Introduction to Data Structure

We study data structures to learn to write more efficient programs. But what is the point of programs being efficient when new computers are faster day by day? Because the more we are capable of excellence, the more our ambition grows. And to tackle that, we need to learn to represent and operate data more efficiently. By studying data structures, we will be able to store and use data more efficiently. In this chapter, we are going to explore the meaning of *data structure* as well as learn what different types of data structures there are.

1.1 Data Structure

To know the meaning of *data structure*, we first have to know what *data* constitutes. Simply put, *data* means raw facts or information that can be processed to get results. There are many variants of *data* to work with. For example, your *age* is one kind of data. Then again, so is your name. We can also say, this whole chapter is a form of data. From this, you can gather the fact that *data* is not only numeric. It takes all kinds of forms. Which brings us to its *structure*.

So, what do we mean by *structure*? Some elementary items constitute a unit and that unit may be considered as a structure. A structure may be treated as a frame where we organize some elementary items in different ways. Everything around us in our reality has a structure. Starting from the teeny-tiny atoms, to something as large as Mount Everest, everything has a structure that is built by the virtue of some other smaller structures, all the way down to some elementary items that constitute them.

Now that we have learned what *data* and *structure* mean separately, we can understand what *data structure* means. A data structure is a structure where we organize elementary data items in different ways and there exits a structural relationship among the items so that it can be used efficiently. In other words, a data structure is a means of structural relationships of elementary data items for storing and retrieving data in the computer's memory. To be more general, a data structure is any data representation and its associated operations. For example, an integer or floating-point number stored on the computer can be considered as a simple data structure. The term *data structure* is used more commonly to mean an organization or structuring for a collection of data items. A sorted list of integers stored in an array is an example of such a structuring.

1.2 Elements of Data Structure

Usually, elementary data items are the *elements* of a data structure. In a programming language, generally, the types of different elementary data items are Character, Integer, Float, etc. However, a data structure may be an element of another data structure. That means a data structure may contain another data structure. For example Array, Structure, Stack, etc. In computer science, we talk about or study data structure in two ways: Basic, Abstract Data Types (ADTs).

Basic data structures are those who have concrete implementation built into the programming language. For example, Variable, Pointer, Array, etc. When we use a variable, we do not need to "build" or "create" a variable from scratch. We just declare it and use it. The same goes for Pointer and Array. We do not need to "create" them because the fundamental of what they are, is already concretely built or implemented in respective programming languages.

ADTs are entities that are the definition of data and operation but do not have any concrete implementation. Example: List, Stack, Queue, etc. An ADT is an abstract idea that we have in our mind for a data structure. We do not have the implementation for it in the programming language built-in. For solving certain problems, we may need to think of such data structures that will be more efficient than the already built-in basic data structures in a programming language. To use those abstract data structures that we have in our minds, we need to build or create them by ourselves in the programming language.

1.3 Operations of Data Structures

Data structures have their operations to manipulate and use data. Without the operations, data structures are only good for storing data but not using them. Therefore, different operations of a data structure play a vital role in it to be useful.

There are some basic operations that all the data structures should have. Those operations are:

- Insertion (addition of a new element in the data structure)
- Deletion (removal of the element from the data structure)
- Traversal (accessing data elements in the data structure)

Most of the times the above-mentioned basic operations do not suffice when it comes to more advanced usage of data for different data structures. In such scenarios, we need to depend on additional operations. Some of those includes:

- Searching (locating a certain element in the data structure)
- Sorting (Arranging elements in a data structure in a specified order)
- Merging (combining elements of two similar data structures)

1.4 Algorithm

So far we have learned about what it means to be a data structure and several kinds of it. We also learned about their operations and such. Now, everything is well and fine but it would not make much of a sense to have all these data structures that we cannot use if you do not know what a program is. But before we dive into knowing that, we will first visit another term called an *algorithm*.

An algorithm simply means a set of instructions that can be followed to perform a task. In other words, the sequence of steps that can be followed to solve a problem. To solve a problem in a programming language, we first deduce an algorithm about it. To write an algorithm we do not strictly follow the grammar of any particular programming language. However, its language may be near to a programming language. Every algorithm can be divided into three sections:

- The first section is the *input* section, where we show which data elements are to be given or fed to the algorithm as an input.
- The second section is the most important one, which is the *operational* or *processing* section. Here we have to do all necessary operations, such as computation, taking a decision, calling other procedures (or algorithms), etc.

- The third section is *output*, where we display or get the result with the help of the previous two sections.

1.5 Program

Now that we know about algorithms, it is much easier for us to learn what a *program* constitutes. It is basically, the sequence of instructions of any programming language that can be followed to perform a task. This definition is quite similar to the one of an *algorithm*. The key difference is, now we are limiting the solution of a problem to be devised using a programming language. That means, we now have to follow a certain programming language's syntax and semantics. Otherwise, we cannot call it a program.

Like an algorithm, a program also has three sections: input, processing, and output. In a program usually, we use a large amount of data. Most of the cases these data are not only elementary items, but there also exists a structural relationship among them. That means a program uses *data structures*.

1.6 Exercises

Answer the following questions:

1. What is a *data structure*? Explain with examples.
2. What is Abstract Data Types (ADTs)? Explain with examples.
3. What do you mean by the terms *algorithm* and *program*? What are their differences? Explain with appropriate examples.

1.7 References

- "Data Structures and Algorithm Analysis", Edition 3.2 (C++ Version), Clifford A. Shaffer.
- https://en.wikipedia.org/wiki/Data_structure