D PROGRAMMING

programming basics

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

D programming language is an object-oriented multi-paradigm system programming language. D programming is actually developed by re-engineering C++ programming language, but it is distinct programming language that not only takes in some features of C++ but also some features of other programming languages such as Java, C#, Python, and Ruby.

This tutorial covers various topics ranging from the basics of the D programming language to advanced OOP concepts along with the supplementary examples.

Audience

This tutorial is designed for all those individuals who are looking for a starting point of learning D Language. Both beginners and advanced users can refer this tutorial as their learning material. Enthusiastic learners can refer it as their on-the-go reading reference. Any individual with a logical mindset can enjoy learning D through this tutorial.

Prerequisites

Before proceeding with this tutorial, it is advisable for you to understand the basics concepts of computer programming. You just need to have a basic understanding of working with a simple text editor and command line.

Disclaimer & Copyright

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

Table of Contents

Part I - D Programming Basics

1. D – OVERVIEW

D programming language is an object-oriented multi-paradigm system programming language developed by Walter Bright of Digital Mars. Its development started in 1999 and was first released in 2001. The major version of D(1.0) was released in 2007. Currently, we have D2 version of D.

D is language with syntax being C style and uses static typing. There are many features of C and C++ in D but also there are some features from these language not included part of D. Some of the notable additions to D includes,

- Unit testing
- True modules
- Garbage collection
- First class arrays
- Free and open
- Associative arrays
- Dynamic arrays
- Inner classes
- Closures
- Anonymous functions
- Lazy evaluation
- Closures

Multiple Paradigms

D is a multiple paradigm programming language. The multiple paradigms includes,

- Imperative
- Object Oriented
- Meta programming
- Functional
- Concurrent

Example

```
import std.stdio;
void main(string[] args)
{
   writeln("Hello World!");
}
```

Learning D

The most important thing to do when learning D is to focus on concepts and not get lost in language technical details.

The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones.

Scope of D

D programming has some interesting features and the official D programming site claims that D is convenient, powerful and efficient. D programming adds many features in the core language which C language has provided in the form of Standard libraries such as resizable array and string function. D makes an excellent second language for intermediate to advanced programmers. D is better in handling memory and managing the pointers that often causes trouble in C++.

D programming is intended mainly on new programs that conversion of existing programs. It provides built in testing and verification an ideal for large new project that will be written with millions of lines of code by large teams.

2. D – ENVIRONMENT

Try it Option Online

You really do not need to set up your own environment to start learning D programming language. Reason is very simple, we already have set up D Programming environment online under “Try it” option. Using this option, you can build and execute all the given examples online at the same time when you are learning theory. This gives you confidence in what you are reading and checking the result with different options. Feel free to modify any example and execute it online.

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

```
import std.stdio;


void main(string[] args)
{
   writeln("Hello World!");
}
```

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

Local Environment Setup for D

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

Text Editor for D Programming

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.

Name and version of 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 D programs are named with the extension "**.d**".

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, build it and finally execute it.

The D Compiler

Most current D implementations compile directly into machine code for efficient execution.

We have multiple D compilers available and it includes the following.

- **DMD** - The Digital Mars D compiler is the official D compiler by Walter Bright.
- **GDC** - A front-end for the GCC back-end, built using the open DMD compiler source code.
- **LDC** - A compiler based on the DMD front-end that uses LLVM as its compiler back-end.

The above different compilers can be downloaded from D downloads

We will be using D version 2 and we recommend not to download D1.

Let's have a helloWorld.d program as follows. We will use this as first program we run on platform you choose.

```
import std.stdio;
void main(string[] args)
{
   writeln("Hello World!");
}
```

Installation of D on Windows

Download the windows installer.

Run the downloaded executable to install the D which can be done by following the on screen instructions.

Now we can build and run a d file say helloWorld.d by switching to folder containing the file using cd and then using the following steps:

```
C:\DProgramming> DMD helloWorld.d
C:\DProgramming> helloWorld
```

We can see the following output.

```
hello world
```

C:\DProgramming is the folder, I am using to save my samples. You can change it to the folder that you have saved D programs.

Installation of D on Ubuntu/Debian

Download the debian installer.

Run the downloaded executable to install the D which can be done by following the on screen instructions.

Now we can build and run a d file say helloWorld.d by switching to folder containing the file using cd and then using the following steps

```
$ dmd helloWorld.d
$ ./helloWorld
```

We can see the following output.

```
$ hello world
```

Installation of D on Mac OS X

Download the Mac installer.

Run the downloaded executable to install the D which can be done by following the on screen instructions.

