Stitch Language Reference Manual

Daniel Cole	**dhc2131**
Megan Skrypek	**ms4985**
Rashedul Haydar	**rh2712**
Tim Waterman	**tbw2105**

1 Introduction

Most "modern" programming languages trace their origins back decades to before the advent of cheap, general purpose multicore CPUs. They were designed for a distinctly mono-threaded environment. While libraries and enhancements to mainstay languages such as C/C++ and Java have added multithreading capabilities, it remains in many ways bolted on kludge. The aim of Stitch is to build a language that has the power and flexibility of a fully compiled C style language, while having native threading support for modern multithreaded applications.

We take much of our inspiration from the previous GPU focused PLT language Sheets, and also from OpenMP. These influences were crucial in developing the overall concept of how we split certain blocks of code into multiple threads, then stitch these threads back together after the block is processed. To this end, we provide the `stitch ... from ... to ... by` syntax, which serves both to delineate which code will be run in multiple threads, and to provide the programmer with an easy way to iterate over data that has been split into these threads. While Stitch aims to provide an improved interface for multithreaded processing, especially when compared to standard C, we will still compile down to C, which will provide the programmer with a plethora of platforms to target.

2 Types

2.1 Primitive Data Types

Stitch supports a number of primitive data types, including integers, characters, floating point numbers, and arrays.

2.1.1 Numerical Data Types

Stitch has support for two numeric data types, int and float.

- `int`
  Integers are 32-bit signed fixed precision numbers.

- `float`
  Floats are single precision floating points.

2.1.2 Characters

Chars in Stitch are exactly the same as their C counterparts; they are one byte variables that hold a value representative of an alphanumeric character or punctuation.

```
char c = ‘h’;
```

2.1.3 Arrays

An array is a data structure that lets you store one or more elements consecutively in memory.

Arrays in Stitch use the reserved word `array`, and in general adhere to the following template:

```
<type> array arrayName[ ]
```

Arrays can store any of the numerical or character data types (float, int, char).

There are two ways to declare an array:

```
int array a[5] ;
int array a[ ] = {1,2,3,4};
```

The first declaration creates an array of size 5, and sets the starting value of each element to be 0. The second declaration will initialize an array with the values passed to it, and infer the size of the array from the number of arguments it is passed.

You can declare an array with either the [size] or {initial elems}, and you must use at least one. So the following are invalid array declarations in Stitch:

```
int array a;
float array a[];
```

The following statement is valid:

```
int array b[6] = {0,7};
```

In this case, an int array of size 6 would be created and the first two elements would be initialized to 0 and 7. The remaining 4 elements would be set to 0, like the default case above.

To access an element of an array, you use C-style square bracket notation:

```
a[0] = 0;
```

2.1.4 Multi-Dimensional Arrays

Multi-dimensional arrays are declared in a very similar fashion to their one-dimensional counterparts:

```
<type> array arrayname[numRows, numCols]
```

will create an array of type <type> with a total number of elements equal to numRows * numCols.

The size parameters are also optional, and an array can be created and initialized with values:

```
int array d[][] = { {2,3,1}, {4,6,5} }
```

This will create a 2D array of ints named d, whose first row is {2,3,1} and whose second is {4,6,5}.

If the size parameters are included, but the number of elements initialized does not match, all rows will be extended to be their proper size and missing elements will be initialized to 0. If the size is inferred, and the row lengths do not match, all the rows will be extended to be the length of the longest row, and any missing values will also be initialized to 0.
An example:

```
float array m[][] = { {1,2,3,4}, {5,6}, {7,8,9} }
```

In this example, the array would be created with a size of 3 * 4, and the second and third rows would be initialized to be {5,6,0,0} and {7,8,9,0}, respectively.

2.2 Structs

A struct is a user-defined data type that can contain primitives and other structs. A struct is a non-primitive data type. The size of a struct is guaranteed to be large enough to contain all the elements.

Structs are defined with the keyword struct, followed by a name, and all the declarations of the variables the struct contains enclosed in curly braces ‘{‘ and ‘}’.
A struct definition must terminate with a semicolon after the right curly brace.

