EBook Gratis

APRENDIZAJE

D Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#d

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: d-language

It is an unofficial and free D Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official D Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con D Language

Observaciones

D es un lenguaje de programación de sistemas con sintaxis tipo C y escritura estática. Combina eficiencia, control y potencia de modelado con seguridad y productividad del programador.

Versiones

Versión	Registro de cambios	Fecha de lanzamiento
re	http://www.digitalmars.com/d/1.0/changelog.html	2007-01-23
D2	https://dlang.org/changelog/2.000.html	2007-06-17

Examples

Instalación o configuración

El compilador estándar DMD del lenguaje de programación D puede ejecutarse en todas las plataformas principales. Para instalar DMD ver aquí . Para instalar por línea de comandos, puede ejecutar el comando (que se encuentra en el sitio web de D):

```
curl -fsS https://dlang.org/install.sh | bash -s dmd
```

Gestores de paquetes

Arco de linux

```
pacman -S dlang
```

Chocolatey

```
choco install dmd
```

Gentoo

```
layman -f -a dlang
```

Homebrew OSX

```
brew install dmd
```

Debian / Ubuntu

La instalación en las distribuciones Debian / Ubuntu necesita que el repositorio APT se agregue a la lista de fuentes.

```
wget http://master.dl.sourceforge.net/project/d-apt/files/d-apt.list -O
/etc/apt/sources.list.d/d-apt.list
wget -qO - https://dlang.org/d-keyring.gpg | sudo apt-key add -
apt-get update
apt-get install dmd-bin
```

Otros compiladores

LDC es un compilador D que utiliza el frontend oficial del compilador DMD y LLVM como backend.

GDC es un compilador D que usa el backend de GCC para generar código.

IDEs

Para facilitarle la vida, es posible que también desee instalar un IDE (Entorno de desarrollo integrado). El Wiki de D-Language tiene una lista de IDE y complementos disponibles para todas las plataformas aquí .

Hola Mundo

```
import std.stdio;

// Let's get going!
void main()
{
    writeln("Hello World!");
}
```

Para compilar y ejecutar, guarde este texto como un archivo llamado `main.d` Desde la línea de comandos, ejecute `dmd main.d` para compilar el programa. Finalmente, ejecute `./main` para ejecutar el programa en un shell bash o puede hacer clic en el ejecutable en windows.

¡Hola Mundo!

Para crear el programa de impresión clásico "Hola, mundo", cree un archivo llamado `hello.d` con un editor de texto que contenga el siguiente código:

```
import std.stdio;

void main() {
    writeln("Hello, World!");    //writeln() automatically adds a newline (\n) to the output
}
```

Explicación:

```
import std.stdio
```

Esta línea le dice al compilador que se usarán las funciones definidas en el módulo de la biblioteca estándar `std.stdio` . Se puede importar cualquier módulo, siempre que el compilador sepa dónde buscarlos. Muchas funciones se proporcionan como parte de la biblioteca estándar masiva de D.

```
void main() {
```

Esta línea declara la función `main` , volviendo `void` . Tenga en cuenta que a diferencia de C y C ++, D permite que main sea de tipo `void` . La función `main` es especial ya que es el punto de entrada del programa, es decir, aquí es donde comienza la ejecución del programa. Algunas notas sobre las funciones en general:

- El nombre de una función puede ser cualquier cosa que comience con una letra y se componga de letras, dígitos y guiones bajos.
- Los parámetros esperados serán una lista separada por comas de nombres de variables y sus tipos de datos.
- El valor que se espera que la función devuelva puede ser cualquier tipo de datos existente, y debe coincidir con el tipo de expresión utilizada en la declaración de retorno dentro de la función.

Las llaves `{ … }` se utilizan en pares para indicar dónde comienza y termina un bloque de código. Se pueden usar de muchas maneras, pero en este caso indican dónde comienza y termina la función.

```
writeln("Hello, World!");
```

`writeln` es una función declarada en `std.stdio` que escribe sus documentos en `stdout` . En este caso, su argumento es `"Hello, World"` , que se escribirá en la consola. Se pueden usar varios caracteres de formato, similares a los utilizados por `printf` de C, como `\n` , `\r` , etc.

Cada declaración debe ser terminada por un punto y coma.

Los comentarios se utilizan para indicar algo a la persona que lee el código y el compilador los trata como un espacio en blanco. En el código anterior, este es un comentario:

```
//writeln() automatically adds a newline (\n) to the output
```

Estas son piezas de código que son ignoradas por el compilador. Hay tres formas diferentes de comentar en D:

1. `//` - Comenta todo el texto en la misma línea, después de `//`
2. `/* comment text */` - Estos son útiles para comentarios multilínea
3. `/+ comment text +` - Estos son también comentarios multilínea

Son muy útiles para transmitir lo que una función / pieza de código está haciendo a un desarrollador.

