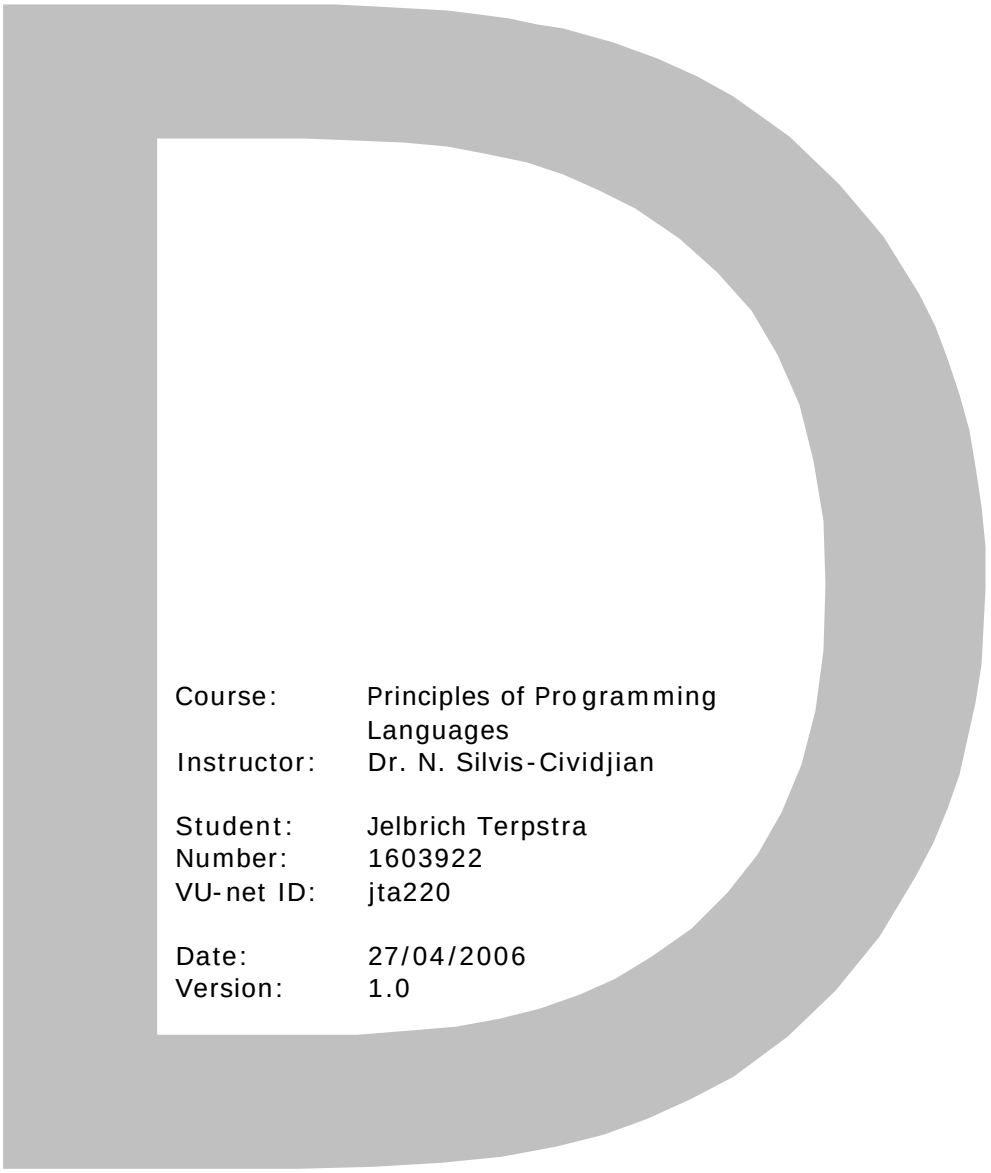
The “D” programming language



Course:	Principles of Programming Languages
Instructor:	Dr. N. Silvis-Cividjian
Student:	Jelbrich Terpstra
Number:	1603922
VU-net ID:	jta220
Date:	27/04/2006
Version:	1.0

Table of contents

History

The original name for the language was “the Mars Programming Language”. But because friends of the inventor kept calling it “D”, it was renamed “D”. The idea to name it “D” can also be explained that this language is the successor of the C-language.

A man called Walter Bright of Digital Mars is the inventor who designed D. He has done so by adding some features and reducing the complexity of the C++ syntax. He started to develop the language way back in 1999. Nowadays it is still under development and the official compiler is still in “Beta-testing” state.

