Chaiscript Overview and Tutorial

Introduction

Recently, I had to train myself on Chaiscript. This is my summary on Chaiscript. Please excause my english. Let me know if there are any mistakes here.

This blog entry is based on Chaiscript 6.0.0. This report will concentrate on the chaiscript language itself, the C++ API is not discussed.

Installation

Chaiscript requires g++/gcc compiler environment. There is also no ready package, so installation requires several and different steps for Windows and Linux:

Chaiscript Installation for Windows

1. Install Min-GW: Execute “mingw-get-setup.exe” from http://www.mingw.org/wiki/getting_started (http://www.mingw.org/wiki/getting_started).
2. Optionally install ConEmu as a terminal for Min-GW: https://conemu.github.io/ (https://conemu.github.io/).
3. Install cmake (Windows win64-x64 Installer (.msi)) from https://cmake.org/download/ (https://cmake.org/download/).
4. Download Chaiscript zip file from https://github.com/ChaiScript/ChaiScript/releases (https://github.com/ChaiScript/ChaiScript/releases). Save the zipfile into a local folder (e.g. “C:\chai”) and extract the zip file.

5. Open a Min-GW (or ConEmu) command window and goto the new Chaiscript folder (which contains CMakeLists.txt).
   Note: “cd C:\chai” will be “cd /c/chai” within Min-GW.
6. Execute cmake:
   cmake -G “MSYS Makefiles” -D MULTITHREAD_SUPPORT_ENABLED=FALSE CMakeLists.txt
7. Now create the Chaiscript interpreter chai.exe by executing make (in the Min-GW shell).

Chaiscript Installation for Ubuntu Linux

1. Install cmake (sudo apt install cmake).
2. Download Chaiscript zip file from https://github.com/ChaiScript/ChaiScript/releases (https://github.com/ChaiScript/ChaiScript/releases). Save the zipfile into a local folder (e.g. “C:\chai”) and extract the zip file.
3. Locate Chaiscript folder (which contains CMakeLists.txt) and execute cmake:
   cmake -D MULTITHREAD_SUPPORT_ENABLED=FALSE CMakeLists.txt
4. Execute make. chai.exe will be created in the same folder.

Calling Chaiscript

The newly created executable can be tested by the following call:
chai -c ‘dump_system()’

It may depend on your system, whether you need to type ./chai.exe or ./chai or just chai. This command will print all registered Chaiscript functions. The usual HelloWorld example will look like this:
chai -c ‘print(“Hello World!”);’

Of course Chaiscript commands can be put into a file:

my_first_script.chai

```
print("Hello World!");
```

Then execute:
chai my_first_script.chai

Summary:

- A single line of Chaiscript code can be executed directly with the “-c” option.
- Chaiscript files are executed by calling chai with the name of the file as argument.

Chaiscript “print” Command

Whatever can be converted to a string, can be printed with the “print” functions. Here are more examples:

```
var v = 3;
print("v=" + to_string(v));
print("v=${v}");
```

- For testing, save the code to a file and execute the file (see above). You could also test a single line of code like this:
  chai -c ‘var v = 3; print(“v=” + to_string(v));’
- “var v = 3;” declares variable v and assigns value 3 to it.
- The “+” operator connects two strings in the example above.
- The expression ${xyz} evaluates xyz and pastes the result of the evaluation into the output.
- print() always prints a carriage return/line feed.
- Statements should be terminated with “;“. I many cases Chaiscript is tolerant if the “;” is not there.

Summary

- Use the print() function to print strings and values.
- Use the to_string() to convert something to a string
- Results of an expression can be embedded into a string with ${xyz}

Conditions and Loops

“if” Statement

```
var v = 3;
if ( v != 3 )
{
    print("v=" + to_string(v));
}
else
{
    print("three");
}
```

- “if” statements are similar to C/C++ or JavaScript.
- In Chaiscript always put the if-block and/or else-block between “{” and “}“. Also a single statement must be placed between “{” and “}“.

- Here is a summary of the compare operators:

Operator	Description
==	Test for “equal”
!=	Test for “not equal”
<=	Lower or equal
>=	Greater or equal
<	Lower
>	Greater

“for” Loop

```
var i;
for( i = 0; i < 5; ++i )
{
    print(i);
}
```

- “for” loop is similar to C/C++ “for” loop: It contains three parts, the statement for the initalization of the loop variable, the test condition for the loop variable and the action for the modification of the loop variable.
- The increment “++” and decrement “—” operator must be placed before the variable.

