Preface to Data Structures Course

Data structures deals with representation and arrangement of data that we use in computer programs. Based on how we represent data, certain features of our program can be highly efficient or inefficient and, often, limits what our programs can do. Therefore, the study of data structures is essential in computer science education. In fact, a part of the reason we can program modern-day computers so easily because programming languages allow us to define variables and constants the way we human understand that languages translate to bits of electrical signals that computer can process. You would be surprised to know that in early days of computers, users have to write binary numbers (that is, just sequences of zeros and ones) to communicate with the computers. Numbers, characters, and even instructions were just sequences of zeros and ones that users needed to tediously write and then send to the machine to let it do any computation. Imagine how hard and error-prone that mode of programming was. So it is a blessing that we can use data structures whose behavior we easily comprehend when writing computer programs.

Just as big structures in the real word such as roads, buildings, cars, electrical grids are made of building block elements such as concrete, brick blocks, aluminum, and wires; complex data structures that our programs use are made out of elementary data structures. These elementary data structures are so common that they have been standardized among languages and hardware. *They are typically called primitive data types as there is no structure below them*. For example, bit (aka. A true/false value), byte, character, integer, single-precision fractional number (aka, floating points), double precision fractional numbers are supported by virtually all hardware and programming languages. Larger structures are combination of these smaller structures and most often collection of them.

In fact, larger data structures are made of multiple layers of intermediate data structures that goes back to primitive data types. For example, to represent a 2D rectangle (a data structure) you need four points (an intermediate data structure) each of which has two integers (for horizontal and vertical placement). Now to compute the total land area covered by a region of rectangular land plots, you would need a collection type data structure that holds all rectangles. Suppose in your computer program you need to know the total number of rectangles frequently. Without any additional info in your collection data structure, you would need to count the rectangles every time you need that information. However, if you keep a counter in the collection that you increase by one every time you add a new rectangular plot, then when querying the total number of rectangles, you could quickly use that counter. So you understand depending on how you structure the data some feature may take constant time or involve a lot of computations. Consider another program, where you know that all plots are of the same size. Then you do not need a collection data structure at all. You just need the counter and a single rectangle. This saves a lot of space.

I hope you understand from this simple example, why the choice of data structures is so important. A good grasp of data structures is indispensible for any programming related job and even in hardware design. If you want to do research in computer science in the future then you need a strong foundation on data structures even more. That is why the data structures course is in the program core. I would take it further to say that it is among the few courses that are at the very center of the core program curriculum.

You now understand the pivotal role of data structures in program and hardware design. But what it means to have a strong foundation on data structures? It means you are capable of designing data new structures from existing when needed, pick appropriate data structures when being asked to choose, correctly assess what is the cost of different operations related to a data structure, and finally describe the composition of complex data structures in terms of their building block parts.

As we are using computers for solving all sorts of problems for so many decades now, the principal building block data structures that we need to solve interesting problems and larger structures are already well-known. The universal realization of the computing community is that these data structures are principally needed for dealing with different types of collections. If you think a little, this makes sense. A computer may be a thousand time faster than a human in doing an addition between two numbers. However, that a normal human being can do that same operation within few seconds makes the speed advantage unnecessary. It is when we are given a large sequence of numbers then the speed difference becomes prominent. A computer can do in minutes that may take a human a decade.

Now, different collection handling data structures have different advantages and disadvantages. There is no one data structure that fits for all. Furthermore, when we need to capture the interrelation among the elements of a collection along with the capacity to list them, there are some collection types that simply cannot achieve that.

In this course, you will learn the various standard data structures for collection handling. We skip basic data type (often called a class) design as you learned that in your earlier programming courses. You will design these data structures, define useful operations on them, and then use them in common programming problems. As you will go through many data structure studies and implementation exercises in this course, you might feel overwhelmed if you do not study regularly and work sincerely to understand things. So be diligent and committed to your study and attend both theory and lab classes regularly. Remember that, the student life is for learning difficult things for making future life easier.

I hope you will learn a great deal and enjoy this course very much.