CScharf

CScharf is an interpreted, general-purpose, imperative, and object-oriented programming language that combines elements of static and dynamic typing. Primitive types are subject to static typing while complex types such as classes are afforded some flexibilities that allow for more dynamic typing. Although a primarily object-oriented language, CScharf provides the ability to apply the procedural and functional programming paradigms with functionality such as immutability and pure functions. To bolster its object-oriented functionality, CScharf supports interface inheritance for classes.

CScharf adopts a scope system alike C#'s that dictates that variables defined in a scope cannot be accessed by code on a lower scope, and upon exiting a scope, any variable defined inside it will be removed.

Data types, variables, and control structures

CScharf supports the following data types: integers, floating point numbers, doubles, booleans, strings, anonymous types, higher-order functions, arrays, class instances, Java classes (through reflection). Void is also available to be used by functions as a return type.

Floating point numbers and doubles are differentiated by the letter used at the end of their literal value e.g. 10.0f (float) and 9.2d (double). Values stored in anonymous types are immutable, while members of classes can have their level of immutability configured; const can be used for variables whose values cannot change after definition, readonly can be used for variables whose values can only be changed in the parent class's constructors, and finally if neither const nor readonly are present, then the variable will always be mutable. Types such as functions and classes can be built inside other functions and classes, but when instantiating a class nested inside a class, its full path must be provided e.g. new ClassOne.NestedClass().

Variable names are limited to containing letters, numbers, and underscores and must start with with a letter or underscore. Due to its mostly static typing, CScharf requires that variables must be declared/defined with a type against which values (represented by expressions that get evaluated to a value) are checked prior to assignment to ensure type safety.

CScharf provides a system to cast variables of different types to others. Simply by providing the type to cast to between angle brackets (and quotation marks if casting to a Java class), the CScharf interpreter will attempt to cast between two types. The casting system can only cast primitive types to other primitive types, and reflection types to other reflection types.

The following control structures are supported in CScharf: if [else] statements, for loops, and while loops.

Syntax

Syntactically, CScharf is very similar to C#, while some exceptions are made regarding its typing, function, and reflection syntaxes. Examples of CScharf code to demonstrate syntax can be seen below (more can be found in demonstration*.csf and test*.csf files). CScharf provides the following operators: +, -, *, /, %, ++, --, ==, =, >, >=, <, <=, and != which all behave like in C#.

```
int integer;                                        // Variable declaration
const double val = <double> 12.0f;                  // Const variable def.
anon obj = new { Age = 12, Colour = "Red"};         // Anonymous type
instance rect = new Rectangle(15.2f, 8.245f);       // Class instantiation
array values = new int[16];                         // Array definition
values[12] = obj.Age * values[0];                   // Array set and get

func squareNumber = double function(double num) {   // Func type definition
    return num * num;
}

public void PrintNumber(int number) {               // Function definition
    print(number);
}

public interface ISortable {                        // Interface definition
    void Sort();
}

public class ArrayList : ISortable {                // Class definition
    public ArrayList() {}
    public void Sort() {}
}

double randomNumber = Reflect(METHOD: "java.lang.Math.random");
```

Reflection

While CScharf natively provides various features that allow for useful applications to be built, there were two areas that needed to be improved: reach and utilising existing code. Natively, CScharf does not provide a way to interact with things outside of the language e.g. files. Additionally, while classes such as collections can be built using CScharf, it is likely that the collection has already been created and so re-writing it in CScharf would be re-inventing the wheel. CScharf's reflection system allows for almost full access to the Java Platform, Standard Edition API with full control. From writing and reading from files on the disk to sending HTTP requests, CScharf provides the means to instantiate Java classes with or without arguments, and call static methods such as Math.random() and Math.round(float). This approach taken with reflection improves on how reflection is applied in languages such as Java and C# by simplifying and shortening the process of calling reflection methods or instantiating reflection classes such that only one line is required.

```
public class HttpResponse
{
    public HttpResponse(int statusCode, string responseMessage) {
        _statusCode = statusCode;
        _responseMessage = responseMessage;
    }

    readonly int _statusCode;
    readonly string _responseMessage;

    public int GetStatusCode() {
        return _statusCode;
    }

    public string GetResponseMessage() {
        return _responseMessage;
    }
}

public instance Delete(string url) {
    reflection result = Reflect(CONSTRUCT: "java.lang.StringBuilder");
    reflection urlObj = Reflect(CONSTRUCT: "java.net.URL", url);
    reflection connection = <"java.net.HttpURLConnection"> urlObj.openConnection();

    connection.setRequestMethod("DELETE");

    return new HttpResponse(connection.getResponseCode());
}
instance response = Delete("https://my-json-server.typicode.com/typicode/demo/posts/1");
```