Some important improvements toward C++ is that it is now possible to use automatic memory management. New is also the possibility to implement “Design by contract” as is shown in the sample code fragments chapter. It has even build-in support for unit testing.
Some other parts consist only of reengineered parts of C++, like the template syntax.

Its purpose

The C++ language needed to be backwards compatible with the old C standard. Because many new concepts were added to the language without the possibility to totally redesign things it got a bit messy. Many weaknesses of the original C design couldn’t be eliminated during the past years. D has the purpose to be the redesign many programmers where waiting for. With D it should be possible to reduce software development costs by at least 10% by adding in proven productivity, enhancing features and by adjusting language features so that common, time-consuming bugs are eliminated from the start. But D's goal is not to prevent dirty programming, but to minimize the need for it in solving routine coding tasks.

Common uses

D is well suited to writing medium to large scale million line programs with teams of developers. D is relative easy to learn, provides many capabilities to aid the programmer. It is mostly used for projects that need build-in testing and verification.

Numerical programmers like the features in D to directly support complex data types en the behaviour for “Not a number” errors and infinities.

It is not intended to be used for small application. Those small programs can better be written in a scripting or interpreted language like Python or Perl.

Language features

D is a high level applications programming language. It is a higher level language than C++, but still retains the ability to write high performance code and interface directly with the operating system hardware and API's.

Syntax and semantics

It is possible to access multidimensional arrays for things like matrix operations directly instead of using a pointer to a pointer. The D syntax for matrixes and arrays is:

```
double[6][6] matrix;   //multidimensional arrays.
matrix[1][4] = 2.0;

bit[10] x;             // array of 10 bits.
x.length               // 10, number of bits.
x.size                 // 4, bytes of storage.

int[char[]] b;         // associative array with integers, indexed by an
                          array of characters.
b[“hello”] = 3;
delete b["hello"];     // remove a particular key, not the value.

int[3] a = [ 1:2, 4 ]; // static initialization of static arrays
                       // a[0] = 0, a[1] = 2, a[2] = 4
```

Here is shown how a constructor can use the keywords this and super. An example with inheritance:

```
class Foo {
  this(int x) {        // declare constructor for Foo
    ...
  }
  this(){
    ...
  }
}
class ExtFoo : Foo {
  this(int x) {        // declare constructor for ExtFoo
    super(x);
    ...
  }
}
```

New is the possibility to include unit tests in the code. Unit tests are a series of test cases applied to a class to determine if it is working properly. An self-explaining example is:

```
class Sum {
  int add(int x, int y) { return x + y; }
  unittest {
    assert(add(3,4) == 7);
    assert(add(-2,0) == -2);
  }
}
```

Types

Besides the keywords void, bool, int, float etc. there is a imaginary float: ifloat, imaginary double (idouble) and imaginary real (ireal). The same for complex numbers; the are prefixed with an c.
Strings are a collection of characters but do not terminate with an “\0” like in C.

Memory management

Memory is normaly managed with garbage collection. But specific objects can be finalized immediately when they go out of scope. Explicit memory management is possible using the overloaded operators new and delete, as well as simply calling C's malloc and free functions directly. It is also possible to disable garbage collection for individual objects, or even for the entire program if more control over memory management is desired.

Garbage collection

Because D is a fully garbage collected language, it is for a programmer never necessary to free memory. Just allocate as much as needed, and the garbage collector will periodically return all unused memory to the pool of available memory. One advantage over manual memory management is that garbage collection only kicks in when the available memory gets tight. When memory is not tight, the program runs at full speed and does not spend any time freeing memory.

‘What differentiates it’

The inventor says “If a language can capture 90% of the power of C++ with 10% of its complexity, I argue that is a worthwhile trade off”. It implies that the language is much easier to read, and thus it should be easier to write a program. Besides more errors are captured during programming time, instead of during run-time.

D compared to C/C++

Some C/C++ features are dropped. Here is a list with dropped features.

- Multiple inheritance.
  It's a complex feature of debatable value. It's very difficult to implement in an efficient manner, and compilers are prone to many bugs in mplementing it. Nearly all the value of MI can be handled with single inheritance coupled with interfaces and aggregation. What's left does not justify the weight of MI implementation.
- Namespaces.
  An attempt to deal with the problems resulting from linking together independently developed pieces of code that have conflicting names. The idea of modules is simpler and works much better.
- Forward declarations.
  C compilers semantically only know about what has lexically preceded the current state. C++ extends this a little, in that class members can rely on forward referenced class members. D takes this to its logical conclusion, forward declarations are no longer necessary at all. Functions can be defined in a natural order rather than the typical inside-out order commonly used in C programs to avoid writing forward declarations.
- Include files.
  A major cause of slow compiles as each compilation unit must reparse enormous quantities of header files. Include files should be done as importing a symbol table.
- Distinction between . and ->.
  This distinction is really not necessary. The . operator serves just as well for pointer dereferencing.

