Author: Tenton Lien

Last Updated: 10/15/2018

# The Zen Programming Language

# Preface

// TODO

Tenton Lien

# Contents

# Chapter 1 Getting Started

## 1.1 Introduction

Zen is multi-paradigm, statically-typed and general-purpose programming language based on the Zen Virtual Machine.

## 1.2 Writing a Single Zen Program

Every Zen program contains one or more functions, one of which must be named main.

```
val main = (args: String[]) -> {
}
```

A function definition has four elements: a return type, a function name, a parameter list and a function body.

### 1.2.1 Building and Executing Our Program

Having written the program, we need the Zen compiler to compile it so that later we can execute it on the Zen Virtual Machine. Therefore, you should download the compiler on the internet at first.

## 1.3 A First Look at Input / Output

The Zen programming language uses a built-in class Console to do input and output (IO).

#### A program That Uses the IO Library

Using the Console class, we can extend our main program to prompt the user to give us his/her name and then print a salute word.

```
val main = () -> {
    var name: String = Console.read('Enter your name: ')
    Console.writeln('Hello, ' + name)
}
```

## 1.3 A Word about Comments

Before our programs get much more complicated, we should see how Zen handles comments. Comments help the human readers of our programs. They are typically used to summarize an algorithm, identify the purpose of a variable, or clarify an otherwise obscure segment of code. The compiler ignores comments, so they have no effect on the program's behavior or performance.

Although the compiler ignores comments, readers of our code do not. Programmers tend to believe comments even when other parts of the system documentation are out of date. An incorrect comment is worse than no comment at all because it may mislead the reader. When you change your code, be sure to update your comments, too.

### Kinds of Comments in Zen

There are two kinds of comments in Zen: single-line and paired, which are inherited from C and C++. A single comment starts with a double slash (//) and ends with a new line. Everything to the right of the slashes on the current line is ignored by the compiler. A comment of this kind can contain any text, including additional double slashes.

The other kind of comment uses two delimiters (/* and */). Such comments begin with a /* and end with the next */. These comments can include anything that is not a */, including newlines. The compiler treats everything that falls between the /* and */ as part of the comment.

# Chapter 2 Variables and Basic Types

## 2.1 Primitive Built-in Types

As a statically-typed programming language, Zen defines a set of primitive types that include the arithmetic types and two special types named void and any. The arithmetic types represent characters, integers, boolean values and floating-point numbers. The void type has no associated values and can be used in only a few circumstances, most commonly as the return type for functions that do not return a value. The any type is actually a template to accept all kinds of data types.

### 2.1.1 Arithmetic Types

The arithmetic types are divided into two categories: integral types (which include characters and boolean types) and floating-point types.

| Zen: Arithmetic Types | | | |
|---|---|---|---|
| **Type** | **Meaning** | **Size** | **Bounds** |
| bool | boolean | 1 byte | True / False |
| char | character | 1 byte | 0 - 127 |
| i8 | byte | 1 byte | -128 - 127 |
| u8 | unsigned byte | 1 byte | 0 - 255 |
| i16 | short integer | 2 bytes | -32,768 - 32,767 |
| u16 | unsigned short integer | 2 bytes | 0 – 65,535 |
| i32 | integer | 4 bytes | |
| u32 | unsigned integer | 4 bytes | |
| i64 | long integer | 8 bytes | |
| u64 | unsigned long integer | 8 bytes | |
| f32 | single-precision floating-point | 4 bytes | |
| f64 | double-precision floating-point | 8 bytes | |

The bool type represents the truth values true and false.

There are several character types, most of which exist to support internationalization. The basic character type is char.

### 2.1.2 Type Conversions

## 2.2 Variables

### 2.2.3 Identifiers

# Chapter 3 Arrays and Strings

An array is a data structure that is similar to the data container linked list but offers a different trade-off between performance and flexibility. Unlike a linked list, arrays have fixed size.

## 3.1 Defining and Initializing Built-in Arrays

Arrays are a compound type. An array declarator has the form array[id], where array is the name being defined and id is the dimension of the array.

```
var arr: i32[]
```

When we define an array, we must specify a type for the array. We cannot use auto to deduce the type from a list of initializers.

