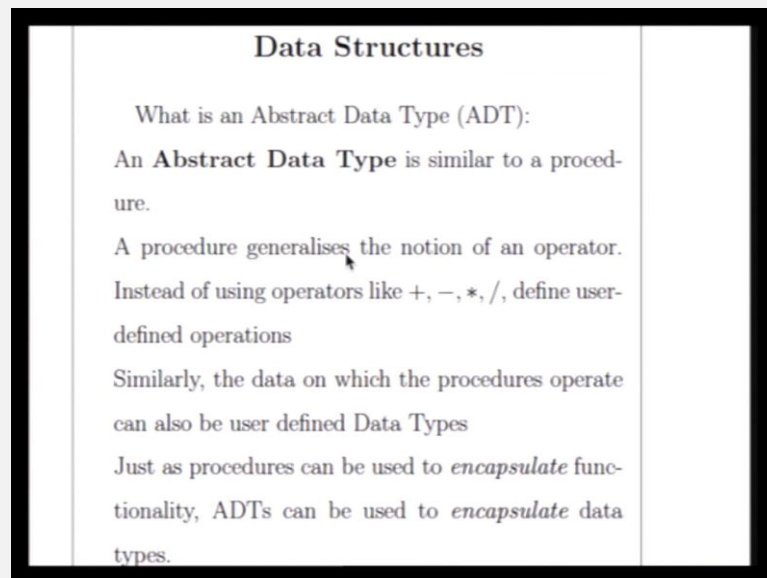


Assignment on Data Structures
Prof. Hema A Murthy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 44

Good morning, this is a supplementary video to help you to do the assignments on ADTs, I am sure all of you are started your assignments on the using abstract data types in solving problems. So, I thought what I will do is, I will work you through some examples of using abstract data types to solve problems.

(Refer Slide Time: 00:32)



Just to recap what is an abstract data type it is similar to a procedure. And just as a procedure generalizes the notion of an operator, instead of using operators like plus, minus, star and division, you define user defined operators. And also the data on which the procedures operators can operates, can also the user defined data types. So, just as in a procedure for example, you can encapsulate the functionality ADTs are used to encapsulate data types also.

So, basically what I mean is if I using for example, the cin function or the cout function in C plus plus, you do not have to know how cin is implemented in the class std, you just use it, as long as you know how to use it, that is basically the idea.

(Refer Slide Time: 01:27)

A procedure generalises the notion of an operator. Instead of using operators like $+$, $-$, $*$, $/$, define user-defined operations

Similarly, the data on which the procedures operate can also be user defined, Data Types

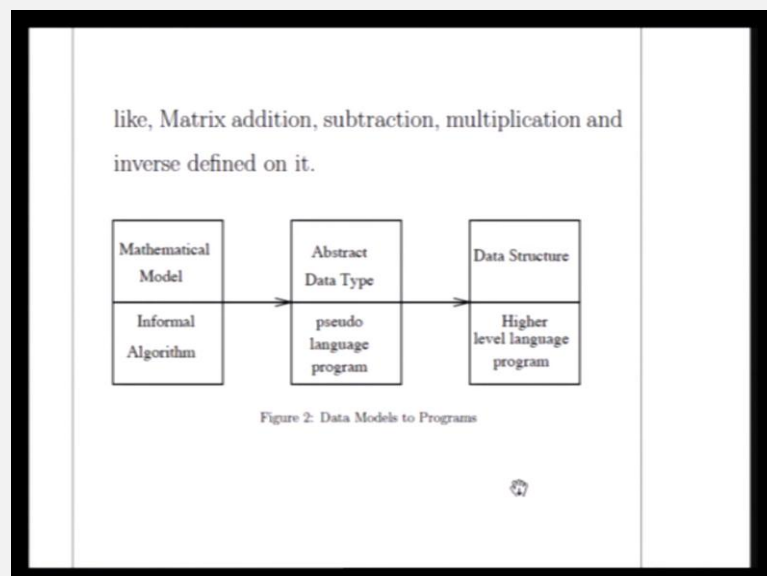
Just as procedures can be used to *encapsulate* functionality, ADTs can be used to *encapsulate* data types.

Example:
Definition of a ADT called Matrix, with operations

Hema A Murthy IIT, Madras

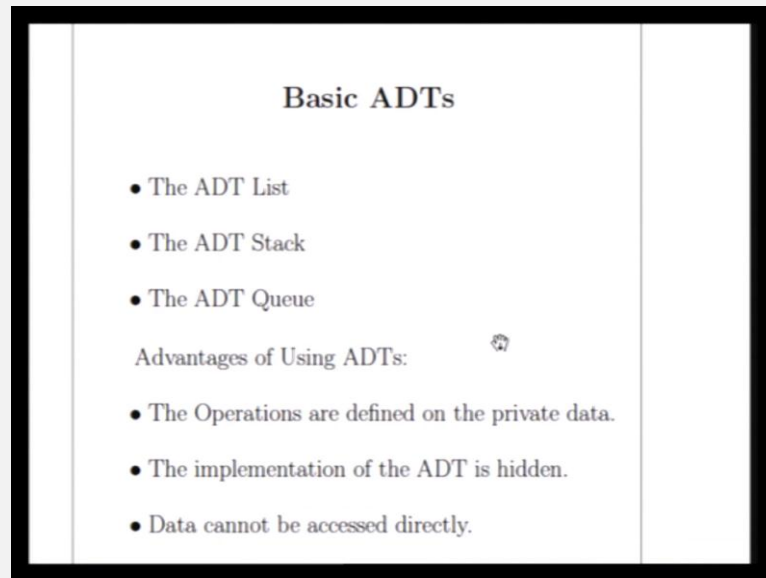
An example of an ADT user defined ADT is a matrix.

(Refer Slide Time: 01:33)



And it has operations like addition, subtraction, multiplication and inverse defined on it.

(Refer Slide Time: 01:39)



So, generally when we are talking about ADTs, we are talking about a mathematical model. For example, when we talk about matrices, you can give a mathematical model, you know it is a two dimensional array. And there are informal algorithms to show what is the multiplication of a matrix, what is the addition of a matrix, what subtraction, how do you compute inverse and so on. Then, a more Computer Science way of looking at it is, an abstract data type and a pseudo language program.

Now finally, when it goes in to the actual programming language, we have to use a concrete data structure to represent this mathematical model of a matrix which was. Kind of more closer to the programming language in an abstract data type and when the corresponding operations that are performed, we had an informal algorithm here, here you have a pseudo language program, because it is not a language specific. Finally, you have a corresponding concrete language over here.

So, from here to here the difference is essentially that you defined it in a language independent way over here and here it becomes specific to the particular language, this is what we saw in the lectures. So, basically I have a data model and from the data model you go through some intermediate step which is like a abstract data type and a pseudo language program and then you do the implementation of the abstract data type in a concrete language and the corresponding algorithms become higher level language programs.

So, now what we in this part of the course, we would decided to use C plus plus primarily because, C plus plus lend themselves, lends itself naturally for the implementation of abstract data types. So, we looked at if you remembering the course we looked at the ADT list, stack and queue.

