Tenon 语言教程

Release 0.1.0

ZDZN

Aug 30, 2018

Contents

Chapter 1

简介

Tenon 是一门用于机械臂程序开发的编程语言，具有可扩展、支持错误处理、中断等特性。它包括独立的语言核心部分，以及机械臂相关的扩展部分。本文档主要介绍其语言核心部分的内容。

1.1 语言特点

Tenon 以 rapid 等传统语言为范本，参考诸多现代语言，形成了自己的语言特点。

与该领域传统语言相比：

- Tenon 大小写敏感，更符合现代编程习惯。
- 语言更为简洁。例如 tenon 通过 `function` 实现了 rapid 的 `routine` 和 `procedure` 。
- 语言更为自由。Tenon 的记录类型支持函数成员, tenon 支持自增、自减表达式等。
- 语言具有扩展性。Tenon 支持通过 c 语言等主流语言进行功能扩展。

1.2 语言展示

首先，我们来编写一个简单的例子程序 hello.t：

```
! This is a simple tenon program 'hello.t'.

module hello

  function void main()
    tpwrite("Hello World!");
  end

end
```

在命令行中执行该程序，可以看到输出信息如下：

```
$ tenon -m base hello.t
Hello World!
```

这个例子展示了 tenon 程序的大概模样:

- 程序由模块组成，每个模块作为一个文件单独保存
- 程序的执行入口函数为 `main` 函数
- 注释以 `!` 开头，直到一行结束
- 语句以 `;` 结束
- 代码块，例如模块，函数等，以 `end` 结束

与 rapid 不同，tenon 语言大小写敏感，标识符由字母、数字和下划线组成，其中起始字符必须是字母或者下划线。

标识符不能与关键字重名。关键字包括：

```
and alias
backward bool
case connect const
default div do
else elseif end error exit
false for from function
goto
if
local
module mod
not num
or
pers private public
raise record retry return
step string switch
task then trap true trynext
undo
var void
while with
xor
```

Chapter 2

数据

2.1 数据种类

数据种类包括：常量、变量、持久量。

- 常量为只读数据，不能被改写
- 变量即可以被读取，也可以被改写
- 持久量也是可以读写的，与变量不同的是，它在程序运行结束后，其值会被持久保存

2.2 数据类型

数据类型包括：基本类型、记录类型和别名类型。

2.2.1 基本类型

有三种基本数据类型：`num`、`bool`、`string` 。分别对应数字类型、布尔类型和字符串类型。

2.2.2 记录类型

记录类型用来表示一个组合类型，具有一个或者多个域。与 rapid 相比，tenon 语言的记录类型支持函数成员。下面的例子，展示了如何定义一个记录类型：

```
record info
  string name;
  num age;
end
```

这段代码定义了一个记录类型 `info` 。

有 3 个内建记录类型：`pos`，`orient`，`pose` 。它们分别定义如下：

```
record pos
  num x;
  num y;
  num z;
end

record orient
  num q1;
  num q2;
  num q3;
  num q4;
end

record pose
  pos trans;
  orient rot;
end
```

2.2.3 别名类型

别名类型用来表示一个类型的别名，它与所表示的实际类型是等价的。下面的例子，展示了如何定义一个别名类型：

```
alias num score;
```

这段代码定义了一个别名类型 `score` 。

有 2 个内建别名类型：`errnum`，`intnum` 。它们分别定义如下：

```
alias num errnum;
alias num intnum;
```

2.3 数据声明

Tenon 程序中，声明一个数据，需要给出该数据的种类和类型。数据可以是一个 1 维或多维数组，最多是 3 维。

2.3.1 常量声明

常量声明必须要给出初始值，初始值为一个常量表达式。下面的例子，展示了如何声明一个常量：

```
const num pi = 3.14;
```

这段代码声明了一个 num 类型的常量 pi ，其初始值为 3.14 。

2.3.2 变量声明

变量声明的初始值是可选的，如果有则为一个常量表达式。下面的例子，展示了如何声明一个变量：

```
var bool status;
```

这段代码声明了一个 bool 类型的变量 status 。

2.3.3 持久量声明

持久量声明的初始值是可选的，如果有则为一个文字表达式。下面的例子，展示了如何声明一个持久量：

```
pers pos move_point = [0, 0, 0];
```

这段代码声明了一个 pos 类型的持久量 move_point ，其初始值为 [0, 0, 0] 。

Chapter 3

表达式

表达式用来表示一个值，以及值的运算。

3.1 运算符

Tenon 支持如下运算符：

```
+ - * / div mod ++
-- < <= > >= == <>
and xor or not
```

与 rapid 相比，tenon 支持自增运算与自减运算。

3.2 数据引用

表达式中，可以通过标识符来引用一个数据。下面的例子展示了如何引用数据：

```
a
b.x
c{1,2}
```

3.3 常量表达式

常量表达式中不能包含变量，持久量，以及函数调用。常量表达式主要用于数据的初始化。

3.4 文字表达式

文字表达式是一种特殊的常量表达式，其只能是一个单个的文字值，或者一个聚合表达式，且表达式的成员都是单个的文字值。文字表达式主要用于持久量的初始化。

3.5 聚合表达式

聚合表达式表示一个组合值，其可以是一个数组，或者一个记录。例如：

```
[1, 2, 3]
```

Chapter 4

语句

Tenon 支持各种常见的语句，包括声明语句，赋值语句，循环，条件分支，函数调用等；同时也支持异常处理和中断处理相关的语句。

4.1 赋值语句

赋值语句是将一个表达式的值赋予一个数据，数据类型和表达式类型必须匹配。

例如：

```
a = 1;
```

4.2 If 语句

If 语句是常用的选择语句。语句以 `if` 开头，以 `end` 结束。由 if、elseif、else 三部分组成。其中 elseif 部分可以有多个或没有, else 部分可省略。

当满足条件时执行 then 语句列表。语法为：

```
if 条件表达式 then 语句列表
{elseif 条件表达式 then 语句列表}
[else 语句列表]
end
```

下面是一个简单示例：

```
if option == "-h" then
  show_help();
elseif option == "-v" then
```

(continues on next page)

(continued from previous page)

```
  show_version();
else
  return;
end
```

此例中的 if 部分由条件表达式 `option == "-h"` 以及 then 语句列表 `show_help();` 组成。当 `option` 的值为 `"-h"` 时，调用函数 `show_help()` 。

4.3 For 语句

For 语句为常用循环语句，以 `for` 开头，以 `end` 结束。由循环变量，起点表达式，终点表达式以及语句列表组成。其中步长表达式可省略，默认值为 `1` 。

语句会重复执行语句列表，每执行一次循环变量的值会增加步长表达式的值，当循环变量的值超过超过终点表达式的值时，循环终止。语法为：

```
for 循环变量 from 起点表达式 to 终点表达式 [步长表达式]    do
  语句列表
end
```

下面是一个简单示例：

```
var num sum = 0;
for i from 1 to 10 do
  sum = sum + i;
end
```

此例中，我们通过 for 循环求得从 1 到 10 的整数之和。

4.4 While 语句

while 语句为常用循环语句，以 `while` 开头，以 `end` 结束。由条件表达式以及语句列表组成。

语句会重复执行语句列表直至不再满足条件。语法为：

```
while 条件表达式   do
  语句列表
end
```

下面是一个简单示例：

```
var num sum = 0;
var num i = 1;
while i <= 10 do
  sum = sum + i;
end
```

此例中，我们通过 while 循环求得从 1 到 10 的整数之和。

4.5 Switch 语句

switch 语句是常用的选择语句。语句以 `switch` 开头，以 `end` 结束。由 switch、case、default 三部分组成。其中 case 部分可以有多个或没有, default 部分可省略。

语句分析表达式的值，如果有与之相等的测试值，则执行对应 case 的语句列表，否则执行 default 部分的语句列表。语法为:

```
switch 表达式
{case 测试值: 语句列表}
[default: 语句列表]
end
```

下面是一个简单示例:

```
switch a;
case 1:
  return true;
default:
  return false;
end
```

此例中，我们通过 witch 语句判断 a 的值是否为 1。

4.6 Return 语句

Return 语句用于函数返回一个表达式的值或空值。

下面是一些简单示例:

```
return;
return 1;
return a;
```

4.7 Exit 语句

Exit 语句用于退出程序。

下面是一个简单示例：

```
var num i = 0;

while true do
  i = i + 1;
  tp_write(i);
  if i >= 10 then
    exit;
  end
end
```

此例用于打印从 0 到 10 的所有整数。

4.8 Raise 语句

错误处理的常用语句，用于将错误报告到上一层，或设置错误。

下面是一个简单示例：

```
function void main()
  safediv(2, 0);

error
  trynext;
end

function num safediv(num x, num y)
  return x / y;

  error
    raise;
end
```

此例中函数 `safediv()` 发生错误时，会将错误报告到 `main` 函数的错误处理处处理。

4.9 Retry 语句

错误处理的常用语句，用于从出错的语句处重新执行。

下面是一个简单示例：

```
function num safediv(num x, num y)
  return x / y;

error
  y = 1;
  retry;
end
```

此例中，如果函数运行出现错误，就会将 y 的值设为 1 然后从出错处重新执行，最终的返回值为 x 的值。

4.10 Trynext 语句

错误处理的常用语句，用于从出错的语句的下一条语句处开始执行。

下面是一个简单示例：

```
function num safediv(num x, num y)
  var num a = 0;
  a = x / y;
  return a;
error
  trynext;
end
```

此例中，如果函数运行出现错误，a = x / y; 赋值失败，函数会继续执行 return 语句 return a，返回值为 0 。

4.11 Connect 语句

用于将中断变量与中断处理函数关联。

下面是一个简单示例：

```
var intnum t1;

function void main()
  connect t1 with traptry;
end

trap traptry
  tp_write("this is a interrupt test");
end
```

Chapter 5

函数

Tenon 程序的执行代码位于函数中，其中入口函数为 `main` 函数。

5.1 函数声明

函数可以有 1 个返回值，如果没有返回值则通过 `void` 来指出。函数的参数分为：必选参数和可选参数。下面的例子展示了如何声明一个函数：

```
function num increase(num x, num y = 1)
  return x + y;
end
```

其中 `x` 为必选参数，`y` 为可选参数。可选参数可以给出一个缺省值，在这里，`y` 的缺省值是 `1` 。

5.2 函数调用

调用函数，涉及到函数的传参和返回值。必选参数必须给出，可选参数可以给出，并且必须是位于所有的必选参数之后。下面的例子展示了函数调用：

```
function void main()
  var num a = 1;
  a = increase(a);
  a = increase(a, y=2);
end
```

第一次调用 `increase`，`y` 使用的是缺省值 `1` ； 第二次调用，`y` 使用的是给定值 `2` 。

Chapter 6

模块

Tenon 程序由一个或者多个模块组成。每个模块可以用来实现独立的功能。一个模块具有一个模块名，以及可选的模块属性。

模块属性用来配置该模块在机械臂运行系统中的权限，加载执行模式等，包括：

- sysmod，表示该模块为系统模块
- noview，表示该模块的源代码对用户是不可见的
- nostepin，表示在单步执行模式下，不能单步进入该模块
- viewonly，表示该模块的源代码不能被用户修改

下面的例子用来声明一个系统模块：

```
module smod(sysmodule, viewonly)


end
```

系统模块在机械臂加电启动之后，会被自动加载到内存中，对所有的程序都可用。缺省没有指定属性的模块为普通模块，其只对当前程序可用。

Chapter 7

错误处理

错误处理是指在函数运行出错时执行的语句，它能够帮助我们从错误中恢复，从而保证任务顺利进行。

错误处理以 error 开头，直至函数声明结束。我们可以为错误处理添加编号，并通过 raise 语句将错误提交到指定的错误处理处。

语法为：

```
error
  语句列表

或：
error(表达式数列)
  语句列表
```

7.1 错误编号

TENON 为每种错误类型提供了一个编号，例如 除数为零的编号为：ERR_DIVZERO 。

7.2 错误变量

错误变量 ERRNO 是一个用于保存错误编号的系统变量。

当函数执行出错后，错误编号就会被写入到 ERRNO 中，我们通过查询 ERRNO 的值来确定当前错误类型。

7.3 错误处理代码

下面是一个简单的错误处理示例：

```
function safediv(num x, num y)
  return x / y;

error
  if ERRNO == ERR_DIVZERO then
    return x;
  end
end
```

这个例子展示了函数中错误处理的大概模样：

在执行函数时，如果出现错误，系统会自动将 ERRNO 的值设为该错误对应的错误编号，并进入函数的错误处理部分。

在这一函数中，如果错误类型是除数为零 ($ERRNO == ERR\_DIVZERO$)，就会返回被除数的值。

Chapter 8

中断

在执行任务的过程中有可能触发中断，我们可以通过中断处理函数 (*trap* 函数) 来处理中断。

8.1 中断变量

中断变量是用户声明的用以和 trap 函数关联的变量，变量类型为 `intnum` 。一个中断变量只能与一个 trap 函数关联。

中断编号声明语句:

```
var intnum interrupt;
```

此外，TENON 在函数库中提供一些用于操作中断变量的函数，包括:

```
idelete()

idisable()

ienable()

ipers()

isleep()

iwatch()
```

