Programming and Data Structures
Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 06
Implementation of List Data Type

Hello, welcome to the 6 lecture of this course and in today's lecture we are going to do something quite exciting, we are going to write a program which will apply, call the abstract data type features that we have designed and implemented. Some of the step that we have designed earlier we have not yet completed the implementation though we have completed discussing it. So, in today's lecture we will go through this implementation in detail and then go through the programming exercise that is going to use this abstract data type.

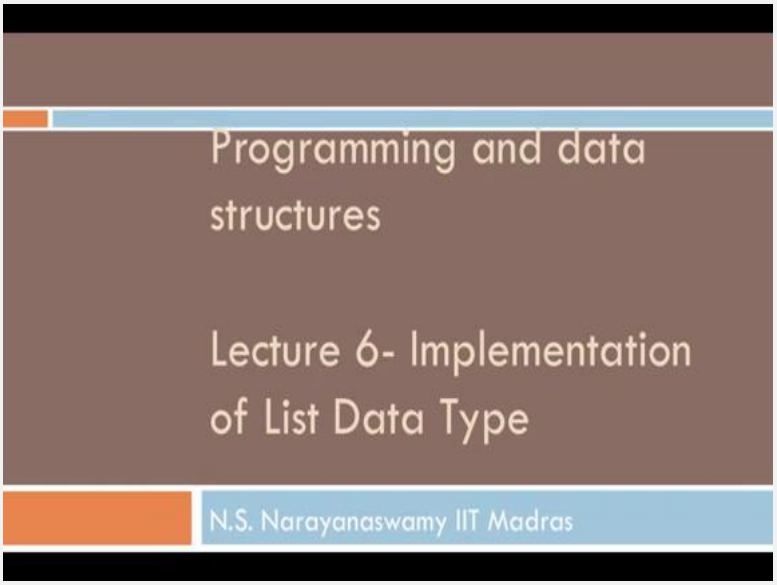
And I encourage you to focus on the programming practices also, what I mean by a programming practices, how do you decompose or break down a single programming exercise into smaller manageable components, most important thing is manageable components. Somebody else, who wants to develop your program and take it forward should be able to do so, because we appropriately documented and it must also be appropriately split into smaller manageable modules.

If those are the things that we will really not talk about, but by looking at my presentation, you should be able to get an idea of how to achieve these things that is the focus of this lecture. I also have some comments about the queries that people always raise on the forums. So, with respect to the programming assignments. So, to the best of our knowledge we can safely say that there are no major errors with the compilers which you get access to when you submit your assignments.

So, these are well tested compilers. So, you do not have to worry about whether the compiler is making mistake. So, if your submission is not being successful than, you might want to look at your implementation and understand how you are probably reading from the input and whether you using the write algorithm in right problem solving technique. But, please do continue to post on the forum and when we will try to answer them to the best of our utilities. Now, let us go to the presentation, the presentation is going to be short, but a large amount of this lecture is going to, go through a C program file and it is likely that a lecture is going to be very detailed on a trick, let us go to the

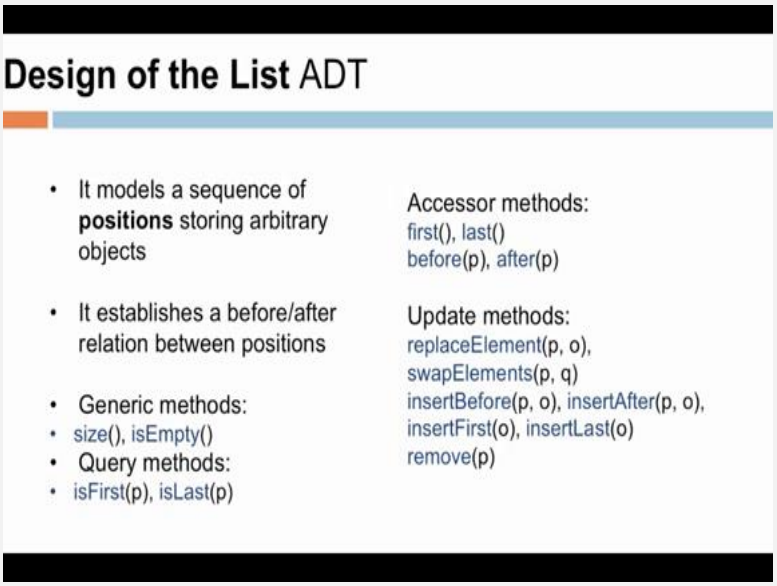
presentation.

(Refer Slide Time: 03:20)



So, let us complete the implementation of the list data type.

(Refer Slide Time: 03:22)

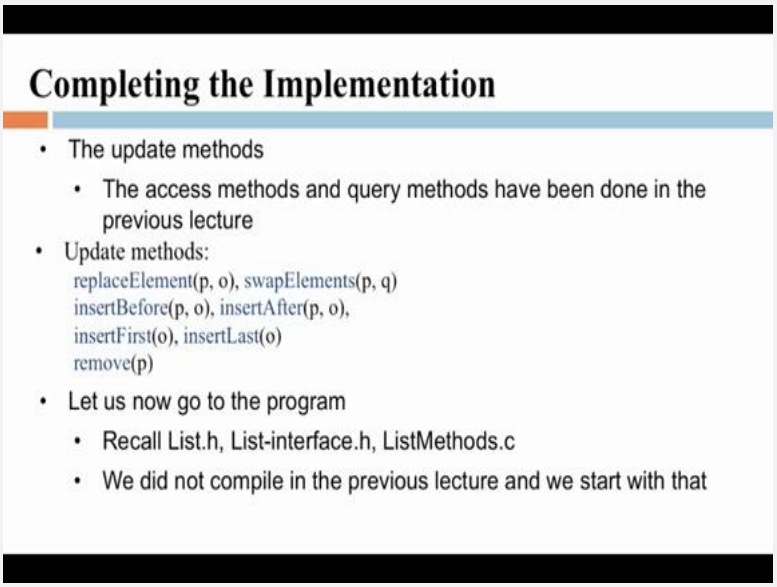


So, in the last class we essentially implemented the generic methods, the query methods and the accessor methods and he wrote the methods where we want to replace a certain element in the list wwere another value or exchange the contents of two positions in the list, insert an element before a certain position in the list, insert an element after the position in the list, insert an element to the first to the big. So, that becomes the first position in the list and insert an element into the last position in the list and remove a

certain position itself.

These are the update methods which we did not implement, when we met a last during the last lecture and today we will complete this.

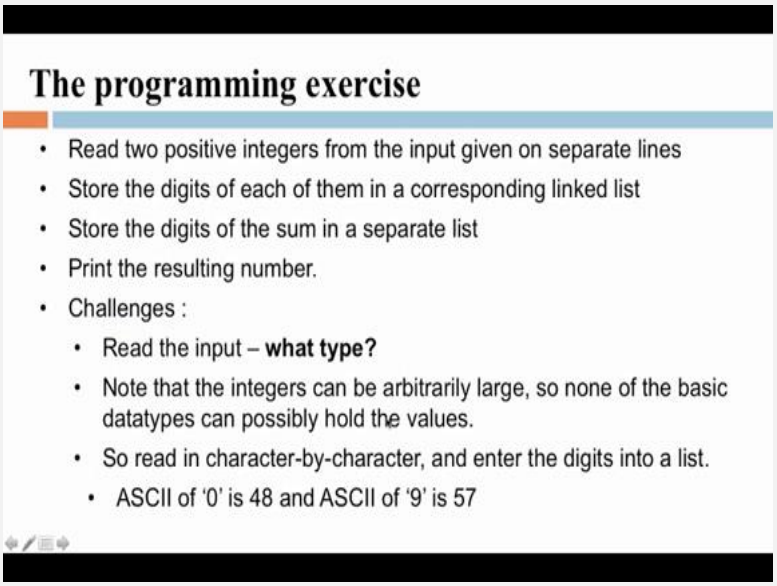
(Refer Slide Time: 04:10)



So, now let us now go to the program and before we go to the program, I recall what are all the three main files that we edited and created to implement this abstract data type. So, when you create and implement an abstract data type, you are essentially creating a collection of files and somehow a programmer who wants to use this abstract data type should be able to include these files. So, this is the most important thing.

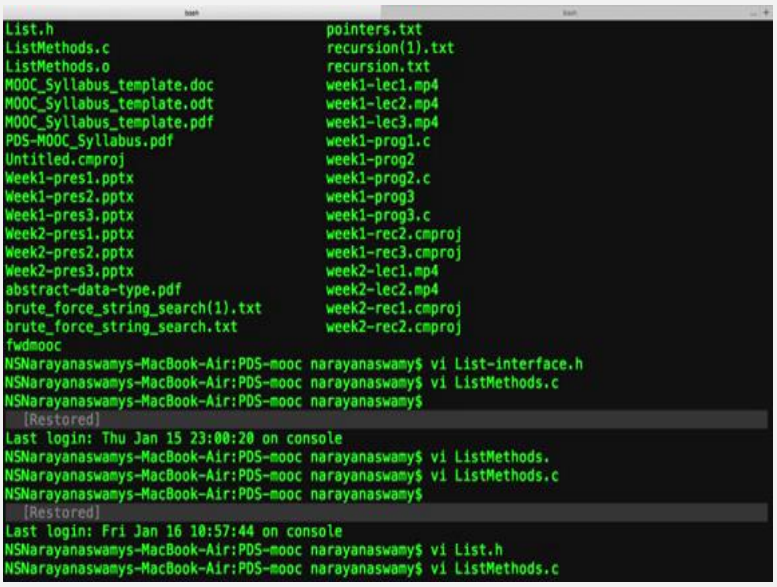
Now, the other thing that we did not do in the last lecture was we did not compile the ListMethods dot c file that are promised to compile that was an over side and we will definitely start with that today.

(Refer Slide Time: 05:02)



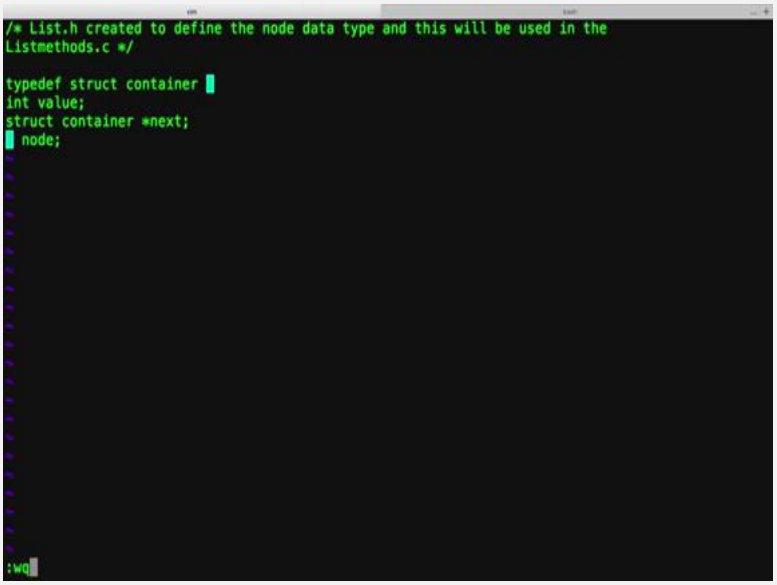
And after that we will look at a programming exercise when we come back to the exercise, come back to the lecture again.

(Refer Slide Time: 05:17)



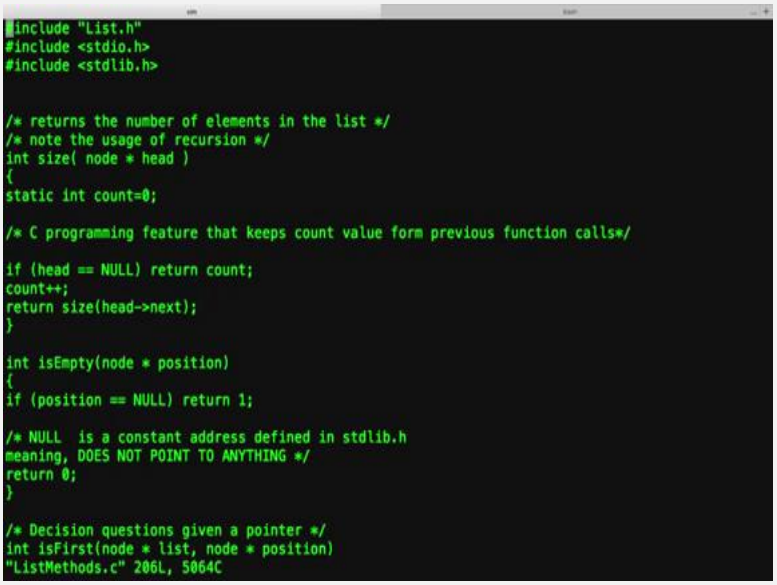
So, now let us go to the... So, recall that this file was the file that contains all the methods associated with the abstract data type list.

