Chaiscript Дополнительные темы

некоторые дополнительные темы и мысли о Chaiscript.

Chaiscript знает три разных задания:

- Постоянное присвоение (которое создает постоянную переменную)
- Справочное задание (которое создает псевдоним)
- Присвоение клона (который копирует значение)

К сожалению, для этого используются только два ключевых слова оператора (“:” и “:=”):

КОД	УСТУПКА	ОПИСАНИЕ
`x=0;`	назначение клонов	Назначьте значение 0, другие значения могут быть назначены позже.
`x:=0;`	постоянное назначение	Назначьте значение 0 и отметьте x как константу. Больше не разрешается изменять x.
`x=y;`	назначение клонов	Присвоить значение y x, только если x не является постоянным.
`x:=y;`	ссылочное задание	Сделайте x псевдоним для y. И x, и y будут содержать одно и то же значение. x не станет постоянным.
`x=clone(0);`	назначение клонов	То же, что и x = 0;
`x:=clone(0);`	назначение клонов	Присвоить значение 0. Если x является псевдонимом для другой переменной, то удалите эту ссылку и сделайте x независимой переменной.

КОД	УСТУПКА	ОПИСАНИЕ
`x=clone(y);`	назначение клонов	То же, что и x = y;
`x:=clone(y);`	назначение клонов	Присвоить значение y. Если x является псевдонимом для другой переменной, то удалите эту ссылку и сделайте x независимой переменной.

В общем: Как только тип назначен, тогда могут быть назначены только значения одного и того же типа. Автоматическое преобразование применяется, если доступно.

Встроенная функция `clone()` создаст копию предоставленного аргумента. Для вложенных объектов данных копируется только корневой объект (b)`clone()` не будет делать глубокую копию).

Заключение

- Присвоение клона будет наиболее часто используемым назначением.
- Постоянное назначение может быть полезным, но может быть заменено назначением клона.
- Справочное назначение может привести к странным побочным эффектам и его следует избегать. Однако это может быть быстрее, чем набор клонов в некоторых особых случаях (см пример quicksort ниже).

Подробности о Переменных

Как отмечалось выше, переменная может находиться в определенном состоянии и определенного типа. Есть некоторые вспомогательные функции, которые предоставляют более подробную информацию о переменной. Вот примеры вместе с возвращаемыми значениями `type_name()`, `is_var_undef()` и `is_var_const()`.

ПРИМЕР	ТИП_ИМЯ(X)	IS_VAR_UNDEF(X)	IS_VAR_CONST(X)
`var x;`	`""`	`true`	`false`
`var x:=5;`	`"int"`	`false`	`true`
`var x=5;`	`"int"`	`false`	`false`
`var x:=clone(5);`	`"int"`	`false`	`false`
`var x=int64_t(5);`	`"int64"`	`false`	`false`
`var x:="abc";`	`"string"`	`false`	`true`
`var x="abc";`	`"string"`	`false`	`false`
`var x:=[];`	`"Vector"`	`false`	`true`
`var x=[];`	`"Vector"`	`false`	`false`
`var x:=Vector();`	`"Vector"`	`false`	`false`

ПРИМЕР	ТИП_ИМЯ(X)	IS_VAR_UNDEF(X)	IS_VAR_CONST(X)
`var x=Vector();`	`"Vector"`	`false`	`false`
`var x:=y;`	зависит от y	`false`	зависит от того, почему
`var x=y;`	зависит от y	`false`	`false`
`var x=fun() {};`	`"Function"`	`false`	`false`

Заключение

- Использование конструктора (`int64_t`, `Vector`, ...) в переменном объявлении всегда приведёт к непостоянной переменной.
- Функция `type_name()` полезно выяснить, как иметь дело с переменной типа unkown.