8.2 系统中断变量

系统中断变量 `INTNO` 是一个用于保存中断编号的系统变量。

当发生中断后，中断编号就会被写入到 INTNO 中，我们通过查询 INTNO 的值来确定当前中断类型。

8.3 中断处理函数

通过 trap 来声明一个 trap 函数，并以 end 结束声明。

通过 connect 语句将 trap 函数与中断编号关联。一个 trap 函数可以与多个中断变量关联。

下面是一个简单的中断处理示例：

```
module trap_test

var intnum t1;
trap traptry
  if INTNO == t1 then
    tp_write("this is an interrupt try.");
  end
end

function void main()
  connect t1 with traptry;
  starttrap(t1);
end

end
```

首先我们声明一个中断变量 t1 ，以及 trap 函数 traptry 。

然后在主函数中通过 connect 语句将中断变量 t1 与 trap 函数 traptry 关联。

在发生中断后，系统会将 t1 的值写入到 INTNO 中，并调用与之关联的 trap 函数。

Chapter 9

语言扩展

与 rapid 相比,tenon 的扩展性更强。Tenon 支持使用 opaque 函数声明一个接口，通过 C 语言进行扩展。opaque 函数以 `opaque function` 开头，以 `end` 结束。与一般函数相比缺省了语句部分。

C 文件:

```
#include "tenon.h"

static void tpwrite_bool(TenonData data)
{
  bool v = tenon_get_bool_value(data);
  char *str = v ? "true" : "false";
  printf(str);
}

static void tpwrite_num(TenonData data)
{
  double v = tenon_get_num_value(data);
  printf("%0.14g", v);
}

static const TenonExtFunc base_funcs[] = {
  {"tpwrite_bool", tpwrite_bool},
  {"tpwrite_num", tpwrite_num},
  {NULL, NULL}, /* sentinel */
};

int init_base()
{
  return tenon_init_lib("base", base_funcs);
}
```

(continues on next page)

(continued from previous page)

```
Tenon 文件:
  opaque function void tpwrite_num(num x)
  end

  function void main()
    tpwrite_num(1);
  end
```

首先我们在 tenon 文件中声明一个 `opaque` 函数，在C语言中将其实现，并将其注册到 tenon 中，最后就可以调用这个函数了。

此示例最后会打印 `1` 。

Chapter 10

Tenon 函数介绍

10.1 abs 函数

用于得到一个数值的绝对值。

语法

`abs(value)`

- 参数类型：num，参数可以是正数、负数或者小数。
- 返回值类型：num，返回值是一个非负数。

例子

下面是一个简单的示例：

```
x = abs(3);
x = abs(-3);
x = abs(-2.53);
```

value	abs(value)
3	3
-3	3
-2.53	2.53

10.2 acos 函数

用于计算一个数值的反余弦值。其中，反余弦值以度表示。

语法

`acos(value)`

- 参数类型：num，取值范围为 [-1, 1]。
- 返回值类型：num，取值范围为 [0, 180]。

例子

下面是一个简单的示例：

```
x = acos(-1);
x = acos(0);
x = acos(1);
```

value	acos(value)
-1	180
0	90
1	0

10.3 argname 函数

用于得到当前参数的原始数据对象的名称。

语法

`argname(parameter)`

- 参数类型：anytype，可以是原子、记录或数组在内的所有类型的数据。
- 返回值类型：string

例子

下面是一个简单的示例：

```
function void test_AN(num x)
  var string name;
  name = argname(x);
  print(name);
end

function void main()
  var num a{2,2} = [[1, 2], [2, 1]];
  var num text;
  test_AN(a{1, 2});
  test_AN(a{2, 2});
```

(continues on next page)

(continued from previous page)

```
  test_AN(text);
end
```

此示例会依次打印 `a{1, 2}` ， `a{2, 2}` ， `text` 。

10.4 asin 函数

用于计算一个数值的反正弦值。其中，反正弦值以度表示。

语法

`asin(value)`

- 参数类型：num，取值范围为 [-1, 1]。
- 返回值类型：num，取值范围为 [-90, 90]。

例子

下面是一个简单的示例：

```
x = asin(-1);
x = asin(0);
x = asin(1);
```

value	asin(value)
-1	-90
0	0
1	90

10.5 atan 函数

用于计算一个数值的反正切值。其中，反正切值以度表示。

语法

`atan(value)`

- 参数类型：num，参数可以取任意数值。
- 返回值类型：num，取值范围为 [-90, 90]。

例子

下面是一个简单的示例：

```
x = atan(1);
x = atan(-1);
```

value	atan(value)
1	45
-1	-45

10.6 atan2 函数

用于计算两个数值的比值的反正切值。其中，反正切值以度表示。

语法

`atan2(y, x)`

- 参数 y：作为分子参数值，数据类型为 num。
- 参数 x：作为分母参数值，数据类型为 num。
- 返回值：等于 atan(y/x)，数据类型为 num，取值范围为 [-180, 180]，并根据两个参数的符号确定返回值的象限。

将通过以下示例说明该函数的用法：

```
var num angle;
var num x_value;
var num y_value;
angle = atan2(y_value, x_value);
```

在本示例中 `angle` 的值等于 `y_value/x_value` 的反正切值。

例子

下面是一个简单的示例：

```
x = atan2(1,1);
x = atan2(1,-1);
x = atan2(-1,-1);
x = atan2(-1,1);
```

y	x	atan2(y, x)
1	1	45
1	-1	135
-1	-1	-135
-1	1	-45

10.7 bitand 函数

对两个数值进行逻辑按位与运算。

语法

`bitand(data1, data2)`

- 参数 data1：数据类型为整型，取值范围 [0, 4503599627370495]。
- 参数 data2：数据类型为整型，取值范围 [0, 4503599627370495]。
- 返回值类型：整型。

例子

下面是一个简单的示例：

```
var num data1 = 38;
var num data2 = 35;
var num data3;
data3 = bitand(data1, data2);
```

在本示例中, 对 `data1` 、`data2` 进行按位与运算，并将结果赋值给 `data3` 。运算过程，如下所示：

数值	52 位表示
data1: 38	0 0 ~ 0 0 1 0 0 1 1 0
data2: 35	0 0 ~ 0 0 1 0 0 0 1 1
data3: 34	0 0 ~ 0 0 1 0 0 0 1 0

10.8 bitcheck 函数

检查一个数值的特定位是否为 1。

语法

`bitcheck(data, check_pos)`

- 参数 data: 即被检查的数据，数据类型为整型，取值范围 [0, 4503599627370495]。
- 参数 check_pos：即指定的位，数据类型为整型，取值范围 [1, 52]。
- 返回值类型：bool。若指定位为 1，则返回 TRUE。若指定位为 0，则返回 FALSE。

例子

下面是一个简单的示例：

```
var bool x;
x = bitcheck(130,8);
print(x);
```

如下表所示，数值 130 的第 8 位为 1 ，故 bitcheck(130,8) 将返回 TRUE ，即本示例会打印输出 x 的值为 TRUE 。

数值	52 位表示
130	0 0 ~ 1 0 0 0 0 0 0 1 0

10.9 bitlsh 函数

对一个数值进行逻辑按位左移操作。

语法

bitlsh(data, ls_step)

- 参数 data：数据类型为整型，取值范围 [0, 4503599627370495]。
- 参数 ls_step：即向左移动的位数。数据类型为整型，取值范围 [1, 52]。
- 返回值类型：数据类型为整型。左移后，右侧位补 0。

例子

下面是一个简单的示例：

```
var num data1 = 38;
var num ls_step = 3;
var num data2;
data2 = bitlsh(data1, ls_step);
print(data2);
```

由下表可知，数值 38 向左移动 3 位后得到数值 48，故本示例将打印 data2 的值 48 。

起始数值	52 位表示	左移 3 位	结果数值
data1: 38	0 0 ~ 0 0 1 0 0 1 1 0	0 0 ~ 0 0 1 1 0 0 0 0	data2: 48

10.10 bitneg 函数

对一个数值实现按位取反。

语法

`bitneg(data)`

- 参数类型：num，以整型表示。取值范围 [0, 4503599627370495]。
- 返回值类型：num，以整型表示。

例子

下面是一个简单的示例：

```
var num data1 = 4;
var num data2;
data2 = bitneg(data1);
print(data2);
```

由下表可知，数值 4 按位取反后得到数值 4503599627370491，故本示例将打印 `data2` 的值 `4503599627370491` 。

起始数值	52 位表示	按位取反	结果数值
data1: 4	0 0 ~ 0 0 0 0 0 1 0 0	1 1 ~ 1 1 1 1 1 0 1 1	data2: 4503599627370491

10.11 bitor 函数

对两个数值进行逻辑按位或运算。

语法

`bitor(data1, data2)`

- 参数 data1：num，以整型表示。取值范围 [0, 4503599627370495]。
- 参数 data2：num，以整型表示。取值范围 [0, 4503599627370495]。
- 返回值类型：num，以整型表示。

例子

下面是一个简单的示例：

```
var num data1 = 39;
var num data2 = 162;
var num data3;
data3 = bitor(data1, data2);
```

在本示例中，对 `data1` 、`data2` 进行按位或运算，并将结果赋值给 `data3` 。运算过程，如下所示：

数值	52 位表示
data1: 39	0 0 ~ 0 0 1 0 0 1 1 1
data2: 162	0 0 ~ 1 0 1 0 0 0 1 0
data3: 167	0 0 ~ 1 0 1 0 0 1 1 1

10.12 bitrsh 函数

对一个数值进行逻辑按位右移操作。

语法

`bitrsh(data, rs_step)`

- 参数 data：数据类型为整型，取值范围 [0, 4503599627370495]。
- 参数 rs_step：即向右移动的位数。数据类型为整型，取值范围 [1, 52]。
- 返回值类型：数据类型为整型。右移后，左侧位补 0。

例子

下面是一个简单的示例：

```
var num data1 = 38;
var num rs_step = 3;
var num data2;
data2 = bitrsh(data1, rs_step);
print(data2);
```

由下表可知，数值 38 向右移动 3 位后得到数值 4，故本示例将打印 `data2` 的值 `4` 。

起始数值	52 位表示	右移 3 位	结果数值
data1: 38	0 0 ~ 0 0 1 0 0 1 1 0	0 0 ~ 0 0 0 0 0 1 0 0	data2: 4

10.13 bitxor 函数

对两个数值进行逻辑按位异或运算。

语法

`bitxor(data1, data2)`

- 参数 data1：num，以整型表示。取值范围 [0, 4503599627370495]。
- 参数 data2：num，以整型表示。取值范围 [0, 4503599627370495]。
- 返回值类型：num，以整型表示。

例子

下面是一个简单的示例：

```
var num data1 = 39;
var num data2 = 162;
var num data3;
data3 = bitxor(data1, data2);
```

在本示例中, 对 data1 、data2 进行按位异或运算，并将结果赋值给 data3 。运算过程，如下所示：

数值	52 位表示
data1: 39	0 0 ~ 0 0 1 0 0 1 1 1
data2: 162	0 0 ~ 1 0 1 0 0 0 1 0
data3: 133	0 0 ~ 1 0 0 0 0 1 0 1

10.14 bytetostr 函数

将一个字节数据转换为一定格式的字符串数据。

语法

```
bytetostr(data, [\Hex]|[\Okt]|[\Bin]|[\Char])
```

- 参数 data：num 类型，取值范围 [0, 255]。
- 可选参数：可指定以下一种格式。如果缺省，则以十进制（Dec）格式转换。
  - Hex：数据将以十六进制格式转换。
  - Okt：数据将以八进制格式转换。
  - Bin：数据将以二进制格式转换。
  - Char：数据将以 ASCII 字符格式转换。
- 返回值类型：string

可选格式的具体信息，见下表：

格式	字符集	字符串长度	范围
Dec	0 ~ 9	1 ~ 3	0 ~ 255
Hex	0 ~ 9, A ~ F	2	00 ~ FF
Okt	0 ~ 7	3	000 ~377
Bin	0, 1	8	00000000 ~ 11111111
Char	any ASCII char	1	one ASCII char

例子

下面是一个简单的示例：

```
var string test{5};
var num data = 122;
test{1} = bytetostr(data);
test{2} = bytetostr(data, Hex=1);
test{3} = bytetostr(data, Okt=1);
test{4} = bytetostr(data, Bin=1);
test{5} = bytetostr(data, Char=1);
```

本示例中，数组被依次赋值 `122` 、`7A` 、`172` 、`01111010` 、`z` 。

10.15 cdate 函数

用于读取当前系统日期。

语法

`cdate()`

- 返回值类型：string。标准日期格式为“年 - 月 - 日”，例如“1998-01-29”。

例子

```
function void main()
  print(cdate());
end
```

本示例会打印当前的系统日期“2018-08-29”。

10.16 cos 函数

计算一个角度值的余弦值。

语法

`cos(angle)`

- 参数类型：num，角度值以度表示。
- 返回值类型：num，范围为 [-1, 1]。

例子

下面是一个简单的示例：

```
function void main()
  var num x;
  x = cos(60);
  print(x);
end
```

本示例将打印输出 x 的值为 0.5 。

10.17 ctime 函数

用于读取当前系统时间。

语法

ctime()