(Refer Slide Time: 05:39)



And the type definition, the most important type definition is inside this file called list dot h. So, as you can see I am intending to separate the definition of the type of the creation of the new type from the methods. It is not a bad programming practice either to put all the methods inside this file,, but if your data type itself gets very large, it is a good idea to separate these two ((Refer Time: 06:03)). So, now let us go to list methods dot c, this contains all the methods which are important for a programmer to use the abstract data type list.

(Refer Slide Time: 06:13)



So, we have included list dot h and we have included the stdio dot h like I said in the last lecture, it is most slightly useless for us and stdlib dot h would be important, because we

will be using constant associated with pointers which are defined inside this library. For example, null is one such constant that we will repeatedly using. So, we will already seen the functions size, size takes in it is inattentive taken a pointer to the first element of the list and return the an integer value which will represent the size of the list.

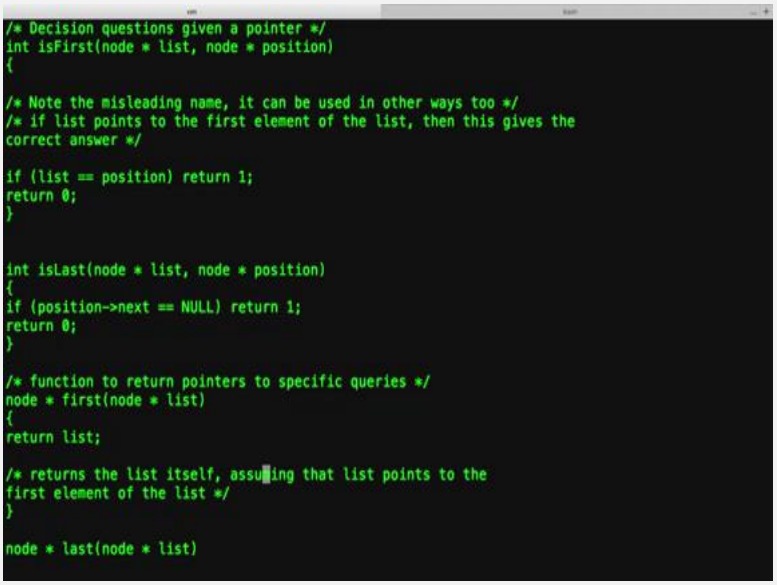
Remember, with this was a recursion function, then recall that isEmpty is a Boolean function, it is the function that returns a Boolean value, in this case it is model, thus an integer return value and I says whether if the list is empty or not. And observe that in the implementation we ensure that the return value is either 1 or 0.

(Refer Slide Time: 07:26)

```
/* C programming feature that keeps count value form previous function calls*/

if (head == NULL) return count;
count++;
return size(head->next);
}

int isEmpty(node * position)
{
if (position == NULL) return 1;

/* NULL  is a constant address defined in stdlib.h
meaning, DOES NOT POINT TO ANYTHING */
return 0;
}

/* Decision questions given a pointer */
int isFirst(node * list, node * position)
{

/* Note the misleading name, it can be used in other ways too */
/* if list points to the first element of the list, then this gives the
correct answer */

if (list == position) return 1;
return 0;
}
```

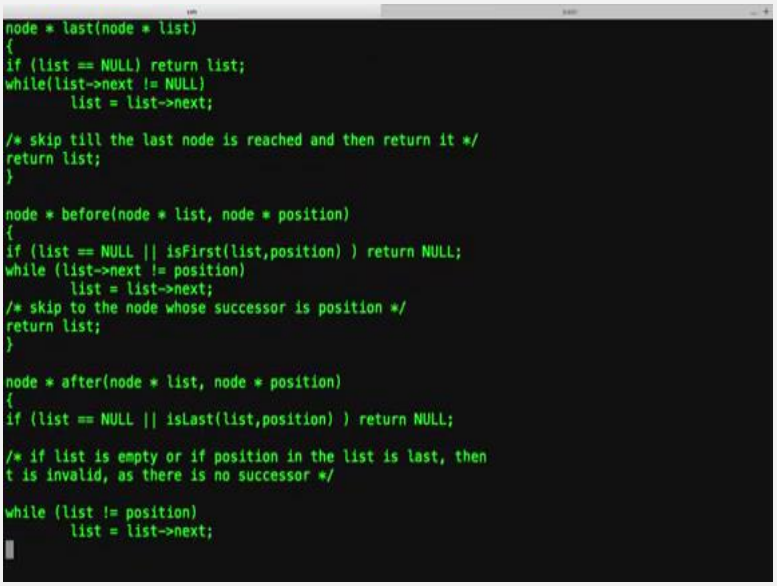
So, it is up to you to just carefully return appropriate values for the data type as an integer. So, let we ask you a question would you be happy with such an implementation, because the Boolean value is a single bit,, but an integer uses close to 4 bytes, you uses 4 bytes or 2 bytes. So, it is just a thought, I am not giving you a solution,, but definitely a kind of question that you should be asked and isFirst is another Boolean function which recognizes if a position is a first element of the list or a second element or not.

(Refer Slide Time: 08:10)



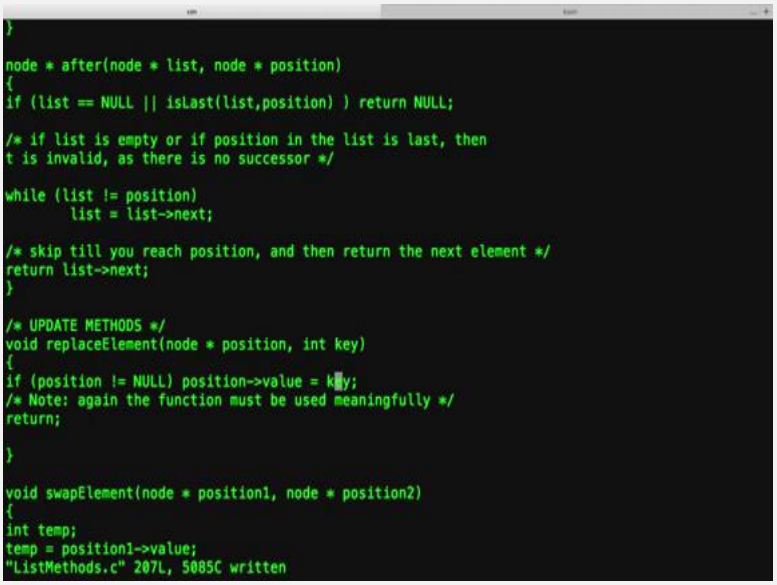
Similarly, isLast returns if a position is a last element of a list or not ((Refer Time: 08:18)). So, those we are all queries to which the responses were of the Boolean data type.

(Refer Slide Time: 08:28)



So, now let us look at some queries which return objects of the node data type itself, the first function take some a list and returns a pointer to the first element to the list. In the last function returns a pointer to the last element we will list. Before returns a pointer to a given to the node position, just before or given position in a given list, after returns a pointer to a position just after a given position in a given list and now we come to the update methods.

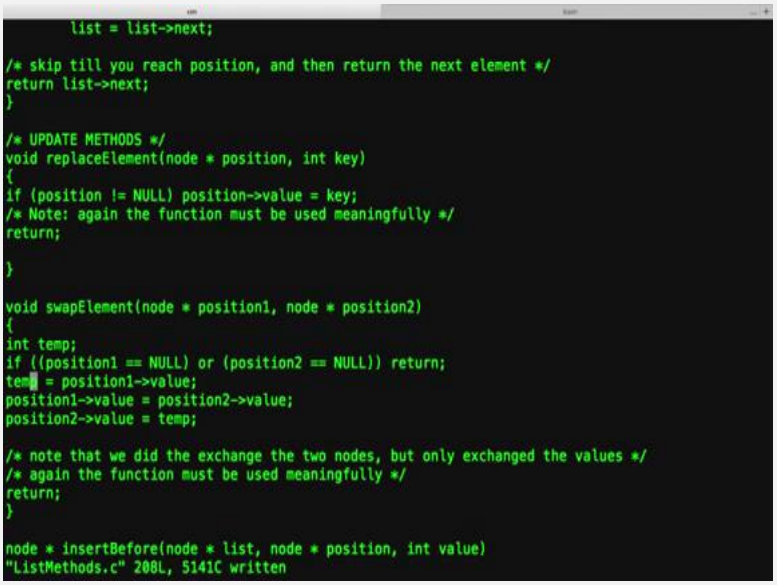
(Refer Slide Time: 09:00)



So, these worked methods that we did not really talk about in the last lecture. So, let us look at this update methods. So, replace element is an update method, it does not return any value and therefore, one has to be careful in making this function call. So, it takes in two arguments, the first argument is a position in a list and it takes in a key and if the position is not null it updates a value of the position with the key; otherwise, it just returns.

So, I repeat if position is not null it updates the value with the given value key and then it returns, if position was indeed null then it returns without doing anything. So, such things a program or a function that calls it is method should be very careful to check that position as a meaningful value.

(Refer Slide Time: 10:29)

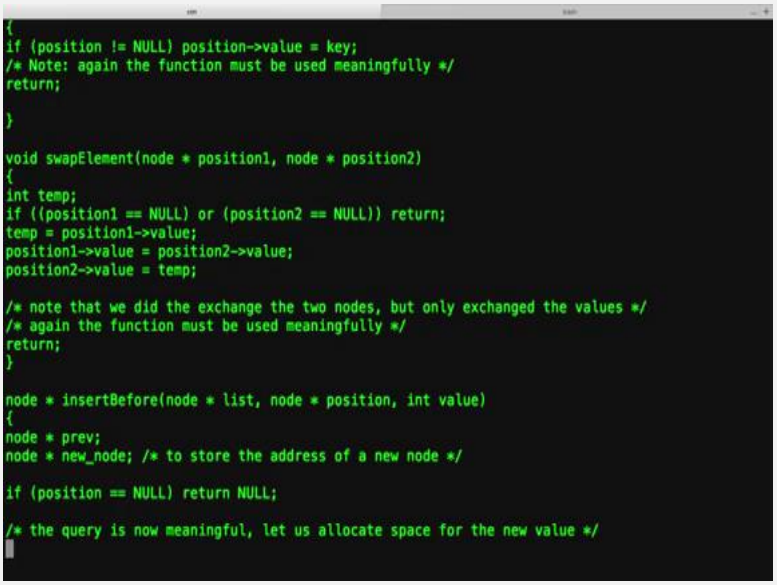


Swap element is a well studied function that most C programming students or programming students, here the two arguments is what elements are the address of two positions, position 1 and position 2 and the result of this function execution is that the values in these two positions are exchanged and again we return a, we do not return any value from this particular function.

Again this function, any such function which does not return a value must be used very meaningfully and it must be ensure that the calling function does all sanity checks of the parameters, that is you must ensure that the position is not null and position 2 is not null and so, on and so, forth. These are things that we are not checking for inside this function. Of course, one can definitely check here and I will do this for you, if position 1 is equal to null or which response says that if either of the two is null, then you return.

But, again one has be careful, there is no message that comes from this function that this function call was meaningless. And therefore, the execution of the calling function make at important, if such meaningless function calls a meaning. So, it is up to the programmer to ensure that the function calls that calls swap element and meaningful arguments.

(Refer Slide Time: 12:24)



This is going to be true for every method that you see in this implementation and in general this will be true for any function call that you make to a library of functions. I repeat this, it is important as a programmer for you to check that the arguments that you give to a function are always checked for their meaningfulness before the function called is made.