Compilando y ejecutando el programa

Para ejecutar este programa, el código debe ser compilado en un ejecutable. Esto se puede hacer con la ayuda del compilador.

Para compilar usando DMD, el compilador de referencia D, abra un terminal, navegue a la ubicación del archivo `hello.d` que creó y luego ejecute:

```
dmd hello.d
```

Si no se encuentran errores, el compilador generará un ejecutable con el nombre de su archivo fuente. Esto ahora se puede ejecutar escribiendo

```
./hello
```

Una vez ejecutado, el programa imprimirá `Hello, World!` , seguido de una nueva línea.

Leer valores de una cadena

```
import std.format;

void main() {
    string s = "Name Surname 18";
    string name, surname;
    int age;
    formattedRead(s, "%s %s %s", &name, &surname, &age);
    // %s selects a format based on the corresponding argument's type
}
```

La documentación oficial para las cadenas de formato se puede encontrar en: https://dlang.org/phobos/std_format.html#std.format

Lea Empezando con D Language en línea: https://riptutorial.com/es/d/topic/1036/empezando-con-d-language

Capítulo 2: Arrays y segmentos dinámicos

Sintaxis

- <tipo> [] <nombre>;

Observaciones

Los segmentos generan una nueva vista en la memoria existente. No crean una nueva copia. Si ya no hay ninguna división que contenga una referencia a esa memoria, o una parte de ella, el recolector de basura la liberará.

Usando segmentos, es posible escribir código muy eficiente para, por ejemplo, analizadores que solo operan en un bloque de memoria y solo cortan las partes en las que realmente necesitan trabajar, sin necesidad de asignar nuevos bloques de memoria.

Examples

Declaración e inicialización.

```
import std.stdio;

void main() {
    int[] arr = [1, 2, 3, 4];

    writeln(arr.length); // 4
    writeln(arr[2]); // 3

    // type inference still works
    auto arr2 = [1, 2, 3, 4];
    writeln(typeof(arr2).stringof); // int[]
}
```

Operaciones de matriz

```
import std.stdio;

void main() {
    int[] arr = [1, 2, 3];

    // concatenate
    arr ~= 4;
    writeln(arr); // [1, 2, 3, 4]

    // per element operations
    arr[] += 10
    writeln(arr); // [11, 12, 13, 14]
}
```

Rebanadas

```
import std.stdio;

void main() {
    int[] arr = [1, 2, 3, 4, 5];

    auto arr2 = arr[1..$ - 1]; // .. is the slice syntax, $ represents the length of the array
    writeln(arr2); // [2, 3, 4]

    arr2[0] = 42;
    writeln(arr[1]); // 42
}
```

Lea Arrays y segmentos dinámicos en línea: https://riptutorial.com/es/d/topic/2445/arrays-y-segmentos-dinamicos

Capítulo 3: Bucles

Sintaxis

- para (<inicializador>; <condición de bucle>; <declaración de bucle>) {<declaraciones>}
- while (<condición>) {<declaraciones>}
- do {<statements>} while (<condition>);
- foreach (<el>, <collection>)
- foreach_reverse (<el>, <collection>)

Observaciones

- `for` bucle en *Programación en D* , especificación
- `while` bucle en *Programación en D* , especificación
- `do while` bucle while *Programación en D* , especificación
- `foreach` en *Programación en D* , opApply , especificación

Puedes jugar con loops y foreach en línea.

Examples

En bucle

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    for (int i = 0; i < arr.length; i++)
    {
        arr[i] *= 2;
    }
    writeln(arr); // [2, 6, 8]
}
```

Mientras bucle

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    int i = 0;
    while (i < arr.length)
    {
        arr[i++] *= 2;
    }
    writeln(arr); // [2, 6, 8]
}
```

hacer mientras

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    int i = 0;
    assert(arr.length > 0, "Array must contain at least one element");
    do
    {
        arr[i++] *= 2;
    } while (i < arr.length);
    writeln(arr); // [2, 6, 8]
}
```

Para cada

Foreach permite una manera menos propensa a errores y mejor legible para iterar colecciones. El atributo `ref` se puede usar si queremos modificar directamente el elemento iterado.

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    foreach (ref el; arr)
    {
        el *= 2;
    }
    writeln(arr); // [2, 6, 8]
}
```

También se puede acceder al índice de la iteración:

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    foreach (i, el; arr)
    {
        arr[i] = el * 2;
    }
    writeln(arr); // [2, 6, 8]
}
```

La iteración en orden inverso también es posible:

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    int i = 0;
    foreach_reverse (ref el; arr)
    {
        el += i++; // 4 is incremented by 0, 3 by 1, and 1 by 2
    }
```

```
    writeln(arr); // [3, 4, 4]
}
```

Romper, continuar y etiquetas

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4, 5];
    foreach (i, el; arr)
    {
        if (i == 0)
            continue; // continue with the next iteration
        arr[i] *= 2;
        if (i == 2)
            break; // stop the loop iteration
    }
    writeln(arr); // [1, 6, 8, 5]
}
```

Las etiquetas también se pueden usar para `break` o `continue` dentro de bucles anidados.