Example:

```
struct Node {
      double data;
      struct Node left;
      struct Node Right;
};
```

defines a struct called Node with 3 elements: A double called data, and two struct Nodes called left and right.

To declare an instance of a struct, you use the following syntax:

- `struct <struct name> <variable name>`

For example,

```
struct Node root;
```

will create an instance of a `struct Node` named root.

To access individual elements of a struct, you use the '`.`' (access) operator, followed by the name of the element. For example:

```
root.data;
```

will access the data element inside the struct Node variable root.

2.5 String Literals

Stitch will support string literals, they will be cast to char arrays and treated as such. As in C, all strings are null terminated.

For example,

```
char name[20];
name = "Bob";
```

2.6 Casting

Stitch supports casting, and the general rule is that casting from a larger data type (in bytes) to a smaller data type will result in precision loss

Casting is allowed under the following conditions:

- Ints can be cast as floats without penalty
- Floats casting to Ints results in undefined behavior
- Chars can be cast to both Ints and Floats
- An Int cast to a char may result in precision loss and generate a warning

- A float cast to a char also results in undefined behavior
- Primitives and non-primitives cannot be cast to each other
- Arrays cannot be cast, but individual elements can

3 Lexical Conventions

3.1 Declarations and Identifiers

A declaration in Stitch associates an identifier with a stitch object. Variables and functions may be so named. The name of a declared identifier in Stitch must begin with an underscore or an alphabetic character, and may contain any further number of alphanumeric characters and underscores. Stitch does not support characters other than [‘0’-’9’ ‘a’-’z’ ‘A’-’Z’ ‘_’] in valid declarable names.

3.2 Literals

- char literals
    - For all common ASCII characters a literal is expressed as the character surrounded by single quotes.
    - Characters that require escaping, because they have no equivalent typable glyph, or because they have special meeting (meaning?) are escaped by a backslash, and then surrounded by single quotes. The following characters must be so escaped:
        - ‘\\’ - backslash
        - ‘\’’ - single quote
        - ‘\”’ - double quote
        - ‘\n’ - newline
        - ‘\t’ - tab

- int literals
    - one or more digits without a decimal point, and with an optional sign component

- float literals

    - one or more digits with a decimal point, and with an optional sign component

For both int and float literals, the maximum representable value is determined by the underlying C implementation.

- array literal
    - an array literal is a comma separated list of literals enclosed by curly braces. Multidimensional arrays are made by nesting arrays within arrays.
- string literal
    - a string literal is a sequence of one or more chars, enclosed by double quotes. Stitch treats all strings as char arrays.

3.4 Whitespace

In Stitch, whitespace consists of the space, tab, and newline characters. Whitespace is used for token delimitation, and has no other syntactic meaning.

3.5 Comments

In Stitch, as in C, single line comments are delimited by the double forward slash characters. Multiline comments begin with the forward slash character, followed by the asterisk character. They continue until they are ended by an asterisk followed by a forward slash.

3.6 Punctuation

- single quote - ‘
    - used to encapsulate a char literal
- double quote - “
    - used to denote strings
- parens - ( )
    - function arguments
    - conditional predicates
    - expression precedence
- square brackets - [ ]

    - array access
    - array declaration
- curly braces - {}
    - array declaration, struct declarations, function definitions, block statements
- comma - ,
    - function parameter separation
    - array literal separation
- semicolon - ;
    - sequencing

3.7 Operators

Stitch includes a simplified subset of the C operators, including all basic arithmetic operators. All operators may be used freely in stitch loops.