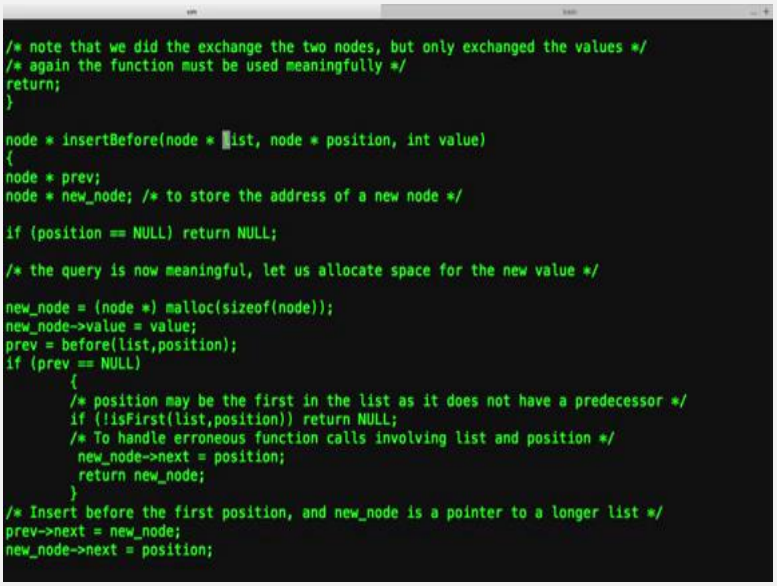
Of course, there are well return functions which may give you reasonable indication that an erroneous behavior has happen while execution and the execution can update itself appropriately, not the program, the execution itself can update based on the result of a certain function call. But, it is good programming practices to ensure with the arguments are sanities, before function calls the main. Having pointed out, let us move on let us look at the other functions by the way observe that the implementation of the swap element function is fairly straight forward, we have an integer variable called temp.

Temp is a temporary location that stores one of the two values, in this case it shows the value of position 1, then it effects the exchange. Most importantly, observe that we did not exchange the two positions,, but we exchange only their values. Insert before is very interesting and among all the functions, this is the one that can be improved significantly. So, what is the result of insert before? The result of the insert before is a pointer, it is a pointer to a position which occurs just before, which is inserted just before the given position in the list, in the value the return position is the value that has been given.

So, let we repeat this again the result of itself before is an insertion. What is inserted? A

node is inserted. Where it is inserted? It is inserted just before the given position in the given list. And in this position, the value that you see is the value which has been given as an argument to this function. This is the behavior of the insert before function.

(Refer Slide Time: 15:03)



```
/* note that we did the exchange the two nodes, but only exchanged the values */
/* again the function must be used meaningfully */
return;
}

node * insertBefore(node * list, node * position, int value)
{
node * prev;
node * new_node; /* to store the address of a new node */

if (position == NULL) return NULL;

/* the query is now meaningful, let us allocate space for the new value */

new_node = (node *) malloc(sizeof(node));
new_node->value = value;
prev = before(list,position);
if (prev == NULL)
        {
       /* position may be the first in the list as it does not have a predecessor */
       if (!isFirst(list,position)) return NULL;
       /* To handle erroneous function calls involving list and position */
        new_node->next = position;
        return new_node;
       }
/* Insert before the first position, and new_node is a pointer to a longer list */
prev->next = new_node;
new_node->next = position;
```

So, let as look at the logic of what we would do. So, let us assume that list points to the first element of the list. Now, what we do is, we find the predecessor of position in this list and insert the new value into a new position between the predecessor and given position. I repeat this in the less assuming that the value list points to the first element of this list, we first find the predecessor of the given position, that is in the list which is the node which occurs just before position, then we create a new node insert that new node between the predecessor and the given position and assign the value in to that position as the given value this is the approach.

So, let us as quickly go through this piece of code, first we check the position is not null given to the position is null, then users return null saying that while the query is not meaningful. Because, a query is meaningful which means position is meaningful, we create a new node, we allocates space for a new node and we call it new node itself. Then, inside new node in the value feel the put in the value that has been passed as an argument.

Then, recall our before function what is before do, before text two arguments it takes the list, it takes a position and then it returns a pointer to the predecessor of position in the list, that is stored in this variable called prev, prev is a short form for previous. And all

we check a previous is null, what is previous being null means that position is the first element of the list, that is position is the first element of the list as a does not have a predecessor.

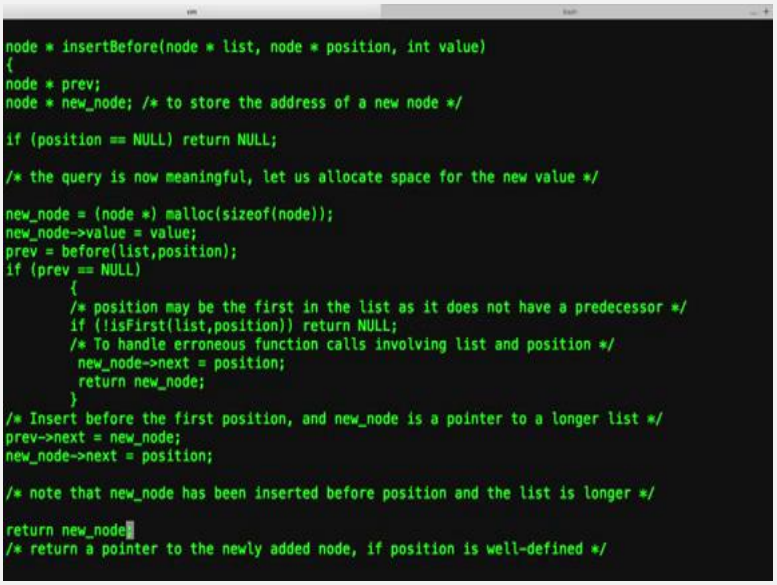
So, if previous is null then recall from the before function that the return values on null only if position as a first element of the list. Because, the first element of the list does not have a predecessor, now let us check this let us do one more meaningful check. So, to handle erroneous function calls is really not necessary, these is going to make function calls very carefully, the first check if position is the first element of the list, why are we doing this here, we found that position does not have a predecessor and which means it must see the first element of the list.

So, now we check if indeed position is a first element of the list, what are we checking, we checking if the function call is a meaningful function call or a meaningless function call. If you terms are that position is not the first element of the list, when we say loop something is wrong with this function call and we return a null, that we will not insert attend to do is insert which is return a null.

So, this is a essentially to handle one type of an erroneous function call, we do not how many erroneous function calls people can make, we cannot check for all of them. But, definitely this seems to be a reasonable kind of error to make and we are catching that. Now, if indeed position is the first element of the list, then we introduce the new node, as the first element to the list, let us see what we do look at this statement, you say new node pointer dot next is position and we return new node.

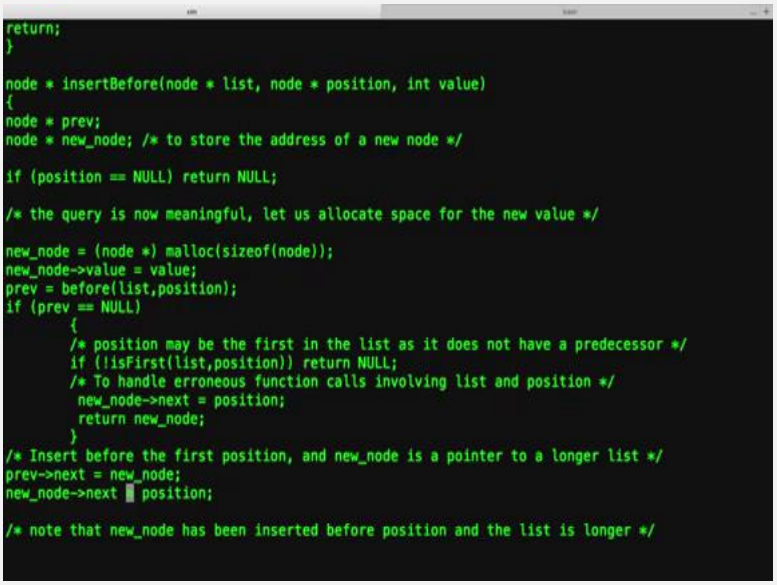
So, let us see what is happening, the insert new node into the list and here we insert new node as a first element of the list and then we connected to the rest of the list by making with point to position and then we a return a pointer to new node.

(Refer Slide Time: 19:34)



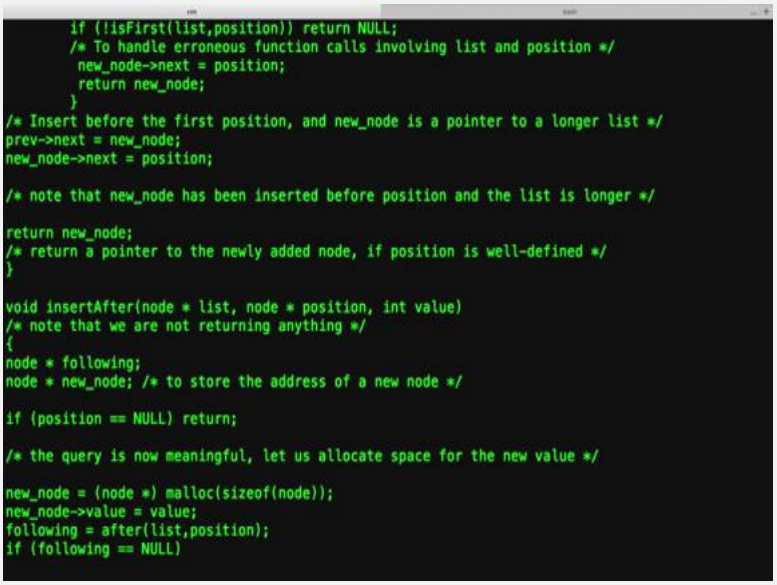
On the other hand, if previous is not equal to null that is what we do here is the previous is not equal the null; that means, that the position has a predecessor. So, now, what we do the predecessor now recall is call previous prev, we make previous point to new node that is previous pointer dot next points to new node and new node pointer dot next points to position. In essences, we are inserting new node into the position between previous at the given position therefore, the new node is now inserted into the list and the list is now longer and we return a pointer to new node.

(Refer Slide Time: 20:24)



This completes, the procedure or the method which is insert before and in this observe the, we have try to handle errors that are common errors that we pull might make.

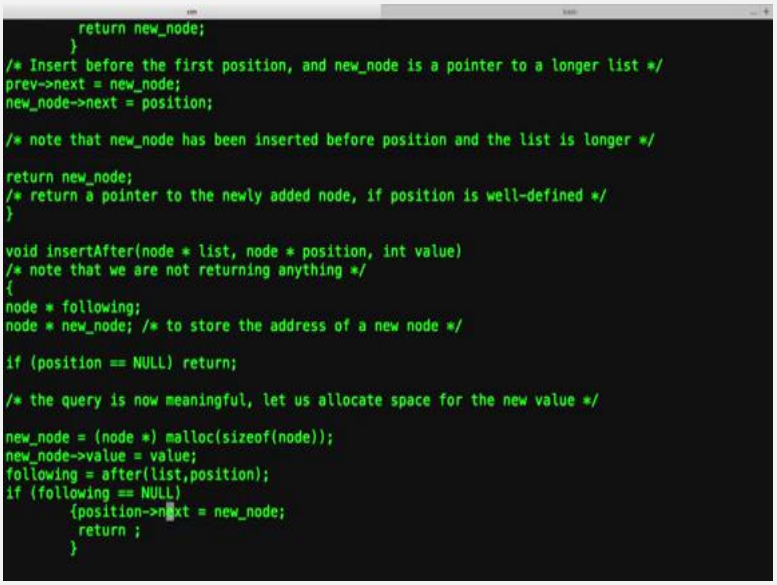
(Refer Slide Time: 20:51)



Similarly, let us go to the next function which is insert after, insert after takes is arguments a list, position in the list and then a value. Now, like previous we say following, the logic is very simple what we will do is, we will fine the successor of position in the list, create a node and inserted between position and it successor and the created node will contain the value that is being passed that is a hole idea.

