Solace Language Specification Document

Noé Garcia

February, 2023

1 Solace Overview

This document contains the specification lists and requirements for the Solace Language. Solace is a statically typed functional programming language with a focus on quick construction and easy to read and write syntax. The following document defines syntax, grammar, and lexical aspects of the langauge. As the definition of the language grows and changes, this document will be updated.

Solace is an implementation of a functional programming language similar to Elixir, while taking aspects from many different languages such as C, Python, and Rust. The main focus with this project is introducing the functional paradigm in a easily digestible manner so as to minimize confusion for users who are most comfortable with the imparative paradigm. I will be exploring different ideas for how to do this, and as such, the specifications of the language will most likely change quite a bit until a more stable foundation is established.

Solace is a small project designed and constructed by one person who is learning more as they are building the language. As such take all implementation with a grain of salt, their main training did not have a large focus (or much) on functional langauge design.

2 Types

This section documents the available types within Solace. Types are broken into Atomic, Composite, and Domain-specific. Solace is going to function as a general programming language, with no specific focus on any domain (as of yet).

2.1 Atomic Types

reserved words	description
null	base type representing no data type
int	integer data type, defaults to int32
float	floating point number type, defaults to float 64
char	character type
string	string type
bool	bolean type
:sym	symbolic type
fn	function type

2.2 Composite Types

type name	description
lst	designating an array type, contains same type collections.
tpl	basic typle type for simple groupings.
struct	designating a structure object definition for more complex data objects.

2.3 Domain-Specific

Solace is meant to be general purpose at this point, so nothin domain specific will be listed.

3 Lexical Rules

Like in many languages, Solace has rules as to what constitutes legal syntax for things such as variable declarations, function definitions, and naming criteria.

Solace defines a number of reserved words that are used in defining types, function definitions, and data structures. The reserved words chosen for Solace are designed to be similar to other languages, such as Python and C, while taking a different approach. A lot of the reserved words are defined for type declarations

shown above, or the legal operators for the language that are defined below. The following are the reserved words for the language that do not fit in under the other sections.

3.1 Reserved Words

Reserved word	description
main	used to declare the main function of the program
use	used to include packages/libraries
module	used to define the module the file belongs to
return	keyword used to return value(s) from function

4 Operators

The following showcase some of the legal oporators for Solace

Oporator character(s)	description
+	addition operation
−	subtraction operation
*	multiplication operation
/	division operation
%	modulo operation
>	greater than comparison
<	less than comparison
>=	greater than or equal to comparison
<=	less than or equal to comparison
==	equal to comparison
!=	not equal to comparison

5 Syntax

This section outlines an overview on the basic syntax of the Solace language. The main goal of the syntax is to remain simple but flexible enough to build interesting programs. This includes the ability for defining variables and functions, higher order functions, and establishing basic program workflow. The following showcase a simple program that defines a fibonacci function and the main function of the program. The main function is necessary for the solace to run, similar to C type languages.

Solace will take an approach to syntax similar to C like syntax. Solace is a statically typed langauge, so variable types must be defined upon their declaration. Code blocks are defined through the use of brackets. nested blocks can be defined within function definitions to

The following is a showcase of a simple program:

```
module.Main

func main int ()
{

        out int = fibonacci(7); // returns 13

        // there is a ret and return keyword, but is not needed.
        // the last statement in a block will be returned.
        ret 0;
}

func fibonacci int (n int)
{
        // embedded blocks can be used when pattern matching
```

```
        // and grouping functionality:
        { (n <= 1) -> 1; }
        fibonacci(n-1) + fibonacci(n-2);
}

func factorial int (n int)
{
        // when matching a single parameter, it is possible to
        // omit the parameter for just the matched values.
        { (0) -> 1; }
        n * factorial(n-1);
}
```

In maintaining with the traditional functional language paradigm, all functions are designed to be pure. There are plans for the allowance of unpure function definitions in the future, but for now all functions are pure.

Higher order functions are also a fundamental aspect to Solace. The following showcase how functions are used in the context of other functions.

```
func main int ()
{
        // things
}
```

6 Summary

Solace is a simple functional programming language built as a toy project language. This language is meant to be simplistic to read and write programs with. Solace is written utilizing Yacc, Bison, and C, with the clang compiler. The goal right now is getting the language up and off the ground by implementing the specifications defined above. From there, the language will be evaluated and this document will be updated on where the language will build from there.