(Refer Slide Time: 03:22)

Lists

The ADT List:

A sequence of zero or more elements of a given *ElementType*:

$a_1, a_2, a_3, \dots, a_n$

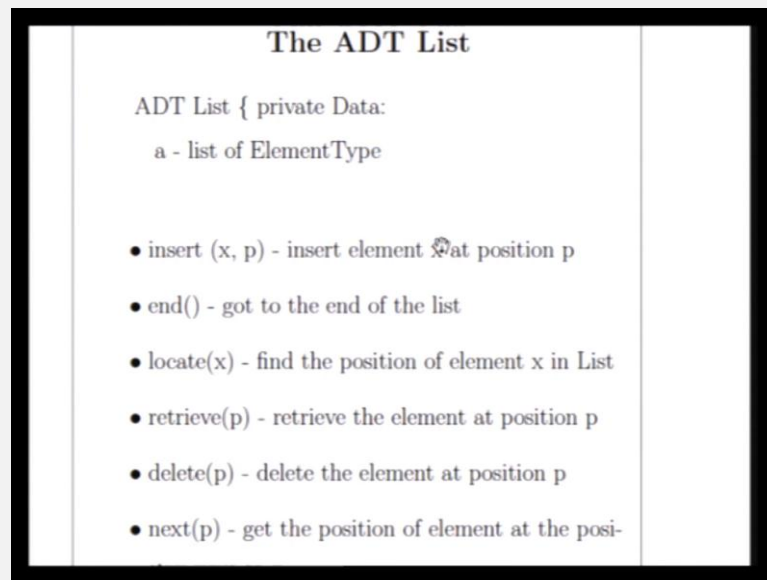
$n \geq 0$ and a_i is of *ElementType*

n – is the length of the List

$n \geq 1$, first *element* is a_1 and last is a_n

And I am always straight away go to the list which we talked about and what it we say is, it is a simplest kind of abstract data type that you can think of. Basically, it is a list of n elements and the elements are of some element type, n is the length of the list and first element is a 1 and last element is a n.

(Refer Slide Time: 03:48)



The ADT List

ADT List { private Data:
a - list of ElementType

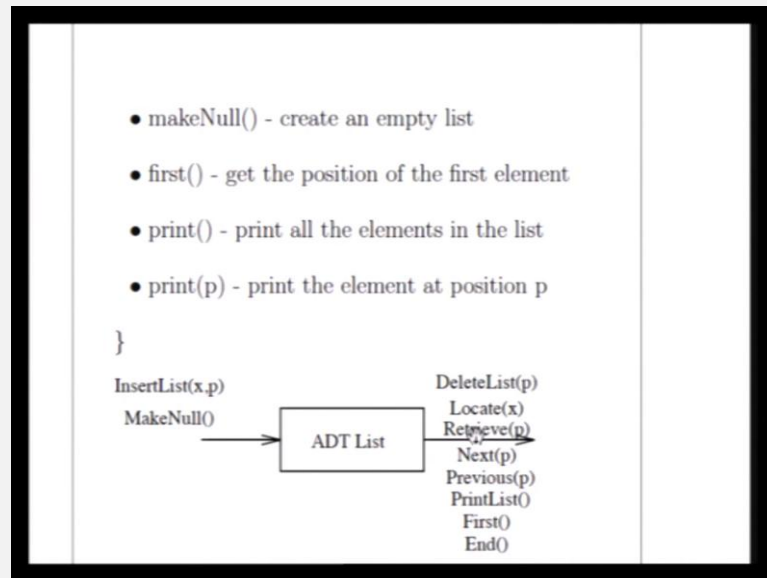
- insert (x, p) - insert element x at position p
- end() - got to the end of the list
- locate(x) - find the position of element x in List
- retrieve(p) - retrieve the element at position p
- delete(p) - delete the element at position p
- next(p) - get the position of element at the posi-

And what is interesting about the list is that you can, if you want to define it is an ADT that is quick closer to the programming language, you will say a is a list of element type. Notice that we are in the ADT here, we are not talked about what the implementation is and then, what are the various operations that I want to perform, I want to insert an element into the list at some particular point. Going back to this example over here ((Refer Time: 04:11)), I have a list of elements a 1 to a n. Suppose, I want to insert an element somewhere in the middle, how would I go back to indeed, talk about that.

And this I want to go to the end of the list, that is after a n, I want to go there, because I may want to do some operation there. Locate, I want to find an element in the given list and retrieve I want to retrieve an element, let us say I have a position let us say 3 over here. At position 3, what is the element that is the element list, delete and I want to delete an element at the position 3. Next for example, find the position of the next element in the list.

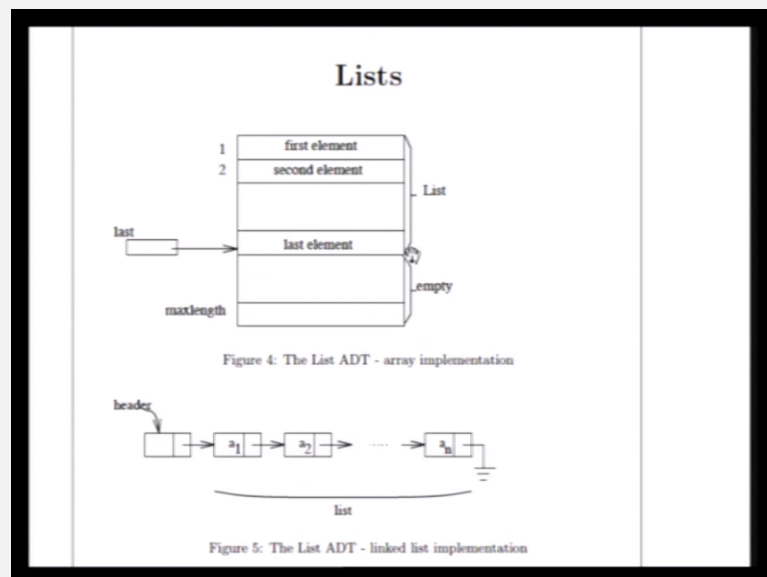
So, previous get the position we want to do why do I want this? Then, I look at next and previous for that matter. Primarily, because we do not know what the underlined implementation of the list is. The list could be implemented using arrays or link list and we just want to know why, where is the next element located.

(Refer Slide Time: 05:08)



Or I might use something called a load space, for I have multiple list that are located. And basically you need index is to point to where these elements are located. So, this is figuratively showing what a list looks like.

(Refer Slide Time: 05:20)



So, these two different kinds of data structures that we have for implementing list is an array, add a link list. Remember, the course we did using the link list of array, if I remember. So, what did we do over there was we said, we did the implementation using the main class. So, basically when you look at list for example using an array, one

corresponds to the first element in the array. And let us say this is the max length of the array and as you keep in certain elements, you also have a pointer which points to the last element in this list.

If you using a link list for example, you have pointer to the header, that is necessary and last if you want you can have a pointer or you need not have a pointer and I keep inserting the elements as I am required. So, what did we do in the course now, let we I am going to take you through the code that we have here. Let us look at this I have two implementations over here. I am going to look at the... So, basically we said I can in the course for example, we did the pointer based implementation.

(Refer Slide Time: 06:25)

```
*
* Bugs:
*
* Change Log: <Date> <Author>
* <Changes>
*
*****

#include <iostream>
using namespace std;
typedef struct cellType* Position;
typedef int elementType;
typedef struct cellType {
    elementType value;
    Position next;
} CellNode;
class List {
public:
    void insert(elementType x, Position p);
    void dellItem(Position p);
    void makeNull();
    void printList();
    Position end();
    Position first();
    Position next(Position p);
private:
    Position listHead;
```

Now, ((Refer Time: 06:09)) just enlarging it a bit, so that you can see it better, so this is the just 1 minute. So, we have this list over here. Notice that, this is the pointer based implementation, notice that in this particular case I am having a list of integers and here is something which is called an element type. An element type is defined by the user of the list, whoever is writing an application using this and here typedef struct cell star pointer is position.

Position for example, becomes the address of the next element as we saw. If it is an array implementation, address becomes an index, it becomes an integer and the various operations are given over here as indicated, because I have written this in C plus plus I

have used dell item instead of delete primarily, because delete is used for specifically deleting an element in C plus plus.