Функция Аргументы

Если в качестве аргумента к функции передаётся переменная (не постоянная), то она передаётся как ссылка: Любая модификация этой переменной внутри функции останется после вызова функции.
Ошибка генерируется, если функция пытается присвоить значение аргументу, который не является переменной:

```
1  def add_dot(s)
2  {
3    print(is_var_return_value(s));
4    print(is_var_const(s));
5    s = s + "."; // error if argument for s is not a variable
6  }
```

ЗВОНИТЬ	ПОВЕДЕНИЕ	IS_VAR_RETURN_VALUE(S)	IS_VAR_CONST(S)
`var a = "a"; add_dot(a);`	`a=="a."`	`false`	`false`
`var a = "a"; add_dot(a+"");`	ошибка	`true`	`false`
`add_dot("");`	ошибка	`false`	`true`

Заключение

- Если `is_var_return_value()` или `is_var_const()` верните true, тогда аргумент не является переменной и функция не должна присваивать значение этому аргументу.
- Если требуется назначение аргументов функции, то чётко опишите это в документации для этой функции.

- Это хорошая пактиция программирования для изменения только значения первого аргумента функции.

Функции Лямбды

Лямбда-функция - это безымянная функция. Вместо “`def function-name`” Chaiscript будет просто использовать “`fun`” как ключевое слово.

```
var f = fun ( x ) { return x * x; };
print ( f ( 3 ) );
```

Лямбда-функции часто используются, если функция ожидает другую функцию в качестве аргумента. В таких случаях использование лямбда-функции более компактно (см. Также пример зыбучих веществ ниже).

Глобальные переменные

Существуют два разных объявления для переменных, которые определяют видимость объявленной переменной.

вар x;

Объявлено на уровне файла: x отображается только на уровне файла, но не в пределах какого-либо определения функции.
Объявленные внутренние функции: x виден только внутри функции, но не внутри каких-либо определений функций внутри функции. Следующий пример не будет работать, потому что x не виден внутри лямбда-функции:

```
def g ( Функция f ) {
  f ( );
}
def h ( ) {
  вар x = 5;
  g ( fun ( ) { print ( x ); } ); // ошибка: не удается найти 'x'
}
h ( );
```

глобальный x;

Объявлено на уровне файла: x доступен везде, особенно внутри определений функций. Объявленная функция: x доступен внутри вложенных определений функций. Это решит вышеуказанную ошибку с помощью функции lamda:

```
def g ( Функция f ) {
  f ( );
}
def h ( ) {
  глобальный x = 5; // x может использоваться в лямбда-функции
  g ( fun ( ) { print ( x ); } ); // g напечатает значение 5
}
h ( );
```

Заключение

- Использовать “`var`” для локальных переменных внутри функции
- Использовать “`global`” для переменных, которые должны быть доступны везде

Quicksort

В Chaiscript нет метода встроенной сортировки, поэтому давайте реализуем quicksort для Chaiscript. Алгоритм Quicksort основан на “https://en.wikipedia.org/wiki/Quicksort (https://en.wikipedia.org/wiki/Quicksort)“. Quicksort будет использовать некоторые из вышеперечисленных тем.

```
def swap(Vector v, int a, int b) {
  var e := v[a];
  v[a] := v[b];
  v[b] := e;
}

def partition(Vector v, int lo, int hi, Function is_lower_than) {
  var i = lo;
  var j;
  for( j = lo; j < hi; ++j ) {
    if ( is_lower_than(v[j], v[hi]) ) {
      swap(v, i, j);
      ++i;
    }
  }
  swap(v, i, hi);
  return i;
}

def quicksort(Vector v, int lo, int hi, Function is_lower_than) {
  if ( lo < hi ) {
    var p = partition(v, lo, hi, is_lower_than);
    quicksort(v, lo, p-1, is_lower_than);
    quicksort(v, p+1, hi, is_lower_than);
  }
}
```

Функция “swap()” будет обмениваться двумя элементами с помощью задания ссылки. Это быстрее, чем использование назначения клона. Во всех остальных случаях используется более распространенное присвоение клона.

Цель состоит в сортировке элементов вектора v, который передается по ссылке на своп, разбиение и фыккорт. Эти элементы вектора v модифицируются свопом, разбиением и фриксором.

Последним аргументом функции quicksort должна быть функция сравнения для двух элементов (return true, если первый аргумент ниже второго аргумента). Это может быть реализовано с лямбда-функцией:

```
var v = [3, 2, 4, 1, 6, 7, 2, 4, 1, 6, 2, 8, 4, 1, 6, 5, 7, 2, 4, 1, 6
quicksort(v, 0, v.size()-1,  fun(a,b) { return a < b; } );
print(v);
```