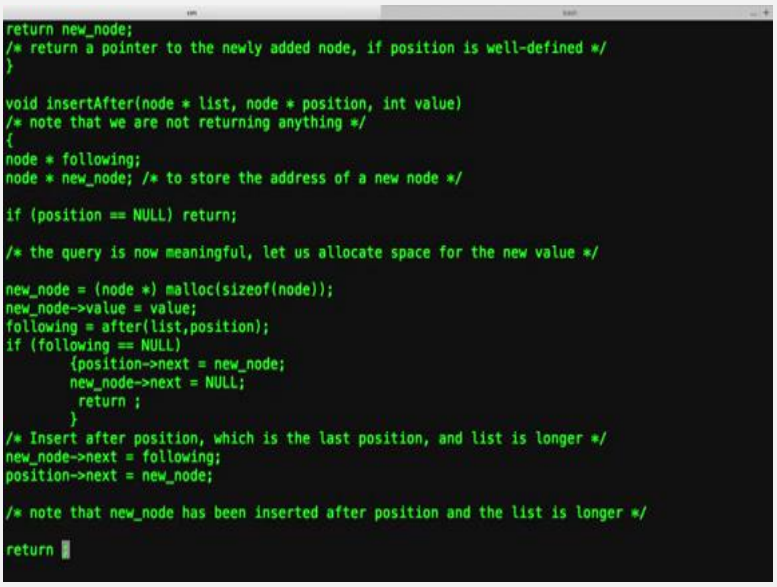
So, following stands for the successor node of position, new node is a point will be a pointer to a new node that we will create. Now, is position is null you just return, as a lecture is going on you may here students in other in the carita. So, I hope that does in get recorder,, but it is... So, now, here we create a new node we assign the value appropriately, now here we compute the successor of position in list.

(Refer Slide Time: 22:09)



Now, you following is null let us understand what this needs, if position does not have a successor then it means that, it is the last element of list.

(Refer Slide Time: 22:22)



Of course, observe here that we are not done any sanity check I should whether position is actually inside the list of not. So, that we should have done,, but we are assuming that you can argument this code, in such a way that it is clear. For a example, it is important to check that this is the meaningful function call we are not doing it here. So, if following is null then it means that position is the last element of the list. So, we need position pointer or next to be new node and then return. So, there is a small error here I should have already set, new node pointer at next is a good programming practice.

So, this error was introduce deliberately. So, that you see this observe what how happen, this was missing. So, what we I do I make position pointer dot next point to new node and I have not done something very important I should have said do not pointer or next is noun value. If I did not do this in the behavior in the program may not be predicable. So, therefore, we add this and then return. So, and you return observe that here we are not returning any pointer, we have just inserting this is inserting after the particular node in to the list.

And you can think about why insert before I return a pointer,, but insert after I am designing it slightly differently I am not returning the value, you should think about it. I can also tell you,, but maybe you should think about it and then if you do not get it, then ask a forum and we will respect. If following is not equal to null, then it means that the successor of position is not the last element of the list.

So, now, we have to insert the new node between following and it is successor, that is what we do. So, we say new node pointer dot next is the successor which is called following and position pointed dot next is new node in essences, new node has been inserted between position and it successor.

(Refer Slide Time: 25:05)

```
following = after(list,position);
if (following == NULL)
        {position->next = new_node;
        new_node->next = NULL;
         return ;
        }
/* Insert after position, which is the last position, and list is longer */
new_node->next = following;
position->next = new_node;

/* note that new_node has been inserted after position and the list is longer */

return ;
/* return a pointer to the newly added node, if position is well-defined */
}

node * insertFirst(node * first, int value)
{
return insertBefore(first, first, value);
/* note the risk that a user must use this function call carefully. If first is not the
first element of a list, the programmer better know what is being done */
}

void insertLast( node * list, int value )
{
insertAfter(list, last(list), value);
/* again, the calling functions have to be careful to check that first is not NULL, else
this functions returns no information about the failure to insert value */
return;
```

This completes the insertion of a particular node into the list, it can be insert before a position or insert after the position, there are two simple functions that are very important. But, now can be immediately implemented this is insert first into a list. How given value? How do we do this, we just say insert before in the list first before first

insert a value. So, let me save this again, insert first is very simple you want to insert a node with a given value and make it the first element of that list, this is the goal of insert first.

The way we achieve this is my making of function called to insert before. What would be the arguments? Remember, the insert before arguments of a first one is a list, the second one is a position and the third one is a value. So, I say insert before in the list call first, before the first element, insert value. Remember, that the return value is a pointer and you return that pointer.

Again this function call must be use very carefully, because the calling function must ensure that first is the first element of the list and if it is not the programmer should have a clear understanding of what here see is doing, insert last does a same thing,, but to the last. So, what is it do it takes a list and inserts a new node with a value which is pass your and makes it the last element of the list. How is this done? First your perform insert after into the list after the last element of the list, remember that last is a function that we have already return, after the last element of the list and populated with the given value and then you return from here.

(Refer Slide Time: 27:29)

```
first element of a list, the programmer better know what is being done */
}

void insertLast( node * list, int value )
{
insertAfter(list, last(list), value);
/* again, the calling functions have to be careful to check that first is not NULL, else
this functions returns no information about the failure to insert value */
return;
}

node * eliminate(node * list, node * position)
{

node *pred, *succ;
pred = before(list,position);
succ = after(list, position);
if (pred == NULL)
/* position does not have a predecessor */
        { free(position);
          return(succ);
        }
if (succ == NULL)
/* position does not have a successor */
        {pred->next = NULL
         free(position);
         return pred;
        }
/* position has both predecessor and successor */
```

So, finally, we come to very important function which is to eliminate from the given list or position. So, how we are going to eliminate. So, let us just visualize this, if you want to eliminate a position from the list, you have to ensure that the list structure is a respect it. So, which means if you remove of certain position, you have to ensure that the

predecessor and successor of this position or now predecessor and successor of each other.

So, this must be ensure. So, it to do this we have to temperate to local variables for predecessor and successor is a pointer, these a pointer should node. Predecessor is a pointer to the node which is before position in the list, successor is a pointer to the node which is after position in the list, it is quite possible that predecessor and successor or both null or at least one of them as a null.

So, let us see what we should if predecessor is a null then it means that, your short in the list by eliminating the first element of the list. Let us see what we do? If predecessor is null it. So, if predecessor is null then it means that position does not have a predecessor, which means you removing position, the removing the first element of the list. So, you free position. So, that is a function call that you must definitely use as a discipline that every programmer should forward.

So, when you are going to remove a certain dynamically allocated memory location that is anything that your allocated using a malloc, the good programming practice to free it. Rather, it is a responsible programming practice to free it. So, that other programs or other functions can successfully run and not failed for the because of the lack of memory, then you return a pointer to the successor. Now, successor is null then it means that the position is a last element of the list.

So, what we do is we make the predecessor pointer dot next is equal to null and you have to observe that there is a very careful that has careful programming that as happened here, I just come to that a minute then you free position and then you return predecessor. So, let us just see. So, for example, first we compute predecessor and successor, let us look at the case where position the list just has only one element, which is position which means go predecessor and successor at null.

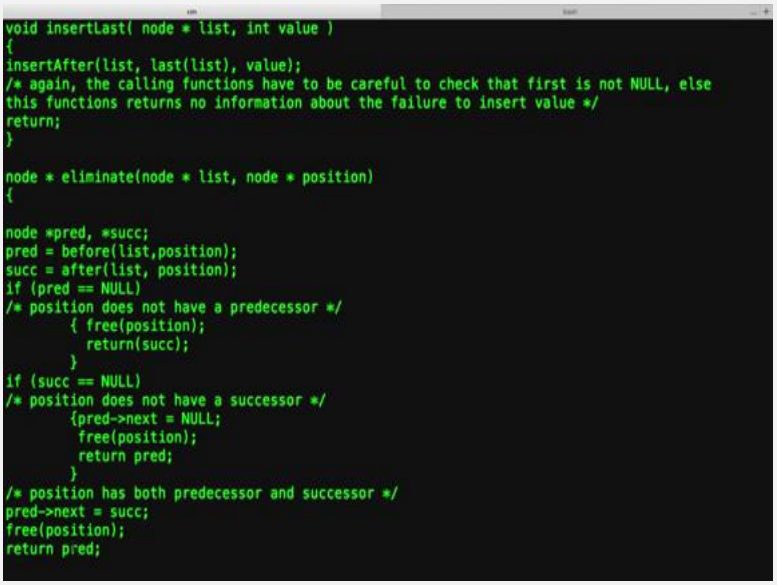
So, if the predecessor is null you free position and you return successor which is null and observe that this is very carefully done by not put in the successor check first. Let us imagine what would of happen with a execution if this piece of code was done before this piece of code let us see what happens. And let us consider the case when position was a single node. So, if position is a single node than it means predecessor and successor or null.

So, if we were doing this comparison first and executing this piece of code, then

successor is null and observe the first statement, you say we assign to predecessor pointer at next the value null,, but predecessor itself is null. So, this statement will fail if this piece of code was place before the check which is involve the predecessor is equal to null, this is what a mend by saying that some careful programming has happened here.

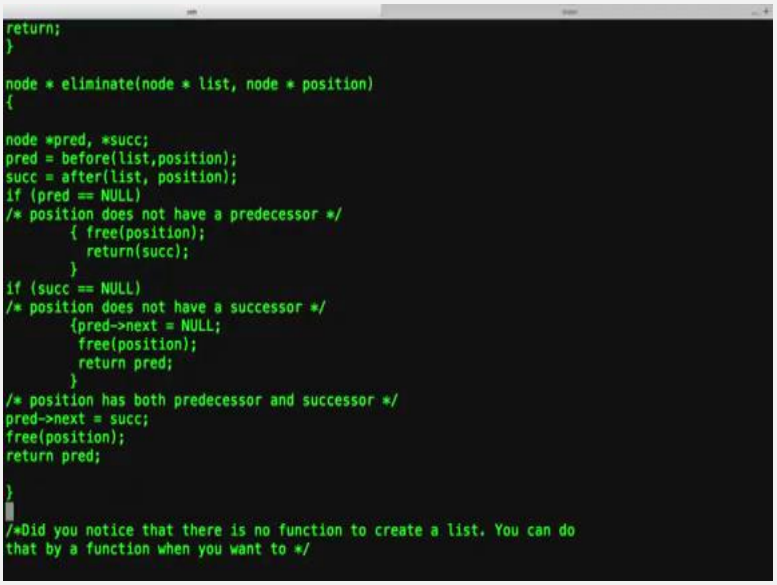
So, you must be aware of such details are you must be checking for such details when you right pieces of code, you must imagine what happens during the computation. Anyway, after ensuring the predecessor point, predecessor point the next is null, you remove position, you free position and then return the pointer to the last element of the list.

(Refer Slide Time: 32:29)



And if control comes this location at means that position has both predecessor and successor. So, what you need to do is make predecessor pointer dot next is successor, short term the list by removing position,, but do not break the list structure, free position and return predecessor.

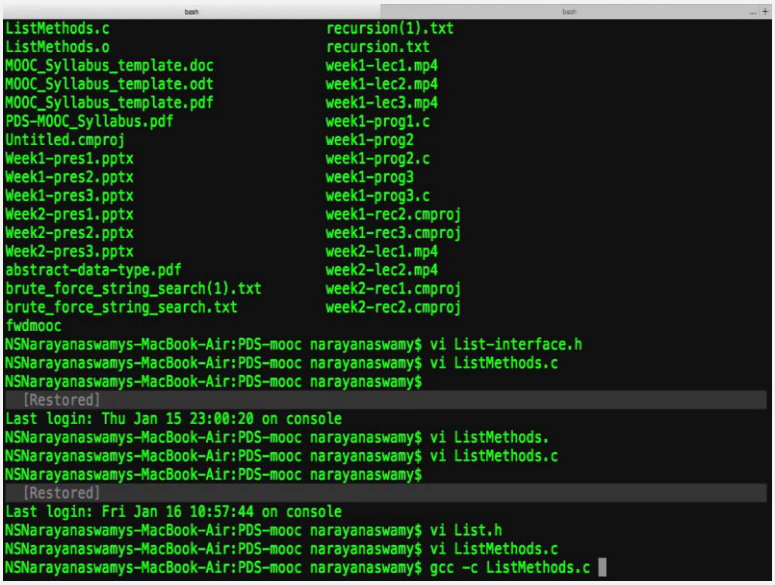
(Refer Slide Time: 32:46)



So, this completes the set of methods that we have implemented with the abstract data type. Observe that, may not given a function to create a list this is for a person who wants to use this data type, he or she creates a list using these methods, that is what we are going to do, in the programming exercise. I hope you appreciate what we have done. So, for. So, what we have been doing is that, we have look at the abstract data type list and we have implemented of hole set of methods which are associated with this particular abstract data type and very importantly we created the data type.