(Refer Slide Time: 07:36)

```
void dellItem(Position p);
void makeNull();
void printList();
Position end();
Position first();
Position next(Position p);
private:
    Position listHead;
};
void List::makeNull() {
    listHead = new CellNode;
    listHead->next = NULL;
}
void List::insert(elementType x, Position p) {
    Position temp;
    temp = p->next;
    p->next = new cellType;
```

Then, the something which makes an empty list, it create a dummy node as we saw in the class and insertion for example, inserts an element at a position p in the given list.

(Refer Slide Time: 07:49)

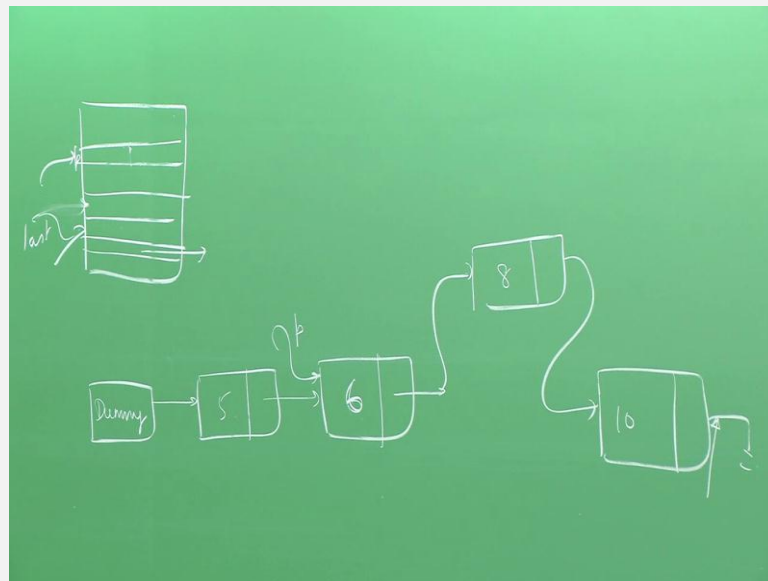
```
while (p != NULL) {
    cout << p->value << " ";
    p = p->next;
}
cout << endl;
}
Position List::end() {
    Position p;
    p = listHead;
    while (p->next != NULL)
        p = p->next;
    return(p);
}
void List::delItem(Position p) {
    p->next = p->next->next;
}
Position List::first() {
    return(listHead);
}
Position List::next(Position p) {
    return (p->next);
}
main() {
    List a;
    int i;
    elementType x;
    cout << "start pgm " << endl;
```

And print prints the entire list and list end points to the end of the list. Let me give that these kind of explain what we mean by this. Delete for example notice we have done this over here, that is it is a single operation. Normally for example, when you look at link

list representation, we have to go to a particular position by to locate the position p and to remove the element at a particular position p , you have to note that pointed in the previous node.

What we do in this implementation, you are little clever and what we do is, we create a dummy node and when you say insert a position at p , we insert at the position next to that p , what we mean by that is the following.

(Refer Slide Time: 08:35)



What we doing is, so there are two implementations as we have already saw, we saw the array and we also have list. So, what we do is we create a dummy node first and we create a make null, then when we say insert at let me say this is a position p at which I want to insert. What we do is the user notes what he or she is doing, then what we will do programmatically is, we insert at this point and point this back.

So, obviously when the user wants an element to be retrieved, let us say this is 5, 6, 8, 10. So, initially it contained 5, 6 and 10 and you said please insert an element at position p , you know that it corresponds to the next element, it is not. When you wanted this element for example, it will can be retrieved or accessed by a pointer to this, you do not have to worry about it.

So, we take care of it internally that whenever we insert an element, it is always inserted at the next element. Primarily, because remember when we did the data structures and

algorithms, we said time complexity is the very important for that event. So, in this algorithm by creating one simple dummy node, we make the delete operation in an order one operation.

(Refer Slide Time: 09:55)

```
void List::delItem(Position p) {
    p->next = p->next->next;
}
Position List::first() {
    return(listHead);
}
Position List::next(Position p) {
    return (p->next);
}
main() {
    List a;
    int i;
    elementType x;
    cout << "start pgm " << endl;
    a.makeNull();
    cout << " initialize List " << endl;
    for (i = 0; i < 10; i++)
        a.insert(i,a.end());
    cout << "before printing " << endl;
    a.printList();
    a.delItem(a.first());
    a.printList();
    a.delItem(a.next(a.first()));
    a.printList();
    return 0;
}
```

So, it basically the idea gives you and there are various operations that have been performed. Now, what I am going to do is this is the linked list implementation, now I am going to work you through the array implementation of list. Now, let us we see this one.

(Refer Slide Time: 10:25)

```
#include <iostream>
using namespace std;
typedef int elementType;
typedef int Position;
typedef struct CellType {
    elementType value;
    Position next;
} CellNode;
class List {
public:
    void insert(elementType x, Position p);
    void delItem(Position p);
    void makeNull();
    void printList();
    Position end();
    Position first();
    Position next(Position p);
private:
    Position lastNode;
    CellNode* list;
};
void List::makeNull() {
    list = new CellNode[100];
    lastNode = 0;
}
void List::insert(elementType x, Position p) {
```

Let us look at this now and what will you see now, what is the typedef int element type prevents the same, because this is a list of integer elements. Notice that, position becomes integer now, it was a pointer to CellType before and then notice the operations, the operations look identical, it do not looked different at all. So, the user of the list the person who is going to use your list to write programs can use either this list, either the link list implementation or the array implementation.

Then, what are we doing here now, we are creating... Make null for example, it creates an array of 100 elements and rest of the operations are very similar.

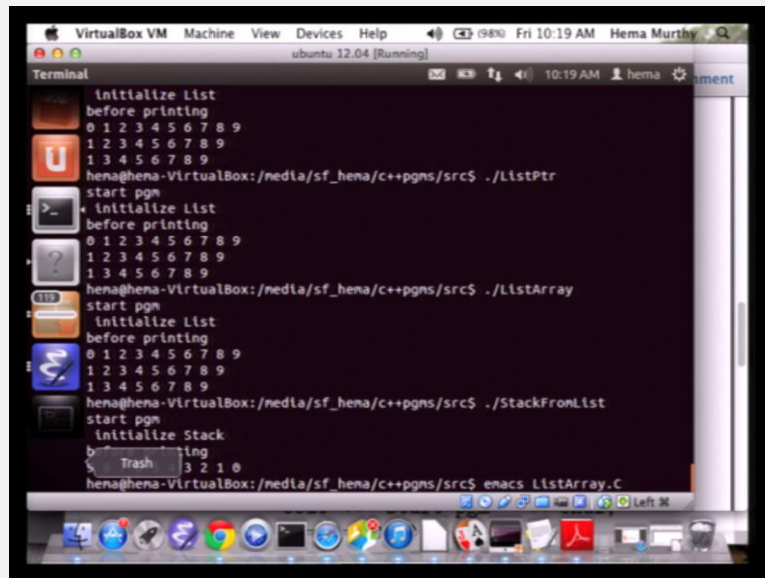
(Refer Slide Time: 11:07)

```
for (i = lastNode+1; i > p; i--)
    list[i] = list[i-1];
list[p].value = x;
lastNode++;
}
void List::printList() {
    Position p;
    p = 1;
    while (p <= lastNode) {
        cout << list[p].value << " ";
        p++;
    }
    cout << "\n";
}
Position List::end() {
    Position p;
    p = lastNode + 1;
    return(p);
}
void List::delItem(Position p) {
    int i;
    for (i = p; i < lastNode; i++)
        list[i] = list[i+1];
    lastNode = lastNode - 1;
}
Position List::first() {
```

And when you asked for the... If you are inserting for example, in the array and this is the position of the last node ((Refer Time: 11:19)). What do we do now, if you want to add an element to the list, if I let us say add a particular point p, I want to put an element here, then this last node must be moved here and put the element here and all these should be moved downwards, that is exactly what is achieved. When we talk about the end, this is the end of the list.