A full list of features can be found as appendix A. It is totally self explaining and all the important features of D are inserted.

The D compiler

I used the official D compiler from www.digitalmars.com. It consist of a DMD.zip for the compiler itself and a DMC.zip for the linker and some utilities.

Installing was really easy; I only had to unzip both files to the root of my C: drive.

Sample code fragments

A simple example of a Windows dialog box.

```
/* Compile with:
 *      dmd Hello_world.d gdi32.lib Hello_world.def
 *
 * Hello_world.def contains:
 *      EXETYPE NT
 *      SUBSYSTEM WINDOWS
 */

import std.c.windows.windows;
extern (C) void gc_init();
extern (C) void gc_term();
extern (C) void _minit();
extern (C) void _moduleCtor();
extern (C) void _moduleDtor();
extern (C) void _moduleUnitTests();

extern (Windows)
int WinMain(HINSTANCE hI, HINSTANCE hPrevI, LPSTR lpCmdLine, int nCmdShow) {
    int result;

    gc_init();                  // initialize garbage collector
    _minit();                   // initialize module constructor table

    try {
        _moduleCtor();          // call module constructors
        _moduleUnitTests();     // run unit tests (optional)

        result = myWinMain(hI, hPrevI, lpCmdLine, nCmdShow);

        _moduleDtor();          // call module destructors
    } catch (Object o) {        // catch any uncaught exceptions
        MessageBoxA(null, cast(char *)o.toString(), "Error",
          MB_OK | MB_ICONEXCLAMATION);
        result = 0;             // failed
    }

    gc_term();                  // terminate garbage collector
    return result;
}

int myWinMain(HINSTANCE hI, HINSTANCE hPrevI, LPSTR lpCmdLine, int nCmdShow) {
    MessageBoxA(null,"Hello world!","Hello World example ", MB_OK);
    return 0;
}
```

A program to retrieve a webpage. It makes use of the import function to use socket specific functions.

```
import std.string, std.conv, std.stream;
import std.socket, std.socketstream;

int main(char[][] args) {
	if(args.length < 2) {
		printf("Usage:\n   get <web-page>\n");
		return 0;
	}
	char[] url = args[1];
	int i;

	i = std.string.find(url, "://");
	if(i != -1)  {
		if(icmp(url[0 .. i], "http"))
			throw new Exception("http:// expected");
	}

	i = std.string.find(url, '#');
	if(i != -1) // Remove anchor ref.
		url = url[0 .. i];

	i = std.string.find(url, '/');
	char[] domain;
	if(i == -1) {
		domain = url;
		url = "/";
	} else {
		domain = url[0 .. i];
		url = url[i .. url.length];
	}

	uint port;
	i = std.string.find(domain, ':');
	if(i == -1) {
		port = 80; // Default HTTP port.
	} else {
		port = std.conv.toUshort(domain[i + 1 .. domain.length]);
		domain = domain[0 .. i];
	}

	auto Socket sock = new TcpSocket(new InternetAddress(domain, port));
	Stream ss = new SocketStream(sock);

	if(port != 80)
		domain = domain ~ ":" ~ std.string.toString(port);
		ss.writeString("GET " ~ url ~ " HTTP/1.1\r\n"
		"Host: " ~ domain ~ "\r\n"
		"\r\n");


	// Skip HTTP header.
	char[] line;
	for(;;) {
		line = ss.readLine();
		if(!line.length)
			break;

		const char[] CONTENT_TYPE_NAME = "Content-Type: ";
		if(line.length > CONTENT_TYPE_NAME.length &&
			!icmp(CONTENT_TYPE_NAME, line[0 .. CONTENT_TYPE_NAME.length])) {
			char[] type;
			type = line[CONTENT_TYPE_NAME.length .. line.length];
			if(type.length <= 5 || icmp("text/", type[0 .. 5]))
				throw new Exception("URL is not text");
		}
```

```
        }

        print_lines:
        while(!ss.eof()) {
                line = ss.readLine();
                printf("%.*s\n", line);

                size_t iw;
                for(iw = 0; iw != line.length; iw++) {
                        if(!icmp("</html>", line[iw .. line.length]))
                                break print_lines;
                }
        }
        return 0;
}
```

Implementation example of design by contract.

```
long square_root(long x)
    in
    {
        assert(x >= 0);                  // function pre condition
    }
    out (result)
    {
        assert((result * result) == x);  // function post condition
    }
    body
    {
        return math.sqrt(x);             // function
    }
```

Index of source material

Digital Mars	http://www.digitalmars.com/d/index.html
Wiki Books	http://en.wikibooks.org/wiki/Programming:D

Appendices

Appendix A: Comparison Sheet

This table is a quick and rough comparison of various features of D with other languages it is frequently compared with. While many capabilities are available with standard libraries, this table is for features built in to the core language itself. Rationale. Only official standardized features are considered, not proposed features, betas, or extensions. And, like all language comparisons, it is biased in terms of what features are mentioned, omitted, and my interpretation of those features.