```
void main()
{
    import std.stdio : writeln;
    int[] arr = [1, 3, 4];
    outer: foreach (j; 0..10) // iterates with j=0 and j=1
        foreach (i, el; arr)
        {
            arr[i] *= 2;
            if (j == 1)
                break outer; // stop the loop iteration
    }
    writeln(arr); // [4, 6, 8] (only 1 reaches the second iteration)
}
```

Lea Bucles en línea: https://riptutorial.com/es/d/topic/4696/bucles

Capítulo 4: Estructuras

Examples

Definiendo un nuevo Struct

Para definir la estructura llamada Persona con una variable de tipo entero edad, altura de variable de tipo entero y variable de tipo flotante ageXHeight:

```
struct Person {
    int age;
    int height;
    float ageXHeight;
}
```

Generalmente:

```
struct structName {
    /+ values go here +/
}
```

Constructores Struct

En D podemos usar constructores para inicializar estructuras como una clase. Para definir una construcción para la estructura declarada en el ejemplo anterior podemos escribir:

```
struct Person {
    this(int age, int height) {
        this.age = age;
        this.height = height;
        this.ageXHeight = cast(float)age * height;
    }
}

auto person = Person(18, 180);
```

Lea Estructuras en línea: https://riptutorial.com/es/d/topic/4075/estructuras

Capítulo 5: Evaluación de la función de tiempo de compilación (CTFE)

Observaciones

CTFE es un mecanismo que permite al compilador ejecutar funciones en tiempo de compilación. No hay un conjunto especial de lenguaje D necesario para usar esta función; cuando una función solo depende del tiempo de compilación de valores conocidos, el compilador D puede decidir interpretarlo durante la compilación.

También puedes jugar interactivamente con CTFE.

Examples

Evaluar una función en tiempo de compilación

```
long fib(long n)
{
    return n < 2 ? n : fib(n - 1) + fib(n - 2);
}

struct FibStruct(int n) { // Remarks: n is a template
    ubyte[fib(n)] data;
}

void main()
{
    import std.stdio : writeln;
    enum f10 = fib(10); // execute the function at compile-time
    pragma(msg, f10); // will print 55 during compile-time
    writeln(f10); // print 55 during runtime
    pragma(msg, FibStruct!11.sizeof); // The size of the struct is 89
}
```

Lea Evaluación de la función de tiempo de compilación (CTFE) en línea: https://riptutorial.com/es/d/topic/4694/evaluacion-de-la-funcion-de-tiempo-de-compilacion--ctfe-

Capítulo 6: Examen de la unidad

Sintaxis

- unittest {...} - un bloque que solo se ejecuta en modo "unittesting"
- assert (<expresión que se evalúa como booleano>, <mensaje de error opcional>)

Examples

Bloques unittest

Las pruebas son una excelente manera de garantizar aplicaciones estables y libres de errores. Sirven como una documentación interactiva y permiten modificar el código sin temor a romper la funcionalidad. D proporciona una sintaxis conveniente y nativa para el bloque `unittest` como parte del lenguaje D. Los bloques `unittest` en cualquier lugar de un módulo D pueden usarse para probar la funcionalidad del código fuente.

```
/**
Yields the sign of a number.
Params:
    n = number which should be used to check the sign
Returns:
    1 for positive n, -1 for negative and 0 for 0.
*/
T sgn(T)(T n)
{
    if (n == 0)
        return 0;
    return (n > 0) ? 1 : -1;
}

// this block will only be executed with -unittest
// it will be removed from the executable otherwise
unittest
{
    // go ahead and make assumptions about your function
    assert(sgn(10)  ==  1);
    assert(sgn(1)   ==  1);
    assert(sgn(-1)  == -1);
    assert(sgn(-10) == -1);
}
```

Ejecutando test unit

Si el indicador `-unittest` se pasa al compilador D, ejecutará todos los bloques de `-unittest` . A menudo es útil dejar que el compilador genere una función `main` aplastada. Usando el compilador y ejecute wrapper `rdmd` , probar su programa D es tan fácil como:

```
rdmd -main -unittest yourcode.d
```

Por supuesto, también puede dividir este proceso en dos pasos si desea:

```
dmd -main -unittest yourcode.d
./yourcode
```

Para proyectos de `dub` , compilar todos los archivos y ejecutar sus bloques de prueba de unidad se puede hacer de manera conveniente con

```
dub test
```

Consejo profesional: define `tdmd` como alias de shell para guardar las propinas.