Similarly, when we talk about the end, this is the end of the list that is what we mean in this as long as we are consistent, end means end of the list.

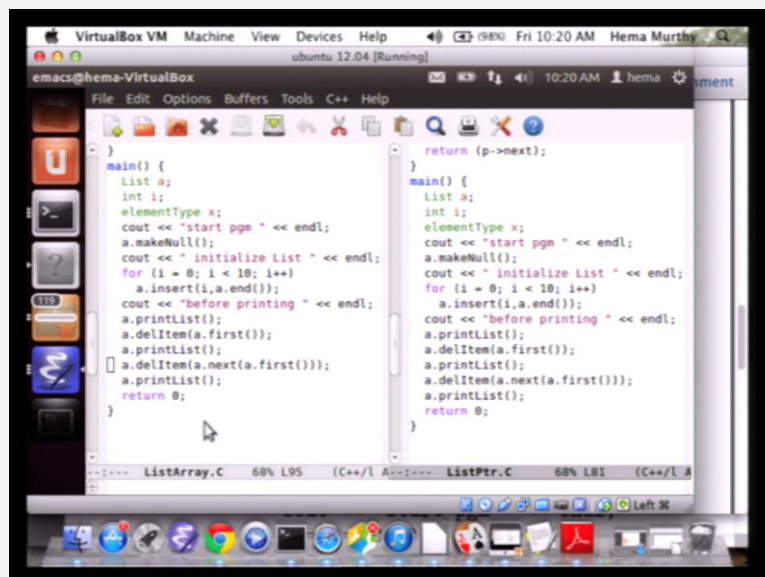
(Refer Slide Time: 12:07)



```
VirtualBox VM Machine View Devices Help
ubuntu 12.04 [Running]
Terminal
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hema@hema-VirtualBox:~/media/sf_hena/c++pgns/src$ ./ListPtr
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hema@hema-VirtualBox:~/media/sf_hena/c++pgns/src$ ./ListArray
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hema@hema-VirtualBox:~/media/sf_hena/c++pgns/src$ ./StackFrontList
start pgn
Initialize Stack
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 2 1 0
hema@hema-VirtualBox:~/media/sf_hena/c++pgns/src$ emacs ListArray.C
```

So, we do this and now what are I am going to do is we will look at, I will open it in the programming windows, so that you can look at it carefully and understand it. I want to open these two programs together and show you something.

(Refer Slide Time: 12:30)



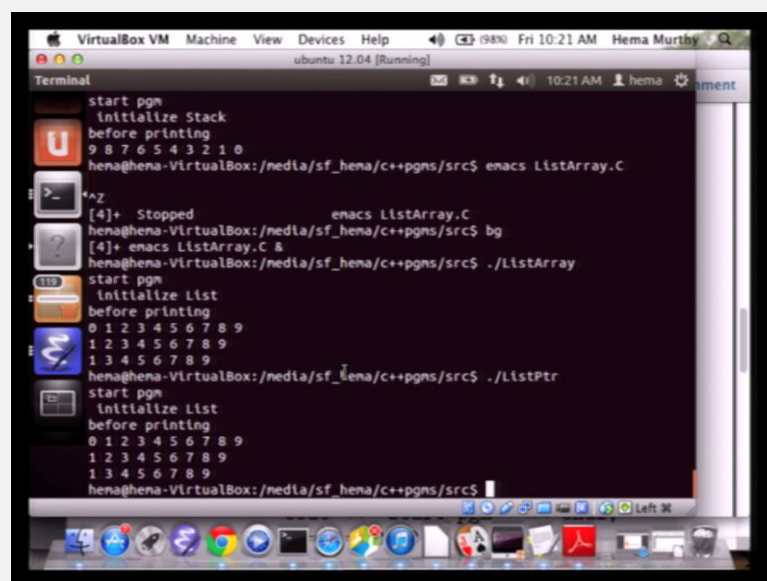
```
VirtualBox VM Machine View Devices Help
ubuntu 12.04 [Running]
emacs@hema-VirtualBox
File Edit Options Buffers Tools C++ Help
}
main() {
List a;
int i;
elementType x;
cout << "start pgn " << endl;
a.makeNull();
cout << " initialize List " << endl;
for (i = 0; i < 10; i++)
a.insert(i,a.end());
cout << "before printing " << endl;
a.printList();
a.delItem(a.first());
a.printList();
a.delItem(a.next(a.first()));
a.printList();
return 0;
}
return (p->next);
}
main() {
List a;
int i;
elementType x;
cout << "start pgn " << endl;
a.makeNull();
cout << " initialize List " << endl;
for (i = 0; i < 10; i++)
a.insert(i,a.end());
cout << "before printing " << endl;
a.printList();
a.delItem(a.first());
a.printList();
a.delItem(a.next(a.first()));
a.printList();
return 0;
}
ListArray.C 68% L95 (C++/L)
ListPtr.C 68% L81 (C++/L)
```

So, I am opening both these programs, we have the list which is represented using an array and the list which is represented using the... So, basically it is a same implementation that we have, I want you to look at this part where we have the same

main program notice this over here. If we observe this over here we create an empty list and we are inserting 10 elements into the list at the end of it.

So, initially where is the end now, end is pointing to the 0th element, the end in C is starting from one in the array, although in C plus plus it will allow you to start with 0. What am I doing, I am inserting I into the end of the list, same program over here. Then, what are we doing, assign print the list, delete item, we delete the first item in the list, print the list, delete item, the next of the first and I can print the list.

(Refer Slide Time: 13:25)



```
VirtualBox VM Machine View Devices Help 98% Fri 10:21 AM Hema Murthy
ubuntu 12.04 [Running]
Terminal
start pgn
Initialize Stack
before printing
9 8 7 6 5 4 3 2 1 0
hema@hena-VirtualBox:/media/sf_hena/c++pgns/src$ enacs ListArray.C
^AZ
[4]+ Stopped          enacs ListArray.C
hema@hena-VirtualBox:/media/sf_hena/c++pgns/src$ bg
[4]+ enacs ListArray.C &
hema@hena-VirtualBox:/media/sf_hena/c++pgns/src$ ./ListArray
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hema@hena-VirtualBox:/media/sf_hena/c++pgns/src$ ./ListPtr
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hema@hena-VirtualBox:/media/sf_hena/c++pgns/src$
```

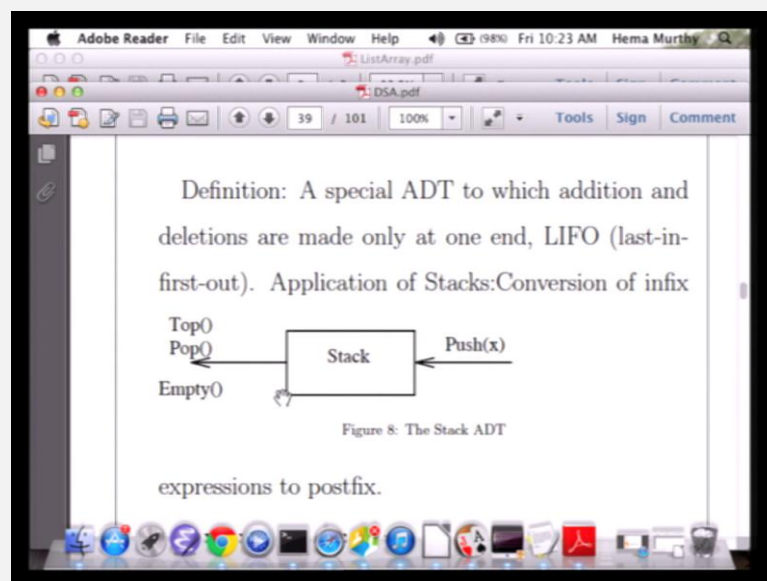
So, let us run these two programs and they should give us identical results, let us see how this can be done. So, what do I mean here, so what did we do, we inserted 10 elements into the list. So, this is what was done 0, 1, 2, 3, 7, 8, 9 then what did we do, we said delete the first element, the 0th element is gone. Then, we said delete the next to the first element which is the next to the first element to therefore, it is called deleted.