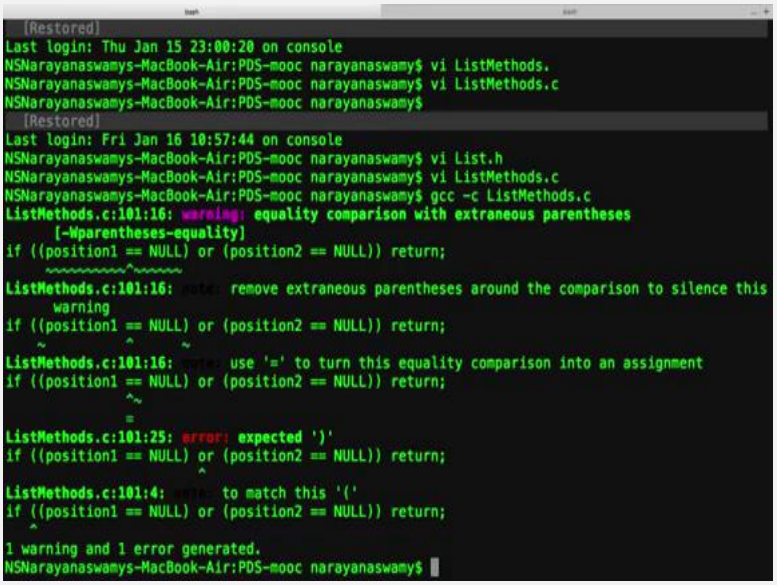
So, from now on if you want to visualize the data type, one is the abstract definition and if you visualize the implementation you have to look at three files, one file is list dot h with defines the type definition which is important. The second file is a list methods dot c which has all the method definitions that are relevant, the third one is a list interface dot h which is what programmers want to use this data type will include and use this particular data type. So, now again I forgotten to show you the compilation let us go back in look at the compilation.

(Refer Slide Time: 34:29)



So, now what I am going do is I am going to compile is this methods dot c, these in a I am showing you this compilation is that, recall that in list methods dot c there is no main function. So, many of few my think for at least I have seen some people who think that you can compile C programs only if they have a main function. So, my answer to you use that known that is not correct, let us see gcc minus c less methods dot c compiles list method dot c and generates a object file.

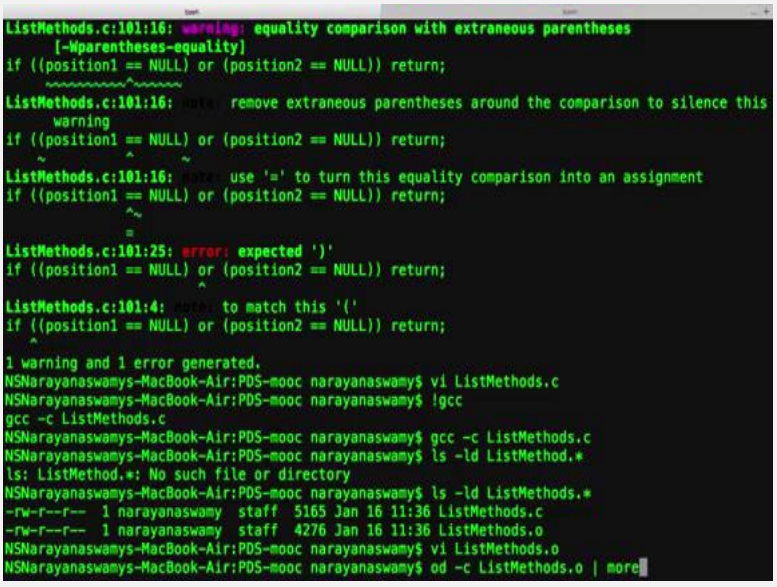
(Refer Slide Time: 35:01)



There are some errors here So, now I fixed was errors let me just show you the error that I made. So, it is also interesting, observe that what I did was I use the English word or. So, when you program it is important to understand the correct operations that you

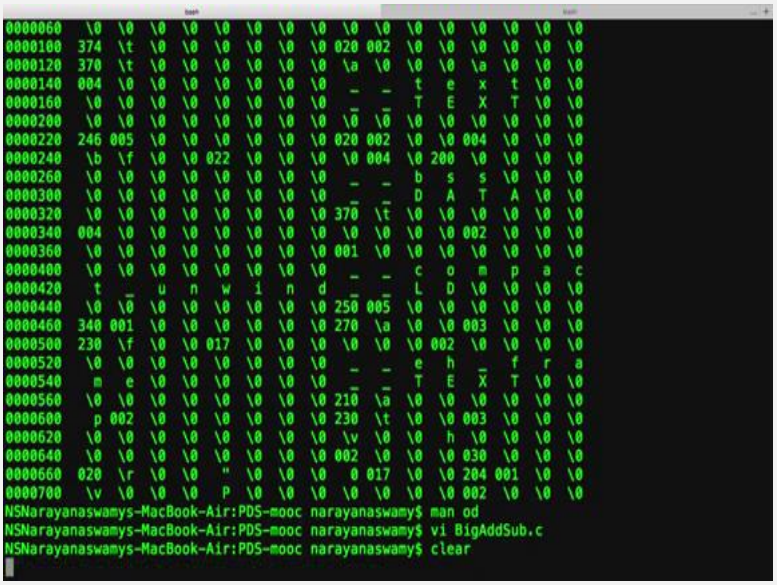
performing and further natural language is very useful. But, when you converted into the C programming language or any other programming language, you have to use the correct words and here I at the mistake that I need us at add use the English word or instead of the operator or.

(Refer Slide Time: 35:24)



Now, let be compile this and as you can see the compilation succeeded there was no main file and let us just look at the file that has been created. So, observe that list methods dot c is the c file that we created and list methods dot o is executed there are been created and which mean created at these particular time, which is Friday, if you hours was after I should uploaded with the file that is I uploaded in a couple of hours you know at this time. So, this is very recently created and this object file is created. So, let us as just take a look at this object file right.

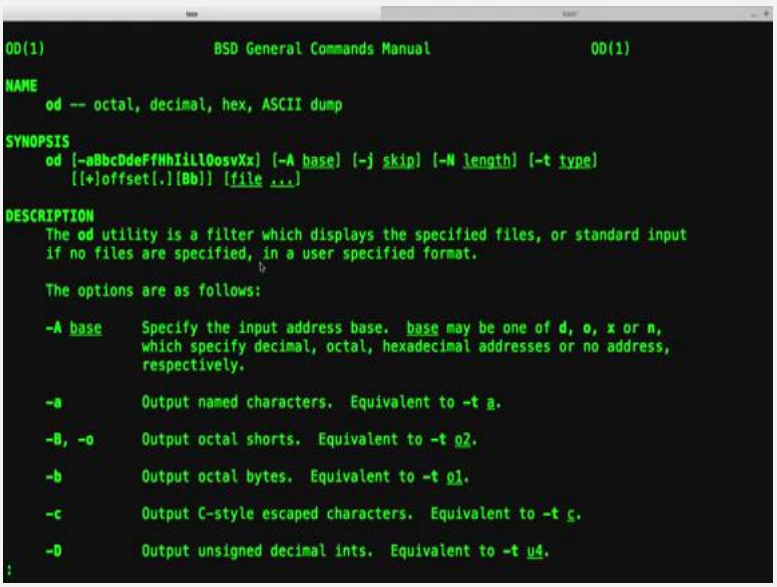
(Refer Slide Time: 37:05)



So, as you can see the seem to be a lot of interesting thinks that are worth while reading. For example, the reason I show this to you is these are all some indices, in this seems to be some information about some tables. Now, you can see my function name, size is visible here, the function name is empty is visible here, this first is visible here and so, on and so, forth. I did not show you the object file to explain any theory to you.

But, to show you that there is something spurious going on objects file and you can actually view it. Now, let me show you the commend on the machine, it is a octal dump o d transfer octal dump and I want everything to be shown as characters and I am doing an octal dump of list methods dot o any.

(Refer Slide Time: 38:33)



And if you want to know what octal dump is, you can go to a lanes machine and you can do man o d and I tells you, it is an octal, decimal, hex, ASCII dump tells you that the o d utilities of filter with displays this specified files or standard input if no files is specified in a users specified format. So, you can actually look at files that you typically cannot look.

So, what are where are we know. So, we are created the list methods dot c, if compile compiled it and made an object. So, we have completed creating the abstract data type we even compile with. But, will should not believe me, because I have not shown you, how to use it then are not shown you the proof that it works, do it did compile and must show you the networks. To show you the networks I am going to take you to my slights first and then show you the networks.

Going to show you this programming exercises ((Refer Time: 39:36)), the programming exercises extremely simple anybody can understand this programming exercises, what do you read two positive integers from input that are given on separate lines, going to sure the digits one after we other from the most significant digit in the left to the least significant digit on the right in a corresponding link list. And I want to do compute the digits of the some into a separate list and print the resulting number, you may think that this is a trivial programming exercise.

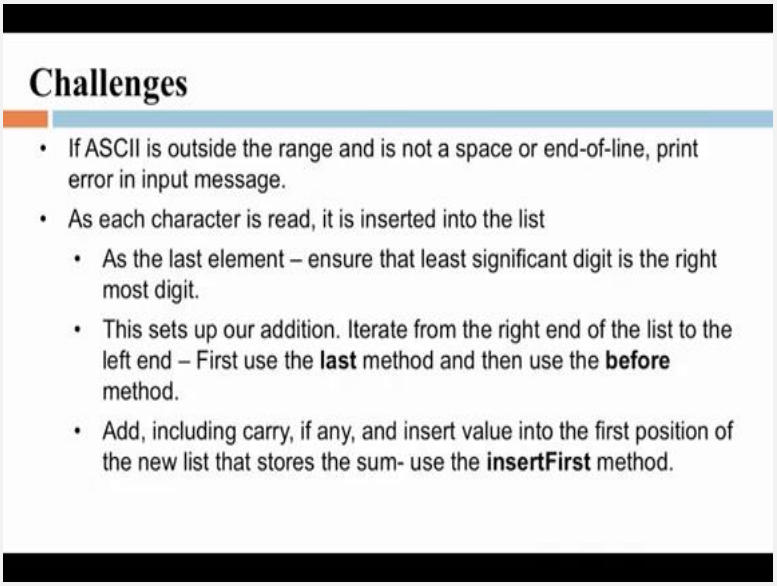
But, really is not trivial, because I did not tell you how large the integers are going to be, where going to see that we will be able to add integers of arbitrarily size. Therefore, when you read the input you cannot assume that your reading one integer. So, you have to choose a careful date type to read, because the none of the basic data types then possibly whole the values that I will give us input.

I promise to keep inputs which have 200 digits or 100 digits or 50 digits and if I give 50 digits observe that, your essentially train to represent numbers which are 10 power 50 and you can represent 10 power 50 in any of your data types. So, let us see 10 power 50 is 2 the power of 150 approximately, it is more than 2 the power of 150. So, you are not going to able to represented even int the long integer data type which occupies 8 bytes treats of 64 bits.

So, what is my idea, the idea is to solve this particular problem to be able to deal with arbitrarily large integers and perform arithmetic. I am going to read and the input character by character and enter the digits into a list. Now, of course, I am as deal with a

situation where meaningless characters are inserted. So, let us now observe the ACSII code for the character corresponding to 0 is 48 and the ASCII code for the character corresponding to 9 is 57, whose the very nice think and we will use this in our program.

(Refer Slide Time: 42:01)



So, what are the challenges in this exercise. So, the challenges are if the ASCII is specified out of range, then or if it is not as space or end of line, we have to printer error message, in this you will see the we will just return the minus 1. Because, we will go with the programming practice if our been follow that the output messages will only consist of this specified value and not have any message to the end user.

Now, you will recall that the reason I am doing is that, this way of thinking helps you participant in programming contest. For example, where you understand the input format and print out the output in the certain input format and not always have the messages in English saying, please enter the input and this is output and so, on, that kind of messages we are going to completely avoid. Therefore, we will not print any error in the output, if there is an error in the input we will not print an error message,, but we will return a value of minus 1.