- 返回值类型：string。标准时间格式为“小时：分钟：秒”，例如“18:20:46”。

例子

```
function void main()
  print(ctime());
end
```

本示例将打印当前的系统时间“21:22:01”。

10.18 dectohex 函数

将可读字符串中的数字由十进制转换为十六进制。

语法

dectohex(str)

- 参数类型：string
- 返回值类型：string，结果字符串由字符集 [0-9，A-F，a-f] 构成。

例子

```
function void main()
  print(dectohex("10"));
end
```

本示例将打印输出 A 。

10.19 dim 函数

用于获得数组指定维度的元素数量。

语法

`dim(arrpar, dimno)`

- 参数 arrpar：数组名称，可以是任意类型。
- 参数 dimno：数组的维度，num 类型。数字 1 表示第一维，数字 2 表示第二维，数字 3 表示第三维。
- 返回值类型：num，指定维度的数组元素数。

例子

下面是一个简单的示例：

```
function void main()
  var num dimno;
  var pos p1 = [1, 2, 3];
  var pos p2 = [2, 3, 4];
  var pos p3 = [3, 4, 5];
  var pos p4 = [4, 5, 6];
  var pos p5 = [5, 6, 7];
  var pos p6 = [6, 7, 8];
  var pos action{3, 2} = [[p1, p2], [p3, p4], [p5, p6]];
  dimno = dim(action,1);
  print(dimno);
  dimno = dim(action,2);
  print(dimno);
  dimno = dim(action,3);
  print(dimno);
end
```

本示例将打印输出数组 `action` 第一维的元素数 `3` ，第二维的元素数 `2` ，第三维的元素数 `3` 。

10.20 distance 函数

用于计算空间中两点之间的距离。

语法

`distance(point1, point2)`

- 参数类型：pos
- 返回值类型：num，始终为正数。

假设两点 (x1,y1,z1) 和 (x2,y2,z2)，距离计算公式为：

```
distance = ((x1 - x2) ^ 2 + (y1 - y2) ^ 2 + (z1 - z2) ^ 2) ^ (1 / 2);
```

例子

下面是一个简单的示例：

```
function void main()
  var pos p1;
  var pos p2;
  var num d;
  p1 = [1,2,5];
  p2 = [3,4,6];
  d = distance(p1,p2);
  print(d);
  p1 = [4,0,4];
  p2 = [-4,4,4];
  d = distance(p1,p2);
  print(d);
  p1 = [1,2,5];
  p2 = [0,0,0];
  d = distance(p1,p2);
  print(d);
end
```

本示例将依次打印三个距离值 3.0 、8.9442719099992 、5.4772255750517 。

10.21 dotprod 函数

用于计算两个 pos 向量的点积。

两个向量 A 和 B 的点积等于 A 和 B 的大小以及它们之间角度的余弦的乘积。

语法

```
dotprod(vector1, vector2)
```

- 参数类型：pos
- 返回值类型：num

点积的取值：

- 小于或等于两个向量大小的乘积。
- 可以是正数也可以是负数，取决于两个向量之间的角度是小于还是大于 90 度。
- 当两个向量彼此垂直时，点积为零。

例子

下面是一个简单的示例：

```
function void main()
  var pos v1;
  var pos v2;
  var num d;

  v1 = [1,1,0];
  v2 = [0,0,1];
  d = dotprod(v1,v2);
  print(d);

  v1 = [1,1,1];
  v2 = [-1,-1,0];
  d = dotprod(v1,v2);
  print(d);

  v1 = [1,1,1];
  v2 = [1,2,3];
  d = dotprod(v1,v2);
  print(d);

  v1 = [1,1,0];
  v2 = [2,2,0];
  d = dotprod(v1,v2);
  print(d);

  v1 = [1,1,0];
  v2 = [-1,-1,0];
  d = dotprod(v1,v2);
  print(d);
end
```

本示例会依次打印两个向量的点积值 `0` 、`-2` 、`6` 、`4` 、`-2` 。

10.22 exp 函数

用于得到一个数值的指数值。

语法

`exp(exponent)`

- 参数类型：num

- 返回值类型：num，得到的指数值。

例子

```
var num x;
var num value;
value = exp(x);
```

在本示例中，x 的指数值将被赋值给 value 。

10.23 filesize 函数

10.24 filetime 函数

10.25 fssize 函数

10.26 getnextsym 函数

10.27 gettime 函数

10.28 hextodec 函数

10.29 isfile 函数

10.30 modexist 函数

10.31 numtostr 函数

10.32 pow 函数

10.33 present 函数

10.34 round 函数

将数值四舍五入到指定的小数或整数值。

语法

`round(val, dec)`

- 参数 val: num 类型。
- 参数 dec: num 类型，用于指定数据 val 的精度。
- 返回值类型：num 类型，返回特定精度的数据类型。

例子

下面是一个简单的示例：

```
v1 = round(0.3852138\Dec:=3);
v2 = round(0.3852138\Dec:=1);
v3 = round(0.3852138);
v4 = round(0.3852138\Dec:=6);
```

val, dec	round(val, dec)
0.3852138Dec:=3	0.385
0.3852138Dec:=1	0.4
0.3852138	0
0.3852138Dec:=6	0.385214

10.35 sin 函数

计算正弦值。

语法

`sin(val)`

- 参数 val: num 类型, 角度值。
- 返回值类型：num 类型，范围是 [-1, 1]。

例子

下面是一个简单的示例：

```
v1 = sin(-1);
v2 = sin(0);
v3 = sin(1);
```

val	sin(val)
-1	-0.841471
0	0
1	0.841471

10.36 sqrt 函数

用于计算平方根。

语法

`sqrt(val)`

- 参数 val: num 类型，正数或 0。
- 返回值类型：num 类型，val 的平方根。

例子

下面是一个简单的示例：

```
v1 = sqrt(9);
```

val	sqrt(val)
9	3

10.37 strdigcmp 函数

比较两个数字字符串。

语法

`strdigcmp(str1, relation, str2)`

- 参数 str1: 字符串类型。
- 参数 relation: 定义符合比较两个字符串，包含 LT, LTEQ, EQ, NOTEQ, GTEQ 或 GT。
- 参数 str2: 字符串类型。
- 返回值类型：bool, 到条件符合时返回 true, 否则是 false。

例子

下面是一个简单的示例：

```
v1 = strdigcmp("1234", EQ, "1256");
```

str1, relation，str2	strdigcmp(“1234”, EQ, “1256”)	说明
“1234”, EQ, “1256”	false	1234 不等于 1256

10.38 strfind 函数

从指定位置开始在字符串中搜索特定字符集的字符。

语法

`strfind(str, chpos, set, notinset)`

- 参数 str: 字符串, 被搜索的字符串。
- 参数 chpos: num 数据类型，指定开始字符串搜索的开始位置。
- 参数 set: 指定字符集，包含 STR_DIGIT、STR_UPPER、STR_LOWER、STR_WHITE。
- 参数 notinset：bool, 搜索 str 中不在 set 字符集合中的字符位置。
- 返回值类型：num, 返回符合要求的字符位置。

例子

下面是一个简单的示例：

```
v1 = strfind("Robotics",1,"aeiou");
v2 = strfind("Robotics",1,"aeiou"\NotInSet);
v3 = strfind("IRB 6400",1,STR_DIGIT);
v4 = strfine("IRB 6400",1,STR_WHITE);
```

str, chpos, set, notinset	strfind(str, chpos, set, notinset)
“Robotics” ,1,” aeiou”	2
“Robotics” ,1,” aeiou” NotInSet	1
“IRB 6400” ,1,STR_DIGIT	5
“IRB 6400” ,1,STR_WHITE	4

10.39 strlen 函数

计算字符串长度。

语法

`strlen(str)`

- 参数 str: 字符串类型。
- 返回值类型：num, 返回字符串长度。

例子

下面是一个简单的示例：

```
v1 = strlen(str);
```

str1	strlen(str)
“Robotics”	8

10.40 strmap 函数

用于创建字符串的副本，其中所有字符都根据指定的映射进行翻译。

语法

`strmap(str, frommap, tomap)`

- 参数 str: 字符串类型。
- 参数 frommap: 字符串类型，指定映射前的字符类型。
- 参数 tomap: 字符串类型，指定映射后的字符类型。
- 返回值类型：string, 返回 str 经过映射变换后的字符串。

例子

下面是一个简单的示例：

```
v1 = strmap("Robotics", "aeiou", "AEIOU");
v2 = strmap("Robotics",STR_LOWER, STR_UPPER);
```

str, frommap, tomap	strmap(str, frommap, tomap)
“Robotics” , “aeiou” , “AEIOU”	RObOtIcs
“Robotics” ,STR_LOWER, STR_UPPER	ROBOTICS

10.41 strmatch 函数

从指定位置开始, 在字符串中搜索指定的模式。

语法

`strmatch(str, chpos, pattern);`

- 参数 str: 字符串类型。
- 参数 chpos: 指定开始位置，从 str 的第 chpos 位置开始搜索。
- 参数 pattern: 字符串类型，被搜索的字符串。
- 返回值类型：num, 返回 pattern 在 str 中的位置。

例子

下面是一个简单的示例：

```
v1 = strmatch("Robotics", 1, "bo");
```

str, chpos, pattern	strmatch(str, chpos, set)	说明
“Robotics” , 1, “bo”	3	在 str 中第 3 个字符的位置找到了 pattern 字符串

10.42 strmem 函数

检查字符串中的指定字符是否属于另一字符串。

语法

`strmem(str, chpos, set)`

- 参数 str: 字符串类型。
- 参数 chpos: 指定选取 str 中的字符位置。
- 参数 set: 字符串类型，指定一组字符。
- 返回值类型：bool, str 中 chpos 位置的字符包含在 set 中。

例子

下面是一个简单的示例：

```
v1 = strmem("Robotics", 2, "aeiou");
v2 = strmem("Robotics", 3, "aeiou");
v3 = strmem("S-721 68 VÄSTERÅS", 3, STR_DIGIT");
```

str, chpos, set	strmem(str, chpos, set)	说明
“Robotics” , 2, “aeiou”	true	在 set 中可以找到 str 的二个字符 o
“Robotics” , 3, “aeiou”	false	在 set 中不能找到 str 的二个字符 b
“S-721 68 VÄSTERÅS” , 3, STR_DIGIT”	true	在 set 中包含 str 的二个字符 7

10.43 strorder 函数

比较两个字符串，判断两个字符串是否按照指定的字符顺序排列。

语法

`strorder(str1,str2,order)`

- 参数 str1: 字符串类型。
- 参数 str2: 字符串类型。
- 参数 order: 指定字符序列类型，包含 STR_DIGIT、STR_UPPER、STR_LOWER、STR_WHITE。
- 返回值类型：bool, 当 str1 <= str2 返回 true, 否则是 false。

例子

下面是一个简单的示例：

```
v1 = strorder("FIRST","SECOND",STR_UPPER);
v2 = strorder("FIRST","FIRSTB",STR_UPPER);
v3 = strorder("FIRSTB","FIRST",STR_UPPER);
```

str1,str2,order	strorder(str1,str2,order)
“FIRST” ,” SECOND” ,STR_UPPER	true
“FIRST” ,” FIRSTB” ,STR_UPPER	true
“FIRSTB” ,” FIRST” ,STR_UPPER	false

10.44 strpart 函数

查找字符串的子串作为新的字符串。

语法

`strpart(str,chpos,len)`

- 参数 str: 字符串类型。
- 参数 chpos: 子字符串的起始位置。
- 参数 len: num 类型，子字符串的长度。
- 返回值类型：string, 字符串。

例子

下面是一个简单的示例：

```
v1 = strpart("Robotics",1,5);
```

str,chpos,len	strpart(str,chpos,len)
“Robotics” ,1,5	Robot

10.45 strtobyte 函数

将字符串转换为字节数。

语法

```
strtobyte(constr,hex|okt|bin|char);
```

- 参数 constr：字符串类型。
- 参数 hex：指定 constr 为十六进制。
- 参数 okt：指定 constr 为八进制。
- 参数 bin：指定 constr 为二进制。
- 参数 char：指定 constr 为 ASCII 字符。
- 返回值类型：byte, 十进制数。

例子

下面是一个简单的示例：

```
v0 = StrToByte("10");
v1 = strtobyte("AE"", Hex = 1);
v2 = strtobyte("176", Okt = 1);
v3 = strtobyte("00001010", Bin = 1);
v4 = StrToByte("A", Char = 1);
```

constr,hex|okt|bin|char	strtoval(str,val)
10	10
“AE” “, Hex = 1	174
“176” , Okt = 1	126
“00001010” , Bin = 1	10
“A” , Char = 1	65