“while” Loop

```
var i;
i = 0;
while( i < 5 )
{
  print(i);
  ++i;
}
```

Loops can be stopped with the break statement:

```
var i;
i = 0;
while( true )
{
  ++i;
  if ( i > 5 )
  {
    break;
  }
}
```

Summary

- Chaiscript supports if/else, for and while loops.

Build-In Objects

Chaiscript includes useful objects. These objects are listed here:

https://codedocs.xyz/ChaiScript/ChaiScript/namespaceChaiScript__Language.html (https://codedocs.xyz/ChaiScript/ChaiScript/namespaceChaiScript__Language.html)

This chapter will discuss “map“, “vector” and “string” objects.

“map” Object

```
var m = ["a":111, "c":333];
print(m);
```

- A map object can contain a set of key/value pairs.
- Key must be a string.
- Key must be unique.
- Value can be any type.

There is a special “for”-loop to loop over all elements of a map:

```
var m = ["a":111, "b":222];
for( i : m ) // "i" will loop over all key/value pairs
{
    print(i.second()); // print the value of the key/value pair
}
```

- The declaration of the loop variable “i” is not required.
- Use pair.first() to access the key of a key/value pair
- Use pair.second() to access the value of a key/value pair.

The following table gives some examples about what you can do with maps:

Syntax	Description	Example
m = Map()	Create a new empty map	

m[key]	Return the value for key	var m = [“a”:111, “c”:333];
		print(m[“c”]);

m[key] = value	Insert or overwrite a key/value pair.	var m = Map();
		m[“a”] = 111;

m[key].is_var_null()	Check whether key is invalid. Returns true or false. Do not use this: Better use .count() function.	if (m[“a”].is_var_null() == true)
m.count(key)	Check how often key exists in the map (Should be 0 or 1).	if (m.count(“a”) > 0)

m.size()	Return number of key/value pairs.	var m = [“a”:111, “c”:333];
		print(m.size());

m1.insert(m2)	Join two maps m1 and m2 and put result into m1.	var m = [“a”:111, “c”:333];
		m.insert([“b”:222,”d”:444]);
		print(m);

m.erase(key)	Delete the key/value pair with the specified key.	

m.clear()	Remove all elements from the map.	m.clear();
		print(m.size());

m.empty()	Return true if the map is empty.	
m.range()	Returns the range object for looping over the elements.	
m.range().front()	Get the first key/value pair.	
m.range().front().first()	Get the key of the first key/value pair (string).	
m.range().front().second()	Get the value of the first key/value pair.	

“vector” Object

A vector is an ordered list of objects. Any objects can be used as elements of the vector.

```
var v = ["a", 1, 2.9];
print(v);
```

Access and modify functions for a vector:

Syntax	Description	Example
v = Vector()	Create a new empty vector	
v[p]	Return the object at position p. v[0] is the first element.	var v = [“a”, “c”]; print(v[1]);
v[p] = value	Replace object at position p. Note: Only objects of the same type can be replaced.	var v = [“a”, “c”]; v[1] = “b”;
v.insert_at(p, value)	Insert value before the object at position p.	var v = [7, 8]; v.insert_at(1, 123);
v.erase_at(p)	Remove object at position p. The total number of objects in the vector is decreased by one.	var v = [“a”, “c”]; v.erase_at(0);
v.size()	Return the number of elements in the vector.	var v = [“a”, “b”]; print(v.size());
v.clear()	Remove all elements from the vector.	m.clear(); print(m.size());
v.empty()	Return true if the vector is empty.	
m.range()	Returns the range object for looping over the elements.	
v.pop_back()	Remove the last object in the vector. Same as “v.erase_at(v.size()-1)“	
v.push_back(value)	Append value at the end of the vector. Same as “v.insert_at(v.size(), value)“	

“string” Object

A string is a ordered list of chars. Access functions a very similar to ‘Vector’ object.