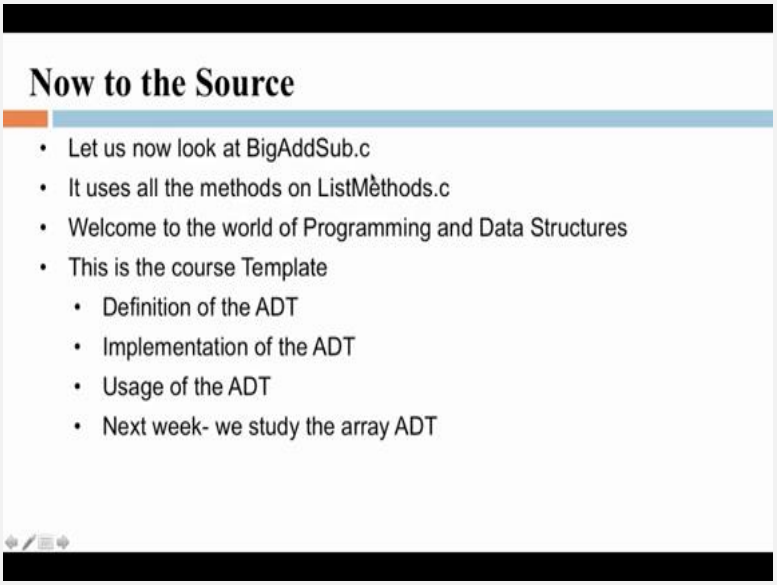
Has each character visitant, we will insert is into the list and we will use the method set we have described in the abstract data type. For example, we will use the last method we will use the before method and we will use the insert first method, repeatedly in a very careful first. So, for a example as you read inputs from the list you have to add it to the... Remember, that the key the value is type is given by you as an input and the processing

of the input happens from the left to right fashion, that is you see the more significant digits first, then the next more significant digit all the way down to re significant digit.

Therefore, we will have to insert the numbers in the correct order to set to the addition, we will start identifying the last element of the number and then use the before method and keep adding the values and keep in crack of the carries. So, what did I mention here I just mention that we are going to use the abstract data type, we adjust describe to compare to perform big integer arithmetic.

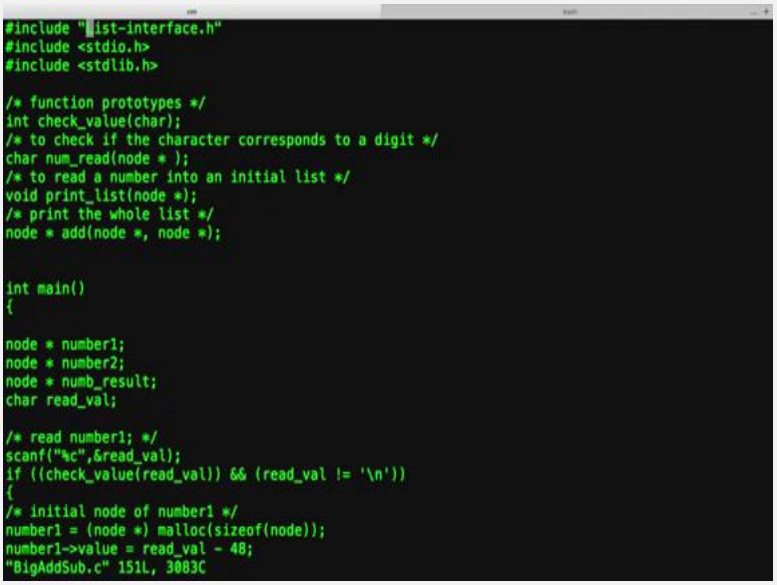
And as an example you are only going to show big integer summation and I mean big integer summation, I do not place any restriction on the size in the integer that will be input.

(Refer Slide Time: 45:00)



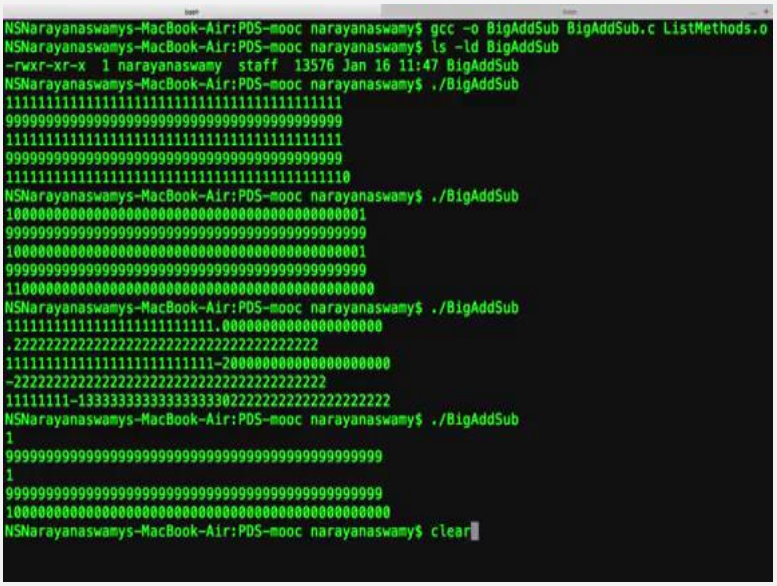
Now, let us go to the source, the source is call big and sub dot c it big addition in subtraction dot c and it will use all the methods in the list method dot c. So, now let us go to the c file and look at the implementation.

(Refer Slide Time: 45:25)



Now, the first hash include is the most important, we use list interface dot h. Now, that your seen the first hash include let me quickly exit, compile and run this program for you.

(Refer Slide Time: 45:49)



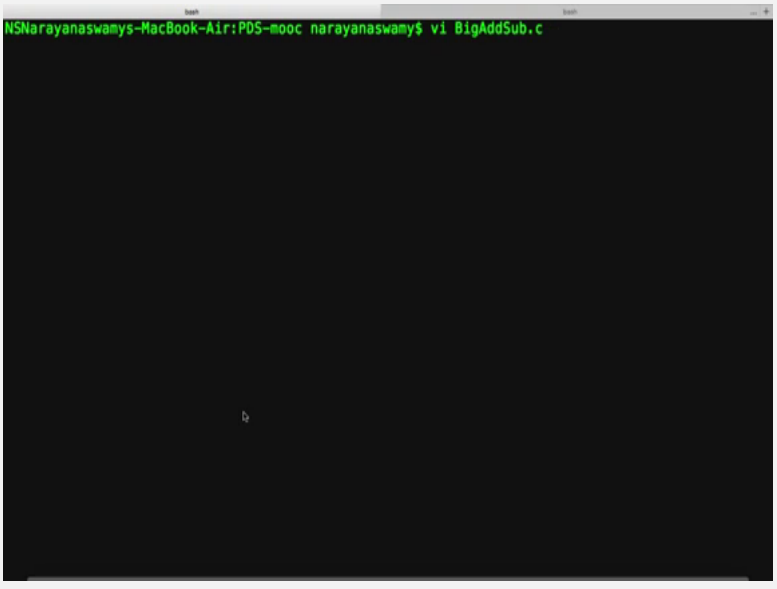
So, that you get a feeling of what the input and output look like gcc minus o sent the output to big add sub big and sub I want to compile big add sup dot c along with list methods dot o. We see this, the object file is now compile along with this c file and the compilation successful, as you can see now I have created this executable which is big add sub. So, now, let us execute big add sub.

Now, let we get a super large integer with 1 1 1 1 1 enter as the second integer 9 9 9. So, what should you see, you should see a one digit longer the every were else there is a 1 and that is the more re significant digit is 0. So, which is understand the input the first two lines where the input by had given, the next two lines was the giving printing out the input again and the third line is a summation let us run it again. These are the two inputs I have no idea have to read this numbers which one followed by a large numbers of 0s followed by a 1 add it 2 as many digits all of which are 9s.

So, as you can see the number is correct it is 1 1 followed by a large number of 0s which is the number of digits is one more than the total number of digits that where. So, let to one more input as you can see the behaviors in the program is, I we am predictable when this happens and you will see why these dashes and all these in show up. So, the behavior of the program is all most the output is meaningless, which of course, I should if card this errors.

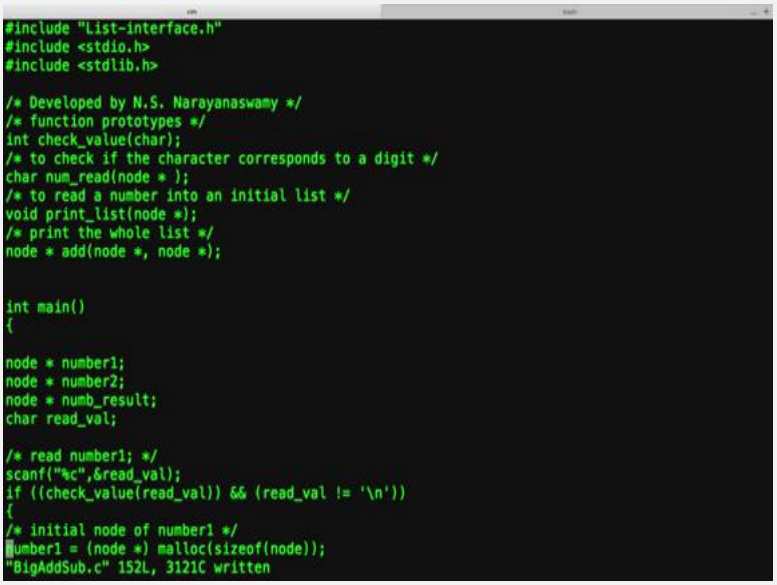
And so, let us do one more. So, this is a meaning full input and we know what the output is which one more then this many large number involving only minutes. So, essentially what have I have done, if use the list data type to perform big integer arithmetic or rather to perform addition of two very large integers.

(Refer Slide Time: 49:49)



So, the next 5 or 10 minutes we will look at the C programmer that I have down which makes function calls to this list abstract data type, then the class will end. So, the file has you know is call big and sub dot c.

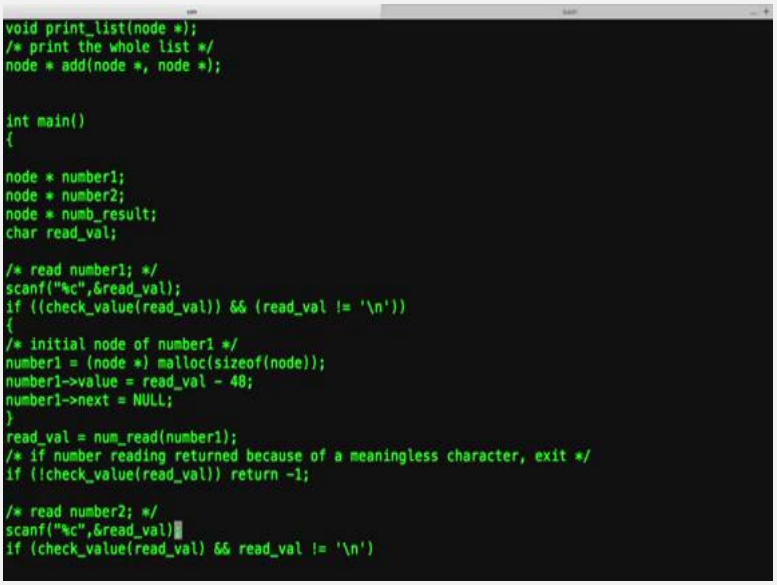
(Refer Slide Time: 50:13)



So, let us just quickly run through this file I will also ensure that by the middle of next week the whole C program is available to you for your exploration. So, I am not put in the appropriate remarks I should have put in the appropriate documentation saying that the develop by etcetera, etcetera this is good programming practice and also describe what the function is that is not mean back. So, now have four functions what you does is, the first one check some meaning fullness of the character value.

The next one is call the num read which reads and number into the initial list. So, what is do you, it takes an argument a pointer and then reach a value and returns the value that was read. And this is very crucial it returns the value that was read, the last one is something that prints a whole list in the format that you want it. And so, that was not the last one the penalty make one, the last one is the add function which takes two list of which encode two large integers access then and returns the pointer to a new list.

(Refer Slide Time: 51:49)



So, let us just look at the different parts of the code. So, number 1 is a pointer to the first number, number 2 is a pointer to the second number, num result is a pointer to the result, character is the read val is the character variable to read one value at a times from the input. So, let is just see what happens, you read the value.

