

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

Module - 02
Lecture – 04
Abstract Data Types-Data plus Methods

Hello, hi. This is the fourth lecture of the course and this is the first lecture of the second week of this course. So, today, we are going to talk about abstract data types. So, let us just recall what we did last week. The last week, we talked about programming in C. In particular, we studied structures, pointers and function calls. And we saw how one can pass pointers to functions and how to return pointers from functions. We also saw how to create new data types, which are of interest. And all these are going to play a very important role and what we do for the rest of the course. In particular, we are going to do a little bit more of sophisticated C programming very relevant to the focus of this course during this week.

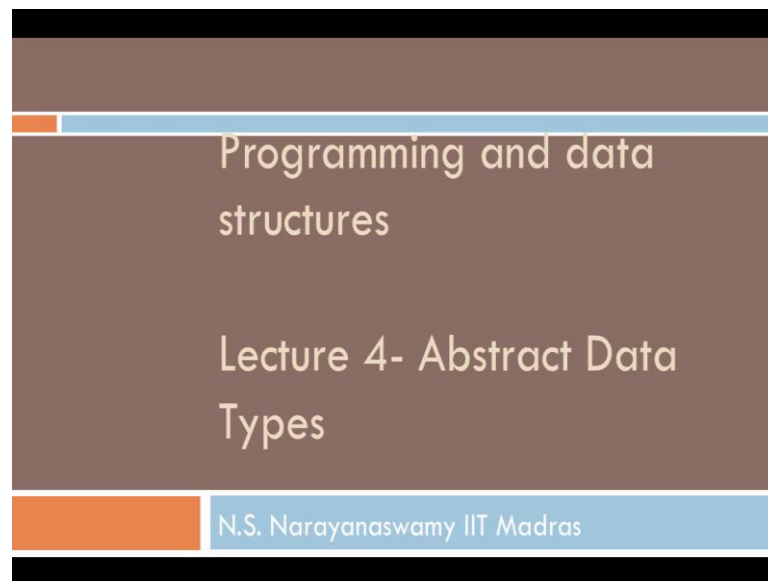
In particular, in today's lecture, we are going to talk about abstract data types, and we are going to look at programming methods in the C programming language to implement abstract data types. This is the focus of today's lecture. Again, it is likely to be just a short lecture for the following reason. In today's lecture, I am not going to show you any C code that has been written; which essentially going to be discussion or a presentation using PowerPoint slides. So, in some sense, you can think of this as a theory lecture – a very important background lecture for what is going to come in the remaining lectures during this week. In particular, during this week, we are going to implement specified different data types and implement different data types. That is the focus for this week.

So, before I go to my slides, let me also respond to some of the most important things that I have seen on the course forums. Those of you, who are attempting to write the program associated with the Pascal's triangle, the most important thing that you will want to remember is that, writing a recursive function to evaluate factorial is dangerous. So, your program will not work when you submit. Again, that is an exercise for you to figure out why. For example, if you use just the int data type; most likely, you will not be able to compute anything more than 17 factorial. So, therefore, taking the approach of

computing the binomial coefficients using the formula that we study in school for the binomial coefficients involving different factorials is unlikely to work for you. So, that is one small piece of advice. The second one is that, you get run time errors when you do not return a proper value from the main function; the submission interface is pedantic to use the correct word; it is configured with the assumption that, programmers will submit programs, where main returns an integer. So, therefore... And therefore, your return data type must be appropriately set.

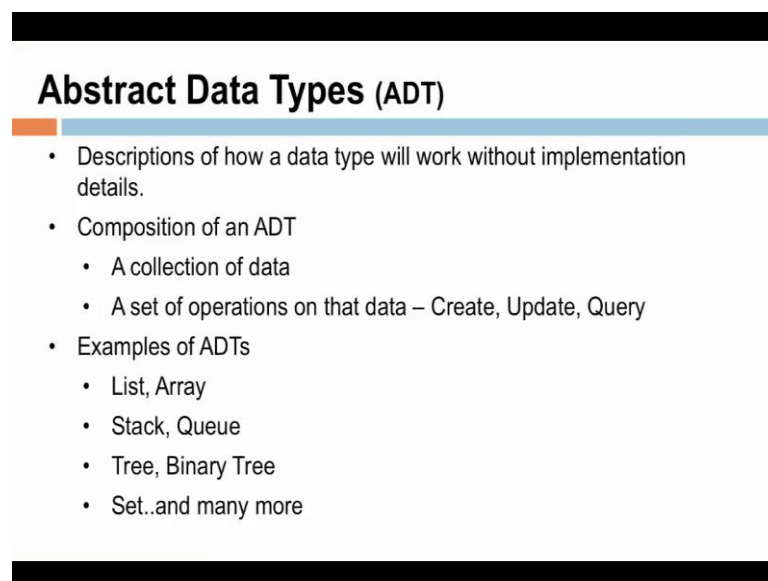
So, for example, to ensure that you do not get runtime errors, see if the following works; change your main function to `int main` and return 0 from there; and check if this will work. Another very important point; and I should not be saying this; I do not think people should be having arguments on the forum. The forum is a very gentle place for you to exchange technical ideas; I request people not to get very emotionally involved with their technical ideas and have arguments on the forum. It makes the whole forum bit unpleasant. Those of you, who want to unsubscribe from the forum; if you look at your gmail – mail box and the mails that come from the forum; there is an option to unsubscribe right there. So, you can unsubscribe; but, remember to check what happens on the forum and remember to look out for the announcements periodically. So, those are a few things that I remember from the experience of last week on a forum. And now let us go to the slides.