```
alias tdmd="rdmd -main -unittest"
```

y luego prueba tus archivos con:

```
tdmd yourcode.d
```

Prueba de unidad anotada

Para el código con plantilla, a menudo es útil verificar que los atributos de la función (por ejemplo, `@nogc` se deducen correctamente. Para garantizar esto para una prueba específica y, por lo tanto, escribir todo el test de unidad puede anotarse)

```
@safe @nogc pure nothrow unittest
{
    import std.math;
    assert(exp(0)  ==  1);
    assert(log(1)  ==  0);
}
```

Tenga en cuenta que, por supuesto, en D, cada bloque se puede anotar con atributos y los compiladores, por supuesto, verifican que sean correctos. Entonces, por ejemplo, lo siguiente sería similar al ejemplo anterior:

```
unittest
{
    import std.math;
    @safe {
        assert(exp(0)  ==  1);
        assert(log(1)  ==  0);
    }
}
```

Lea Examen de la unidad en línea: https://riptutorial.com/es/d/topic/6201/examen-de-la-unidad

Capítulo 7: Gamas

Observaciones

Si el compilador encuentra un foreach

```
foreach (element; range) {
```

Es reescrito internamente similar al siguiente:

```
for (auto it = range; !it.empty; it.popFront()) {
    auto element = it.front;
    ...
}
```

Cualquier objeto que cumpla con la interfaz anterior se denomina rango de entrada y, por lo tanto, es un tipo que se puede iterar sobre:

```
struct InputRange {
    @property bool empty();
    @property T front();
    void popFront();
}
```

Examples

Las cuerdas y matrices son rangos.

```
import std.stdio;

void main() {
    auto s = "hello world";
    auto a = [1, 2, 3, 4];

    foreach (c; s) {
        write(c, "!"); // h!e!l!l!o! !w!o!r!l!d!
    }
    writeln();

    foreach (x; a) {
        write(x * x, ", "); // 1, 4, 9, 16,
    }
}
```

Haciendo un nuevo tipo de rango de entrada

El concepto `InputRange` tiene tres funciones, ejemplo:

```
struct InputRange(T) {
```

```
    @property bool empty();
    @property T front();
    void popFront();
}
```

En definitiva, una forma de

1. comprueba si el rango está vacío
2. obtener el elemento actual
3. pasar al siguiente elemento

Para hacer nuestro propio tipo un `InputRange` , debemos implementar estas tres funciones. Echemos un vistazo a la secuencia infinita de cuadrados.

```
struct SquaresRange {
    int cur = 1;

    @property bool empty() {
        return false;
    }

    @property int front() {
        return cur^^2;
    }

    void popFront() {
        cur++;
    }
}
```

Vea la gira D para ver un ejemplo con Fibonacci.

Lea Gamas en línea: https://riptutorial.com/es/d/topic/3106/gamas

Capítulo 8: Guardias de alcance

Sintaxis

- alcance (salida): las declaraciones se llaman sin importar cómo se salió del bloque actual
- ámbito (éxito): las declaraciones se invocan cuando el bloque actual se cerró normalmente
- ámbito (error): se invocan declaraciones cuando se sale del bloque actual mediante el lanzamiento de excepciones

Observaciones

El uso de guardas de alcance hace que el código sea mucho más limpio y permite colocar la asignación de recursos y limpiar el código uno al lado del otro. Estos pequeños ayudantes también mejoran la seguridad porque se aseguran de que cierto código de limpieza siempre se llame independiente de qué rutas se toman realmente en tiempo de ejecución.

La función de alcance D reemplaza efectivamente el lenguaje RAII usado en C ++ que a menudo conduce a objetos de guardas de alcance especiales para recursos especiales.

Las guardas de alcance se llaman en el orden inverso al que están definidas.

Juega con guardias de alcance o ve un extenso tutorial .

Examples

Coloque la asignación y el código de limpieza uno al lado del otro

Las guardas de alcance permiten ejecutar sentencias en ciertas condiciones si se deja el bloqueo actual.

```
import core.stdc.stdlib;

void main() {
    int* p = cast(int*)malloc(int.sizeof);
    scope(exit) free(p);
}
```

Múltiples ámbitos anidados

```
import std.stdio;

void main() {
    writeln("<html>");
    scope(exit) writeln("</html>");
    {
        writeln("\t<head>");
        scope(exit) writeln("\t</head>");
```

```
        "\t\t<title>%s</title>".writefln("Hello");
    } // the scope(exit) on the previous line is executed here

    writeln("\t<body>");
    scope(exit) writeln("\t</body>");

    writeln("\t\t<h1>Hello World!</h1>");
}
```

huellas dactilares

```
<html>
    <head>
        <title>Hello</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Lea Guardias de alcance en línea: https://riptutorial.com/es/d/topic/4343/guardias-de-alcance

Capítulo 9: Importaciones y módulos.