Now, we check the value of read value of the character read, if it is valid and if it is not the end of line character. Then, we create the first node of number 1 and observe this formula, we using the fact that the number 0 has an ASCII value or rather the printable character 0 as the ASCII value of 48 and c allows as to subtract and integer value from a character value. So, here this is the character, subtracted with 48 it means that the ASCII value of this character is from the ASCII value of the this character the value 48 is subtract.

And then the number is make two, number 1 pointer are next is make to pointer null, now then you read the next value. So, let us just the understand what this function does, this fact computed the it created a node for the number 1. Now, num read is a function which takes a pointer to the list and returns the character, it is a last character that was read before exit from the function call. So, let us just understand this with num read returns the last character read form the input before the function call.

So, num read does, now why do we need this, if the last character that was a read as a meaningless value, we just exit.

(Refer Slide Time: 54:35)

```
/* read number1; */
scanf("%c",&read_val);
if ((check_value(read_val)) && (read_val != '\n'))
{
/* initial node of number1 */
number1 = (node *) malloc(sizeof(node));
number1->value = read_val - 48;
number1->next = NULL;
}
read_val = num_read(number1);
/* if number reading returned because of a meaningless character, exit */
if (!check_value(read_val)) return -1;

/* read number2; */
scanf("%c",&read_val);
if (check_value(read_val) && read_val != '\n')
{
number2 = (node *) malloc(sizeof(node));
number2->value = read_val - 48;
number2->next = NULL;
}
read_val = num_read(number2);
/* if number reading returned because of a meaningless character, exit */
if (!check_value(read_val)) return -1;
numb_result = add(number1,number2);
print_list(number1);
print_list(number2);
print_list(numb_result);
return 0;
```

Then we similarly read number 2 I am going to skip this spice of code, because most the work the function calls the abstract data type happen in the num read function. So, I am going to go directly do num read function.

(Refer Slide Time: 54:55)

```
{
/* initial node of number1 */
number1 = (node *) malloc(sizeof(node));
number1->value = read_val - 48;
number1->next = NULL;
}
read_val = num_read(number1);
/* if number reading returned because of a meaningless character, exit */
if (!check_value(read_val)) return -1;

/* read number2; */
scanf("%c",&read_val);
if (check_value(read_val) && read_val != '\n')
{
number2 = (node *) malloc(sizeof(node));
number2->value = read_val - 48;
number2->next = NULL;
}
read_val = num_read(number2);
/* if number reading returned because of a meaningless character, exit */
if (!check_value(read_val)) return -1;
numb_result = add(number1,number2);
print_list(number1);
print_list(number2);
print_list(numb_result);
return 0;
}

int check_value(char read_val)
```

Now, what you do is you print the number list and print the 3 numbers. So, number 1 is a list, number 2 is a list, in print number underscore result which contains a saw. And num result is what? It is another list which is obtain by performing an addition on the list number 1 and number 2.

(Refer Slide Time: 55:14)

```
/* read number2; */
scanf("%c",&read_val);
if (check_value(read_val) && read_val != '\n')
{
number2 = (node *) malloc(sizeof(node));
number2->value = read_val - 48;
number2->next = NULL;
}
read_val = num_read(number2);
/* if number reading returned because of a meaningless character, exit */
if (!check_value(read_val)) return -1;
numb_result = add(number1,number2);
print_list(number1);
print_list(number2);
print_list(numb_result);
return 0;
}

int check_value(char read_val)
{
if (read_val == ' ') return -1;
if (read_val < 48 || read_val > 57)
        return -1;
return 1;
}

char num_read(node* number)
{char read_val;
node * another_node;
```

So, now let us just look at this. So, check value is a very simple function, it check is the character is the meaning full value. So, let says if the value that is read is a space then you return a minus 1 and if it is not a space,, but some other character we check if it is value smaller than 48 r is a value is larger than 57 in that case, we return a minus 1. So, we just come to this later. So, this may requires small amount of correction.

So, I guess the correct think the right is the if the read value is outside the range 48 and 57 that is, it is not in the interval 48 and 57 that is the ASCII code is not an interval 48 and 57, then you return a minus 1. That means, it must be outside this name that is it must be either smaller than 48 or larger than 57.

(Refer Slide Time: 56:51)

```
print_list(number1);
print_list(number2);
print_list(numb_result);
return 0;
}

int check_value(char read_val)
{
if (read_val == ' ') return -1;
if (read_val < 48 || read_val > 57)
        return -1;
return 1;
}

char num_read(node * number)
{char read_val;
node * another_node;

scanf("%c",&read_val);
while (check_value(read_val) && read_val != '\n')
        {

        another_node = (node *) malloc(sizeof(node));
        another_node->value = read_val - 48;
        another_node->next = NULL;
        last(number)->next = another_node;

        scanf("%c",&read_val);
        }
```

Otherwise, you return a minus 1 saying that the value is val. Now, let us go to the number read function. So, observe now that one digit of the number has been read when this function call has been make, this is a very crucial property. When this function called is meet one read of the function is one digit of a number has been read and number is pointing to some meaning full value. So, that is now you read the next character from the input that while loop now checks whether the values meaning full.

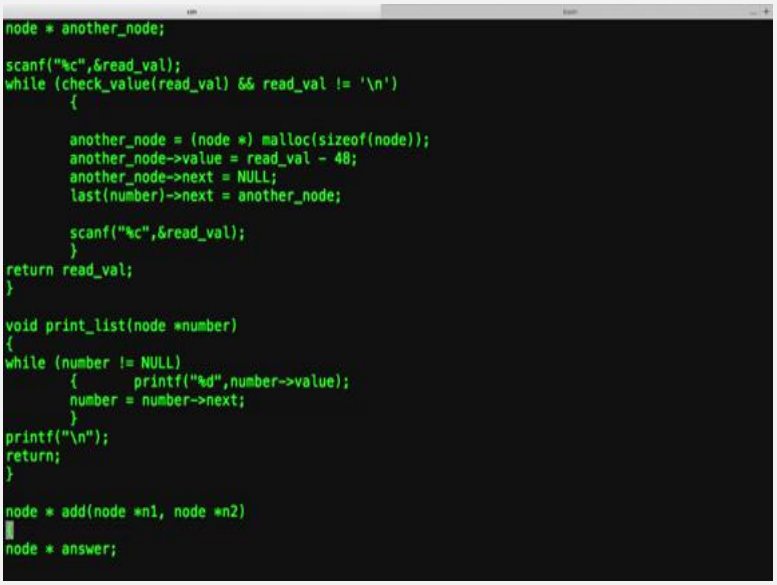
Let us just neither of end of line, nor as return meaningless value, you create a new node called another node, you put in the corresponding integer value by saying read value minus 48, make another node pointer null and enter int as the last element of the list. Observe,, but we are now in this use one of the editor functions, in the list number identify the last element and make this new node the last element of the list, let you are reading a input from left to right. So, therefore, every number that you read should go to the last element of the list, now you read the value next value and iterate.

(Refer Slide Time: 58:17)

```
return 1;
}

char num_read(node * number)
{char read_val;
node * another_node;

scanf("%c",&read_val);
while (check_value(read_val) && read_val != '\n')
        {

        another_node = (node *) malloc(sizeof(node));
        another_node->value = read_val - 48;
        another_node->next = NULL;
        last(number)->next = another_node;

        scanf("%c",&read_val);
        }
return read_val;
}

void print_list(node *number)
{
while (number != NULL)
        {       printf("%d",number->value);
        number = number->next;
        }
printf("\n");
return;
```

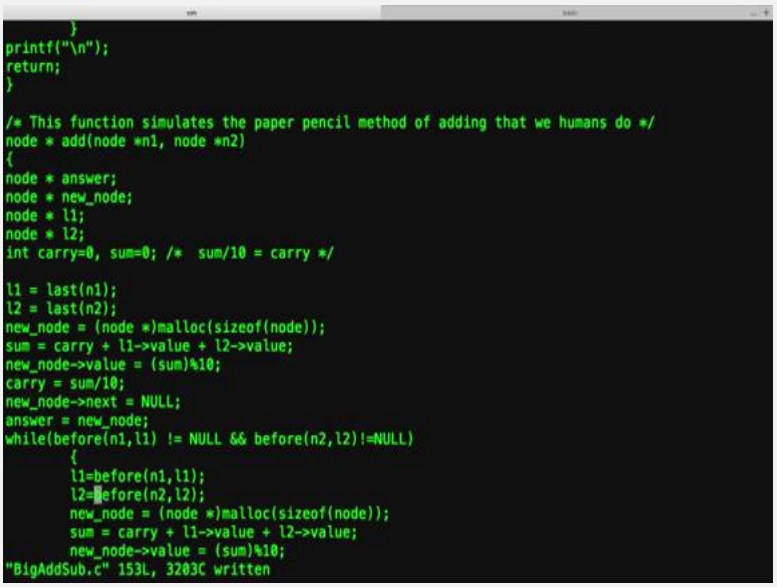
So, this is the essential either num read function. Now, let us look at the print function, the print function is a fairly straight forward, it checks if number is null while as long as number is not null, you print the value do not give any unnecessarily space and move on to the next number, then you exit from here print a new line character.

(Refer Slide Time: 58:48)



Now, let us go to the add method. So, we are in the last part of this lecture, where we are going to use all the features of an add that we have develop. Already observe that we have use the last method in the add, the last of number returns a pointer to the last element of the list call number and therefore, last of number is a pointer.

(Refer Slide Time: 59:18)



So, now let us look at the add function, what are we going to simulate. So, the add function is going to simulate the pencil paper method of add. So, this function simulates the paper pencil, this function just simulates a paper pencil method of adding that we humans. So, let us look at the variable that we have, we have one pointer called answer, one pointer of a new node. Because, we have to create a new list which contains as a

sum.

And then we have two pointers l 1 and l 2, l 1 and l 2 are design to store the last values in the un process part of the numbers l 1 and l 2 are design to store the last value in the un process part of the two numbers. So, l 1 is the last of n 1 to start with, that is it is a last digit, the right most digit of n 1, l 2 corresponds is points to the location that has a right most digit of n 2. Now, you create a new node and carry is initialize to 0, sum is also initialize to 0.

Now, you say sum is carry plus the value of n 1 value and l 1 plus value of l 2. Now, you take the digit by taking the reminder that is they divides some by 10 and take the reminder that will be the value that has to go on to the new node, this is clear this is the paper and pencil method. The carry is the value that you have to carry over to the next digit summation.

So, now this computes the value in the new node, this computes the carry, this carry will be use for the next summation and make answer point to new node. Now, observe with the first digit of the answer list has been created here now we go on to the loop.

(Refer Slide Time: 61:45)

```
node * answer;
node * new_node;
node * l1;
node * l2;
int carry=0, sum=0; /*  sum/10 = carry */

l1 = last(n1);
l2 = last(n2);
new_node = (node *)malloc(sizeof(node));
sum = carry + l1->value + l2->value;
new_node->value = (sum)%10;
carry = sum/10;
new_node->next = NULL;
answer = new_node;
while(before(n1,l1) != NULL && before(n2,l2)!=NULL)
        {
        l1=before(n1,l1);
        l2=before(n2,l2);
        new_node = (node *)malloc(sizeof(node));
        sum = carry + l1->value + l2->value;
        new_node->value = (sum)%10;
        carry = sum/10;
/* equivalently, insertBefore(answer,first(answer),new_node) */
        new_node->next = answer;
        answer = new_node;
        }
if (before(n1,l1) == NULL)
        {
        while(before(n2,l2) !=NULL)
"BigAddSub.c" 154L, 3267C written
```

Let us just look at what are the three spaces in the loop,, but three cases in the loop are this loop runs as long as before of l 1 comma n 1 is not null. And similarly before of l 2 comma n 2, that is it move from right to left, the summation has you know is happening from right to left. So, as long as if not reach the left most digit of either of the two numbers n 1 and n 2, the control will be inside this loop.

So, let we repeat this, the implementation happens in a few cases in three cases to be besides. This while loop runs as long as the control has not reached the left most digit of either number n 1 are the left most digits of either number of number n 2. Control keep running if it reaches the left most digit of one of the two, then some other action has to be taken, the control exits from the while loop let us see what happens.

So, l 1 is null the digit before, the current candidate for the left most digit at remember we are processing from right to left adding a digit by digit, like the paper pencil method. Now, here l 1 and l 2 where the two digits that were process, pointers two digits that were process in a previous iteration, after the execution of this before functions on the list n 1 and n 2, l 1 and l 2 point to the predecessor of the digits processed in the previous iteration.