(Refer Slide Time: 04:55)



So, this lecture – the fourth one in this whole course is about abstract data types. And this is a very fundamental concept in the whole study and implementation of different data structures. So, you have to pay attention to what the definition of an abstract data types.

(Refer Slide Time: 05:16)



So, what does an abstract data type actually do? An abstract data type tells you how a

data type will work. And this description of how a data type will work will not involve any of the implementation details. This is extremely important. It is called an abstract data type, because you specify the kind of data that you are going to work with. And you are also going to specify what are all the different methods or different functions that will manipulate this data; but, you do not specify anything about the implementation; whether you are going to implement it in C or in C plus plus or Pascal or Java; or whether you are going to implement it using algorithm x or algorithm y; none of these will be specified when you define an abstract data type. This is a very important concept. What this enables we will see.

Now, what does an abstract data type consist of? The specification on an abstract data type consists of the specification of a collection of data and a set of operations of that data. Typical set of operations that you specify; we will see in a few examples that, this is not – by no means exhaustive. Depending on the data type that you are designing, you can come up with the appropriate set of operations. So, the normal set of operations typically fall into the following classes. You can have a create operation, which will create the data type or rather create the data structure; and you can update the data structure or query the data structure. Observe that, we started off by defining a data type; but, the operations are on the data structure. So, the data type is an abstract specification. Along with a data type, you describe a collection of operations. Now, if you ask the question where are the operations going to be applied upon? The answer is – the operations will be applied once this data type is instantiated; let me repeat – the operations will be applied once the data type is instantiated; in other words, when one instance of the data type is created.

When you create an instance of the data type, you have created the data structure. Subsequently, you can update the data structure with new pieces of data of the appropriate type. You can query the data structure for the existence of certain data and so on and so forth. I hope this is clear. Therefore, in this slide, so far we have seen two concepts: the abstract specification – something that you can write on your note book. That consists of description of the collection of data associated with your data type. Apart from describing the data, you will also list the kind of operations that you will perform on that data. The operations typically fall into the categories of creating the data

structure; in other words, instantiating the data type; in other words, creating one instance of the data type. Then subsequently, you can update the data structure and query the data structure.

So, let us look at examples of abstract data types that all of us would be familiar with. Unless this is the first time, you are actually looking at this. So, the examples of abstract data types are the most popular ones and array is an abstract data type. List is an abstract data type. Stack and queue are abstract data types. Tree and binary tree are abstract data types. Set is an abstract data type. And there are many more. So, therefore, in this slide, we have seen what an abstract data type is. So, this slide essentially was the definition of an abstract data type. So, if you ever have to answer this question – what is an abstract data type; the second point in this slide gives you the precise definition. An abstract data type is a description of a collection of data associated with a type and a set of operations that may be performed on that data; a description of the set of operations that may be performed on the data. This is what constitutes an abstract data type.

(Refer Slide Time: 09:56)

Data Structure

- An **implementation** of an abstract data type
 - An organization of data
 - A description of operations as functions (C functions)
- Data Structures are containers: they hold other data
 - Arrays and lists
 - Stack, queue.
 - Tree, Binary Tree
 - Binary search tree, hash table.
 - Dictionary, map, set etc.
- Each data structure is optimized for a special class of operations

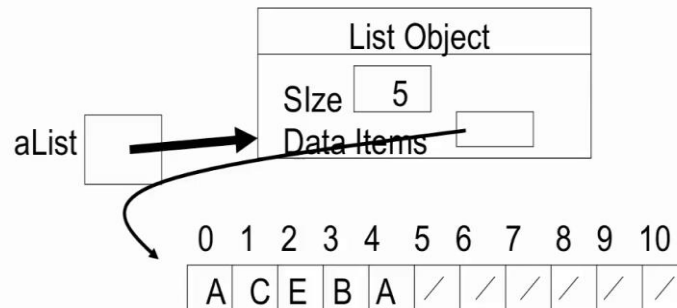
Now, we come to the central theme of this course; which is a data structure. A data structure first of all is an implementation of an abstract data type; it is not just a specification of an abstract data type; it is an implementation of an abstract data type. So,

the implementation tells you how the data is organized; and you also describe the operations as C functions. Remember in the abstract data type, you only say what the functions are by specifying the behaviour of the functions; we will definitely see an example. But, when an abstract data type is converted; when an abstract data type is implemented in a programming language; and in this case, since we are working with the C programming language; the abstract data type is implemented and you have access to a data structure.