10.46 strtoval 函数

将字符串转换为任何数据类型的值。

语法

```
strtoval(str,val)
```

- 参数 str：字符串类型。
- 参数 val：任何一种有效的数据类型。

- 返回值类型：bool, 转换成功返回 true, 转换失败返回 false。

例子

下面是一个简单的示例：

```
v1 = strtoval("daf\"", pos);
v2 = strtoval("{2, 2, 2}", pos);
```

str,val	strtoval(str,val)
“daf” “, pos	false
“{2, 2, 2}” , pos	true

10.47 tan 函数

计算正切值。

语法

`tan(angle)`

- 参数 angle：num 类型，角度值用度表示。
- 返回值类型：num。

例子

下面是一个简单的示例：

```
v1 = tan(0);
v2 = tan(45);
```

angle	tan(angle)
0	0
45	1

10.48 testandset 函数

可以与 bool 类型的对象一起使用，作为二进制信号量，检索特定快速代码区域或系统资源的独占权。可以在不同的程序任务和级别间使用。

语法

`testandset(object)`

- 参数类型：bool。

- 返回值类型：bool。

例子

下面是一个简单的示例：

```
v1 = testandset(false);
v2 = testandset(true);
```

object	testandset(object)	object
false	true	true
true	false	true

10.49 trunc 函数

用于将数值截断为指定数量的小数或整数值。

语法

`trunc(val,dec)`

- 参数 val：num 类型，正数、负数或者小数。
- 参数 dec：num 类型, 正数。
- 返回值类型：num，返回值是一个非负数。

例子

下面是一个简单的示例：

```
v1 = trunc(0.3852138,3);
v2 = trunc(0.3852138,1);
v3 = trunc(0.3852138);
v4 = trunc(0.3852138,6);
```

val,dec	trunc(val,dec)
0.3852138,3	0.385
0.3852138,1	0.3
0.3852138	0
0.3852138,6	0.385213

10.50 type 函数

用于获取指定变量的数据类型。

语法

`type(data,basename)`

- 参数 data：任何类型的数据。
- 参数 basename：bool 类型。
- 返回值类型：string，返回值是一个字符串。

说明

当使用 basename 时，指定的数据类型是一种数据类型的别名时，函数将返回基础数据类型名称：

- 参数类型：intnum 类型是 num 的别名。

例子

下面是一个简单的示例：

```
function void main()
  var pos p1;
  var num a;
  var string str;

  print(type(p1));
  print(type(a));
  print(type(str));
end
```

本示例会依次打印相应的数据类型名 `pos` 、`num` 、`string` 。

Chapter 11

cookbook

cookbook 是为了 Tenon 的初学者和使用者更加快速和便捷的实现所需要的功能，而提供的使用 Tenon 实现的算法实例。目前，cookbook 的内容有两个部分：一个部分位于源码的 doc/cookbook 目录下；另外一个部分，位于源码的 testsuite/cookbook 目录下。这两个部分并没有实质性的差别，只是 testsuite/cookbook 目录下以测试用例的形式进行了展现。

截止目前为止，cookbook 共包含了 29 个算法实例，覆盖了数组、队列、堆、栈、哈希表、排序和图的相关算法。在此之外，也提供了一些常用的经典算法。

11.1 数组

数组作为基本的数据结构，在任何一门语言之中，都有着相当重要的作用。Tenon 也不例外，本部分将分别对 cookbook 中一维数组、二维数组和三维数组的算法实例进行介绍，并且会附上算法实例的代码。值得注意的是，Tenon 的数组下标是从 1 开始，不同于 C 语言等从 0 开始，这点需要特别注意。

11.1.1 一维数组

一维数组的算法实例包括 array_1dim_init.t、array_1dim_search.t 和 array_1dim_update.t，分别演示了一维数组的初始化、查找和更新。

array_1dim_init.t

array_1dim_init.t 实现了一维数组的初始化，包含了使用常量初始化和使用变量初始化。具体来说，array_1dim_init.t 中声明了 6 个数组，分别对 num、string、pos 三种类型各声明了两个数组。之后，针对每种类型的两个数组，分别采用了常量和变量进行初始化。具体源码如下：

```
module array
  function void main()
```

(continues on next page)

(continued from previous page)

```
var num a{10};
var string b{10};
var pos c{10};
var num d{10};
var string e{10};
var pos f{10};

! init the arrays with the constants
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
b = ["one", "two", "three", "four", "five", "six", "seven",
     "eight", "nine", "ten"];
c = [[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5],
     [0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]];

! traverse the arrays
for i from 1 to 10 do
  tpwrite_num(a{i});
end

for i from 1 to 10 do
  tpwrite_string(b{i});
end

for i from 1 to 10 do
  tpwrite_pos(c{i});
end

! init the arrays with the vars
for i from 1 to 10 do
  d{i} = i;
  f{i} = [0, 0, i];
end

var string str1 = "one";
e{1} = str1;
var string str2 = "two";
e{2} = str2;
var string str3 = "three";
e{3} = str3;
var string str4 = "four";
e{4} = str4;
var string str5 = "five";
e{5} = str5;
```

(continues on next page)

(continued from previous page)

```
    var string str6 = "six";
    e{6} = str6;
    var string str7 = "seven";
    e{7} = str7;
    var string str8 = "eight";
    e{8} = str8;
    var string str9 = "nine";
    e{9} = str9;
    var string str10 = "ten";
    e{10} = str10;


    ! traverse the arrays
    for i from 1 to 10 do
      tpwrite_num(d{i});
    end

    for i from 1 to 10 do
      tpwrite_string(e{i});
    end

    for i from 1 to 10 do
      tpwrite_pos(f{i});
    end

  end
end
```

array_1dim_search.t

array_1dim_search.t 实现了对一维数组的查找，分别对 num、string、pos 三种类型的数组进行了按下标查找和按存储值查找。具体源码如下：

```
module array
  function void main()
    var num a{10};
    var string b{10};
    var pos c{10};

    a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    b = ["one", "two", "three", "four", "five", "six", "seven",
         "eight", "nine", "ten"];
    c = [[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5],
```

(continues on next page)

(continued from previous page)

```
      [0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]];

! search the array by index
var num index = 4;
tpwrite_num(a{4});
tpwrite_string(b{4});
tpwrite_pos(c{4});

! search the num array by value
var num num_value = 4;
var num num_value_index = 0;
for i from 1 to 10 do
  if a{i} == num_value then
    num_value_index = i;
  end
end
if num_value_index == 0 then
  tpwrite("Can't find the value in array.");
else
  tpwrite_num(num_value_index);
end

! search the string array by value
var string string_value = "four";
var num string_value_index = 0;
for i from 1 to 10 do
  if b{i} == string_value then
    string_value_index = i;
  end
end
if string_value_index == 0 then
  tpwrite("Can't find the value in array.");
else
  tpwrite_num(string_value_index);
end

! search the pos array by value
var pos pos_value = [0, 0, 4];
var num pos_value_index = 0;
for i from 1 to 10 do
  if c{i} == pos_value then
    pos_value_index = i;
  end
```

(continues on next page)

(continued from previous page)

```
    end
    if pos_value_index == 0 then
      tpwrite("Can't find the value in array.");
    else
      tpwrite_num(pos_value_index);
    end

  end
end
```

array_1dim_update.t

array_1dim_update.t 实现了对一维数组的更新，分别针对 num、string、pos 三种类型的数组进行了按下标更新和按值更新。具体源码如下：

```
module array
  function void main()
    var num a{10};
    var string b{10};
    var pos c{10};

    a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    b = ["one", "two", "three", "four", "five", "six", "seven",
         "eight", "nine", "ten"];
    c = [[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5],
         [0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]];

    ! update the num array vualue by index
    var num update_num_index = 5;
    var num update_num_value = 15;
    a{update_num_index} = update_num_value;
    tpwrite_num(a{update_num_index});

    ! update the string array value by index
    var num update_string_index = 5;
    var string update_string_value = "fifth";
    b{update_string_index} = update_string_value;
    tpwrite_string(b{update_string_index});

    ! update the pos array vualue by index
    var num update_pos_index = 5;
    var pos update_pos_value = [0, 0, 15];
    c{update_pos_index} = update_pos_value;
```

(continues on next page)

(continued from previous page)

```
  tpwrite_pos(c{update_pos_index});

  ! update the num array value by value
  var num update_num_value_old = 6;
  var num update_num_value_new = 16;
  var num index_temp = 0;
  for i from 1 to 10 do
    if a{i} == update_num_value_old then
      a{i} = update_num_value_new;
      index_temp = i;
    end
  end
  tpwrite_num(a{index_temp});

  ! update the string array value by value
  var string update_string_value_old = "six";
  var string update_string_value_new = "sixteen";
  index_temp = 0;
  for i from 1 to 10 do
    if b{i} == update_string_value_old then
      b{i} = update_string_value_new;
      index_temp = i;
    end
  end
  tpwrite_string(b{index_temp});

  ! update the pos array value by value
  var pos update_pos_value_old = [0, 0, 6];
  var pos update_pos_value_new =  [0, 0, 16];
  index_temp = 0;
  for i from 1 to 10 do
    if c{i} == update_pos_value_old then
      c{i} = update_pos_value_new;
      index_temp = i;
    end
  end
  if index_temp == 0 then
    tpwrite("Can't find the value in array.");
  else
    tpwrite_pos(c{index_temp});
  end

end
```

(continues on next page)

(continued from previous page)

```
end
```

11.1.2 二维数组

二维数组的算法实例包括 array_2dim_init.t、array_2dim_search.t 和 array_2dim_update.t，分别演示了二维数组的初始化、查找和更新。

array_2dim_init.t

array_2dim_init.t 分别针对 num、string、pos 三种类型的二维数组（2X5），进行了常量初始化和变量初始化。具体源码如下：

```
module array
  function void main()
    var num a{2, 5};
    var string b{2, 5};
    var pos c{2, 5};
    var num d{2, 5};
    var string e{2, 5};
    var pos f{2, 5};

    ! init the arrays with the constants
    a = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]];
    b = [["one", "two", "three", "four", "five"],
         ["six", "seven", "eight", "nine", "ten"]];
    c = [[[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5]],
         [[0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]]];

    ! traverse the arrays
    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_num(a{i, j});
      end
    end

    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_string(b{i, j});
      end
    end
```

(continues on next page)

(continued from previous page)

```
    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_pos(c{i, j});
      end
    end

    ! init the arrays with the vars
    for i from 1 to 2 do
      for j from 1 to 5 do
        d{i, j} = (i - 1) * 5 + j;
        f{i, j} = [0, 0, (i - 1) * 5 + j];
      end
    end

    var string str1{5} = ["one", "two", "three", "four", "five"];
    var string str2{5} = ["six", "seven", "eight", "nine", "ten"];
    for i from 1 to 5 do
      e{1, i} = str1{i};
      e{2, i} = str2{i};
    end

    ! traverse the arrays
    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_num(d{i, j});
      end
    end

    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_string(e{i, j});
      end
    end

    for i from 1 to 2 do
      for j from 1 to 5 do
        tpwrite_pos(f{i, j});
      end
    end

  end
end
```

array_2dim_search.t

array_2dim_search.t 实现了对二维数组的查找，分别对 num、string、pos 三种类型的二维数组进行了按下标查找和按值查找。具体源码如下：

```
module array
  function void main()
    var num a{2, 5};
    var string b{2, 5};
    var pos c{2, 5};

    a = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]];
    b = [["one", "two", "three", "four", "five"],
         ["six", "seven", "eight", "nine", "ten"]];
    c = [[[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5]],
         [[0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]]];

    ! search the arrays by index
    var num index1 = 2;
    var num index2 = 3;
    tpwrite_num(a{index1, index2});
    tpwrite_string(b{index1, index2});
    tpwrite_pos(c{index1, index2});

    ! search the num array by value
    var num num_value = 4;
    index1 = 0;
    index2 = 0;
    for i from 1 to 2 do
      for j from 1 to 5 do
        if a{i, j} == num_value then
          index1 = i;
          index2 = j;
        end
      end
    end
    if ((index1 == 0) or (index2 == 0)) then
      tpwrite("Can't find the value in array.");
    else
      tpwrite_num(index1);
      tpwrite_num(index2);
    end

    ! search the string array by value
    var string string_value = "four";
```

(continues on next page)

(continued from previous page)

```
    index1 = 0;
    index2 = 0;
    for i from 1 to 2 do
      for j from 1 to 5 do
        if b{i, j} == string_value then
          index1 = i;
          index2 = j;
        end
      end
    end
    if ((index1 == 0) or (index2 == 0)) then
        tpwrite("Can't find the value in array.");
    else
      tpwrite_num(index1);
      tpwrite_num(index2);
    end

    ! search the pos array by value
    var pos pos_value = [0, 0, 4];
    index1 = 0;
    index2 = 0;
    for i from 1 to 2 do
      for j from 1 to 5 do
        if c{i, j} == pos_value then
          index1 = i;
          index2 = j;
        end
      end
    end
    if ((index1 == 0) or (index2 == 0)) then
      tpwrite("Can't find the value in array.");
    else
      tpwrite_num(index1);
      tpwrite_num(index2);
    end

  end
end
```

array_2dim_update.t

array_2dim_update.t 实现了对二维数组的更新，分别对 num、string、pos 三种类型的二维数组进行了按下标更新和按值更新。具体源码如下：

```
module array
  function void main()
    var num a{2, 5};
    var string b{2, 5};
    var pos c{2, 5};

    a = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]];
    b = [["one", "two", "three", "four", "five"],
         ["six", "seven", "eight", "nine", "ten"]];
    c = [[[0, 0, 1], [0, 0, 2], [0, 0, 3], [0, 0, 4], [0, 0, 5]],
         [[0, 0, 6], [0, 0, 7], [0, 0, 8], [0, 0, 9], [0, 0, 10]]];

    ! update the arrays by index
    var num index1 = 2;
    var num index2 = 3;
    var num update_num_value = 0;
    a{index1, index2} = update_num_value;
    tpwrite_num(a{index1, index2});

    var string update_string_value = "zero";
    b{index1, index2} = update_string_value;
    tpwrite_string(b{index1, index2});

    var pos update_pos_value = [0, 0, 0];
    c{index1, index2} = update_pos_value;
    tpwrite_pos(c{index1, index2});

    ! update the num array by value
    var num num_value_old = 5;
    var num num_value_new = 15;
    index1 = 0;
    index2 = 0;
    for i from 1 to 2 do
      for j from 1 to 5 do
        if a{i, j} == num_value_old then
          a{i, j} = num_value_new;
          index1 = i;
          index2 = j;
        end
      end
    end
    if ((index1 == 0) or (index2 == 0)) then
      tpwrite("Can't find the value in array.");
    else
```