Now, we create a new node to show the sum, sum is now going to be the carry which came from the previous digit, you take the l 1 value and the l 2 value at the make and when you get the carry. The least significant digit obtain by taking the remainder and division by 10 tells you what values should be new node, then the carry is updated and new node is now made the first element of the longer list, new node pointer are next is answer.

Of course, I could have make this function call here equivalently that could present among, equivalently we could have insert before answer that is in the list answer, before answer, before first of answer, insert new. This is exactly what I should a done, that is insert before in the list answer and where should insert I am insert the before the first element of answer and I will insert new.

(Refer Slide Time: 65:29)

```
l1 = last(n1);
l2 = last(n2);
new_node = (node *)malloc(sizeof(node));
sum = carry + l1->value + l2->value;
new_node->value = (sum)%10;
carry = sum/10;
new_node->next = NULL;
answer = new_node;
while(before(n1,l1) != NULL && before(n2,l2)!=NULL)
        {
        l1=before(n1,l1);
        l2=before(n2,l2);
        new_node = (node *)malloc(sizeof(node));
        sum = carry + l1->value + l2->value;
        new_node->value = (sum)%10;
        carry = sum/10;
/* equivalently, insertBefore(answer,first(answer),new_node) */
        new_node->next = answer;
        answer = new_node;
        }
if (before(n1,l1) == NULL)
        {
        while(before(n2,l2) !=NULL)
                { l2 = before(n2,l2);
        new_node = (node *)malloc(sizeof(node));
        sum = carry + l2->value;
        new_node->value = (sum)%10;
        carry = sum/10;
```

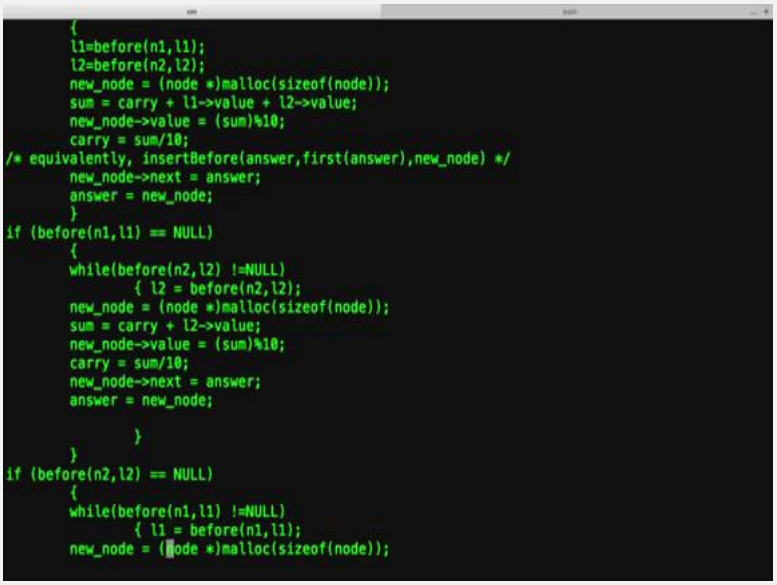
Now, when control exits from this while loop it means that, either we have reach the left most digit of the either n 1 and n 2. Let us assume here that we have reach the left most digit of n 1 which means what n 2 has further more many more digits to the left NULL n 1. Therefore, you should now visualize n 1 has lot of 0s prefix that is all we have to perform. So, here what to be do. So, till me reach the left most digit of n 2 what to be do we take the left most digit of n 2 and repeatedly add the carry.

(Refer Slide Time: 66:14)

```
new_node->next = NULL;
answer = new_node;
while(before(n1,l1) != NULL && before(n2,l2)!=NULL)
        {
        l1=before(n1,l1);
        l2=before(n2,l2);
        new_node = (node *)malloc(sizeof(node));
        sum = carry + l1->value + l2->value;
        new_node->value = (sum)%10;
        carry = sum/10;
/* equivalently, insertBefore(answer,first(answer),new_node) */
        new_node->next = answer;
        answer = new_node;
        }
if (before(n1,l1) == NULL)
        {
        while(before(n2,l2) !=NULL)
                { l2 = before(n2,l2);
        new_node = (node *)malloc(sizeof(node));
        sum = carry + l2->value;
        new_node->value = (sum)%10;
        carry = sum/10;
        new_node->next = answer;
        answer = new_node;
                }
        }
if (before(n2,l2) == NULL)
        {
```

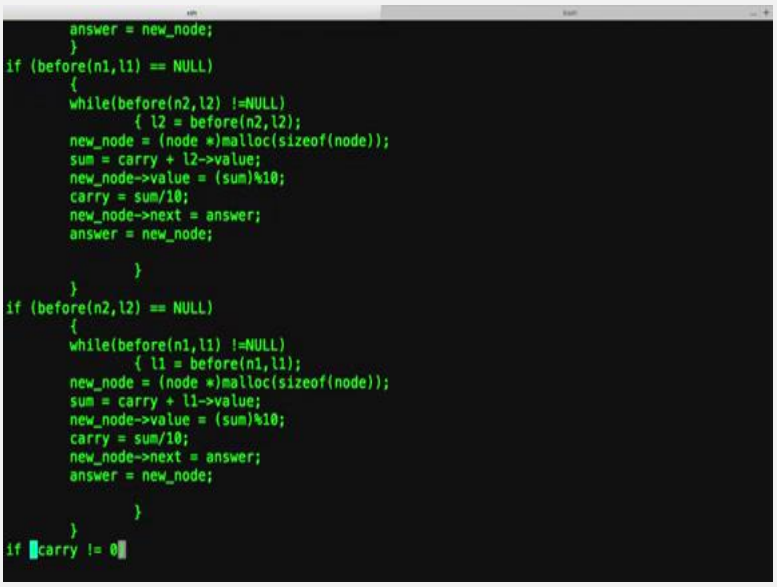
And insert it into the, let us insert the new node into the list which is answer.

(Refer Slide Time: 66:22)



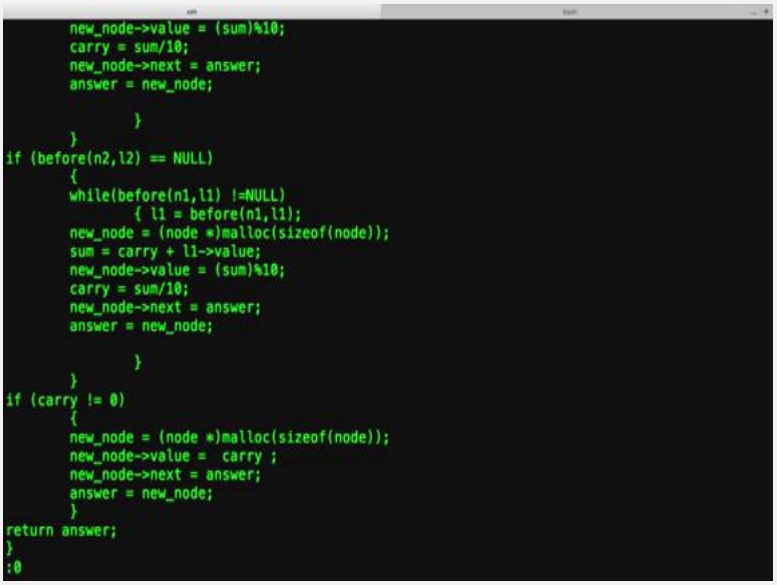
On the other hand, if symmetrically if n 2 becomes null, if reach you reach the left most digit of n 2.

(Refer Slide Time: 66:29)



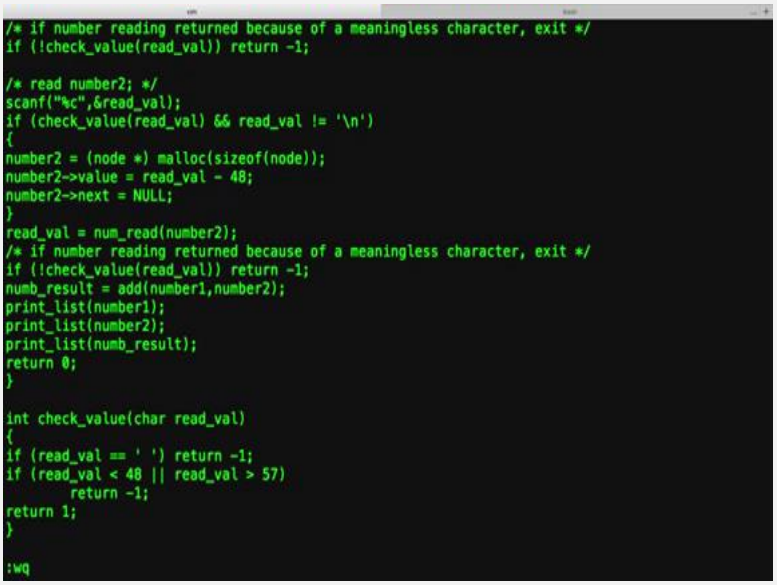
Then, you repeat is by adding two the digits of n 1, the carry iteratively tell the whole number is processed.

(Refer Slide Time: 66:38)



At the end of all this, if you still have a carry have to add a new node, this is what happens, at a end of all this if you still have a non zero carry, then you insert it into the list answer which has your summation finally. And you return a pointer to this newly created list, I hope the add function was clear, we add function basically made function calls to the methods in the abstract data type and created a new list which has the summation of the two given numbers which are represented in the list, list number 1 and number 2.

(Refer Slide Time: 67:31)

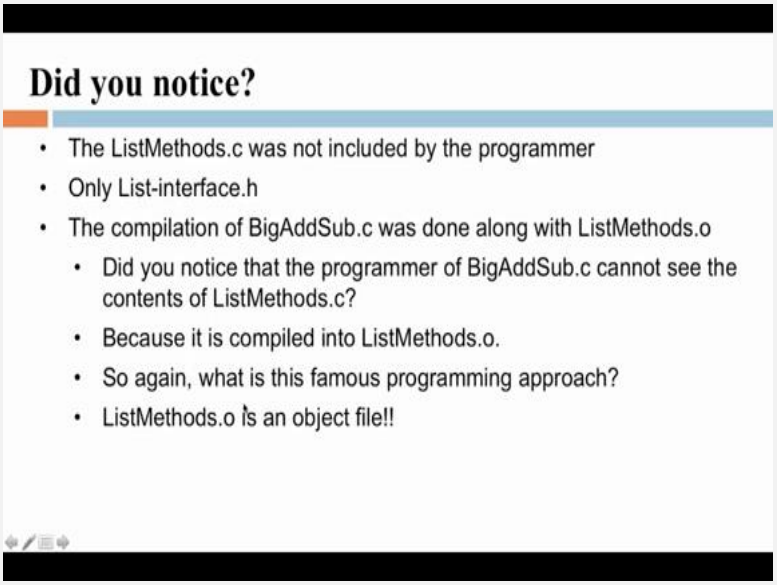


Finally, we print the result and exit from the function. So, this completes the description of this whole program and observe that we have use features of abstract data type

extensively in this next sums. So, that completes the code what through let us get back to over slides ((Refer Time: 68:03)), really now I can confidentially tell you, that you welcome to the world of programming and data structures.

And really what we have done today is the course step like, what we will do is, we will define abstract data type, we will implementing the abstract data type, we will use the abstract data type and we will meet next week we will study the array abstract data type.

(Refer Slide Time: 68:27)



Let us notice some important features in today's lecture, observe that the list methods was the c file was not use by the programmer who program big add sub dot c. This version add access only to list interface dot h and this was done along with methods, this was compiled along with list method dot o which is the object file. In particular I want did you notice that the programmer of big adds sub dot c cannot see the contents of list methods of dot c.

Because, it is compile into list method dot o of course,, I have access list method dot c if I wish to I put c them. But, as a programming practice, a good programmer or a good system of the group of programmers do not exchange the c file unnecessarily they only exchange the dot also it is. So, again for the third time in this course I am asking, what is the famous programming approach, observe that list methods dot o this called an object file.

So, let us end this lecture and let us end the lectures of this week, the programming assignment will be release to you on Sunday evening and observe the first programming

assignment the dead line has been extended to Monday morning. So, some time also Sunday evening you can book forward or by Monday afternoon you should be able to look forward to the second programming assignment.

Thank you very much.