1. **Abstract Data Type (ADT)**
* An abstract data type (ADT) is the way we look at a data structure, focusing on what it does and ignoring how it does its job.
* For example, stacks and queues are perfect examples of an ADT. We can implement both these ADTs using an array or a linked list. This demonstrates the ‘abstract’ nature of stacks and queues.
* To further understand the meaning of an abstract data type, we will break the term into ‘**data type**’ and ‘**abstract**’, and then discuss their meanings.
* Data type of a variable is the **set of values** that the variable can take.
* Basic data types in C include int, char, float, and double.
* When we talk about a primitive type (built-in data type), we actually consider two things: a **data item with certain characteristics** and the **permissible operations on that data.** For example, an **int** variable can contain any whole-number value from –32768 to 32767 and can be operated with the operators +, –, \*, and /.
* The operations that can be performed on a data type are a part of its identity. Therefore, when we declare a variable of an abstract data type (e.g., stack or a queue), we also need to specify the operations that can be performed on it.
* The word ‘abstract’ in the context of data structures means considered apart from the detailed specifications or implementation.
* In C, an **abstract data type** can be a **structure** considered without regard to its implementation. It can be thought of as a ‘description’ of the **data** in the structure with **a list of operations** that can be performed on the data within that **structure.**
* The end-user is not concerned about the details of how the methods carry out their tasks. They are only aware of the methods that are available to them and are only concerned about calling those methods and getting the results. For example, when we use a stack or a queue, the user is concerned only with the type of data and the operations that can be performed on it. Therefore, the fundamentals of how the data is stored should be invisible to the user. They should not be concerned with how the methods work or what structures are being used to store the data. They should just know that to work with stacks, they have push () and pop () functions available to them.

**Advantage of using ADTs**

* In the real world, programs evolve as a result of new requirements or constraints, so a modification to a program commonly requires a change in one or more of its data structures.
* For example, if you want to add a new field to a student’s record to keep track of more information about each student, then it will be better to replace an array with a linked structure to improve the program’s efficiency. In such a scenario, rewriting every procedure that uses the changed structure is not desirable. Therefore, a better alternative is to separate the use of a data structure from the details of its implementation. This is the principle underlying the use of abstract data types.