Contents

Introduction

Let us begin by explaining the differences between *data types*, *abstract data types* and *data structures*.

Data Types

Data types are the most basic classification of data, usually given as

- a set of possible values,
- a set of allowed operations on them,
- a concrete representation for computer to manipulate.

An examples is `int` with its arithmetic operators, represented using 32 bits with the `Int32 Struct`.

Abstract Data Types

An *abstract data type* (ADT) is a mathematical model for data types, typically giving

- possible values,
- possible operations on data of this type,
- behavior of those operations.

They are very close to data types, with the following exceptions:

- They exist only conceptually, they have no concrete existence in the context of a language,
- They define the "contractual agreement" regarding their behavior.

An example is the notion of "sets", given as a universe, a union operation that returns the union of its elements, a subset predicates that returns "true" if the first argument is a subset of the second, etc.

Data Structures

Data structures are concrete representations of data (comprised of multiple values), from the point of view of a programmer, and not from a user's perspective. It is in general given by

- data values,
- the relationships among them,
- the functions or operations that can be applied to the data.

They are generally useful for storing and retrieving data, that is, collection of values.

Summarizing

- An abstract data type is the "description", the interface, the contract: this is the most abstract perspective, describing the *behavior* of the structure you want to manipulate.
- A data type is the implementation of an abstract data type describing "atoms" of data, giving concrete instructions to the computer for how to manipulate simple (isolated) values.
- A data structure is the implementation of an abstract data type describing how to manipulate "collections" of data, giving concrete instructions to the computer for how to manipulate multiple values at the same time.

Note that some abstract data types are implemented as data types: integers can be given as an abstract data type and are (imperfectly) implemented in C# as `int`[1] Some abstract data types are (imperfectly) implemented as data structures, for example strings of text are implemented in the `String` class.

Arrays

Arrays are data structures that allow you to store multiple values in memory using a single name and indexes. Internally, an array contains a fixed number of variables (called *elements*) of a particular type[2]. The elements in an array are always stored in a contiguous block of memory, providing fast and efficient access.

An array can be:

- Single-Dimensional,
- Multidimensional.

[1] Typically, `int` cannot correctly add 2,147,483,647 (the value of `int.MaxValue`) and 2, making it an imperfect implementation of integers.

[2] Usually, all the elements of an array have the same type, but an array can store elements of different types if `object` is its type, since any element is actually of type `object`.

Multidimensional arrays can be

- Jagged,
- Rectangular.

Arrays are useful, for instance,

- When you want to store a collection of related values,
- When you do not know in advance how many variables will be needed,
- When you need a large number of variables (say, 10) of the same type,
- When you want to represent matrices (as you can use an array of arrays to represent 2-dimensional objects).