Now we can build and run a d file say helloWorld.d by switching to folder containing the file using cd and then using the following steps

```
$ dmd helloWorld.d
$ ./helloWorld
```

We can see the following output.

```
$ hello world
```

Installation of D on Fedora

Download the fedora installer.

Run the downloaded executable to install the D which can be done by following the on screen instructions.

Now we can build and run a d file say helloWorld.d by switching to folder containing the file using cd and then using the following steps:

```
$ dmd helloWorld.d
$ ./helloWorld
```

We can see the following output.

```
$ hello world
```

Installation of D on OpenSUSE

Download the OpenSUSE installer.

Run the downloaded executable to install the D which can be done by following the on screen instructions.

Now we can build and run a d file say helloWorld.d by switching to folder containing the file using cd and then using the following steps

```
$ dmd helloWorld.d
$ ./helloWorld
```

We can see the following output.

```
$ hello world
```

D IDE

We have IDE support for D in the form of plugins in most cases. This includes,

- Visual D plugin is a plugin for Visual Studio 2005-13
-
- DDT is a eclipse plugin that provides code completion, debugging with GDB.
-
- Mono-D code completion, refactoring with dmd/ldc/gdc support. It has been part of GSoC 2012.
-
- Code Blocks is a multi-platform IDE that supports D project creation, highlighting and debugging.

3. D – BASIC SYNTAX

D is quite simple to learn and let's start creating our first D program!

First D Program

Let us write a simple D program. All D files will have extension ".d". So put the following source code in a test.d file.

```
import std.stdio;


/* My first program in D */
void main(string[] args)
{
   writeln("test!");
}
```

Assuming D environment is setup correctly, lets run the programming using:

```
$ dmd test.d
$ ./test
```

We will get the following output.

```
test
```

Let us now see the basic structure of D program, so that it will be easy for you to understand basic building blocks of the D programming language.

Import in D

Libraries which are collections of reusable program parts can be made available to our project with the help of import. Here we import the standard io library which provides the basic I/O operations. writeln which is used in above program is a function in D's standard library. It is used for printing a line of text. Library contents in D are grouped into modules which is based on the types of tasks that they intend perform. The only module that this program uses is std.stdio, which handles data input and output.

Main Function

Main function is the starting of the program and it determines the order of execution and how other sections of the program should be executed.

Tokens in D

A D program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following D statement consists of four tokens:

```
writeln("test!");
```

The individual tokens are:

```
writeln
(
"test!"
)
;
```

Comments

Comments are like supporting text in your D program and they are ignored by the compiler. Multi line comment starts with /* and terminates with the characters */ as shown below:

```
/* My first program in D */
```

Single comment is written using // in the beginning of the comment.

```
// my first program in D
```

Identifiers

A D identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

D does not allow punctuation characters such as @, $, and % within identifiers. D is a **case sensitive** programming language. Thus *Manpower* and *manpower* are two different identifiers in D. Here are some examples of acceptable identifiers:

```
mohd       zara    abc    move_name   a_123
myname50   _temp   j      a23b9       retVal
```

Keywords

The following list shows few of the reserved words in D. These reserved words may not be used as constant or variable or any other identifier names.

abstract	alias	align	asm
assert	auto	body	bool
byte	case	cast	catch
char	class	const	continue
dchar	debug	default	delegate
deprecated	do	double	else
enum	export	extern	false
final	finally	float	for
foreach	function	goto	if
import	in	inout	int
interface	invariant	is	long
macro	mixin	module	new
null	out	override	package
pragma	private	protected	public
real	ref	return	scope
short	static	struct	super
switch	synchronized	template	this

throw	true	try	typeid
typeof	ubyte	uint	ulong
union	unittest	ushort	version
void	wchar	while	with

Whitespace in D

A line containing only whitespace, possibly with a comment, is known as a blank line, and a D compiler totally ignores it.

Whitespace is the term used in D to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the interpreter to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

```
local age
```

There must be at least one whitespace character (usually a space) between local and age for the interpreter to be able to distinguish them. On the other hand, in the following statement

```
int fruit = apples + oranges   //get the total fruits
```

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

4. D – VARIABLES

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in D has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because D is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types:

Type	Description
char	Typically a single octet (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

D programming language also allows to define various other types of variables such as Enumeration, Pointer, Array, Structure, Union, etc., which we will cover in subsequent chapters. For this chapter, let us study only basic variable types.