Sintaxis

- módulo my.package;
- importar mi.paquete;
- importar my.package: function;
- import fancyName = mypackage;
- importar my.package: fancyFunctionName = function;

Observaciones

Los módulos proporcionan automáticamente un ámbito de espacio de nombres para sus contenidos. Los módulos se asemejan superficialmente a las clases, pero difieren en eso:

- Solo hay una instancia de cada módulo, y se asigna de forma estática.
- No hay mesa virtual.
- Los módulos no se heredan, no tienen super módulos, etc.
- Sólo un módulo por archivo.
- Los símbolos del módulo pueden ser importados.
- Los módulos siempre se compilan a nivel global y no se ven afectados por los atributos circundantes u otros modificadores.
- Los módulos se pueden agrupar en jerarquías llamadas paquetes.

Los módulos ofrecen varias garantías:

- El orden en que se importan los módulos no afecta a la semántica.
- La semántica de un módulo no se ve afectada por lo que lo importa.
- Si un módulo C importa los módulos A y B, cualquier modificación a B no cambiará de manera silenciosa el código en C que depende de A.

Examples

Importaciones globales

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

Las importaciones múltiples se pueden especificar en la misma línea, separadas por una `comma` o en una nueva línea.

```
import std.stdio, std.math;
```

```
import std.datetime;
void main()
{
    writeln("2^4: ", pow(2, 4));
    writeln("Current time: ", Clock.currTime());
}
```

Importaciones selectivas

Las importaciones selectivas pueden ayudar a limpiar el espacio de nombres y acelerar el tiempo de compilación aún más, porque el compilador solo necesita analizar las funciones específicas seleccionadas.

```
import std.stdio: writeln;
void main()
{
    writeln("Hello world");
}
```

Importaciones locales

También puede importar símbolos en cualquier ámbito, la importación solo se buscará cuando el ámbito sea necesario (es decir, compilado) y los nombres importados solo se expondrán en el ámbito importado. Más comúnmente, el alcance de las importaciones locales son funciones, estructuras y clases.

```
void main()
{
    import std.stdio: writeln;
    writeln("Hello world");
}
// writeln isn't defined here
```

Importaciones publicas

Los módulos pueden ser expuestos a otros módulos con `public imports` .

```
public import std.math;
// only exports the symbol 'pow'
public import std.math : pow;
```

Importaciones renombradas

Se puede dar un nombre local para una importación, a través del cual todas las referencias a los símbolos del módulo se deben calificar con:

```
import io = std.stdio;
void main()
{
    io.writeln("Hello world");
```

```
    std.stdio.writeln("hello!"); // error, std is undefined
    writeln("hello!");           // error, writeln is undefined
}
```

Las importaciones renombradas son útiles cuando se trata de nombres de importación muy largos.

Importaciones renombradas y selectivas.

Las importaciones selectivas también pueden ser renombradas.

```
void main()
{
    import std.stdio : fooln = writeln;
    fooln("Hello world");
}
```

Declaración del módulo

Los módulos tienen una correspondencia de uno a uno con los archivos de origen. El nombre del módulo es, de forma predeterminada, el nombre del archivo con la ruta y la extensión eliminadas, y se puede establecer explícitamente con la declaración del módulo. La `ModuleDeclaration` establece el nombre del módulo y el paquete al que pertenece. Si está ausente, el nombre del módulo se toma como el mismo nombre (eliminado de la ruta y la extensión) del nombre del archivo fuente.

```
module my.fancy.module;
```

Lea Importaciones y módulos. en línea: https://riptutorial.com/es/d/topic/4344/importaciones-y-modulos-

Capítulo 10: Instrumentos de cuerda

Observaciones

- Las cuerdas en D son inmutables; utilizar `.dup` hacer una mutable `char` matriz si desea editar en el lugar.