Arithmetic Operators:

`*`	`Multiplication`
`/`	`Division`
`+`	`Addition`
`++`	`Postfix increment`
`--`	`Postfix decrement`
`-`	`Subtraction`
`%`	`Mod`

Assignment, Access, and Equivalence Operators:

`.`	`Access`
`=`	`Assignment`
`==`	`Equivalence`
`!`	`Negation`
`!=`	`Non-Equivalence`

Logical and Bitwise Operators:

`&&`	`Logical AND`
`||`	`Logical OR`
`&`	`Bitwise AND`
`|`	`Bitwise OR`
`~`	`Bitwise NOT`
`<<`	`Bit Shift Left`
`>>`	`Bit Shift Right`

Comparison Operators:

`>`	`Greater Than`
`<`	`Less Than`
`>=`	`Greater Than or Equal To`
`=<`	`Less Than or Equal To`

3.7 Operator Precedence

In Stitch, arithmetic operator precedence will follow standard arithmetic conventions. Comparison operators have precedence as in C.

3.8 Keywords

- `if(condition)`
- `else`
- `while(condition)`
- `for(assignment; condition; expression)`

- `stitch` *`variable`* `from` *`startRange`* `to` *`endRange`* `by` *`stepsize`*
- `break`
- `return`
- `const`
- `struct`
- `void`
- `main(expression, expression)`
- `NULL`

4 Stitch Loops & Multi-threading

A key feature in Stitch is the inclusion of multithreading on certain loop constructs. When you use these loops, the body of the loop will be split into separate threads and run concurrently. The splitting, thread management, and cleanup are all handled by the compiler. The loops are called stitch loops, and can be called using the following syntax:

```
stitch variable from startRange to endRange by stepsize
```

Variable is a counter variable that must be an integer that must be declared before the loop. startRange and endRange are either numeric literals or expressions that evaluate to numeric literals. The variable will begin at the value of `startRange` and increment by the value of stepsize (which is a signed integer value) until the value of `endRange`. In keeping with traditional C paradigms, the range represented by `startRange,endRange` is [startRange,endRange). That is, it is inclusive on the start but exclusive on the end. What follows is an example of a typical C-style for loop with an equivalent stitch loop.

```
for(i = 0; i < 10; i++)

stitch i from 0 to 10 by 1
```

The body of the for loop will then be executed in parallel while the main program thread blocks and waits for the threads to return. The variable, while it can be used as an index to access the current iteration, can never be assigned to; that is, it cannot be an **lvalue** inside a loop of this structure where it is used as an assignment. Vector operations are not allowed inside asynchronous loops, and so having vector operations in a stitch loop will result in compilation errors.

5 Syntax

5.1 Program Structure

The overall syntax of Stitch is very similar to C's syntax, with some minor differences when it comes to the asynchronous parts of the program. The general structure of the program will contain a `main()` function. When the program executes, the body of the `main()` function will be executed along with any functions and variables defined outside of the `main()` function. All other statements will not be run.

5.2 Expressions

Expressions in Stitch have a type and value associated with them, and consist of operators and operands. The order of evaluation of the expressions is from left to right, unless there are parentheses, in which case the expression inside the innermost parentheses gets evaluated first.

5.2.1 Assignment & Access

Assignment is done using the = symbol. The value of the expression on the right hand side is stored in the non-const variable on the left hand side. Assignment of string literals is treated as char arrays.

```
//examples of assignment
a = 5;            //a is of type int
b[0] = a;         //b is of type int array
c[6] = “hello”;   //c if of type char array
```

The access operation is done using the `.` symbol. This is only to be used for accessing members of structs. Below is an example of its usage.

```
//example of member access
struct Name {
      char first[20];
      char last[20];
};

struct Name joe;
joe.first = “Joseph”;
joe.last = “Larson”;
```

5.2.2 Arithmetic

Arithmetic operators are plus `+`, minus `-`, multiplication `*`, division `/`, and modulus `%`. The operands of arithmetic operators can only be expressions of type int, float, or char. The evaluated value is of the same type.

```
//examples of arithmetic operations
x = 6.3 * 9.0;
b = c % 4;
t = ‘d’ + 5;
```

The unary operators increment `++` and decrement `--` are also available for use on an expression of type int.

```
//example of unary operation
x ++;
```

These are both postfix-only operators. `++ x` is not a legal Stitch expression.