Feature	**D**	**C**	**C++**	**C#**	**Java**
Garbage Collection	Yes	-	-	Yes	Yes
Functions					
Function delegates	Yes	-	-	Yes	-
Function overloading	Yes	-	Yes	Yes	Yes
Out function parameters	Yes	Yes	Yes	Yes	-
Nested functions	Yes	-	-	-	-
Function literals	Yes	-	-	-	-
Dynamic closures	Yes	-	-	-	-
Typesafe variadic arguments	Yes	-	-	Yes	Yes
Arrays					
Lightweight arrays	Yes	Yes	Yes	-	-
Resizeable arrays	Yes	-	-	-	-
Built-in strings	Yes	-	-	Yes	Yes
Array slicing	Yes	-	-	-	-
Array bounds checking	Yes	-	-	Yes	Yes
Associative arrays	Yes	-	-	-	-
Strong typedefs	Yes	-	-	-	-
String switches	Yes	-	-	Yes	-
Aliases	Yes	Yes	Yes	-	-
OOP					
Object Oriented	Yes	-	Yes	Yes	Yes
Multiple Inheritance	-	-	Yes	-	-
Interfaces	Yes	-	Yes	Yes	Yes
Operator overloading	Yes	-	Yes	Yes	-
Modules	Yes	-	Yes	Yes	Yes
Dynamic class loading	-	-	-	Yes	Yes
Nested classes	Yes	Yes	Yes	Yes	Yes
Inner (adaptor) classes	Yes	-	-	-	Yes
Covariant return types	Yes	-	Yes	-	Yes
Properties	Yes	-	-	Yes	-
Performance					
Inline assembler	Yes	Yes	Yes	-	-
Direct access to hardware	Yes	Yes	Yes	-	-
Lightweight objects	Yes	Yes	Yes	Yes	-
Explicit memory allocation control	Yes	Yes	Yes	-	-
Independent of VM	Yes	Yes	Yes	-	-
Direct native code gen	Yes	Yes	Yes	-	-

Feature	D	C	C++	C#	Java
Generic Programming					
Class Templates	Yes	-	Yes	Yes	Yes
Function Templates	Yes	-	Yes	-	Yes
Implicit Function Template Instantiation	-	-	Yes	-	-
Partial and Explicit Specialization	Yes	-	Yes	-	-
Value Template Parameters	Yes	-	Yes	-	-
Template Template Parameters	Yes	-	Yes	-	-
Mixins	Yes	-	-	-	-
static if	Yes	-	-	-	-
is expressions	Yes	-	-	-	-
typeof	Yes	-	-	Yes	-
foreach	Yes	-	-	Yes	Yes
Implicit Type Inference	Yes	-	-	-	-
Reliability					
Contract Programming	Yes	-	-	-	-
Unit testing	Yes	-	-	-	-
Static construction order	Yes	-	-	Yes	Yes
Guaranteed initialization	Yes	-	-	Yes	Yes
RAII (automatic destructors)	Yes	-	Yes	Yes	-
Exception handling	Yes	-	Yes	Yes	Yes
Scope guards	Yes	-	-	-	-
try-catch-finally blocks	Yes	-	-	Yes	Yes
Thread synchronization primitives	Yes	-	-	Yes	Yes
Compatibility					
C-style syntax	Yes	Yes	Yes	Yes	Yes
Enumerated types	Yes	Yes	Yes	Yes	Yes
Support all C types	Yes	Yes	-	-	-
80 bit floating point	Yes	Yes	Yes	-	-
Complex and Imaginary	Yes	Yes	-	-	-
Direct access to C	Yes	Yes	Yes	-	-
Use existing debuggers	Yes	Yes	Yes	-	-
Struct member alignment control	Yes	-	-	-	-
Generates standard object files	Yes	Yes	Yes	-	-
Macro text preprocessor	-	Yes	Yes	-	-
Other					
Conditional compilation	Yes	Yes	Yes	Yes	-
Unicode source text	Yes	Yes	Yes	Yes	Yes
Documentation comments	Yes	-	-	Yes	Yes