Again let us just visualize what data structures are. You can imagine data structure to be a container. A container is something that holds other data; container is a very loosely used word; it is not a technical word in this context. For example, arrays are containers; they contain values of a base data type. Stack and queue are also containers; they can be implemented using either arrays and lists, and you can either have a stack of integers or you can have a stack of say characters or you may even have stack of what are called activation records; I repeat – stack of activation records. Find out what activation records are. We will visit this in a few lectures. Trees are containers; binary trees are containers; binary search trees are containers; they all contain other pieces of data. Now, each data structure... So, if you ask the question – why do we have so many different data structures? The answer to this question is that, each data structure is designed carefully and it is optimized for a special class of operations.

(Refer Slide Time: 12:00)

Data Structures



So, here is a pictorial representation of one instance of a data structure. In this case, you will see that, we are talking about a list. So, a list is essentially a list object. And the list object has five elements; it has five elements and you have a pointer to the five data elements. The five data elements as you can see are sitting inside an array of size 11; and the first five elements are essentially the elements inside the list object. So, this is essentially what the physical list. Therefore, this is an instance of a data type. So, let us just look at what the data type is. The data type consists of a description saying that, there is one data item, which says what the size is; and there is another data item, which is actually an array of size 11; and we maintain our list inside this array of size 11. So, therefore, this data structure is an instantiation of the abstract data type, where the fields are size- and integer-valued element; and an array of size 11; and which in this instance shows 5 values. So, this is a data structure. I repeat – in this example, I have pointed out this is a data structure obtained by instantiating a particular data type; of course, this data type was not written in the earlier slides; I have spoken about it.

(Refer Slide Time: 13:43)

List ADT

- It models a sequence of positions storing arbitrary objects
 - It establishes a before/after relation between positions
 - Generic methods:
 - `size()`, `isEmpty()`
 - Query methods:
 - `isFirst(p)`, `isLast(p)`
- Accessor methods:
`first()`, `last()`
`before(p)`, `after(p)`
- Update methods:
`replaceElement(p, o)`,
`swapElements(p, q)`
`insertBefore(p, o)`, `insertAfter(p, o)`,
`insertFirst(o)`, `insertLast(o)`
`remove(p)`

So, now, let us move on and get more specific and look at the list abstract data type. So, we will now study the list data type and implement the list data structure and so on and so forth in the next lecture. For that, we take the approach; that is the focus of this lecture. That first, we studied a abstract data type without worrying about how we are going to implement; we are not going to worry about which programming language we are going to implement this abstract data type; and we are not going to worry about what algorithms we are going to use. We are just going to describe the data type in an abstract fashion. It is an abstract data type.

And the abstract data type is called the list data type. What does the list abstract data type do? It models a sequence of locations, which store arbitrary objects. When I mean arbitrary objects, let us assume that, these are objects of a particular basic data type. So, for example, let us assume it models a sequence of positions storing integers. Then it becomes a list of integers. Now, the most important thing is that, the list data type is a sequence of position storing arbitrary objects. Remember that, it is a sequence of positions and it is not a sequence of values; it is a sequence of indexes; sequence of positions. For every position in the list, there is a well-defined before and after except for the last position and the first position.

So, let me just go back and repeat; we are talking about the list abstract data type. The list abstract data type manipulates data. What kind of data? The data is the set of positions storing a collection of objects. And every position has a well-defined before and after, that is, a predecessor and a successor except for the first element, which does not have a predecessor and the last element, which does not have a successor. This is essentially... Observe – we have only spoken about the data items; data items are positions. Now, let us look at the appropriate methods that go with the list data type. So, let us look at the generic methods. Size is a method. And we can imagine size will take him an argument, which is the list and ask how many elements are there. Now, you can think of another method, which returns a Boolean value, which is the IsEmpty method. The argument is a list; and the method will return empty or true if the list is empty. And it will return false if the list is not empty.