A string is defined and assigned by

```
var s = "abc";
```

A string may cross multiple lines:

```
var s ="abc
def";
```

Many string operations accept or return a value of type ‘char’. Use the following functions to convert from and to type ‘char’:

- ‘A’: Object of type ‘char’ with the uppercase A.

- char(65): Object of type ‘char’ with the uppercase A.
- int(‘A’): The integer value 65. Note: to_int() does not work.
- to_string(‘A’): String with a single letter “A”. Note: string(‘A’) does not work.
- to_char(” ABC”): Returns the first none-whitespace char of a string, ‘A’ in this case.

Syntax	Description	Example
s = string()	Create a new string.	
var s = “”	Create a new string.	var s = “a”; print(s);
s.size()	returns the number of chars in a string.	var s = “abc”; print(s.size());
s[p]	Access a char at the specified position ‘p’. ‘p’ starts with 0. Returns ‘char’.	var s = “abc”; print(s[0]);
s.erase_at(p)	Remove the char at the specified position ‘p’.	var s = “abc”; s.erase_at(0);
s.insert_at(p, c)	Insert char ‘c’ at the specified position ‘p’.	var s = “abc”; s.insert_at(1, ‘A’);
s.substr(p, cnt)	Returns ‘cnt’ chars as a string, starting at position ‘p’. ‘p’ must be lower or equal to ‘s.size()’. ‘p+cnt’ can be larger than ‘s.size()’.	var s = “abc”; print(s.substr(0), 1, 2);
s.clear()	Empty the string.	s.clear(); print(s.size());
s.empty()	Return true if the vector is empty.	
s + t	Concatenate two strings.	
s += t	Append string ‘t’ to string ‘s’.	
s.push_back(c)	Append char ‘c’ at the end of a string.	var s = “abc”; s.push_back(‘\n’);
find(s, p) s.find(p)	Find string ‘p’ inside string ‘s’. Returns the position of ‘p’ inside ‘s’. Return 4294967295 if ‘p’ can’t be found in ‘s’.	find(“abc”, “b”); // 1
to_int(s)	Interprets string ‘s’ as a number and returns the value as an integer.	print(to_int(” 1234″));

Here is a complete reference for char, int and string conversions. The table shows the result of the function in the leftmost column applied to the value in the first line. Some combinations between value and function may lead to an error (in 6.0.0).

Update: I added the results from Chaiscript 6.1.0 after a slash, if the result differs from the previous version.

Function	‘A’	65	“A”
char	‘A’	‘A’	error
to_char	error / ‘A’	error / ‘A’	‘A’
int	65	65	error

Function	‘A’	65	“A”
to_int	error / 65	error / 65	0
string	error	error	“A”
to_string	“A”	“65”	“A”

Summary

- Chaiscript supports maps, vectors and strings
- Use the “for(:)” loop to access all elements of a map, vector or string

Functions

Example:

```
def my_function (x)
{
    var y;
    y = 3;
    return x*y;
}
```

- Function definitions are started with the „def“ keyword.
- Values are returned via „return“ keyword.
- The „return“ keyword also stops the execution of the function and jumps back to the calling function.
- Arguments can be preceeded by the type name: def my_function (int x)

Useful types are:

Type	Construction	Function

int	var x = int(123);
	var x = 123;
		def fn(int x)

int64_t	var x = int64_t(123);	def fn(int64_t x)

double	var x = double(1.23);
	var x = 1.23;
		def fn(double x)

string	var x = string(“abc”); var x = “”;	def fn(string x)

Map	var x = Map();
	var x = [“a”:111, “b”:222];
		def fn(Map x)

Vector	var x = Vector();
	var x = [];
		def fn(Vector x)

Summary

- Use the “def” keyword to define a function.
- Unlike normal variables, a function argument can be preceded with a type name.

Class

```
class MyClass
{
    var x; // member variable
    def MyClass() // constructor (same name as class)
    {
        this.x = 5; // init your member variables here
    }
    def show()   // member function
    {
        print("x=" + this.x.to_string() );
    }
}

// test
var o = MyClass();
o.show();    // this will print x=5
```

- A class is defined by using the “class” keyword, followed by the class name.
- Member functions and member variables are defined within the class.
- A special member function, which has the same name as the class, is always called before any other function is called (constructor).
- A class MUST have a constructor function.
- Within the member functions, prefix member variables with “this.“.