: 19:40)

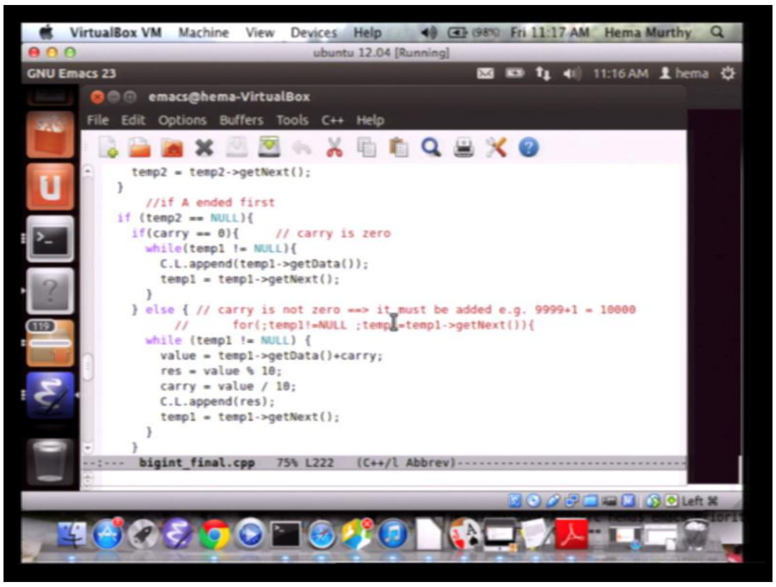


```
  temp2 = A.L.getHead();

      // for(;temp1!=NULL && temp2!=NULL;temp1=temp1->getNext(),temp2=temp2->get
Next()){
  while (temp1 != NULL && temp2 != NULL) {
    value = temp1->getData() + temp2->getData()+carry;
    res = value % 10;
    carry = value / 10;
    C.L.append(res);
    temp1 = temp1->getNext();
    temp2 = temp2->getNext();
  }
      //if A ended first
  if (temp2 == NULL){
    if(carry == 0){      // carry is zero
      while(temp1 != NULL){
        C.L.append(temp1->getData());
        temp1 = temp1->getNext();
      }
```

```
--:---  bigint_final.cpp    70% L217    (C++/l Abbrev)----------------------
```

So, the first part what is done is by both of them are the same length till that is what is done is by temp 1 not equal to null and temp 2 not equal to null. What is being done is you get that data from both this lists you find the remainder you find the carry. Then you appended 2 other lists and you keep on to repeating this process until you come to the end of at least one of the list.

(Refer Slide Time: 20:06)

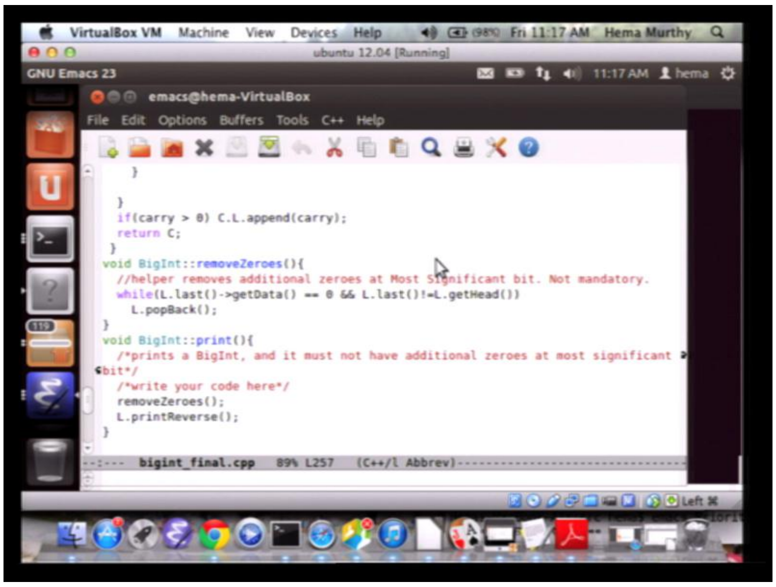


```
    temp2 = temp2->getNext();
  }
      //if A ended first
  if (temp2 == NULL){
    if(carry == 0){      // carry is zero
      while(temp1 != NULL){
        C.L.append(temp1->getData());
        temp1 = temp1->getNext();
      }
    } else { // carry is not zero ==> it must be added e.g. 9999+1 = 10000
        //      for(;temp1!=NULL ;temp1=temp1->getNext()){
      while (temp1 != NULL) {
        value = temp1->getData()+carry;
        res = value % 10;
        carry = value / 10;
        C.L.append(res);
        temp1 = temp1->getNext();
      }
    }
```

```
--:---  bigint_final.cpp    75% L222    (C++/l Abbrev)----------------------
```

Then what we you do is as I said in this particular case for example, list 1 ended first and then list 2 ends later. So, it check which one ends first which one ends later and if there is a carry you add the carry also same thing is done for if the other one ended first and the carry is append..