(continues on next page)

(continued from previous page)

```
  tpwrite_num(a{index1, index2});
end

! update the string array by value
var string string_value_old = "five";
var string string_value_new = "fifth";
index1 = 0;
index2 = 0;
for i from 1 to 2 do
  for j from 1 to 5 do
    if b{i, j} == string_value_old then
      b{i, j} = string_value_new;
      index1 = i;
      index2 = j;
    end
  end
end
if ((index1 == 0) or (index2 == 0)) then
  tpwrite("Can't find the value in array.");
else
  tpwrite_string(b{index1, index2});
end

! search the pos array by value
var pos pos_value_old = [0, 0, 5];
var pos pos_value_new = [0, 0, 15];
index1 = 0;
index2 = 0;
for i from 1 to 2 do
  for j from 1 to 5 do
    if c{i, j} == pos_value_old then
      index1 = i;
      index2 = j;
      c{i, j} = pos_value_new;
    end
  end
end
if ((index1 == 0) or (index2 == 0)) then
  tpwrite("Can't find the value in array.");
else
  tpwrite_pos(c{index1, index2});
end
```

(continues on next page)

(continued from previous page)

```
  end
end
```

11.1.3 三维数组

三维数组的算法实例包括 array_3dim_init.t、array_3dim_search.t 和 array_3dim_update.t，分别演示了三维数组的初始化、查找和更新。

array_3dim_init.t

array_3dim_init.t 实现了对三维数组的初始化，分别针对 num、string、pos 三种类型的三维数组使用常量和变量进行初始化。具体源码如下：

```
module array
  function void main()
    var num a{2, 2, 2};
    var string b{2, 2, 2};
    var pos c{2, 2, 2};
    var num d{2, 2, 2};
    var string e{2, 2, 2};
    var pos f{2, 2, 2};

    ! init the arrays with the constants
    a = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]];
    b = [[["one", "two"], ["three", "four"]],
         [["five", "six"], ["seven", "eight"]]];
    c = [[[[0, 0, 1], [0, 0, 2]], [[0, 0, 3], [0, 0, 4]]],
         [[[0, 0, 5],[0, 0, 6]], [[0, 0, 7], [0, 0, 8]]]];

    ! traverse the arrays
    for i from 1 to 2 do
      for j from 1 to 2 do
        for k from 1 to 2 do
          tpwrite_num(a{i, j, k});
        end
      end
    end

    for i from 1 to 2 do
      for j from 1 to 2 do
        for k from 1 to 2 do
          tpwrite_string(b{i, j, k});
```

(continues on next page)

(continued from previous page)

```
    end
  end
end

for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      tpwrite_pos(c{i, j, k});
    end
  end
end

! init the arrays with the vars
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      d{i, j, k} = (i - 1) * 4 + (j - 1) * 2 + k;
      f{i, j, k} = [0, 0, (i - 1) * 4 + (j - 1) * 2 + k];
    end
  end
end

var string str1{2, 2} = [["one", "two"], ["three", "four"]];
var string str2{2, 2} = [["five", "six"], ["seven", "eight"]];
for i from 1 to 2 do
  for j from 1 to 2 do
    e{1, i, j} = str1{i, j};
    e{2, i, j} = str2{i, j};
  end
end

! traverse the arrays
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      tpwrite_num(d{i, j, k});
    end
  end
end

for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
```

(continues on next page)

(continued from previous page)

```
          tpwrite_string(e{i, j, k});
        end
      end
    end

    for i from 1 to 2 do
      for j from 1 to 2 do
        for k from 1 to 2 do
          tpwrite_pos(f{i, j, k});
        end
      end
    end

  end
end
```

array_3dim_search.t

array_3dim_search.t 实现了对三维数组的查找，分别对 num、string、pos 三种类型的三维数组进行了按下标查找和按值查找。具体源码如下：

```
module array
  function void main()
    var num a{2, 2, 2};
    var string b{2, 2, 2};
    var pos c{2, 2, 2};

    ! init the arrays with the constants
    a = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]];
    b = [[["one", "two"], ["three", "four"]],
         [["five", "six"], ["seven", "eight"]]];
    c = [[[[0, 0, 1], [0, 0, 2]], [[0, 0, 3], [0, 0, 4]]],
         [[[0, 0, 5],[0, 0, 6]], [[0, 0, 7], [0, 0, 8]]]];

    ! search the arrays by index
    var num index1 = 2;
    var num index2 = 2;
    var num index3 = 1;
    tpwrite_num(a{index1, index2, index3});
    tpwrite_string(b{index1, index2, index3});
    tpwrite_pos(c{index1, index2, index3});

    ! search the num array by value
```

(continues on next page)

(continued from previous page)

```
var num num_value = 6;
index1 = 0;
index2 = 0;
index3 = 0;
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      if a{i, j, k} == num_value then
        index1 = i;
        index2 = j;
        index3 = k;
      end
    end
  end
end
if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
  tpwrite("Can't find the value in array.");
else
  tpwrite_num(index1);
  tpwrite_num(index2);
  tpwrite_num(index3);
end

! search the string array by value
var string string_value = "six";
index1 = 0;
index2 = 0;
index3 = 0;
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      if b{i, j, k} == string_value then
        index1 = i;
        index2 = j;
        index3 = k;
      end
    end
  end
end
if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
  tpwrite("Can't find the value in array.");
else
  tpwrite_num(index1);
```

(continues on next page)

(continued from previous page)

```
      tpwrite_num(index2);
      tpwrite_num(index3);
    end

    ! search the pos array by value
    var pos pos_value = [0, 0, 6];
    index1 = 0;
    index2 = 0;
    index3 = 0;
    for i from 1 to 2 do
      for j from 1 to 2 do
        for k from 1 to 2 do
          if c{i, j, k} == pos_value then
            index1 = i;
            index2 = j;
            index3 = k;
          end
        end
      end
    end
    if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
      tpwrite("Can't find the value in array.");
    else
      tpwrite_num(index1);
      tpwrite_num(index2);
      tpwrite_num(index3);
    end

  end
end
```

array_3dim_update.t

array_3dim_update.t 实现了对三维数组的更新，分别对 num、string、pos 三种类型的三维数组进行了按下标更新和按值更新。具体源码如下：

```
module array
  function void main()
    var num a{2, 2, 2};
    var string b{2, 2, 2};
    var pos c{2, 2, 2};

    ! init the arrays with the constants
```

(continues on next page)

(continued from previous page)

```
a = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]];
b = [[["one", "two"], ["three", "four"]],
     [["five", "six"], ["seven", "eight"]]];
c = [[[[0, 0, 1], [0, 0, 2]], [[0, 0, 3], [0, 0, 4]]],
     [[[0, 0, 5],[0, 0, 6]], [[0, 0, 7], [0, 0, 8]]]];

! update the arrays by index
var num index1 = 2;
var num index2 = 2;
var num index3 = 1;

var num update_num_value = 0;
a{index1, index2, index3} = update_num_value;
tpwrite_num(a{index1, index2, index3});

var string update_string_value = "zero";
b{index1, index2, index3} = update_string_value;
tpwrite_string(b{index1, index2, index3});

var pos update_pos_value = [0, 0, 0];
c{index1, index2, index3} = update_pos_value;
tpwrite_pos(c{index1, index2, index3});

! update the num array by value
var num num_value_old = 8;
var num num_value_new = 18;
index1 = 0;
index2 = 0;
index3 = 0;
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      if a{i, j, k} == num_value_old then
        a{i, j, k} = num_value_new;
        index1 = i;
        index2 = j;
        index3 = k;
      end
    end
  end
end
if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
  tpwrite("Can't find the value in array.");
```

(continues on next page)

(continued from previous page)

```
else
  tpwrite_num(a{index1, index2, index3});
end

! update the string array by value
var string string_value_old = "eight";
var string string_value_new = "eighteen";
index1 = 0;
index2 = 0;
index3 = 0;
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      if b{i, j, k} == string_value_old then
        b{i, j, k} = string_value_new;
        index1 = i;
        index2 = j;
        index3 = k;
      end
    end
  end
end
if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
  tpwrite("Can't find the value in array.");
else
  tpwrite_string(b{index1, index2, index3});
end

! update the pos array by value
var pos pos_value_old = [0, 0, 8];
var pos pos_value_new = [0, 0, 18];
index1 = 0;
index2 = 0;
index3 = 0;
for i from 1 to 2 do
  for j from 1 to 2 do
    for k from 1 to 2 do
      if c{i, j, k} == pos_value_old then
        index1 = i;
        index2 = j;
        index3 = k;
        c{i, j, k} = pos_value_new;
      end
```

(continues on next page)

(continued from previous page)

```
        end
      end
    end
    if ((index1 == 0) or (index2 == 0) or (index3 == 0)) then
      tpwrite("Can't find the value in array.");
    else
      tpwrite_pos(c{index1, index2, index3});
    end

  end
end
```

11.2 队列

队列作为一种抽象的数据结构，在实际运用中有很多的场景。cookbook 中包括了对 num、string、pos、自定义 record 类型的队列的 Tenon 实现，并且针对队列的入队、出队、判空、判满和清空等操作，都以函数的方式抽取出来实现。cookbook 中共有 queue_num.t、queue_string.t、queue_pos.t、queue_record.t 四个队列相关的算法实例，都是基于数组进行实现的，其最大容量设定为 100（头和尾不占用这个空间），在具体使用中可根据实际情况进行调整。

11.2.1 queue_num.t

queue_num.t 是针对 num 类型的队列算法实例。它针对 num 类型进行了队列的构建，并且对入队、出队、判空、判满和清空等操作都进行了实现和调用。具体源码如下：

```
module queue_num

  ! define the max num of queue(contain front and rear)
  const num queue_max_size = 102;
  ! define the queue size
  var num size = 0;

  var num queue{queue_max_size};
  var num front = 1;
  var num rear = 2;

  function void main()
    ! judge the queue is empty or not
    var bool bEmpty = isEmpty();
    if (bEmpty == true) then
      tpwrite("The queue is empty.");
```

(continues on next page)

(continued from previous page)

```
  else
    tpwrite("The queue is not empty.");
  end

  for i from 1 to 10 do
    en_queue(i);
  end
  output();

  for i from 1 to 5 do
    de_queue();
  end
  output();

  for i from 1 to 100 do
    en_queue(i);
  end
  output();

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The queue is full.");
  else
    tpwrite("The queue is not full.");
  end

  clear_queue();
  output();

  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end
end


function bool isEmpty()
  if (size == 0) then
    return true;
```

(continues on next page)

(continued from previous page)

```
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void en_queue(num en_num)
  if (size == 100) then
    tpwrite("The queue is full.");
    return;
  end

  if ((size < 100) and (rear == 102)) then
    queue{rear} = en_num;
    rear = 1;
    size =  size + 1;
    return;
  end

  queue{rear} = en_num;
  rear = rear + 1;
  size = size + 1;
end

function void de_queue()
  if (size == 0) then
    tpwrite("The queue is empty.");
    return;
  end

  if ((size < 100) and (front == 102)) then
    front = 1;
    size =  size - 1;
    return;
  end
```

(continues on next page)

(continued from previous page)

```
    front = front + 1;
    size = size - 1;
  end

  function void output()
    tpwrite_num(front);
    tpwrite_num(rear);
    if (size == 0) then
      tpwrite("The queue is empty.");
      return;
    end
    tpwrite_num(queue{front + 1});
    tpwrite_num(queue{rear - 1});
  end

  function void clear_queue()
    size = 0;
    front = 1;
    rear = 2;
  end

end
```

11.2.2 queue_string.t

queue_string.t 是针对 string 类型的队列算法实例。它针对 string 类型进行了队列的构建，并且对入队、出队、判空、判满和清空等操作都进行了实现和调用。具体源码如下：

```
module queue_string

  ! define the max num of queue(contain front and rear)
  const num queue_max_size = 102;
  ! define the queue size
  var num size = 0;

  var string queue{queue_max_size};
  var num front = 1;
  var num rear = 2;

  function void main()
    ! judge the queue is empty or not
    var bool bEmpty = isEmpty();
```

(continues on next page)

(continued from previous page)

```
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end

  for i from 1 to 10 do
    en_queue("hello world");
  end
  output();

  for i from 1 to 5 do
    de_queue();
  end
  output();

  for i from 1 to 100 do
    en_queue("test");
  end
  output();

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The queue is full.");
  else
    tpwrite("The queue is not full.");
  end

  clear_queue();
  output();

  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end
end


function bool isEmpty()
```

(continues on next page)