Variable Definition in D

A variable definition tells the compiler where and how much space to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid D data type including char, wchar, int, float, double, bool, or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j, and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Example

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

When a variable is declared in D, it is always set to its 'default initializer', which can be manually accessed as **T.init** where **T** is the type (ex. **int.init**). The default initializer for integer types is 0, for Booleans false, and for floating-point numbers NaN.

Variable Declaration in D

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

Example

Try the following example, where variables have been declared at the start of the program, but are defined and initialized inside the main function:

```
import std.stdio;


int a = 10, b =10;
int c;
float f;
```

```

int main ()
{
 writeln("Value of a is : ", a);
  /* variable re definition: */
  int a, b;
  int c;
  float f;

  /* Initialization */
  a = 30;
  b = 40;
  writeln("Value of a is : ", a);
  c = a + b;
  writeln("Value of c is : ", c);

  f = 70.0/3.0;
  writeln("Value of f is : ", f);
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of a is : 10
Value of a is : 30
Value of c is : 70
Value of f is : 23.3333
```

Lvalues and Rvalues in D

There are two kinds of expressions in D:

- **lvalue :** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.

- **rvalue :** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and cannot appear on the left-hand side.

The following statement is valid:

```
int g = 20;
```

But the following is not a valid statement and would generate a compile-time error:

```
10 = 20;
```

5. D – DATATYPES

In the D programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the stored bit pattern is interpreted.

The types in D can be classified as follows:

Sr. No.	Types and Description
1	**Basic Types:** They are arithmetic types and consist of the three types: (a) integer, (b) floating-point, and (c) character.
2	**Enumerated types:** They are again arithmetic types. They are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types, and (e) Function types.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section whereas other types will be covered in the upcoming chapters.

Integer Types

The following table gives lists standard integer types with their storage sizes and value ranges:

Type	Storage size	Value range
bool	1 byte	false or true

byte	1 byte	-128 to 127
ubyte	1 byte	0 to 255
int	4 bytes	-2,147,483,648 to 2,147,483,647
uint	4 bytes	0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
ushort	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
ulong	8 bytes	0 to 18446744073709551615

To get the exact size of a type or a variable, you can use the **sizeof** operator. The expression *type.(sizeof)* yields the storage size of the object or type in bytes. The following example gets the size of int type on any machine:

```
import std.stdio;


int main()
{
   writeln("Length in bytes: ", ulong.sizeof);
   return 0;
}
```

When you compile and execute the above program, it produces the following result:

```
Length in bytes: 8
```

Floating-Point Types

The following table mentions standard float-point types with storage sizes, value ranges, and their purpose:

Type	Storage size	Value range	Purpose
float	4 bytes	1.17549e-38 to 3.40282e+38	6 decimal places
double	8 bytes	2.22507e-308 to 1.79769e+308	15 decimal places
real	10 bytes	3.3621e-4932 to 1.18973e+4932	either the largest floating point type that the hardware supports, or double; whichever is larger
ifloat	4 bytes	1.17549e-38i to 3.40282e+38i	imaginary value type of float
idouble	8 bytes	2.22507e-308i to 1.79769e+308i	imaginary value type of double
ireal	10 bytes	3.3621e-4932 to 1.18973e+4932	imaginary value type of real
cfloat	8 bytes	1.17549e-38+1.17549e-38i to 3.40282e+38+3.40282e+38i	complex number type made of two floats
cdouble	16 bytes	2.22507e-308+2.22507e-308i to 1.79769e+308+1.79769e+308i	complex number type made of two doubles
creal	20 bytes	3.3621e-4932+3.3621e-4932i to 1.18973e+4932+1.18973e+4932i	complex number type made of two reals

The following example prints storage space taken by a float type and its range values:

```
import std.stdio;
int main()
{
   writeln("Length in bytes: ", float.sizeof);
   return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux:

```
Storage size for float : 4
```

Character Types

The following table lists standard character types with storage sizes and its purpose.

Type	Storage size	Purpose
char	1 byte	UTF-8 code unit
wchar	2 bytes	UTF-16 code unit
dchar	4 bytes	UTF-32 code unit and Unicode code point

The following example prints storage space taken by a char type.

```
import std.stdio;
int main()
{
   writeln("Length in bytes: ", char.sizeof);
   return 0;
}
```

When you compile and execute the above program, it produces the following result:

```
Storage size for float : 1
```

The void Type

The void type specifies that no value is available. It is used in two kinds of situations:

Sr. No.	Types and Description
1	**Function returns as void** There are various functions in D which do not return value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in D which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**

The void type may not be understood to you at this point, so let us proceed and we will cover these concepts in upcoming chapters.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**