# Chapter 4 Expressions

## 4.1 Fundamentals

## 4.2 Arithmetic Operators

## 4.3 Logical and Relational Operators

## 4.4 Assignment Operators

## 4.5 Increment and Decrement Operators

# Chapter 5 Statements

## 5.1 Simple Statements

Most statements in C and C++ end with a semicolon, but in Zen the semicolon is not put at the end of each statement.

## 5.2 Statement Scope

## 5.3 Conditional Statements

Zen provides two statements that allow for conditional execution. The if statement determines the flow of control based on a condition. The switch statement evaluates an integral expression and chooses one of several execution paths based on the expression’s value.

### 5.3.1 The if Statement

An if statement conditionally executes another statement based on whether a specified condition is true. There are two forms of the if: one with an else branch and one without. The syntactic form of the simple if is

```
if (condition)
    statement
```

An if else statement has the form

```
if (condition)
    statement
else
    statement2
```

## 5.4 Iterative Statements

## 5.5 Jump Statements

## 5.6 Exception Handling

# Chapter 6 Functions

## 6.1 Function Basics

```
val main: void = function(args: String) -> {


}
```

### 10.2.1 Main Function

# Chapter 7 Classes and Objects

## 7.3 Class and Method

```
val Student = class -> {

    private val num: i32 = 123

    public val student:i32 = (x: i32) -> {
        x + num
    }
}



Student student1(12)
```

# Chapter 8 Object-Oriented Programming

# Chapter 9 Functional Programming

High Orderer Function

# Chapter 10 The IO Library

# Chapter 11 Data Containers and Algorithms

# Chapter 12 System and Process

## 12.1 Date and Time

# Chapter 13 Network

```
// Server.zn
  import zen.net
  import zen.console

  val main = () => {
      var ss = net.socket(8080)
      while (true) {
          if (ss.accept()) {
           var str: string = ss.read()
              console.writeln("Received from Client: " + str)
              ss.write("success")
          }
      }
  }

// Client.zn
  import zen.console
  import zen.net

  val main = () => {
      var s = net.socket("192.168.1.22", 8080)
      s.write("Hello!")
      console.println(s.read())
  }
```

# Chapter 14 Multi-Thread

# Chapter 15 Web Development

## 15.1 A Simple Server Program

## 15.2 Connect with Databases

## 15.3 JSON and XML Parser

# Appendix I The Library

2 Zen Internal Library Functions

| Function Declaration | Return Value | Definition |
| --- | --- | --- |
| console.read() | | |
| console.readln() | | |
| console.write() | null | |
| console.writeln() | null | |
| | | |
| | | |
| | | |
| file.close() | | |
| file.compress() | | |
| file.exist(string file_path) | bool | |
| file.extract(string file_path) | | |
| file.open() | | |
| math.sqrt() | | |
| memory.allocate(long memory_size) | bool | |
| memory.release(long memory_address) | bool | |
| net.socket() | | |
| system.call() | bool | |
| system.info() | | |
| system.kill() | bool | |
| thread.create() | | |
| time.current() | class | |
| time.sleep(long millisecond) | | |

| | |
|---|---|
| time.stamp() | long |

type.bool()
type.byte()
type.char()
type.double()
type.float()
type.integer()
type.long()
type.short()
type.string()

## 3 Zen Standard Extended Library Functions

| Function Declaration | Return Value | Definition |
|---|---|---|
| algorithm.add(string num1, string num2) | string | |
| algorithm.division(string divided, string divisor) | string | |
| algorithm.max(array) | | |
| algorithm.min(array) | | |
| algorithm.multiply() | | |
| algorithm.sort(array) | array | |
| algorithm.substract() | | |

database.connect(string host, string dbname, string username, string password)  bool

database.delete()  bool

database.disconnect()  bool

database.execute(string sql_command)

database.insert()  bool

database.select()  string

database.update()  bool

json.decode()

json.encode()

media.play()

security.md5()

security.sha1()

string.length()

string.find()

string.replace()

string.split()

xml.decode()

xml.encode()

Zen and Web Server Development

Zen and Natural Language Processing