Lecture Transcripts
Abstract Data Types

So welcome to this session of the course on Data Structures and Algorithms. In this session we will be looking at the notion of an abstract data type. What it means and why is it useful and how do we use these data types. So if you think of any programming problem like the examples we have seen earlier every program takes some data as input and it computes some output data from that and in order to do that we need to define variables and functions inside the program. The variables are used to represent the data. Input data as well as output data. Whereas the function defines how to manipulate the data that you have represented. So any typical programming problem will require you to define variables for representing data and functions from manipulating data.

And variables as we have seen have what is called a type. So we have seen int, built-in types like int and float and so on. The type of a variable defines the possible values that it can take and when you declare a variable as int, for example, it specifies that this variable can take integer values and it also defines the operations that can be performed on it. So integers can be added multiplied and so on. And the type of a variable is actually independent of the data structure used to represent that variable, the actual programming language used or even the computer used. Hence, we call this type as an abstract data type. It only specifies what are the possible values of that variable and what are the operations that can be performed on it. It does not say about how the variable is represented inside the computer or which programming language is used and so on. We are only worried about the possible values and how you can manipulate those values.

So as examples we have this integer type. Informally we know integers are all numbers 0, 1, 2, 3 and so on and their negations. The operations are +, -, *, and so on. The abstract data type integer is actually an infinite set. There are an infinite number of integers, you can go as large as you want. But, the built-in data structure or data type called int is a particular implementation of the abstract data type integer. So, it actually it cannot represent all possible integers as you know int is typically 32 bit or 64 bit, so there is only a finite set of integers that can be represented by the int variable. So it is a particular implementation of the abstract data type integer. However when we want to work with integers we don't really care about this restrictions. We think of integers as abstract objects on which you can do arithmetic operations and other operations. And then choose the appropriate data structure for representing integers so that our operations work correctly. So there is another built-in data structure called long int or long int also which implements the same abstract data type. The only difference is that it can represent a larger set of values than what the int type can represent. So if int is 32-bit long long int would typically be 64 bit so you can represent larger integers using that. However when we are working with integers we would rather not think about these restriction. We think of integers as an infinite set and on which we can perform these operation.

So if you look at another example of the real numbers so possible values of reals are all fractions, decimals like 2.5, 1.333, pi is a real number whose value is 3.1415926, it's an infinite expansion. So you cannot actually represent pi exactly inside the computer. Each real value has an infinite decimal expansion and there is no way you can write down all those infinite digits exactly. So real numbers, in fact, cannot be represented inside the computer, however, we think of real numbers as abstract objects with certain property and we can work with real numbers as easily as we work with integers and then we have built-in types available which represented real numbers approximately. So reals is an abstract type which has this set of possible values and these operations. Float, double and so on are implementations of that type which impose some restrictions on what values can be represented. But when we work with real numbers we would rather not worry about those restrictions we think of real numbers as abstract objects on which we perform operations. So we think of reals as an abstract type and these as data structures which implement that abstract type with different restrictions. So there can be errors in the representation of real and you may need to choose different data structures to reduce these, the built-in data structures may not be adequate and so on.

However all of them implement the same abstract data type real numbers and why does this help? So this allows us to separate the implementation of a data type from its definition. So, when I define an abstract data type I am only talking about what are the possible values that, that type can take and what are the operations that can be performed. And this can be defined independently of any programming language, or any computer, or any specific data structure, or implementation of the type. So in a programming problem we just first need to define what are the abstract data types that we want to work with and what are the operations that we want to worry about. The actual implementation of these types and their use can be done separately and for many common data types we already have either built-in types available or libraries available which give you an implementations of those types. So if you know exactly what abstract types you need for your solving your problem then many times you will find that you have already implementations available for that type. Either as built-in types or in terms of some library and you can directly use those without worrying about how that is implemented at all. You only need to know that this type has these setup values and these are the operations that can be performed on it. And you use those operations and these values as implemented without actually worrying about how it is done. This is exactly how you used built-in types like integers you don't really care about how the integers are represented inside. All you know is that you can define variables of type int which specifies that its values are integers and you can use these arithmetic operations on those like addition and so on, comparison some set of operations that are defined.
So this allows us to use the same implementation of an abstract data type in many different programs. So if you have a different implementation for integers, if you do it once any program that needs that abstract data type can use that implementation and anybody using that type need not worry about the implementation at all. All they need to worry about is what is the abstract type data type that they want and then look for an implementation of that.

So here is an example of what you can try to do. So here is a rational numbers is another type. Right, so we know that a rational number is a ratio of a two integers. But there is no built-in type available to represent rational numbers. So you may want to define your own way of representing rational numbers and all the arithmetic operations sum them. So you think of rational numbers as an abstract data type which has these possible values. The value of a rational number is a ratio of two integers. What are the operations all possible arithmetic, operations comparisons and so on? Whatever you do with integers you can also do with rational numbers. So you want to define these arithmetic operations for rational numbers also. So now they can choose the data structure for implementing this abstract type. They can define the abstract type as rational numbers and now we can separately choose data structures for implementing this type and we write functions for implementing all operations on this type on a rational number. The advantage of this is now anybody who wants to represents rational numbers once you have done this, you have built an implementation of this abstract type. Anyone can use that implementation without actually looking at the code for that it becomes a library with these operations available somebody can just declare a rational number x and work with x as a rational number just like you work with integers. So the same implementation can be very easily used in many different programs because we know it implements the same abstract data type called rational number which has these values and these operations that can be performed done.

So let's look at the outline of this course that we will be doing. So the first part of this course will actually concentrate on definitions of some standard abstract data types and there uses. So there are many data types the occur very commonly and we will look at their definitions and their properties and some operations of those types. So this is actually independent of any programming language or any use in any particular implementation we are only defining what are the possible values and their operations. The next part of the course will concentrate on implementations of these types and including some that are readily available as part of standard libraries. So here we are worried about how to actually represent those values and how to write functions with manipulate them, whereas the third part of the course will look at uses of these types in algorithms for solving problems. So many algorithms require you to define variables of these types and you need algorithms for manipulating those which will use the operations on these variables. So next will start by looking at some particular abstract data types in the next session and will continue with that later.

Thank you.