What are the query methods? You can give a location and ask – is this the first location? Similarly, you can give a location called p in this example – the variable p; and you can ask the query – whether this is the last location; which means it must not have a predecessor; and it must not have... it must not have a successor if it is a last ((Refer Slide Time: 17:17)) What are the accessor methods? You can ask for the first position and the last position. In other words, first and last will return pointers to the first position and the last position. Then you can ask a question saying that, if p is a position that is being; what is before p if... If p is the first element; then before p is not defined; you can immediately say that, this is an erroneous question to ask.

Similarly, if p is the last element; then after p, it does not have a meaningful value; otherwise, p has a well-defined before p and a well-defined after p; which is what will be returned. In this slide, I have really not specified the return values; I am speaking them out; encourage the student to actually write down what the return types are. And I can even say this – return type of before p is a position or an error value. Similarly, after p is also a position or an error value. First and last will take an arguments, which is a list and it will return to you the first position and the last position. Similarly, you can... These – isFirst and isLast methods return Boolean values; same with isEmpty; size returns an integer value; it tells you how many elements are in the list. So, observe that, these are structural methods, which ask you questions about the list themselves.

So, now, the update methods are here. Here you ask – replace the element in p by o. So, o is the value; in other words, o is the object that is stored in position p. The replace element method takes two arguments: one is a position p; and o is an object o; and replaces the object in position p by the object o. Of course, there is no return value here. The return value can either be a success or a failure depending upon whether p is a well-defined position or not. Another update method is this swap elements method, which swaps the contents of two locations p and q. Then there are insert methods, which say insert before p the object o. This means that, you argument to the list another location; store the value o there; make p the successor and update the predecessor appropriately and return the modified list. Similarly, insert after is that, you insert a new position after p. And in that position, store the value o; update the predecessor and successor relationships of all the locations concept. Insert first o and insert last o – are to create

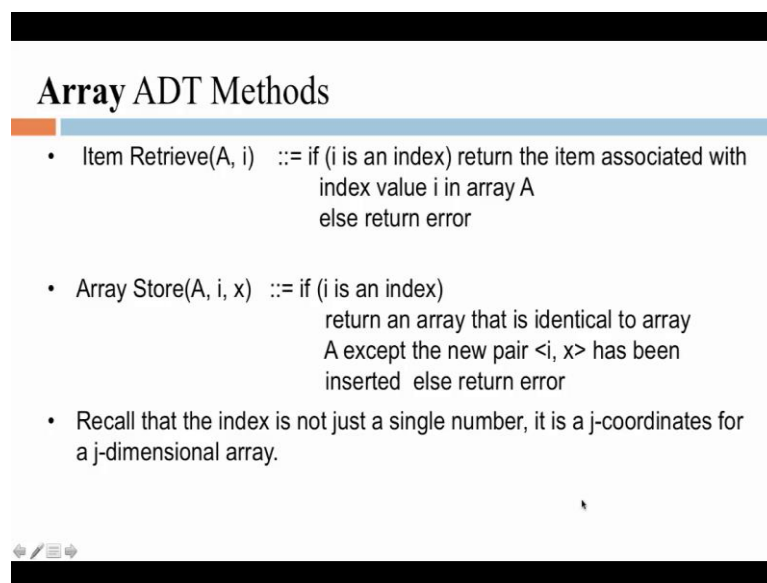
positions; make them the first position; store the value 0. Similarly, create a position; make it the last position; store the value 0. And then the last method is to remove the position p from the list. As you can see that, we have completely described all the basic operations that one might want to perform with a list, that is, an instance of the list ADT. We have really not spoken about how to implement these methods. And that is going to be the focus of the next lecture when we will discuss a C program, which implements all these methods.

Now, let us move on and look at another data type, which we are very comfortable with; which is the array abstract data type. Again, here we do not worry about how the array is implemented and how the indexing of the array happens etcetera, etcetera. These are all the things that we repeatedly use when we write a program involving arrays. But, we will not worry about how these are implemented when we discuss the array data type. So, let us look at the data that is associated with the array data type. The data consists of a set of pairs. A pair is an index and a value, which is present in that index. So, let us just see. If it is a one-dimensional array; clearly, the indices are single values – single integers – 0 to some value n minus 1 if the array has n elements; if the array has space for n elements. On the other hand, if it is a two-dimensional array; then this set gives you an example of the different indices, which are possible. 0,0 is a location in a two-dimensional array; 0,1 is a location in the two-dimensional array; 0,2 is the index of a location in a two-dimensional array; so on and so forth. So, this is essentially the data that we are working with. So, what is a data? The data has an index; and at that index, you want to store a certain value.