(continued from previous page)

```
  if (size == 0) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void en_queue(string en_str)
  if (size == 100) then
    tpwrite("The queue is full.");
    return;
  end

  if ((size < 100) and (rear == 102)) then
    queue{rear} = en_str;
    rear = 1;
    size =  size + 1;
    return;
  end

  queue{rear} = en_str;
  rear = rear + 1;
  size = size + 1;
end

function void de_queue()
  if (size == 0) then
    tpwrite("The queue is empty.");
    return;
  end

  if ((size < 100) and (front == 102)) then
    front = 1;
    size =  size - 1;
    return;
```

(continues on next page)

(continued from previous page)

```
    end

    front = front + 1;
    size = size - 1;
  end

  function void output()
    tpwrite_num(front);
    tpwrite_num(rear);
    if (size == 0) then
      tpwrite("The queue is empty.");
      return;
    end
  end

  function void clear_queue()
    size = 0;
    front = 1;
    rear = 2;
  end

end
```

11.2.3 queue_pos.t

queue_pos.t 是针对 pos 类型的队列算法实例。它针对 pos 类型进行了队列的构建，并且对入队、出队、判空、判满和清空等操作都进行了实现和调用。具体源码如下：

```
module queue_pos

  ! define the max num of queue(contain front and rear)
  const num queue_max_size = 102;
  ! define the queue size
  var num size = 0;

  var pos queue{queue_max_size};
  var num front = 1;
  var num rear = 2;

  function void main()
    ! judge the queue is empty or not
    var bool bEmpty = isEmpty();
```

(continues on next page)

(continued from previous page)

```
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end

  for i from 1 to 10 do
    en_queue([0, 0, i]);
  end
  output();

  for i from 1 to 5 do
    de_queue();
  end
  output();

  for i from 1 to 100 do
    en_queue([0, 0, i]);
  end
  output();

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The queue is full.");
  else
    tpwrite("The queue is not full.");
  end

  clear_queue();
  output();

  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end
end


function bool isEmpty()
```

(continues on next page)

(continued from previous page)

```
  if (size == 0) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void en_queue(pos en_pos)
  if (size == 100) then
    tpwrite("The queue is full.");
    return;
  end

  if ((size < 100) and (rear == 102)) then
    queue{rear} = en_pos;
    rear = 1;
    size =  size + 1;
    return;
  end

  queue{rear} = en_pos;
  rear = rear + 1;
  size = size + 1;
end

function void de_queue()
  if (size == 0) then
    tpwrite("The queue is empty.");
    return;
  end

  if ((size < 100) and (front == 102)) then
    front = 1;
    size =  size - 1;
    return;
```

(continues on next page)

(continued from previous page)

```
    end

    front = front + 1;
    size = size - 1;
  end

  function void output()
    tpwrite_num(front);
    tpwrite_num(rear);
    if (size == 0) then
      tpwrite("The queue is empty.");
      return;
    end
    tpwrite_pos(queue{front + 1});
    tpwrite_pos(queue{rear - 1});
  end

  function void clear_queue()
    size = 0;
    front = 1;
    rear = 2;
  end

end
```

11.2.4 queue_record.t

queue_record.t 是针对自定义 record 类型的队列算法实例。它针对自定义 record 类型进行了队列的构建，并且对入队、出队、判空、判满和清空等操作都进行了实现和调用。具体源码如下：

```
module queue_record

  ! define the max num of queue(contain front and rear)
  const num queue_max_size = 102;
  ! define the queue size
  var num size = 0;

  record position
    num no;
    string name;
    pos posdata;
  end
```

(continues on next page)

(continued from previous page)

```
var position queue{queue_max_size};
var num front = 1;
var num rear = 2;

function void main()
  ! judge the queue is empty or not
  var bool bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end

  for i from 1 to 10 do
    en_queue([i, "Lili", [0, 0, i]]);
  end
  output();

  for i from 1 to 5 do
    de_queue();
  end
  output();

  for i from 1 to 100 do
    en_queue([i, "Lili", [0, 0, i]]);
  end
  output();

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The queue is full.");
  else
    tpwrite("The queue is not full.");
  end

  clear_queue();
  output();

  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
```

(continues on next page)

(continued from previous page)

```
    tpwrite("The queue is empty.");
  else
    tpwrite("The queue is not empty.");
  end
end


function bool isEmpty()
  if (size == 0) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void en_queue(position en_posi)
  if (size == 100) then
    tpwrite("The queue is full.");
    return;
  end

  if ((size < 100) and (rear == 102)) then
    queue{rear} = en_posi;
    rear = 1;
    size =  size + 1;
    return;
  end

  queue{rear} = en_posi;
  rear = rear + 1;
  size = size + 1;
end

function void de_queue()
  if (size == 0) then
```

(continues on next page)

(continued from previous page)

```
      tpwrite("The queue is empty.");
      return;
    end

    if ((size < 100) and (front == 102)) then
      front = 1;
      size =  size - 1;
      return;
    end

    front = front + 1;
    size = size - 1;
  end

  function void output()
    tpwrite_num(front);
    tpwrite_num(rear);
    if (size == 0) then
      tpwrite("The queue is empty.");
      return;
    end
    tpwrite_num(queue{front + 1}.no);
    tpwrite_string(queue{front + 1}.name);
    tpwrite_pos(queue{front + 1}.posdata);
    tpwrite_num(queue{rear - 1}.no);
    tpwrite_string(queue{rear - 1}.name);
    tpwrite_pos(queue{rear - 1}.posdata);
  end

  function void clear_queue()
    size = 0;
    front = 1;
    rear = 2;
  end

end
```

11.3 栈

cookbook 中包含了对栈的相关算法实例，它们分别是：stack_num.t、stack_string.t、stack_pos.t 和 stack_record.t。这几个算法实例分别针对 num、string、pos 和自定义 record 类型，实现了栈的相关操作。栈的最大容量设置为 100，在实际的使用中可根据需求调整。

11.3.1 stack_num.t

stack_num.t 针对 num 类型构建了栈，并且对于栈的入栈、出栈、判空和判满，都以函数的形式进行了单独实现，方便在实际使用的过程中参考。具体源码如下：

```
module stack_num

  ! define the max num of stack
  const num stack_max_size = 100;
  ! define the empty tos
  const num empty_tos = 0;

  var num stack{stack_max_size};
  var num top_of_stack = 0;

  function void main()
    ! judge the stack is empty or not
    var bool bEmpty = isEmpty();
    if (bEmpty == true) then
      tpwrite("The stack is empty.");
    else
      tpwrite("The stack is not empty.");
    end

    for i from 1 to 10 do
      push(i);
    end

    for i from 1 to 11 do
      pop();
    end

    for i from 1 to 101 do
      push(i);
    end

    ! judge the stack is full or not
    var bool bFull = isFull();
    if (bFull == true) then
      tpwrite("The stack is full.");
    else
      tpwrite("The stack is not full.");
    end
```

(continues on next page)

(continued from previous page)

```
  end

  function bool isEmpty()
    if (top_of_stack == empty_tos) then
      return true;
    else
      return false;
    end
  end

  function bool isFull()
    if (top_of_stack == stack_max_size) then
      return true;
    else
      return false;
    end
  end

  function void push(num pushnum)
    if (top_of_stack == 100) then
      tpwrite("The stack is full.");
      return;
    end
    top_of_stack = top_of_stack + 1;
    stack{top_of_stack} = pushnum;
    tpwrite_num(stack{top_of_stack});
  end

  function void pop()
    if (top_of_stack == empty_tos) then
      tpwrite("The stack is empty.");
      return;
    end
    tpwrite_num(stack{top_of_stack});
    top_of_stack = top_of_stack - 1;
  end

end
```

11.3.2 stack_string.t

stack_string.t 针对 string 类型构建了栈，并且对于栈的入栈、出栈、判空和判满，都以函数的形式进行了单独实现，方便在实际使用的过程中参考。具体源码如下：

```
module stack_string

  ! define the max num of stack
  const num stack_max_size = 100;
  ! define the empty tos
  const num empty_tos = 0;

  var string stack{stack_max_size};
  var num top_of_stack = 0;

  function void main()
    ! judge the stack is empty or not
    var bool bEmpty = isEmpty();
    if (bEmpty == true) then
      tpwrite("The stack is empty.");
    else
      tpwrite("The stack is not empty.");
    end

    for i from 1 to 10 do
      push("hello");
    end

    for i from 1 to 11 do
      pop();
    end

    for i from 1 to 101 do
      push("world");
    end

    ! judge the stack is full or not
    var bool bFull = isFull();
    if (bFull == true) then
      tpwrite("The stack is full.");
    else
      tpwrite("The stack is not full.");
    end

  end

  function bool isEmpty()
    if (top_of_stack == empty_tos) then
      return true;
```

(continues on next page)

(continued from previous page)

```
    else
      return false;
    end
  end

  function bool isFull()
    if (top_of_stack == stack_max_size) then
      return true;
    else
      return false;
    end
  end

  function void push(string pushstr)
    if (top_of_stack == 100) then
      tpwrite("The stack is full.");
      return;
    end
    top_of_stack = top_of_stack + 1;
    stack{top_of_stack} = pushstr;
    tpwrite_string(stack{top_of_stack});
  end

  function void pop()
    if (top_of_stack == empty_tos) then
      tpwrite("The stack is empty.");
      return;
    end
    tpwrite_string(stack{top_of_stack});
    top_of_stack = top_of_stack - 1;
  end

end
```

11.3.3 stack_pos.t

stack_pos.t 针对 pos 类型构建了栈，并且对于栈的入栈、出栈、判空和判满，都以函数的形式进行了单独实现，方便在实际使用的过程中参考。具体源码如下：

```
module stack_pos

  ! define the max num of stack
```

(continues on next page)

(continued from previous page)

```
const num stack_max_size = 100;
! define the empty tos
const num empty_tos = 0;

var pos stack{stack_max_size};
var num top_of_stack = 0;

function void main()
  ! judge the stack is empty or not
  var bool bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The stack is empty.");
  else
    tpwrite("The stack is not empty.");
  end

  for i from 1 to 10 do
    push([0, 0, i]);
  end

  for i from 1 to 11 do
    pop();
  end

  for i from 1 to 101 do
    push([0, 0, i]);
  end

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The stack is full.");
  else
    tpwrite("The stack is not full.");
  end

end

function bool isEmpty()
  if (top_of_stack == empty_tos) then
    return true;
  else
    return false;
```

(continues on next page)

(continued from previous page)

```
    end
  end

  function bool isFull()
    if (top_of_stack == stack_max_size) then
      return true;
    else
      return false;
    end
  end

  function void push(pos pushpos)
    if (top_of_stack == 100) then
      tpwrite("The stack is full.");
      return;
    end
    top_of_stack = top_of_stack + 1;
    stack{top_of_stack} = pushpos;
    tpwrite_pos(stack{top_of_stack});
  end

  function void pop()
    if (top_of_stack == empty_tos) then
      tpwrite("The stack is empty.");
      return;
    end
    tpwrite_pos(stack{top_of_stack});
    top_of_stack = top_of_stack - 1;
  end

end
```

11.3.4 stack_record.t

stack_record.t 针对自定义 record 类型构建了栈，并且对于栈的入栈、出栈、判空和判满，都以函数的形式进行了单独实现，方便在实际使用的过程中参考。具体源码如下：

```
module stack_record

  ! define the max num of stack
  const num stack_max_size = 100;
  ! define the empty tos
```

(continues on next page)

(continued from previous page)

```
const num empty_tos = 0;

record position
  num no;
  string name;
  pos posdata;
end

var position stack{stack_max_size};
var num top_of_stack = 0;

function void main()
  ! judge the stack is empty or not
  var bool bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The stack is empty.");
  else
    tpwrite("The stack is not empty.");
  end
  record_output();

  for i from 1 to 10 do
    push([i, "world", [0, 0, i]]);
  end
  record_output();

  for i from 1 to 11 do
    pop();
  end
  record_output();

  for i from 1 to 101 do
    push([i, "world", [0, 0, i]]);
  end
  record_output();

  ! judge the stack is full or not
  var bool bFull = isFull();
  if (bFull == true) then
    tpwrite("The stack is full.");
  else
    tpwrite("The stack is not full.");
  end
```

(continues on next page)

(continued from previous page)

```
end

function bool isEmpty()
  if (top_of_stack == empty_tos) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (top_of_stack == stack_max_size) then
    return true;
  else
    return false;
  end
end

function void record_output()
  if (top_of_stack == empty_tos) then
    tpwrite("The stack is empty, can't output data.");
    return;
  end

  tpwrite_num(stack{top_of_stack}.no);
  tpwrite_string(stack{top_of_stack}.name);
  tpwrite_pos(stack{top_of_stack}.posdata);
end



function void push(position pushpos)
  if (top_of_stack == 100) then
    tpwrite("The stack is full.");
    return;
  end
  top_of_stack = top_of_stack + 1;
  stack{top_of_stack} = pushpos;
end

function void pop()
  if (top_of_stack == empty_tos) then
    tpwrite("The stack is empty.");
```

(continues on next page)

(continued from previous page)

```
      return;
    end
    top_of_stack = top_of_stack - 1;
  end

end
```

11.4 堆

堆作为一种抽象数据结构，其实现方式有多种。cookbook 采用数组去实现完全二叉树架构的二叉堆，实现了最大堆和最小堆，对堆的基本操作也都做了具体实现。同时，对最大堆和最小堆也都实现了相关的堆排序。值得指出的是，二叉堆的插入是在最尾端，删除是在最顶端。