Now, let us look at list ptr, list ptr also does the same thing, remember the main program does the same. In one case, we have use the array implementation of the ADT, in other case we have use the link list implementation of the ADT, it simply does not matter. What is important is the main program is an application that uses an ADT list, and as long as you are consistently using the same ADT, it does not matter.

Now, in the assignment what you would see is that somebody would have given you a particular list implementation and ask you to use that implementation in the program. Now, what are the things that the two of the problems that are using list in your first week assignments are big int and the other one is the hash tables. Now, what I am going to do is I am going to show you one more example and then I will go to the assignment problems.

Now, so for what we have seen, we saw the array implementation and pointer implementation and we saw that the main program is the same irrespective of array or list. Primarily, because we do not operate on the data which is internal to the ADT list, that is a fundamental point. Now, what we will see is how can I use an ADT list that is being defined to do something else, can I use the list to make something else. By now, many of you might have seen the stack video.

(Refer Slide Time: 15:38)



What is the stack now? A stack is essentially as we saw in the course what did we see about the stack, thus a stack is essentially an ADT where you insert at one end and delete at a other end and we also looked at the conversion of infix to postfix in the stack. The next video actually explains how this is done using the ADT stack. Now, in a sense the stack is also a list, why a list ADT allows you to insert elements anywhere, delete elements anywhere.

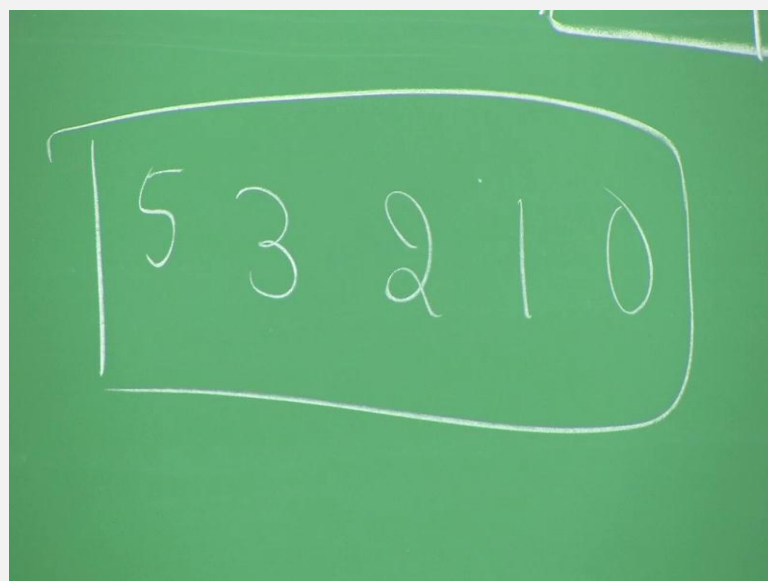
A stack on the other hand allows you to insert at only one end and delete only from the same end. Therefore, we call the stack a last in first out data structure.

(Refer Slide Time: 16:22)



So, if I insert elements into the stack for example, suppose this is the stack, stack is empty and again you can have a link list or an array implementation and so on. Let us say I have inserted 5 elements into the stack. So, this was the, the stack was empty, I put this first element and push this second, third, fourth and fifth.

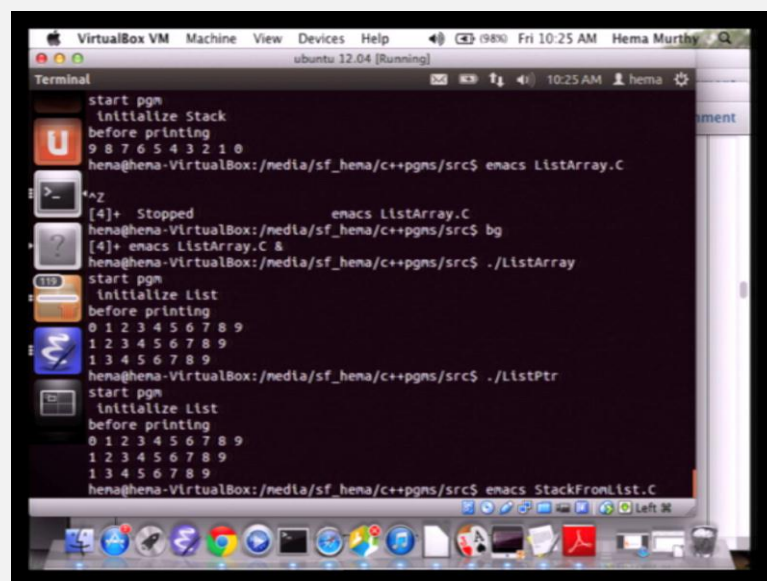
(Refer Slide Time: 16:53)



Now, if I pop the elements from the stack what are we going to see is going to pop 5 out, then it is going to pop 3, 2, 1 and 0 in reverse order. So, basically that is what we mean by last in first out and as you all know, a stack is something that is used which you might have either read about or heard about. In fact, a stack is something that we are always using even in practical live.

Programs for example, when you have functions or recursive functions in particular, what is done it puts everything on the stack, the recursive function calls itself again, you have already seen the C part of the program where you did recursion. And finally, what happens the last recursive call completes and then all the others complete in sequence in the reverse order, this is what we have seen in stacks. So, now let us see the stack is also a list, so can we implement a stack using a list ADT and that is what I am going to talk about.

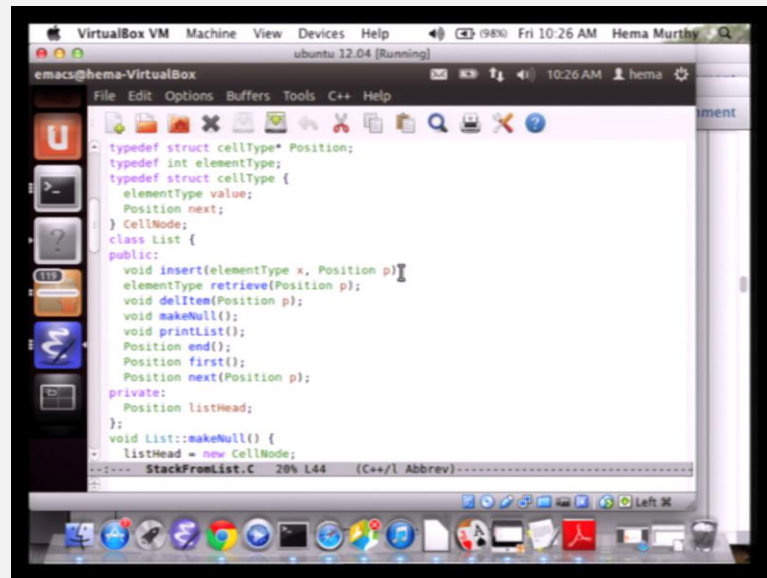
(Refer Slide Time: 17:48)



```
VirtualBox VM Machine View Devices Help
ubuntu 12.04 [Running]
Terminal
start pgn
Initialize Stack
before printing
9 8 7 6 5 4 3 2 1 0
hena@hena-VirtualBox:/media/sf_hena/c++pgns/src$ enacs ListArray.C
^AZ
[4]+ Stopped          enacs ListArray.C
hena@hena-VirtualBox:/media/sf_hena/c++pgns/src$ bg
[4]+ enacs ListArray.C &
hena@hena-VirtualBox:/media/sf_hena/c++pgns/src$ ./ListArray
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hena@hena-VirtualBox:/media/sf_hena/c++pgns/src$ ./ListPtr
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hena@hena-VirtualBox:/media/sf_hena/c++pgns/src$ enacs StackFromList.C
```

So, let us look at this example, let us say I look at stack from list is what I am calling this program.