Examples

Invertir una cadena

`string` se define como `alias string = immutable(char)[];` : así que necesitas usar `dup` para hacer una matriz de caracteres mutable, antes de que pueda revertirse:

```
import std.stdio;
import std.string;

int main() {

    string x = "Hello world!";
    char[] x_rev = x.dup.reverse;

    writeln(x_rev); // !dlrow olleH

    return 0;

}
```

Prueba de una cadena vacía o nula

Cuerda vacía

La cadena vacía no es nula pero tiene longitud cero:

```
string emptyString = "";
// an empty string is not null...
assert(emptyString !is null);

// ... but it has zero lenght
assert(emptyString.length == 0);
```

Cadena nula

```
string nullString = null;
```

una cadena nula es nula (De Lapalisse)

```
assert(nullString is null);
```

pero, a diferencia de C #, leer la longitud de una cadena nula no genera error:

```
assert(nullString.length == 0);
assert(nullString.empty);
```

Prueba de vacío o nulo

```
if (emptyOrNullString.length == 0) {
}

// or
if (emptyOrNullString.length) {
}

// or
import std.array;
if (emptyOrNullString.empty) {
}
```

Prueba para nulo

```
if (nullString is null) {
}
```

Referencias

- ¿Cuál es la forma correcta de probar una cadena vacía?
- ¿D tiene la cadena de C #? ¿Vacía?

Convertir cadena a ubyte [] y viceversa

Cadena a `ubyte[]` inmutable `ubyte[]`

```
string s = "unogatto";
immutable(ubyte[]) ustr = cast(immutable(ubyte)[])s;

assert(typeof(ustr).stringof == "immutable(ubyte[])");
assert(ustr.length == 8);
assert(ustr[0] == 0x75); //u
assert(ustr[1] == 0x6e); //n
```

```
assert(ustr[2] == 0x6f); //o
assert(ustr[3] == 0x67); //g
assert(ustr[7] == 0x6f); //o
```

Cadena a `ubyte[]`

```
string s = "unogatto";
ubyte[] mustr = cast(ubyte[])s;

assert(typeof(mustr).stringof == "ubyte[]");

assert(mustr.length == 8);
assert(mustr[0] == 0x75);
assert(mustr[1] == 0x6e);
assert(mustr[2] == 0x6f);
assert(mustr[3] == 0x67);
assert(mustr[7] == 0x6f);
```

`ubyte[]` a cadena

```
ubyte[] stream = [ 0x75, 0x6e, 0x6f, 0x67];
string us  = cast(string)stream;
assert(us == "unog");
```

Referencias

- Foro DLang

Lea Instrumentos de cuerda en línea: https://riptutorial.com/es/d/topic/5760/instrumentos-de-cuerda

Capítulo 11: Las clases

Sintaxis

- la clase Foo {} // hereda de Object
- clase Bar: Foo {} // Bar también es un Foo
- Foo f = nuevo Foo (); // instanciar nuevos objetos en el montón

Observaciones

Vea la especificación , explore un capítulo de un libro sobre clases , herencia y juegue de manera interactiva .

Examples

Herencia

```
class Animal
{
    abstract int maxSize(); // must be implemented by sub-class
    final float maxSizeInMeters() // can't be overridden by base class
    {
        return maxSize() / 100.0;
    }
}

class Lion: Animal
{
    override int maxSize() { return 350; }
}

void main()
{
    import std.stdio : writeln;
    auto l = new Lion();
    assert(l.maxSizeInMeters() == 3.5);

    writeln(l.maxSizeInMeters()); // 3.5
}
```

Instanciación

```
class Lion
{
    private double weight; // only accessible with-in class

    this(double weight)
    {
        this.weight = weight;
    }
```

```
    double weightInPounds() const @property // const guarantees no modifications
    // @property functions are treated as fields
    {
        return weight * 2.204;
    }
}

void main()
{
    import std.stdio : writeln;
    auto l = new Lion(100);
    assert(l.weightInPounds == 220.4);

    writeln(l.weightInPounds); // 220.4
}
```

Lea Las clases en línea: https://riptutorial.com/es/d/topic/4695/las-clases

Capítulo 12: Los contratos

Observaciones

Las afirmaciones se optimizarán en una versión de lanzamiento.

Examples

Contratos de funciones

Los contratos de funciones le permiten al programador verificar las inconsistencias. Las incoherencias incluyen parámetros no válidos, verifica el valor de retorno correcto o un estado no válido del objeto.

Las comprobaciones pueden realizarse antes y después de que se ejecute el cuerpo de la función o el método.

```
void printNotGreaterThan42(uint number)
in {
    assert(number < 42);
}
body {
    import std.stdio : writeln;
    writeln(number);
}
```

Las afirmaciones se optimizarán en una versión de lanzamiento.