(Refer Slide Time: 20:25)

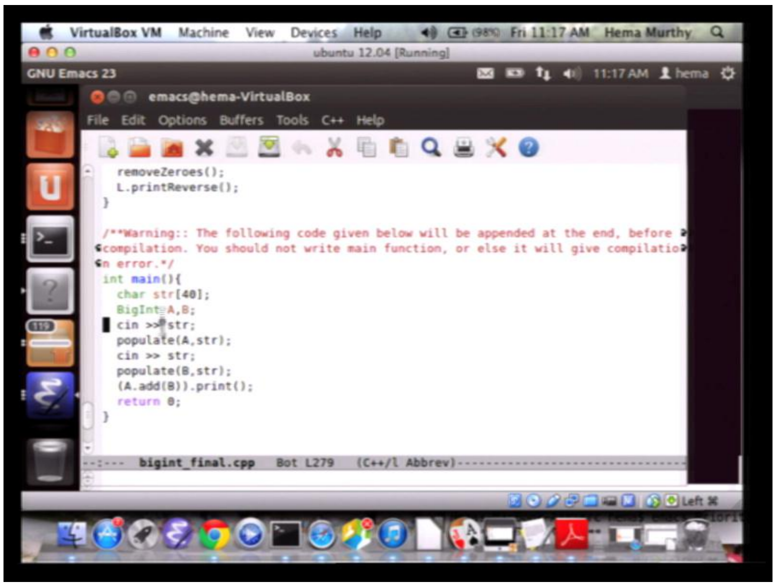


```
    }

  }
  if(carry > 0) C.L.append(carry);
  return C;
 }
void BigInt::removeZeroes(){
  //helper removes additional zeroes at Most Significant bit. Not mandatory.
  while(L.last()->getData() == 0 && L.last()!=L.getHead())
    L.popBack();
}
void BigInt::print(){
  /*prints a BigInt, and it must not have additional zeroes at most significant bit*/
  /*write your code here*/
  removeZeroes();
  L.printReverse();
}
```

```
--:---  bigint_final.cpp    89% L257    (C++/l Abbrev)----------------------------
```

Then if you want you can remove the leading zeros of they are there in, because you are writing it in reverse order..

(Refer Slide Time: 20:30)

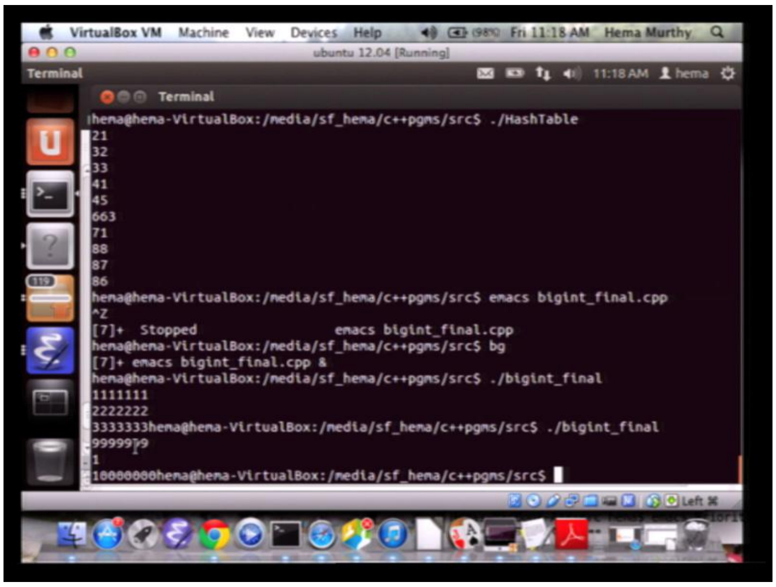


```
  removeZeroes();
  L.printReverse();
}

/**Warning:: The following code given below will be appended at the end, before compilation. You should not write main function, or else it will give compilation error.*/
int main(){
  char str[40];
  BigInt A,B;
  cin >> str;
  populate(A,str);
  cin >> str;
  populate(B,str);
  (A.add(B)).print();
  return 0;
}
```

```
--:---  bigint_final.cpp    Bot L279    (C++/l Abbrev)----------------------------
```

And basically what is done in this main program as was told you are not supposed to write this function. You are suppose to populate the string a corresponding big int a big int b and then perform the, add of the 2 of them and print them.

(Refer Slide Time: 20:48)



```
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./HashTable
21
32
33
41
45
663
71
88
87
86
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ emacs bigint_final.cpp
^Z
[7]+  Stopped                 emacs bigint_final.cpp
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ bg
[7]+ emacs bigint_final.cpp &
hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./bigint_final
1111111
2222222
3333333hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$ ./bigint_final
9999999
1
10000000hema@hema-VirtualBox:/media/sf_hema/c++pgms/src$
```

So, let us run this program what are what should expect now let me let say 1 1 1 1. So, here I gave 2 reasonably big integers 1 1 1 1 2 2 2 2 and it gives me () let us run it again. And that me gave 9 9 9 9 and give 1 let gives you notice that 9 9 9 9 is a 2 numbers this is for the one in the one given a () is big int and other one has only 1, but, 9 plus 1 10. So, what happens that is the carry and that is added to the next number and so, on this is the repeated it. So, this is the big solution to the big int problem. So, what we have looked at is the solution is to 2 of the problem that have been given as part of the data structures big 5 assignments where remember right.

.