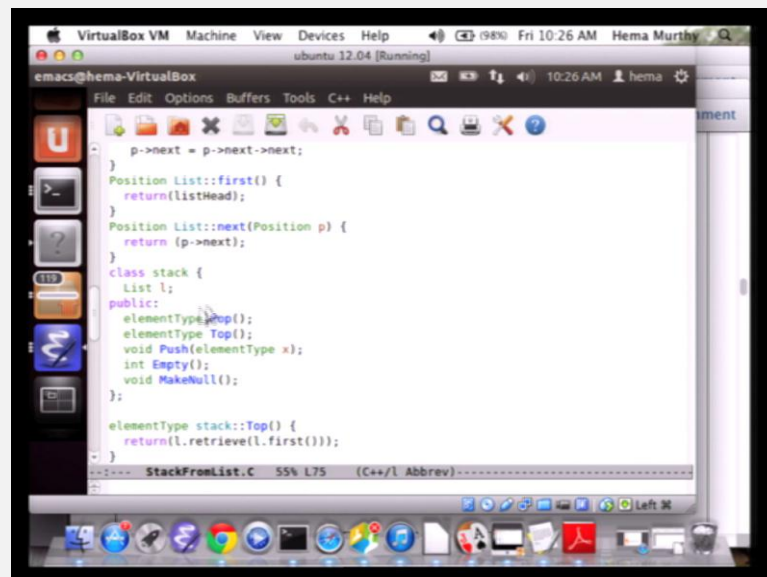
(Refer Slide Time: 18:05)



```
VirtualBox VM Machine View Devices Help
ubuntu 12.04 [Running]
emacs@hema-VirtualBox
File Edit Options Buffers Tools C++ Help
typedef struct cellType* Position;
typedef int elementType;
typedef struct cellType {
    elementType value;
    Position next;
} CellNode;
class List {
public:
    void insert(elementType x, Position p);
    elementType retrieve(Position p);
    void delItem(Position p);
    void makeNull();
    void printList();
    Position end();
    Position first();
    Position next(Position p);
private:
    Position listHead;
};
void List::makeNull() {
    listHead = new CellNode;
}
StackFromList.C 29% L44 (C++/I Abbrev)
```

So, what I am doing over here is I am assuming that I have a list which is already implemented and what we are doing is we are trying to make a stack out of it. So, what happen doing here, I am reusing the list. So, if I look at it over here, there is a class list which we have already defined that is a set of operations which are defined on it.

(Refer Slide Time: 18:30)



```
VirtualBox VM Machine View Devices Help
ubuntu 12.04 [Running]
emacs@hema-VirtualBox
File Edit Options Buffers Tools C++ Help
p->next = p->next->next;
}
Position List::first() {
    return(listHead);
}
Position List::next(Position p) {
    return (p->next);
}
class stack {
    List l;
public:
    elementType top();
    elementType Top();
    void Push(elementType x);
    int Empty();
    void MakeNull();
};
elementType stack::Top() {
    return(l.retrieve(l.first()));
}
StackFromList.C 55% L75 (C++/I Abbrev)
```

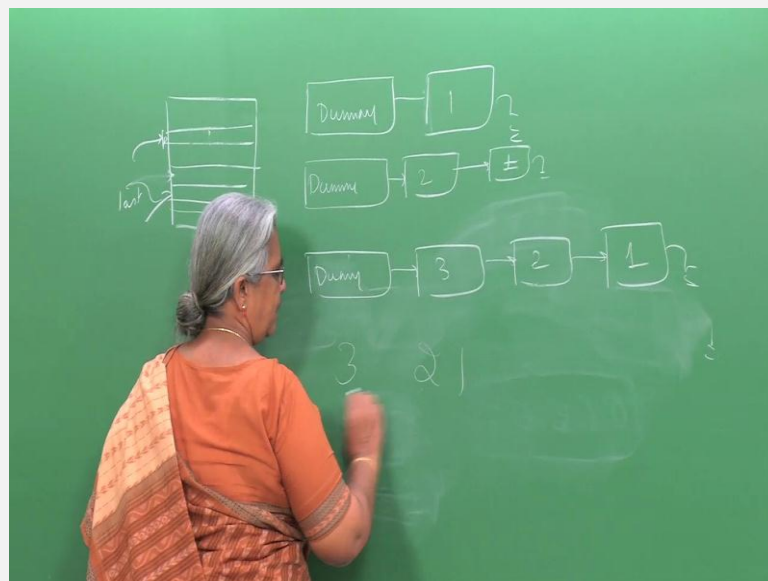
And now I want you to see what I have done in terms of implementing the class stack. What am I doing when I am implementing the class stack, I am representing a part of it is private data a list l, that is I am including a data type which is of type list l. Remember,

this is an ADT that I am defining over here. Then, what are the operations I want to implement? Pop, top, push, empty, make null, we know all this.

So, now what we do is why deriving top then, top should corresponds to retrieve the element, I am using I am always inserting at the first element and popping from the first element. So, top means for me corresponds to 1 dot first in this implementation that I have. So, I say retrieve the first element from the stack. What is pop now, it not only retrieves the first element, it should also delete the element at the first position.

So, basically I am using list operations to perform the stack operations. Now, what is the insertion now, notice that when I am pushing, I am pushing at the first element. So, what is happening now, as we keep if I go back to this list over here, so what are we doing when we are making this as a stack, we are simply doing the following.

(Refer Slide Time: 19:52)



So, this is what we have as the list and purposely we used a link list implementation here, it is simply does not matter. So, when we say insert it at first, it is going to keep, so let us say initially I had nothing, then I inserted 1, I pushed 1 into the stack. Now, next if I want to push 2 on to the stack what I am going to do is, I am going to put it here and then this will be moved forward. This is the meaning of the stack list being used as a stack.