Contratos de funciones

Por ejemplo, si se invoca un método, es posible que el estado del objeto no permita que se llame a un método con parámetros específicos o que no lo sea.

```
class OlderThanEighteen {
    uint age;

    final void driveCar()
    in {
         assert(age >= 18); // variable must be in range
    }
    body {
         // step on the gas
    }
}
```

Lea Los contratos en línea: https://riptutorial.com/es/d/topic/6830/los-contratos

Capítulo 13: Matrices asociativas

Examples

Uso estándar

```
int[string] wordCount(string[] wordList) {
    int[string] words;
    foreach (word; wordList) {
        words[word]++;
    }
    return words;
}

void main() {
    int[string] count = wordCount(["hello", "world", "I", "say", "hello"]);
    foreach (key; count.keys) {
        writefln("%s: %s", key, count[key]);
    }
    // hello: 2
    // world: 1
    // I: 1
    // say: 1

    // note: the order in the foreach is unspecified
}
```

Literales

```
int[string] aa0 = ["x": 5, "y": 6];  //int values, string keys
auto aa1 = ["x": 5.0, "y": 6.0];  // double values, string keys
string[int] aa2 = [10: "A", 11: "B"];  //string values, int keys
```

Añadir pares clave-valor

```
int[string] aa = ["x": 5, "y": 6];
// The value can be set by its key:
aa["x"] = 7;
assert(aa["x"] == 7);
// if the key does not exist will be added
aa["z"] = 8;
assert(aa["z"] == 8);
```

Eliminar pares clave-valor

Supongamos una matriz asociativa `aa` :

```
int[string] aa = ["x": 5, "y": 6];
```

Los elementos se pueden eliminar utilizando `.remove()` , si se eliminarán las salidas clave y si se

`remove` devolverá `true`

```
assert(aa.remove("x"));
```

si la clave dada no existe, `remove` no hace nada y devuelve `false` :

```
assert(!aa.remove("z"));
```

Compruebe si existe una clave

```
int[string] numbers = ["a" : 10, "b" : 20];

assert("a" in numbers);
assert("b" in numbers);
assert("c" in numbers);
```

Lea Matrices asociativas en línea: https://riptutorial.com/es/d/topic/3159/matrices-asociativas

Capítulo 14: Memoria y punteros

Sintaxis

- & <variable> - acceso por referencia (= lleva el puntero a los datos de la variable)
- * <variable> - operador de deferencia (= obtiene el objeto de datos de un puntero)
- <tipo> * - tipo de datos que apunta a <tipo> (por ejemplo, `int *)

Examples

Punteros

D es un lenguaje de programación del sistema y, por lo tanto, le permite administrar y desordenar manualmente su memoria. Sin embargo, D utiliza un recolector de basura por defecto para liberar memoria no utilizada.

D proporciona tipos de punteros T * como en C:

```
void main()
{
    int a;
    int* b = &a; // b contains address of a
    auto c = &a; // c is int* and contains address of a

    import std.stdio : writeln;
    writeln("a ", a);
    writeln("b ", b);
    writeln("c ", c);
}
```

Asignación en el montón

Un nuevo bloque de memoria en el montón se asigna usando la `new` expresión, que devuelve un puntero a la memoria administrada:

```
void main()
{
    int* a = new int;
    *a = 42; // dereferencing
    import std.stdio : writeln;
    writeln("a: ", *a);
}
```

@safe D

Tan pronto como la memoria a la que hace referencia no esté referenciada por ninguna otra variable en el programa, el recolector de basura liberará su memoria.

D también permite la aritmética de punteros, excepto en el código que está marcado como `@safe` .

```
void safeFun() @safe
{
    writeln("Hello World");
    // allocating memory with the GC is safe too
    int* p = new int;
}

void unsafeFun()
{
    int* p = new int;
    int* fiddling = p + 5;
}

void main()
{
    safeFun();
    unsafeFun();
}
```

Para obtener más información sobre SafeD, consulte el artículo del equipo de diseño de D.

Lea Memoria y punteros en línea: https://riptutorial.com/es/d/topic/6374/memoria-y-punteros

Capítulo 15: Plantillas

Sintaxis

- identificador de plantilla (TemplateParameterList) {...}
- identificador de estructura (TemplateParameterList) {...}
- identificador de clase (TemplateParameterList) {...}
- Identificador de tipo de retorno (TemplateParameterList) (ParameterList) {...}
- identificador! (TemplateInvocationList)

Examples

Funcionar con una plantilla

```
import std.stdio;

T min(T)(in T arg1, in T arg2) {
    return arg1 < arg2 ? arg1 : arg2;
}

void main() {
    //Automatic type inference
    writeln(min(1, 2));

    //Explicit type
    writeln(min!(ubyte)(1, 2));

    //With single type, the parenthesis might be ommited
    writeln(min!ubyte(1, 2));
}
```

modelo

Se puede introducir una `template` con `template` . Puede contener funciones y clases y otras construcciones.

```
template StaticArray(Type, size_t Length) {
     class StaticArray {
          Type content[Length];

          size_t myLength() {
            return getLength(this);
          }
     }

     private size_t getLength(StaticArray arr) {
          return Length;
     }
}

void main() {
```

```
    StaticArray!(int, 5) arr5 = new StaticArray!(int, 5);
    import std.stdio;
    writeln(arr5.myLength());
}
```

Lea Plantillas en línea: https://riptutorial.com/es/d/topic/3892/plantillas

Capítulo 16: Rasgos

Sintaxis

- __traits (TraitsKeyword, TraitsArguments ...)