11.4.1 heap_min.t

heap_min.t 实现了最小堆的构建，并在最小堆上实现了插入和删除节点。最后，还实现了基于最小堆的堆排序（递归）。具体源码如下：

```
module heap_min

  const num heap_max_size = 100;
  ! define the heap size
  var num size = 0;

  var num heap{heap_max_size};
  var num front = 1;
  var num rear = 1;

  function void main()
    ! judge the heap is empty or not
    var bool bEmpty = isEmpty();
    if (bEmpty == true) then
      tpwrite("The heap is empty.");
    else
      tpwrite("The heap is not empty.");
    end

    insert_num(1, 20);
    insert_num(2, 2);
    insert_num(3, 35);
    insert_num(4, 46);
```

(continues on next page)

(continued from previous page)

```
insert_num(5, 22);

for i from 1 to 5 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");

delete_num();
delete_num();

for i from 1 to 3 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");

insert_num(4, 20);
insert_num(5, 2);
insert_num(6, 70);
insert_num(7, 26);
insert_num(8, 110);
insert_num(9, 42);
insert_num(10, 120);

for i from 1 to 10 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");



heap_sort();
for i from 1 to 10 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");

for i from 11 to 100 do
  insert_num(i, i);
end

! judge the stack is full or not
var bool bFull = isFull();
if (bFull == true) then
  tpwrite("The heap is full.");
```

(continues on next page)

(continued from previous page)

```
  else
    tpwrite("The heap is not full.");
  end

  clear_heap();

  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The heap is empty.");
  else
    tpwrite("The heap is not empty.");
  end
end



function bool isEmpty()
  if (size == 0) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void insert_num(num index, num value)
  if(index <> size + 1) then
    tpwrite("The insert position is wrong.");
    return;
  end
  heap{index} = value;
  size++;
  min_heap_fixup(index);
end

function void min_heap_fixup(num index)
```

(continues on next page)

(continued from previous page)

```
  var num j = index div 2;
  var num temp = heap{index};
  while ((j >= 1) and (index <> 1)) do
    if (heap{j} < temp) then
      break;
    end
    heap{index} = heap{j};
    index = j;
    j = index div 2;
  heap{index} = temp;
  end
end

function void delete_num()
  heap{1} = heap{size};
  size--;
  min_heap_fixdown(1, size);
end

function void min_heap_fixdown(num i,num size)
  var num j = 2 * i;
  var num temp = heap{i};
  while (j <= size) do
    if (((j + 1) <= size) and (heap{j + 1} < heap{j})) then
      j++;
    end
    if (heap{j} > temp) then
      break;
    end
    heap{i} = heap{j};
    i = j;
    j = 2 * i;
  end
  heap{i} = temp;
end

function void heap_sort()
  for i from size to 2 step -1 do
    var num temp = heap{i};
    heap{i} = heap{1};
    heap{1} = temp;
    min_heap_fixdown(1, i - 1);
  end
```

(continues on next page)

(continued from previous page)

```
  end

  function void clear_heap()
    size = 0;
  end

end
```

11.4.2 heap_max.t

heap_max.t 实现了最大堆的构建，并在最大堆上实现了插入和删除节点。最后，还实现了基于最大堆的堆排序（递归）。具体源码如下：

```
module heap_max

  const num heap_max_size = 100;
  ! define the heap size
  var num size = 0;

  var num heap{heap_max_size};
  var num front = 1;
  var num rear = 1;

  function void main()
    ! judge the heap is empty or not
    var bool bEmpty = isEmpty();
    if (bEmpty == true) then
      tpwrite("The heap is empty.");
    else
      tpwrite("The heap is not empty.");
    end

    insert_num(1, 20);
    insert_num(2, 2);
    insert_num(3, 35);
    insert_num(4, 46);
    insert_num(5, 22);

    for i from 1 to 5 do
      tpwrite_num(heap{i});
    end
    tpwrite("-------------");
```

(continues on next page)

(continued from previous page)

```
delete_num();
delete_num();

for i from 1 to 3 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");

insert_num(4, 80);
insert_num(5, 62);
insert_num(6, 70);
insert_num(7, 26);
insert_num(8, 110);
insert_num(9, 42);
insert_num(10, 120);

for i from 1 to 10 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");


heap_sort();
for i from 1 to 10 do
  tpwrite_num(heap{i});
end
tpwrite("-------------");

for i from 11 to 100 do
  insert_num(i, i);
end

! judge the stack is full or not
var bool bFull = isFull();
if (bFull == true) then
  tpwrite("The heap is full.");
else
  tpwrite("The heap is not full.");
end

clear_heap();
```

(continues on next page)

(continued from previous page)

```
  bEmpty = false;
  bEmpty = isEmpty();
  if (bEmpty == true) then
    tpwrite("The heap is empty.");
  else
    tpwrite("The heap is not empty.");
  end
end


function bool isEmpty()
  if (size == 0) then
    return true;
  else
    return false;
  end
end

function bool isFull()
  if (size == 100) then
    return true;
  else
    return false;
  end
end

function void insert_num(num index, num value)
  if(index <> size + 1) then
    tpwrite("The insert position is wrong.");
    return;
  end
  heap{index} = value;
  size++;
  max_heap_fixup(index);
end

function void max_heap_fixup(num index)
  var num j = index div 2;
  var num temp = heap{index};
  while ((j >= 1) and (index <> 1)) do
    if (heap{j} > temp) then
      break;
    end
```

(continues on next page)

(continued from previous page)

```
      heap{index} = heap{j};
      index = j;
      j = index div 2;
    heap{index} = temp;
    end
  end

  function void delete_num()
    heap{1} = heap{size};
    size--;
    max_heap_fixdown(1, size);
  end

  function void max_heap_fixdown(num i,num size)
    var num j = 2 * i;
    var num temp = heap{i};
    while (j <= size) do
      if (((j + 1) <= size) and (heap{j + 1} > heap{j})) then
        j++;
      end
      if (heap{j} < temp) then
        break;
      end
      heap{i} = heap{j};
      i = j;
      j = 2 * i;
    end
    heap{i} = temp;
  end

  function void heap_sort()
    for i from size to 2 step -1 do
      var num temp = heap{i};
      heap{i} = heap{1};
      heap{1} = temp;
      max_heap_fixdown(1, i - 1);
    end
  end

  function void clear_heap()
    size = 0;
  end

end
```

11.5 哈希表

哈希表作为一种抽象的数据结构，其实现方式有多种。cookbook 采用了二维数组来实现哈希表，实现了 hash_table.t 和 rehash_table.t 两个实例，并将其空间设置为 10X10 个元素空间。这两个实例中的元素，采用自定义的 record 类型，是由一个 num 类型和一个 string 组成，在使用的过程中可根据实际需求自己定义对应的数据类型。

11.5.1 hash_table.t

hash_table.t 不仅对哈希表进行了构建，还对插入、删除、查找哈希表的函数进行了单独实现。其中，为了让实例更加简单明了，哈希函数只是简单的对 10 求 mod，并且加了个 1，将其结果的范围限制在从 1 到 10，好直接根据哈希函数的结果去找对应的数组下标，这也跟 Tenon 数组的下标从 1 开始有关。具体源码如下：

```
module hash

  record hash_item
    num key;
    string content;
  end

  ! create and init the hash table
  var hash_item hash_table{10, 10};


  function void main()
  for i from 1 to 10 do
    for j from 1 to 10 do
      hash_table{i, j} = [-1, "null"];
    end
  end

    var hash_item item = [11, "eleven"];
    if (insert_item(item) == true) then
      tpwrite("Insert item success.");
    end;
    item = [12, "twelve"];
    if (insert_item(item) == true) then
      tpwrite("Insert item success.");
    end;
    for i from 1 to 10 do
      if (insert_item([i, "hash"]) == true) then
        tpwrite("Insert item success.");
      end;
```

(continues on next page)

(continued from previous page)

```
  end

  item = [6, "hash"];
  if (find_item(item) == true) then
    tpwrite("Find item.");
  end;

  item = [8, "hash"];
  if (delete_item(item) == true) then
    tpwrite("Delete item success.");
  end;

end

function num hash_func(num key)
  return (key mod 10) + 1;
end

function bool insert_item(hash_item item)
  var bool b_insert = false;
  var num index = hash_func(item.key);
  for i from 1 to 10 do
    if ((hash_table{index, i}.key == -1) and
        (hash_table{index, i}.content == "null")) then
      hash_table{index, i}.key = item.key;
      hash_table{index, i}.content = item.content;

      tpwrite_num(hash_table{index, i}.key);
      tpwrite_string(hash_table{index, i}.content);
      b_insert = true;
      break;
    end
  end
  return b_insert;
end

function bool find_item(hash_item item)
  var bool b_find = false;
  var num key_hash = hash_func(item.key);
  for i from 1 to 10 do
    if ((hash_table{key_hash, i}.key == item.key) and
        (hash_table{key_hash, i}.content == item.content)) then
      tpwrite_num(hash_table{key_hash, i}.key);
```

(continues on next page)

(continued from previous page)

```
        tpwrite_string(hash_table{key_hash, i}.content);
        b_find = true;
      end
    end
    return b_find;
  end

  function bool delete_item(hash_item item)
    var bool b_del = false;
    var num key_hash = hash_func(item.key);
    for i from 1 to 10 do
      if ((hash_table{key_hash, i}.key == item.key) and
          (hash_table{key_hash, i}.content == item.content)) then
        hash_table{key_hash, i}.key = -1;
        hash_table{key_hash, i}.content = "null";
        b_del = true;
      end
    end
    return b_del;
  end

end
```

11.5.2 rehash_table.t

rehash_table.t 和 hash_table.t 一样，不仅对哈希表进行了构建，还对插入、删除、查找哈希表的函数进行了单独实现。rehash_table.t 的重点是 rehashing，主要体现在哈希函数里，实现了两个哈希函数，并且通过调用对 key 形成了两次计算的过程。具体源码如下：

```
module rehash

  record hash_item
    num key;
    string content;
  end

  ! create and init the hash table
  var hash_item hash_table{5, 10};


  function void main()
  for i from 1 to 5 do
```

(continues on next page)

(continued from previous page)

```
  for j from 1 to 10 do
    hash_table{i, j} = [-1, "null"];
  end
end

  var hash_item item = [11, "eleven"];
  if (insert_item(item) == true) then
    tpwrite("Insert item success.");
  end;
  item = [12, "twelve"];
  if (insert_item(item) == true) then
    tpwrite("Insert item success.");
  end;
  for i from 1 to 10 do
    if (insert_item([i, "hash"]) == true) then
      tpwrite("Insert item success.");
    end;
  end

  item = [6, "hash"];
  if (find_item(item) == true) then
    tpwrite("Find item.");
  end;

  item = [8, "hash"];
  if (delete_item(item) == true) then
    tpwrite("Delete item success.");
  end;

end

function num hash_func1(num key)
  return (key mod 10) + 1;
end

function num hash_func2(num key)
  return (key mod 5) + 1;
end

function bool insert_item(hash_item item)
  var bool b_insert = false;
  var num index = hash_func2(hash_func1(item.key));
  for i from 1 to 10 do
```

(continues on next page)

(continued from previous page)

```
    if ((hash_table{index, i}.key == -1) and
       (hash_table{index, i}.content == "null")) then
      hash_table{index, i}.key = item.key;
      hash_table{index, i}.content = item.content;

      tpwrite_num(hash_table{index, i}.key);
      tpwrite_string(hash_table{index, i}.content);
      b_insert = true;
      break;
    end
  end
  return b_insert;
end

function bool find_item(hash_item item)
  var bool b_find = false;
  var num key_hash = hash_func2(hash_func1(item.key));
  for i from 1 to 10 do
    if ((hash_table{key_hash, i}.key == item.key) and
        (hash_table{key_hash, i}.content == item.content)) then
      tpwrite_num(hash_table{key_hash, i}.key);
      tpwrite_string(hash_table{key_hash, i}.content);
      b_find = true;
    end
  end
  return b_find;
end

function bool delete_item(hash_item item)
  var bool b_del = false;
  var num key_hash = hash_func2(hash_func1(item.key));
  for i from 1 to 10 do
    if ((hash_table{key_hash, i}.key == item.key) and
        (hash_table{key_hash, i}.content == item.content)) then
      hash_table{key_hash, i}.key = -1;
      hash_table{key_hash, i}.content = "null";
      b_del = true;
    end
  end
  return b_del;
end

end
```

11.6 排序算法

排序算法作为算法中的重要的部分，是任何有关算法的内容都避不开的。前文在堆的部分，已经介绍了堆排序的相关内容。除此之外，cookbook 还实现了插入排序、合并排序、快速排序和希尔排序，分别对应 insertion_sort.t、merge_sort.t、quick_sort.t 和 shell_sort.t。cookbook 中的这四个排序算法，都是对一个包含了 10 个 num 类型数据的一维数组进行排序。