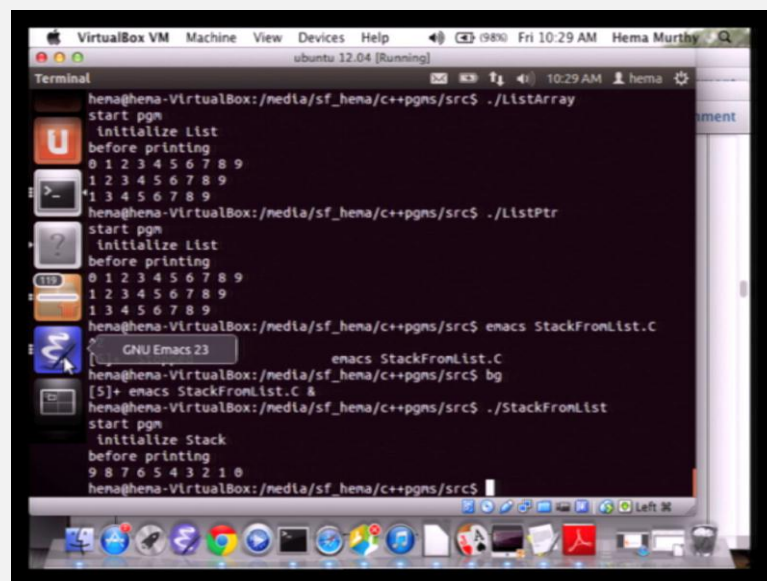
So, next if I want to push the element 3 on to the stack and what am I doing here, I am using the list like a stack, so I am pushing the element 3 here which will point to the

element 2 notice that 2 and 1 are being moved ahead in the list. So, now when I want to pop, print all the elements in the or pop all the elements in the stack, what is going to happen, it is going to put it in reverse order and that is what we will see in this particular example.

So, let us look at this program over here, so what I have done now I am not done anything, I am just simply used implemented the stack using the list which we already know about. So, let us look at this program over here, it is have a stack a. The stack a has been implemented as we have already saw and using the list ADT, then what am I doing I am creating an empty stack and then I am pushing elements 0 through 9 in to the stack and then what am I doing, I am popping out all the elements that are there in the stack.

So, what do we expect when I am pushing them, I am pushing them 0 through 9, so when I pop them it will come out in the reverse order, then it behaves like a stack.

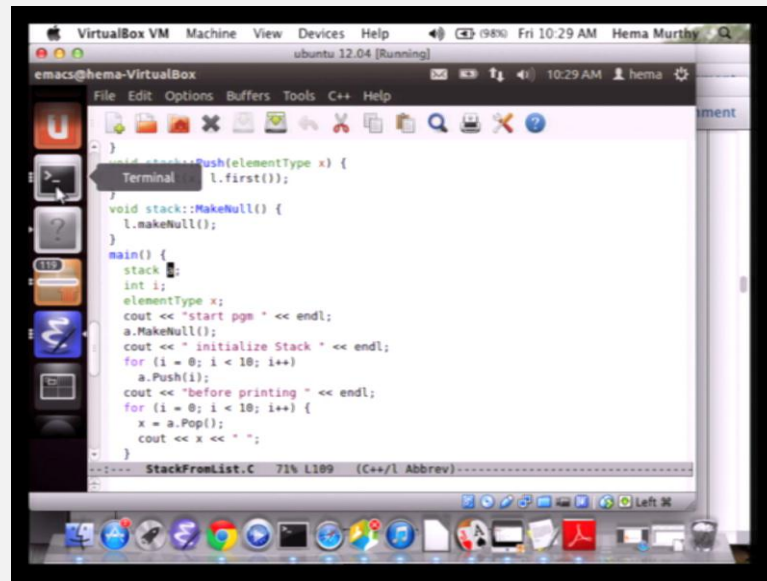
(Refer Slide Time: 21:43)



```
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$ ./ListArray
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
*1 3 4 5 6 7 8 9
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$ ./ListPtr
start pgn
Initialize List
before printing
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 4 5 6 7 8 9
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$ emacs StackFromList.C
GNU Emacs 23
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$ bg
[5]+  emacs StackFromList.C &
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$ ./StackFromList
start pgn
initialize Stack
before printing
9 8 7 6 5 4 3 2 1 0
hena@hena-VirtualBox:~/media/sf_hena/c++pgns/src$
```

So, let us see this particular example, list this is what is doing. So, notice that we insert it from 0 through 9. So, first that is an initialization at the stack and then we inserted it to be remember and finally, when I pop to elements from the stack which is what I printed, it is coming in reverse order.

(Refer Slide Time: 22:08)



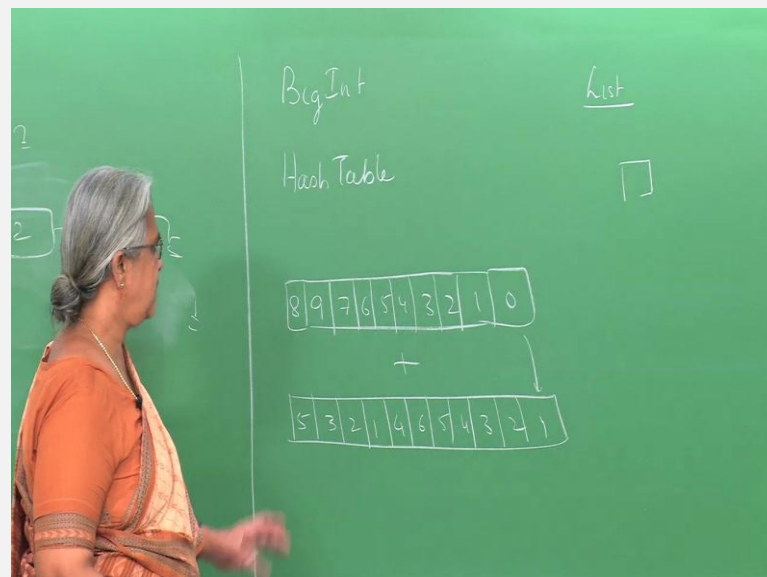
```
void stack::Push(elementType x) {
    l.first();
}

void stack::MakeNull() {
    l.makeNull();
}

main() {
    stack s;
    int i;
    elementType x;
    cout << "start pgn " << endl;
    s.MakeNull();
    cout << " initialize Stack " << endl;
    for (i = 0; i < 10; i++)
        s.Push(i);
    cout << "before printing " << endl;
    for (i = 0; i < 10; i++) {
        x = s.Pop();
        cout << x << " ";
    }
}
```

So, I hope you understand what we have done over here, so what did we do, we said pushing the elements 0 through 9 and then what are we doing, we are printing the elements that are popped out the stack and this is what we saw. So, this kind of should give you an idea about how to use an ADT in a given list.

(Refer Slide Time: 22:34)



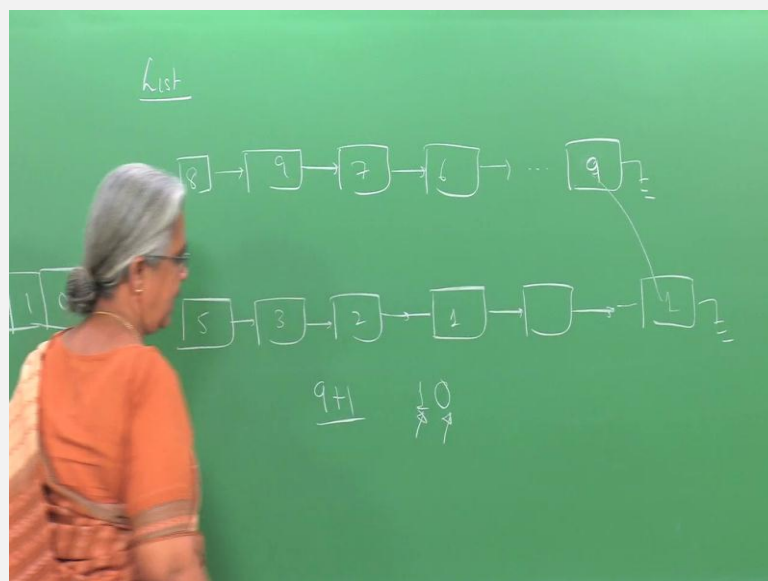
So, now let us get on to the two problems that have been given to you to solve using list, one was the big int problem and other is the hash table problem. So, let us look at each one of them in detail, what are the big int problem, they said a particular definition of list

is given in your assignment, try, the reason for giving that it is you should be able to use any user defined ADT. The definition of list can be different, I can it is a list of elements and how do you use that list of elements to perform what is called big integer arithmetic is what we said.