Examples

Iterando sobre los miembros de una estructura.

```
import std.stdio;

struct A {
    int b;
    void c();
    string d;
};

void main() {
    // The following foreach is unrolled in compile time
    foreach(name; __traits(allMembers, A)) {
        pragma(msg, name);
    }
}
```

Los rasgos `allMembers` devuelven una tupla de cadena que contiene los nombres de los miembros del tipo dado. Estas cadenas son conocidas en tiempo de compilación.

Iterando sobre miembros de una estructura / clase sin sus miembros heredados

```
module main;

auto getMemberNames(T)() @safe pure {
    string[] members;

    foreach (derived; __traits(derivedMembers, T)) {
        members ~= derived;
    }

    return members;
}

class Foo {
    int a;
    int b;
}

class Bar : Foo {
    int c;
    int d;
    int e;
```

```
}

void main() {
    import std.stdio;

    foreach (member; getMemberNames!Bar) {
        writeln(member);
    }
}
```

DeriveMembers devuelve una tupla de literales de cadena, donde cada cadena es el nombre del miembro.

Las salidas de ejemplo:

```
c
d
e
```

Lea Rasgos en línea: https://riptutorial.com/es/d/topic/3416/rasgos

Capítulo 17: UFCS - Sintaxis de llamada de función uniforme

Sintaxis

- aThirdFun (anotherFun (myFun (), 42); // notación común (también válida)
- myFun (). anotherFun (42) .aThirdFun (); // UFCS
- myFun.anotherFun (42) .aThirdFun; // se pueden quitar las llaves vacías

Observaciones

En una llamada `ab(args...)` , si el tipo `a` no tiene un método llamado `b` , el compilador intentará volver a escribir la llamada como `b(a, args...)` .

Examples

Comprobando si un número es primo

```
import std.stdio;

bool isPrime(int number) {
    foreach(i; 2..number) {
        if (number % i == 0) {
            return false;
        }
    }

    return true;
}

void main() {
    writeln(2.isPrime);
    writeln(3.isPrime);
    writeln(4.isPrime);
    5.isPrime.writeln;
}
```

UFCS con rangos

```
void main() {
    import std.algorithm : group;
    import std.range;
    [1, 2].chain([3, 4]).retro; // [4, 3, 2, 1]
    [1, 1, 2, 2, 2].group.dropOne.front; //  tuple(2, 3u)
}
```

UFCS con duraciones de std.datetime

```
import core.thread, std.stdio, std.datetime;

void some_operation() {
    // Sleep for two sixtieths (2/60) of a second.
    Thread.sleep(2.seconds / 60);
    // Sleep for 100 microseconds.
    Thread.sleep(100.usecs);
}

void main() {
    MonoTime t0 = MonoTime.currTime();
    some_operation();
    MonoTime t1 = MonoTime.currTime();
    Duration time_taken = t1 - t0;

    writeln("You can do some_operation() this many times per second: ",
            1.seconds / time_taken);
}
```

Lea UFCS - Sintaxis de llamada de función uniforme en línea:
https://riptutorial.com/es/d/topic/4155/ufcs---sintaxis-de-llamada-de-funcion-uniforme

Creditos

S. No	Capítulos	Contributors
1	Empezando con D Language	Alessio Sacco, André Puel, Bauss, Bennet Leff, Cauterite, Community, EsmaeelE, Gassa, Sirsireesh Kodali, Some coder, T.Furholzer, TuxCopter
2	Arrays y segmentos dinámicos	André Puel, Bauss, greenify, Harry, Shriken
3	Bucles	greenify, o3o
4	Estructuras	Bennet Leff
5	Evaluación de la función de tiempo de compilación (CTFE)	André Puel, greenify
6	Examen de la unidad	greenify
7	Gamas	André Puel, Cauterite, Shriken
8	Guardias de alcance	André Puel, greenify
9	Importaciones y módulos.	greenify
10	Instrumentos de cuerda	Harry, o3o, RamenChef
11	Las clases	greenify, o3o
12	Los contratos	Quonux
13	Matrices asociativas	Bauss, greenify, o3o, Shriken
14	Memoria y punteros	greenify
15	Plantillas	André Puel, Bauss, Quonux
16	Rasgos	André Puel, Bauss
17	UFCS - Sintaxis de llamada de función uniforme	André Puel, greenify, tre0n