Now, let us look at the more generic situation, where we want to maintain a j -dimensional array; where, j is an integer – a positive integer greater than or equal to 1. I repeat we are going to look at the operations associated with a j -dimensional array. We have seen examples of a one-dimensional array, a two-dimensional array; we are going to talk about a generic j -dimensional array. In particular, we are going to describe the operations. In the description of the operations, A will stand for an array; i is an index; x is an item that you are going to store in the array; j is an integer; size is also an integer. Now, let us look at the first operation. The operation is called create j and a list. Let us just look at this example – create 2 and list of 2 comma 3. The result of executing this

operation or performing this operation is that, a two-dimensional array is created. The first dimension is of size 2 and a second dimension is of size 3. What do I mean by this? The first dimension is a row; the number of rows. The first dimension is the rows; there are two rows. And the second dimension are the columns and there are three columns here; and the return value is the array. This is exactly what is specified in an abstract way here. So, create is a function; create is an operation that returns a j-dimensional array. The size of each other dimensions is given by a j-tuple of integers. This was a 2-tuple of integers; a j-tuple of integers, where the k-th element tells you the size of the k-th dimension. There is something that I forgot to save in the example. The items are undefined. So, essentially, just the array is created and there are no values, which are populated in the array. So, this is like an array constructor function. It creates the array and leaves the items in the array unpopulated.

(Refer Slide Time: 25:50)



Array ADT Methods

- Item Retrieve(A, i) ::= if (i is an index) return the item associated with index value i in array A
else return error
- Array Store(A, i, x) ::= if (i is an index)
return an array that is identical to array A except the new pair $\langle i, x \rangle$ has been inserted else return error
- Recall that the index is not just a single number, it is a j-coordinates for a j-dimensional array.

Now, comes the other methods, which are the update methods, which are retrieve and store. So, retrieve method takes the array A as an argument, i as an index and returns a value, which is an item or an error. So, if i is a valid index; then you look at the location in the array, which is indexed by the value i and return the item associated with that particular index; otherwise, you return an error message. So, therefore, the return value of this particular function retrieve is either a valid item, which is present inside the array

or an error code. Store A, i comma x will... If i is a valid index, the array location indexed by i will contain the value x; otherwise, you will get an error. Like you to recall that; the index is just not a single number; it has j coordinates for a j-dimensional array though I have written it as a single value i here. Depending on the dimension of the array, this index i will be an appropriate entity. If it is a one-dimensional array, it is a single value; if it is a two-dimensional array, it is a 2-tuple indexing into the array and so on.

So, what did we do so far? So far we studied what constitutes an abstract data type. So, we observed that, an abstract data type has two components; you specify what the different kinds of data are; and then you specify a list of methods, which are of interest to working with that particular abstract data type. In particular, we saw two examples of data types, which we are familiar with; we saw the list data type and the array data type. What are we going to do now? We are going to now look at the details of implementing abstract data types in the C programming language; that is, in other words, how does one create data structures in the C programming language. We are going to look at the most important practices to become a good C programmer working with sophisticated data structures. That is going to be the focus of the rest of the lecture now.

(Refer Slide Time: 29:02)

Implementing ADTs in C

- The ADT is implemented in a separate C file along with a header file that is a file with extension .h.
- The lower-level implementation details, contents of the C file, of the data structure are hidden from view of the rest of the program.
- To instantiate the implemented data structure `#include "ADT.h"`.
- `#include` provides a clean, simple interface between the ADT implementation and the program(s) that use it.
- The implementation details can be changed without altering the ADT interface, that is without changing the ADT.h file.
- Of course, if you want to add more features to the ADT, then ADT.h should be changed by you.

So, implementing abstract data types in C. And this is extremely important. So, let us

assume that, ADT is a data type that we want to implement in the C programming language. So, the most important thing is that, with the abstract – with every abstract data type that you want to create; you will have a separate C file along with a set of header files, that is, file with extension dot h. So, therefore, whenever you want to implement a certain data type, which you have designed; there are two things that you will do; you will identify a C file and you will identify a typically about two header files; I will encourage you to use two header files and we will talk about these two header files. What does a C file contain? The C file contains all the implementation details associated with an abstract data type that you are implementing.

So, for example, the functions – the definition of the functions and the algorithms that you have implemented inside the functions – all these will be in the C file. Whenever you want to instantiate the implemented data structure, you will include one of the dot h files; I will call it ADT dot h. Now, why should you include this? Because hash include provides a clean way of using the features of the ADT implementation in the programs that want to use this abstract data type. I repeat – hash include is a way by which your programs that want to use the data type ADT; hash include provides a clean way of accessing the features of this data type by just looking at the dot h header file; that is, as a programmer, all that you will do is – you will look at what is there inside the dot h file; and that will tell you all that you need to know about using the abstract data type.