So, what is being done over there in that we essentially; that means, this begin int is actually operating on large numbers. Let us say I have a 10 digit number 1, 2, 3. Let us say I have a 10 digit number and something like this and I want to find the product of two such large numbers. It may be I would not take 2s exactly the same size numbers 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 let us say I have 5, 3, 2, 1, let us have something like this.

So, the problem is many of them will so; obviously, if you running with 32 bit machine or even a 64 bit machine, computations of this kind can cannot be done, because the numbers are too large. So, how do we use a list to do this?

(Refer Slide Time: 24:29)



We can create a list, it does not matter whether it is an array or user list ADT and we can create a list of all these numbers. So, what we can do is you can use when you are writing a program to compute let us say, you are asked to find out sum of these two numbers. So, what do we have to do now, so we have to add these two numbers, then if there is a carry, then this should be carried over here and so on and repeat this process

and maybe the number becomes larger than this 11 digit over here, you have to make that happen.

So, what we do the way to do this using list that I am just going to give you an idea here, what we want to do is you create a list with the elements 9, 7, 6, 5 and so on. It may be actually convenient to put them in reverse order. I am not going to give you the details for you to do that. And then what do we do I have 5, this number is written like this 2, that is every single digit number is put into a, made an element in the given list and so on have that. It will be convenient to put it in the reverse order, because then the first element becomes the first element in both the cases; otherwise, you have to go the end of the list, anyways it is a little bit more of cookie that is all.

So, what do I do, I perform the sum between these two elements. So, your objective is to get define a new ADT called big int which uses a definition of list that is given. Remember that there are append, prepend and so on and so forth, various kinds of operations where you can prefix or you can suffix and so on and so forth.

Append is put it to the end of the list and prepend is put it at the beginning of the list. We compute these operations over here, then what do we have to do now, this will generate. For example, suppose this was 9 and what will happens I basically, if I am using this version of the list, I have to pick both the last elements in the list. Then, I bring them out. Because, the element type I know what they are, because I as a user I have just to built the big int, I gave a string of two integers.

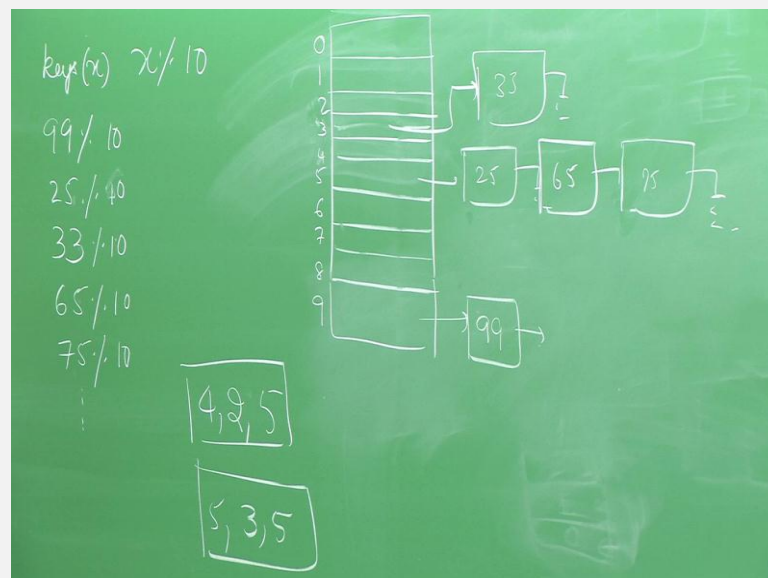
So, what am I doing I am writing it as a string, then I have to do the conversion, so on and so forth, then we perform this arithmetic. So, 9 plus 1 now this will be 10, so 10 what is happening, 10 means there is a 0 followed by a 1. So, what is this now, this is the number that goes into that and there is a carry, now what should you do, go to propagate the carry. So, what do we do, we take the previous element of the last, add the two of them and add the carry to this, you keep repeating this until you move to the end of the list.

So, the big int problem that is being given, it is primarily a particular definition of list has been given to which you have been ask to add some code write some code and the operation that it is expected to perform is the sum of two large integers. So, what will you do now, once you got the result, again the result possibly cannot be represented in an

integer variable in the computers. So, you after again converted back to string and give the result.

So, that is the important point about the big int ADT that you are asked to implement. And let us look at the hash table, what is the hash table tell you, it says that you are given hash table is defined for you. Now, notice that big int is using the list to store large numbers. So, what did we do we took these two strings, took the numbers of the strings, took all the characters one by one, each one of the characters is it is digits and put it in the list, that is what we did and then we use list to perform operations on the list, get elements out of it. Then, perform arithmetic put it back to the list, so what we did. In the hash table, it is slightly different.

(Refer Slide Time: 29:09)



What it is says that the hashing is defined, if we given a set of numbers. Let us say 99, 25, 33, 65, 75 and so on. You are asked to, these are called let us call these the keys and you will given a hash function, let us say x percent 10. So, it means when the key is x let us say and you taking the remainder when it is divided by 10, it goes into an array. And this array has how many elements can it have maximum, clearly because this is percent 10 and we are doing in base 10 over here.

So, the numbers that remainders that you can have are 7, 8 and 9, so what will you expected to do in this now. So, there is a large number of numbers that are given and for these numbers you find out what it hashes to, 99 it hashes to. So, when I take percent 10

over here, this will give me 9. So, goes into these are called buckets 0, 1, 2, 9 are the buckets are put it in this particular bucket, put 99 over here, then 25 percent 10 gives me 5.

Therefore, I put it in this bucket over here, then percent 10 gives me 3 and again I have missed 2, 3 goes into 3 over here, this is 33 and 65. Now, there is a problem. When I do a percent 10 of this what is happening, it gives me again 5 was the remainder therefore, it goes into the 5th bucket. That means, now what done, it is already an element in this bucket, therefore there is a collision.

So, what do we do now, we put 65 here and what you are expected to do as part of the program is that the assignment that is given says as soon as you find a collision, if the position in the input where the collision is occur. Therefore, 1, 2, 3, 4 it is the fourth element and the number of elements in that particular bucket do not remember exactly the order, let us say 2 here and index of the bucket. So, it is a 3 tuple answer, so this is what is expected. Now, when you come to 75 percent 10 will give me 5 again. So, goes over here therefore, I have the number 75 over here and what do we do now, it is the 5th element in the list 1, 2, 3, 4, 5.

There are three elements now in this list and it is again the 5th bucket. Just check this order, I am not show whether this is the order which we expected to get either the in. But, make sure that you presenting it in exactly the same way as expected in the question. So, this is the hash table question. Now, where is the need for list here?

So, basically you going to implement the hash table has an array of list as suggested in the question, use arrays in list is what we says, so you use arrays in list. Now, what is this now this list is the ADT list that has been defined as part of the problem, in that ADT list it is very similar to what we did in class, except that it has also get count gives you the total number of elements in the given list.

So, I hope now you are clear how you would solve the big int and the hash table problems using the ADT list. In one case what we are doing, in the big int problem we using the list like we did for the stack from the list, the list is used as a private variable and operations on the list are performed such that, it limits the operations of a stack. Here in the other hand what we are doing, we define a new ADT and we defining again as part of the private part of the hash table an array of list.

Each element of the array is called a bucket and each element of the bucket pass points to a list which consists of the list of elements that fall into the particular bucket given in the particular hash function.