Data Structures & Algorithms

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

Table of Contents

1. Overview

Data Structure is a systematic way to organize data in order to use it efficiently. Following terms are the foundation terms of a data structure.

- **Interface** – Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.

- **Implementation** – Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

Characteristics of a Data Structure

- **Correctness** – Data structure implementation should implement its interface correctly.

- **Time Complexity** – Running time or the execution time of operations of data structure must be as small as possible.

- **Space Complexity** – Memory usage of a data structure operation should be as little as possible.

Need for Data Structure

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

- **Data Search** – Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.

- **Processor Speed** – Processor speed although being very high, falls limited if the data grows to billion records.

- **Multiple Requests** – As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

Execution Time Cases

There are three cases which are usually used to compare various data structure's execution time in a relative manner.

- **Worst Case** – This is the scenario where a particular data structure operation takes maximum time it can take. If an operation's worst case time is ƒ(n) then this operation will not take more than ƒ(n) time, where ƒ(n) represents function of n.

- **Average Case** – This is the scenario depicting the average execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then m operations will take mƒ(n) time.

- **Best Case** – This is the scenario depicting the least possible execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then the actual operation may take time as the random number which would be maximum as ƒ(n).

Basic Terminology

- **Data** – Data are values or set of values.

- **Data Item** – Data item refers to single unit of values.

- **Group Items** – Data items that are divided into sub items are called as Group Items.

- **Elementary Items** – Data items that cannot be divided are called as Elementary Items.

- **Attribute and Entity** – An entity is that which contains certain attributes or properties, which may be assigned values.

- **Entity Set** – Entities of similar attributes form an entity set.

- **Field** – Field is a single elementary unit of information representing an attribute of an entity.

- **Record** – Record is a collection of field values of a given entity.

- **File** – File is a collection of records of the entities in a given entity set.

2. Environment Setup

Try it Option Online

You really do not need to set up your own environment to start learning C programming language. Reason is very simple, we already have set up C Programming environment online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try the following example using the **Try it** option available at the top right corner of the sample code box –

```
#include <stdio.h>


int main(){
   /* My first program in C */
   printf("Hello, World! \n");


   return 0;
}
```

For most of the examples given in this tutorial, you will find Try it option, so just make use of it and enjoy your learning.

Local Environment Setup

If you are still willing to set up your environment for C programming language, you need the following two tools available on your computer, (a) Text Editor and (b) The C Compiler.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

The name and the version of the text editor can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on Windows as well as Linux or UNIX.

The files you create with your editor are called source files and contain program source code. The source files for C programs are typically named with the extension "**.c**".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it, and finally execute it.

The C Compiler

The source code written in the source file is the human readable source for your program. It needs to be "compiled", to turn into machine language so that your CPU can actually execute the program as per the given instructions.

This C programming language compiler will be used to compile your source code into a final executable program. We assume you have the basic knowledge about a programming language compiler.

Most frequently used and free available compiler is GNU C/C++ compiler. Otherwise, you can have compilers either from HP or Solaris if you have respective Operating Systems (OS).

The following section guides you on how to install GNU C/C++ compiler on various OS. We are mentioning C/C++ together because GNU GCC compiler works for both C and C++ programming languages.

Installation on UNIX/Linux

If you are using **Linux or UNIX**, then check whether GCC is installed on your system by entering the following command from the command line −

```
$ gcc -v
```

If you have GNU compiler installed on your machine, then it should print a message such as the following −

```
Using built-in specs.

Target: i386-redhat-linux

Configured with: ../configure --prefix=/usr .......

Thread model: posix

gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

If GCC is not installed, then you will have to install it yourself using the detailed instructions available at http://gcc.gnu.org/install/

This tutorial has been written based on Linux and all the given examples have been compiled on Cent OS flavor of Linux system.

Installation on Mac OS

If you use Mac OS X, the easiest way to obtain GCC is to download the Xcode development environment from Apple's website and follow the simple installation instructions. Once you have Xcode setup, you will be able to use GNU compiler for C/C++.

Xcode is currently available at developer.apple.com/technologies/tools/

Installation on Windows

To install GCC on Windows, you need to install MinGW. To install MinGW, go to the MinGW homepage, www.mingw.org, and follow the link to the MinGW download page. Download the latest version of the MinGW installation program, which should be named MinGW-<version>.exe.

While installing MinWG, at a minimum, you must install gcc-core, gcc-g++, binutils, and the MinGW runtime, but you may wish to install more.

Add the bin subdirectory of your MinGW installation to your **PATH** environment variable, so that you can specify these tools on the command line by their simple names.

When the installation is complete, you will be able to run gcc, g++, ar, ranlib, dlltool, and several other GNU tools from the Windows command line.

3. Algorithms – Basics

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms –

- **Search** – Algorithm to search an item in a data structure.
- **Sort** – Algorithm to sort items in a certain order.
- **Insert** – Algorithm to insert item in a data structure.
- **Update** – Algorithm to update an existing item in a data structure.
- **Delete** – Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics –

- **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** – An algorithm should have 0 or more well-defined inputs.
- **Output** – An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** – Algorithms must terminate after a finite number of steps.
- **Feasibility** – Should be feasible with the available resources.
- **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

How to Write an Algorithm?

There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.

We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.