So, let me repeat this again – organizing the relevant information for a programmer in the ADT dot h provides a simple interface that enables you the programmer to use the features of the abstract data type without worrying about the implementation. The beauty of this whole thing is that, if you program in this particular fashion, if somebody changes the implementation without changing the ADT dot h – otherwise, called the interface; your program will still continue to work. Of course, if you want to add more features to the ADT; then the ADT dot h should be changed by you or the person you created it. So, what is the summary of this slide? If you are going to implement an abstract data type in a C programming language, what you should do is that, you identify the methods and the data items in the abstract data type; you put the data items in a dot h file; you put the definitions of the methods. So, in other words, the C functions, which are going to implement your different operations – you put them into a C file. And a programmer who

wants to use the abstract data type will include the dot h file that you have created and go ahead and use the methods, which are associated with the abstract data type without worrying about how you have implemented them. Now, this has the very nice advantage that, if I ever change the implementation of the methods; the program which a programmer has written; which uses this abstract data type – does not get affected as long as the implementation is correct.

(Refer Slide Time: 33:20)

Implementing ADTs in C - Details

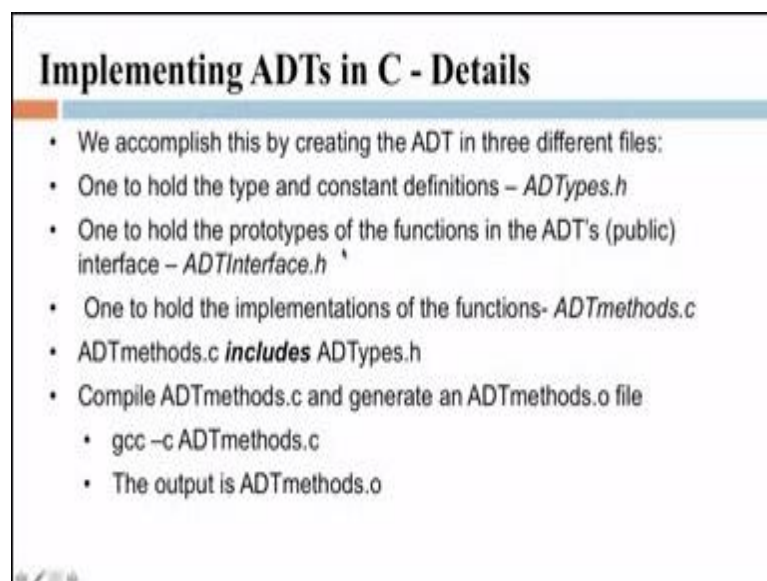
- We accomplish this by creating the ADT in three different files:
- One to hold the type and constant definitions – *ADTypes.h*
- One to hold the prototypes of the functions in the ADT's (public) interface – *ADTInterface.h*
- One to hold the implementations of the public and private functions- *ADTmethods.c*
- *ADTmethods.c* **includes** *ADTypes.h*
- Compile *ADTmethods.c* and generate an *ADTmethods.o* file
 - `gcc -c ADTmethods.c`
 - The output is *ADTmethods.o*

Now, let us get to the details. Please pay attention. So, this next couple of slides essentially tells you what you should exactly do in particular. I will tell you even the names of the files that, you should really create; and this is a good programming practice. So, let us look at the context you want to implement an abstract data type; you have written down the definition of the abstract data type on a sheet of paper; you have designed it very carefully; you know what the data items are; you know what the operators are. So, what you do is you create this abstract data type by creating three different files. What is the first file? I am going to call this *ADTypes dot h* – standing for abstract data types dot h. It is a header file; it contains the type definitions. Remember the typedef that we studied in the last week; it contains a type definitions and other constant definitions, which are important for describing the data type further. Secondly, you remember that, since you have already designed the abstract data type; you also have

the functions or the operation that you are going to support on the abstract data type. The creating the operations for construction, the operation for update, the operations for query – all these functions you have already decided.

Now, you create a second file called ADTInterface dot h; your abstract data type interface dot h. This ADT interface dot h contains prototypes of these functions. Recall from the previous week what the prototype of the function is. So, it essentially is a function name; you list the arguments and the return values. It is not the function definition; it is just a function prototype. And you put in the function prototypes in the interface dot h. So, this is a second header file.