11.6.1 insertion_sort.t

insertion_sort.t 实现的是插入排序。其排序功能在函数 insertion_sort_func() 中实现。具体源码如下：

```
module insertion_sort

  var num a{10};

  function void main()
    a = [18, 2, 34, 55, 67, 12, 1, 88, 79, 50];

    for i from 1 to 10 do
      tpwrite_num(a{i});
    end

    tpwrite("-----------------------");

    insertion_sort_func();
    for i from 1 to 10 do
      tpwrite_num(a{i});
    end
  end

  function void insertion_sort_func()
    var num temp;
    var num j;
    for i from 2 to 10 do
      temp = a{i};
      j = i;
      while ((j > 1) and (a{j - 1} > temp)) do
        a{j} = a{j - 1};
        j = j - 1;
      end
      a{j} = temp;
    end
  end
```

(continues on next page)

(continued from previous page)

```
end
```

11.6.2 merge_sort.t

merge_sort.t 实现了合并排序。其具体排序算法在函数 merge_sort_func() 中实现，此处采用的是递归的实现方式。具体源码如下：

```
module merge_sort

  function void main()
    var num a{10} = [18, 2, 34, 55, 67, 12, 1, 88, 79, 50];
    var num b{10};

    for i from 1 to 10 do
      tpwrite_num(a{i});
    end

    tpwrite("----------------------");

    merge_sort_func(a, 1, 10, b);

    for i from 1 to 10 do
      tpwrite_num(a{i});
    end

  end

  function void merge_sort_func(alias num a{*}, num first, num last, alias num b{*})
    if (first < last) then
      var num mid = (first + last) div 2;
      merge_sort_func(a, first, mid, b);
      merge_sort_func(a, mid + 1, last, b);
      merge_array(a, first, mid, last, b);
    end
  end

  function void merge_array(alias num a{*}, num first, num mid, num last, alias num b{*})
    var num i = first;
    var num j = mid + 1;
    var num m = mid;
    var num n = last;
```

(continues on next page)

(continued from previous page)

```
    var num k = 1;

    while ((i <= m) and (j <= n)) do
      if (a{i} <= a{j}) then
        b{k} = a{i};
        k++;
        i++;
      else
        b{k} = a{j};
        k++;
        j++;
      end
    end

    while (i <= m) do
      b{k} = a{i};
      k++;
      i++;
    end

    while (j <= n) do
      b{k} = a{j};
      k++;
      j++;
    end

    for i from 1 to (k - 1) do
      a{first + i - 1} = b{i};
    end

  end
end
```

11.6.3 quick_sort.t

quick_sort.t 实现了快速排序。排序算法在 quick_sort_func() 函数中实现，此处才用了递归的实现方式。具体源码如下：

```
module quick_sort

  var num a{10};
```

(continues on next page)

(continued from previous page)

```
  function void main()
    a = [18, 2, 34, 55, 67, 12, 1, 88, 79, 50];

    for i from 1 to 10 do
      tpwrite_num(a{i});
    end

    tpwrite("----------------------");

    quick_sort_func(a, 1, 10);
    for i from 1 to 10 do
      tpwrite_num(a{i});
    end
  end

  function void quick_sort_func(alias num a{*}, num first, num last)
    if (first < last) then
      var num i = first;
      var num j = last;
      var num temp = a{i};
      while (i < j) do
        while((i < j) and (a{j} > temp)) do
          j--;
        end
        if (i < j) then
          a{i} = a{j};
          i++;
        end
        while((i < j) and (a{i} < temp)) do
          i++;
        end
        if (i < j) then
          a{j} = a{i};
          j--;
        end
      end
      a{i} = temp;
      quick_sort_func(a, first, i - 1);
      quick_sort_func(a, i + 1, last);
    end
  end

end
```

11.6.4 shell_sort.t

shell_sort.t 实现了希尔排序。排序算法在函数 shell_sort_func() 中实现。具体源码如下：

```
module shell_sort

  var num a{10};

  function void main()
    a = [18, 2, 34, 55, 67, 12, 1, 88, 79, 50];

    for i from 1 to 10 do
      tpwrite_num(a{i});
    end

    tpwrite("------------------------");

    shell_sort_func();
    for i from 1 to 10 do
      tpwrite_num(a{i});
    end
  end

  function void shell_sort_func()
    var num d = 10 div 2;
    while (d >= 1) do
      shell_insert(d);
      d = d div 2;
    end
  end

  function void shell_insert(num d)
    var num index = d + 1;
    for i from index to 10 do
      var num j =  i - d;
      var num temp = a{i};

      while ((j > 0) and (a{j} > temp)) do
        a{j + d} = a{j};
        j = j - d;
      end

      if (j <> i - d) then
        a{j + d} = temp;
      end
```

(continues on next page)

(continued from previous page)

```
    end
  end
end
```

11.7 图论算法

图论算法作为算法中的一大类，在计算机算法领域有十分重要的地位，其直接关系到很多实际问题的解决。由于 Tenon 不支持指针，无法实现链表，所以在图论算法的实现过程中只能使用数组，导致很多图论的算法无法有效实现。目前，cookbook 在图论相关的算法中，只实现了拓扑排序 (graph_topsort.t)。

11.7.1 graph_topsort.t

graph_topsort.t 是针对有向无圈图的顶点的一种排序。它的算法原理是先找出一个没有入边的顶点，然后显示出该顶点，将该顶点和它的边一起删除掉，对剩余的部分做同样的操作，直到显示出所有顶点，形成一个顶点的排序。graph_topsort.t 在实现的过程中，采取了同哈希表算法同样的二维数组存储数据，并且在实现过程中使用了队列算法。具体源码如下：

```
module graph_topsort

  var num graph{10, 10};
  var num queue{12};
  var num queue_size = 0;
  var num queue_front = 1;
  var num queue_rear = 2;
  var num top_sort{7};
  var num top_sort_index = 1;

  function void main()
    for i from 1 to 10 do
      for j from 1 to 10 do
        graph{i, j} = -1;
      end
    end

    ! init the graph
    graph{1, 1} = 2;
    graph{1, 2} = 3;
    graph{1, 3} = 4;
    graph{2, 1} = 4;
    graph{2, 2} = 5;
    graph{3, 1} = 6;
```

(continues on next page)

(continued from previous page)

```
  graph{4, 1} = 3;
  graph{4, 2} = 6;
  graph{4, 3} = 7;
  graph{5, 1} = 4;
  graph{5, 2} = 7;
  graph{7, 1} = 6;

  for i from 1 to 7 do
    if (indegree_zero(i)) then
      en_queue(i);
    end
  end

  while (queue_size > 0) do
    var num temp = de_queue();
    top_sort{top_sort_index} = temp;
    top_sort_index++;

    var num vertex_index = 1;
    while (graph{temp, vertex_index} <> -1) do
      var num temp1 = graph{temp, vertex_index};
      graph{temp, vertex_index} = -1;
      if (indegree_zero(temp1)) then
        en_queue(temp1);
      end
      vertex_index++;
    end
  end

  for i from 1 to 7 do
    tpwrite_num(top_sort{i});
  end
end

function bool indegree_zero(num n)
  var bool zero_inde = true;
  if (n > 7) then
    tpwrite("The graph doesn't has this point");
    return false;
  end

  for i from 1 to 7 do
    for j from 1 to 10 do
```

(continues on next page)

(continued from previous page)

```
      if(graph{i, j} == n) then
        zero_inde = false;
      end
    end
  end

  return zero_inde;
end



function void en_queue(num en_num)
  if (queue_size == 10) then
    tpwrite("The queue is full.");
    return;
  end

  if ((queue_size < 10) and (queue_rear == 12)) then
    queue{queue_rear} = en_num;
    queue_rear = 1;
    queue_size =  queue_size + 1;
    return;
  end

  queue{queue_rear} = en_num;
  queue_rear = queue_rear + 1;
  queue_size = queue_size + 1;
end

function num de_queue()
  if (queue_size == 0) then
    tpwrite("The queue is empty.");
    return -1;
  end

  if ((queue_size < 10) and (queue_front == 12)) then
    queue_front = 1;
    queue_size =  queue_size - 1;
    return queue{queue_front};
  end

  queue_front = queue_front + 1;
  queue_size = queue_size - 1;
    return queue{queue_front};
```

(continues on next page)

(continued from previous page)

```
  end
end
```

11.8 经典算法

除了前文所介绍的算法之外，cookbook 也在尽力搜集常用的经典算法。目前，cookbook 已经包括了 union-find 算法 (union_find_quickfind.t 和 union_find_quickunion.t) 和裴波那契数列算法（fibonacci.t）。

11.8.1 union-find 算法

不相交集合的 union-find 算法通常用来解决动态等价问题。cookbook 中实现了 union_find_quickfind.t 和 union_find_quickunion.t，这两个实例虽然都是解决同样的问题，但是侧重点不同。同时，这二者对于 union、find、connected 和 get_count 等操作都做了函数实现，保证基本操作完全独立。

union_find_quickfind.t

union_find_quickfind.t 作为 union-find 算法中比较侧重查找的算法，其 union 算法相对操作复杂一点，每次做 union 动作，要将全部元素都遍历一遍，将由 union 动作导致合并的两个集合中的所有标识统一。这就导致 union 动作过于复杂，但是 find 动作十分快速，直接按照下标返回存储标识即可。具体源码如下：

```
module union_find_quickfind

    const num N = 10;
    var num a{N};
    var num count = N;

  function void main()

    for i from 1 to N do
      a{i} = i;
    end

    union(1, 9);
    union(1, 6);
    union(4, 9);
    union(5, 6);
    tpwrite_num(get_count());

    union(2, 3);
    tpwrite_num(get_count());
```

(continues on next page)

(continued from previous page)

```
  union(7, 8);
  tpwrite_num(get_count());

  union(7, 4);
  tpwrite_num(get_count());

  var bool b_connect = connected(1, 10);
  if (b_connect == true) then
    tpwrite("The two elements are connected.");
  else
    tpwrite("The two elements are not connected.");
  end

  b_connect = connected(1, 5);
  if (b_connect == true) then
    tpwrite("The two elements are connected.");
  else
    tpwrite("The two elements are not connected.");
  end

end

function void union(num i, num j)
  var num iID = find(i);
  var num jID = find(j);
  if (iID == jID) then
    return;
  end
  for m from 1 to N do
    if(a{m} == iID) then
      a{m} = jID;
    end
  end
  count--;
end

function num find(num i)
  return a{i};
end

function bool connected(num i, num j)
  return find(i) == find(j);
```

(continues on next page)

(continued from previous page)

```
  end

  function num get_count()
    return count;
  end
end
```

union_find_quickunion.t

union_find_quickunion.t 作为 union-find 算法中比较侧重 union 的算法，其 find 算法相对操作复杂一点。每次做 union 动作，等于直接合并二叉树，节点的标识符直接设为父节点的下标，树上节点的标识符则都逐步递归到根节点的存储标识上。这样实现的 union 算法非常便捷，但是带来的负面影响是 find 动作比较复杂。find 动作每次都要追溯到根节点，才能获取标识符。具体源码如下：

```
module union_find_quickunion

    const num N = 10;
    var num a{N};
    var num count = N;

  function void main()

    for i from 1 to N do
      a{i} = i;
    end

    union(1, 9);
    union(1, 6);
    union(4, 9);
    union(5, 6);
    tpwrite_num(get_count());

    union(2, 3);
    tpwrite_num(get_count());

    union(7, 8);
    tpwrite_num(get_count());

    union(7, 4);
    tpwrite_num(get_count());

    var bool b_connect = connected(1, 10);
```

(continues on next page)

(continued from previous page)

```
    if (b_connect == true) then
      tpwrite("The two elements are connected.");
    else
      tpwrite("The two elements are not connected.");
    end

    b_connect = connected(1, 5);
    if (b_connect == true) then
      tpwrite("The two elements are connected.");
    else
      tpwrite("The two elements are not connected.");
    end

  end

  function void union(num i, num j)
    var num iID = find(i);
    var num jID = find(j);
    if (iID == jID) then
      return;
    end
    a{iID} = jID;
    count--;
  end

  function num find(num i)
    while (i <> a{i}) do
      i = a{i};
    end
    return i;
  end

  function bool connected(num i, num j)
    return find(i) == find(j);
  end

  function num get_count()
    return count;
  end
end
```

11.8.2 fibonacci.t

fibonacci.t 是计算斐波那契数列的算法实例。斐波那契数列的计算是经典算法，实现方式也有多种，cookbook 中的 fibonacci.t 则采用递归和非递归两种方法进行了实现。其中，递归的方法在 fibonacci_recursion() 中实现，非递归的方法在 fibonacci_no_recursion() 中实现。具体源码如下：

```
module fibonacci

  function void main()
    for i from 1 to 20 do
      var num temp = fibonacci_recursion(i);
      tpwrite_num(temp);
    end
    tpwrite("--------------------");

    for i from 1 to 20 do
      var num temp1 = fibonacci_no_recursion(i);
      tpwrite_num(temp1);
    end
  end

  function num fibonacci_recursion(num n)
    if (n <=2 ) then
      return 1;
    else
      return (fibonacci_recursion(n - 1) + fibonacci_recursion(n - 2));
    end
  end

  function num fibonacci_no_recursion(num n)
    var num last;
    var num nextToLast;
    var num answer;

    if (n <= 2) then
      return 1;
    end

    last = 1;
    nextToLast = 1;
    for i from 3 to n do
      answer = last + nextToLast;
      nextToLast = last;
      last = answer;
```

(continues on next page)

(continued from previous page)

```
    end

    return answer;
  end

end
```