(Refer Slide Time: 35:33)



Implementing ADTs in C - Details

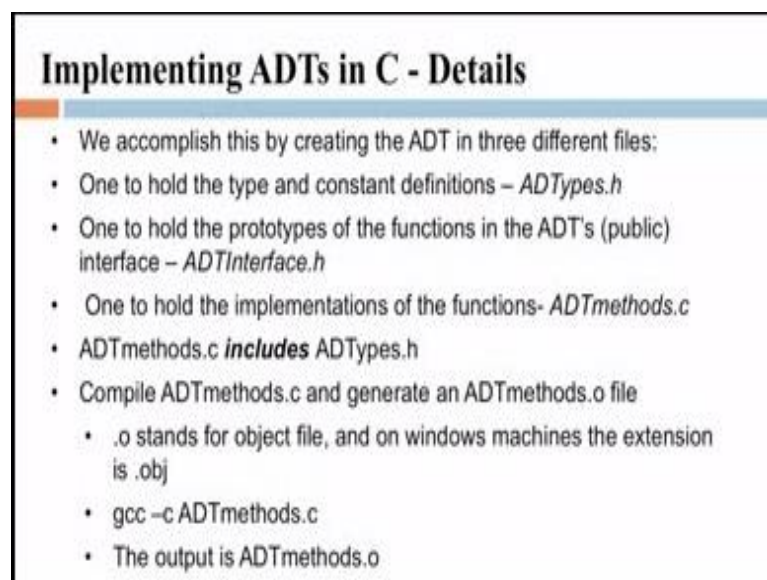
- We accomplish this by creating the ADT in three different files:
- One to hold the type and constant definitions – *ADTypes.h*
- One to hold the prototypes of the functions in the ADT's (public) interface – *ADTInterface.h*
- One to hold the implementations of the functions- *ADTmethods.c*
- *ADTmethods.c* **includes** *ADTypes.h*
- Compile *ADTmethods.c* and generate an *ADTmethods.o* file
 - `gcc -c ADTmethods.c`
 - The output is *ADTmethods.o*

Now, you come to the dot C file; I am just going to edit something here. Then in the dot C file, which I have named *ADTmethods dot c*; you put all the implementations of the functions. In this case, you put in all the function definitions; which function definitions? The functions that you have put in the *ADT interface dot h*; those functions definitely. And any other internal functions, which are necessary for the implementation. I hope this point is clear. These are the functions, which are available in the interface. To write these functions down, you may make other function calls. These function calls will be present inside the... You may write other functions. These functions along with the functions,

which are described in the ADT interface dot h will be defined inside the ADT methods dot C. So, now, you have realized the data type almost completely in the C programming language.

Let us go through a little bit more of the details to complete everything. Now, the ADT methods dot C, which has all the appropriate functions; that is, the functions in the interface dot h and any other internal functions, which are not in interface dot h; this dot C file will include the abstract data types dot h; that is, the file that contains the data organization – the appropriate structure definition and the type names that you have created – they will all be there in ADT types – ADTypes dot h. So, therefore, ADT methods dot C will include ADTypes dot h. Now, you are done. So, what do you do first? You compile your ADT methods dot C and generate not an executable; it is called a dot o file; otherwise, an object file on windows machine; this may come with the extension dot obj.

(Refer Slide Time: 37:51)



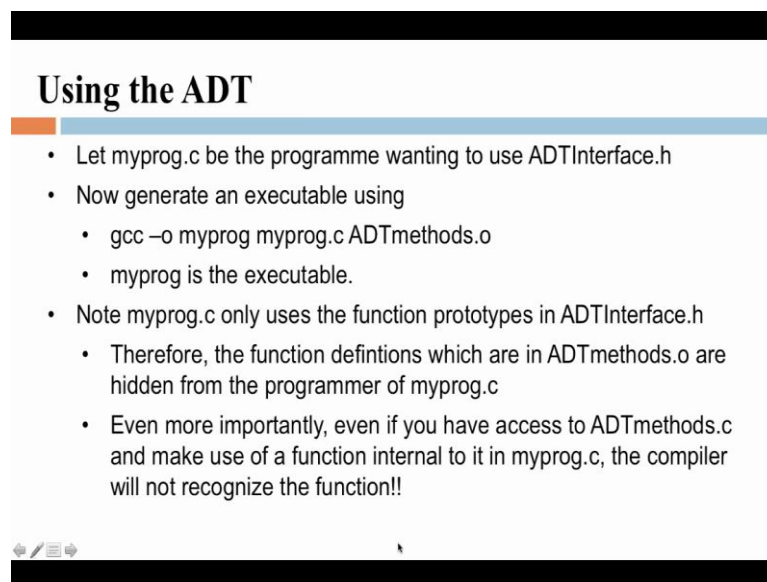
Implementing ADTs in C - Details

- We accomplish this by creating the ADT in three different files;
- One to hold the type and constant definitions – *ADTypes.h*
- One to hold the prototypes of the functions in the ADT's (public) interface – *ADTInterface.h*
- One to hold the implementations of the functions- *ADTmethods.c*
- *ADTmethods.c* **includes** *ADTypes.h*
- Compile *ADTmethods.c* and generate an *ADTmethods.o* file
 - .o stands for object file, and on windows machines the extension is .obj
 - `gcc -c ADTmethods.c`
 - The output is *ADTmethods.o*

Let me add this right in front of you – dot o stands for object file and on machines the extension is dot obj. How do you do this on a Linux machine, which is running g in your C compiler? This is the command; you type in gcc minus c – ADTmethods dot c. It will take you through exactly the compilation that you have done so far except that, it will not

create an executable that you can run immediately; it creates an object file; and the name of the object file will be ADTmethods dot o. For example, if the ADT was list; then the C file would be list methods dot c; the output will be list methods dot o. So, you are almost done. You have essentially created; or rather, you have implemented your abstract data type and you have created an object file, which has now been compiled and which contains all the important... It contains the whole description of the abstract data type that you can use in some other C program.

(Refer Slide Time: 39:22)



Using the ADT

- Let myprog.c be the programme wanting to use ADTInterface.h
- Now generate an executable using
 - gcc -o myprog myprog.c ADTmethods.o
 - myprog is the executable.
- Note myprog.c only uses the function prototypes in ADTInterface.h
 - Therefore, the function definitions which are in ADTmethods.o are hidden from the programmer of myprog.c
 - Even more importantly, even if you have access to ADTmethods.c and make use of a function internal to it in myprog.c, the compiler will not recognize the function!!

How do you use the abstract data type that you have implemented? For this, let us imagine that, myprog dot c is a C program that I am writing that, in which I want to use the abstract data type ADT what do I do? I include ADT interface dot h. Remember that, this has the appropriate interface functions into the data type that we have created. You write down your myprog dot c using all these functions and any other functions that you would want internal to this. Now, you compile to make an executable. The executable is called myprog. You compile myprog dot c along with the ADT method dot o; and you give the flag dash o; myprog is the executable that is created. Let us understand what we do. myprog dot c is the program, which wants to use this abstract data type that you have designed. Remember that, the functions that you can use to access the data items in this abstract data type are the function prototypes are written down in ADTInterface dot h. If

you go back to the previous slide, that is what we have. ADTInterface dot h contains the prototypes. So, that tells you what are the function calls that you need to make. So, using those function calls, you can write your C program and then compile your C program along with your ADTmethods dot o. Now, myprog becomes the executable.

Now, very importantly, note that, you have a C program, that is, myprog dot c – only uses the function prototype in ADT interface dot h. Therefore, in a manner of speaking the function definitions, which are in ADT methods dot o – are hidden from the programmer of myprog dot c. Even more importantly; even if you have access to ADT methods dot c; and if you try to make use of function internal to it in myprog dot c, the compiler will not recognize that function. When you compile myprog dot c; if you end up using a method, which is defined inside ADTmethods dot c; observe that, the compiler will not recognize this function. Therefore, it forces you to use only the function prototypes that are present in the interface. Now, observe that, this is a very good programming practice; whatever details you must use in the application program are the only ones that are present in the ADTInterface dot h. And that is all that a programmer who wants to use the abstract data type will use. So, this kind of force is a certain programming discipline.

(Refer Slide Time: 42:20)

Finally

- Design the ADT
- Implement it
 - Create ADTypes.h and include it in ADTmethods.c
 - Also create ADTInterface.h
 - Compile and generate ADTmethods.o
 - Now include ADTInterface.h in myprog.c, and compile myprog.c and link with ADTmethods.o.
- Very powerful programming technique.
 - Do you recognize it?!!

So, let us just summarize what we have done. You design the abstract data type. That is what we did in the first part of today's lecture. Then we went ahead and implemented this. Now, how did we implemented it? We created the ADT types dot h and included it in the ADTmethods dot c. We also created an ADTInterface dot h. Now, you compile and generate ADTmethods dot o. Now, include ADTInterface dot h in myprog dot c; you compile myprog dot c along with ADTmethods dot o. And what have you done? You have written a program that uses the implementation of this abstract data type that you had designed at the beginning. This is a very powerful programming technique. Do you recognize it? Think about it. I am not going to tell you what is programming technique is, so that I do not confuse you. You will... As we go towards the tail of the course and as you write more programs using this discipline, you yourself will realize that, this is a very powerful programming technique and a very famous programming technique.

(Refer Slide Time: 43:31)

Next lectures this week

- Designing and implementing different ADTs

What are we going to do in the next lectures in this week? We are going to design and implement different abstract data types. So, this brings to end today's lecture. Like I said, this has essentially been what you can call a theory lecture. Please go through the video very carefully; and this is all that you need to look at the programs that we will write in the next two lectures.

Thank you.