5.2.3 Comparison

Comparison operators are less-than-or-equal-to `<=`, less-than `<`, greater-than `>`, greater-than-or-equal-to `>=`, equal-to `==`, and not-equal-to `!=`. The operands can be of any type, but the return type is always int, and the value returned is either 0 (false) or nonzero (true).

```
//examples of comparison operations
a = {‘a’, ‘b’, ‘c’};
b = {‘b’, ‘c’, ‘d’};
```

```
a[1] == b[0];           //returns 1
```

Stitch only supports comparison on primitive data types. Therefore, comparison on arrays is not possible.

```
a == b;           //syntax error
```

5.2.4 Logical

Logical operators are AND `&&`, and OR `||`, bitwise AND `&`, bitwise OR `|`, bitwise NOT `~`. The operands of logical operators must have type int, and the return value is of type int and has values 0 or 1.

```
//example of logical operation
a = 0;
b = -2;
c = a && b;       //returns 0
```

5.2.5 Bit Operations

Bitwise operators are shift-left `<<`, and shift-right `>>`. The operands must be of type int, and the results are also of type int.

```
//example of bit operation
a = 1;
b = a<<2;   //returns 4
```

5.3 Statements

A statement in Stitch is a full instruction, the end of which must be denoted by a semicolon `;`. Multiple statements can be encapsulated by `{` and `}`, and becomes a block.

5.3.1 Conditional Statements

Conditional statements use the `if` and `else` keywords and express decisions. The syntax is as follows:

```
if(expression)
```

```
        statement1
else
        statement2
```

If the expression evaluates to an integer >0, then statement1 executes, otherwise statement2 would execute.

Alternatively, for multiple decisions there can be `else if` blocks, the same as C. The syntax for that is:

```
if(expression1)
statement1
else if(expression2)
        statement2
else
        statement3
```

In this situation, if expression1 evaluates to >0, then statement1 would execute, and the rest of the `else if` and else blocks are terminated. The expressions are evaluated in order. The last `else` is optional, and in general, an `else` always attaches itself to the preceding `else`-less `if`.

5.3.2 Loops

There are three types of loops in Stitch: `for`, `while`, and `stitch` loops. The `for` and `while` loops have the same structure as in C, but the `stitch` loop has a different syntax. The following shows how to use the `stitch` loop.

```
stitch variable from startRange to endRange by stepsize
{
        statement
}
```

Further explanation of the `stitch` loop is provided in section 4.

5.3.3 Loop Disruptions

The keyword `break` can be used inside of all three types of loops. It will cause the innermost loop containing the `break` statement to terminate.

5.3.4 Returns

The keyword `return` is used to return the value of an expression from a function to the caller. Anything after the `return` statement is not executed.

5.3.5 Functions

A function statement calls a function and returns a value if the called function has a `return` statement, otherwise the return type is `void` by default. The syntax for a function definition is the following:

```
returnType functionName(argument1, argument2, ...)
{
      statements
      optional return statement
}
```

The syntax for a function declaration is:

```
returnType functionName(argument1, argument2, ...);
```

5.4 NULL

`NULL` is a reserved keyword that is defined in Stitch to be the integer zero.

6 Standard Library Functions

Stitch provides a relatively small number of standard library functions. These are used to facilitate I/O, and as a convenience to facilitate basic operations.

6.1 I/O Functions

Stitch provides the following functions for both file I/O and user I/O. These are drastically simplified versions of their C counterparts. Files are referenced by their file descriptor, which is stored as an integer value.

- `int open(char array)` - open a file for reading and writing at the path specified in the char array. Returns a file descriptor.

- `int write(int, array)` - write the data held in array to the file specified by the int file descriptor. Returns the number of elements written. Warning: if the file is not empty, fwrite() will overwrite some or all of the data stored in the file.

- `int read(int, array)` - read data from the file specified by the int file descriptor into the array. If there is more data in the file than can be stored in the array, the array will be filled, and and the read will stop. Returns the number of elements read.

- `void print(char array)` - prints the specified character array to stdout.

- `void error(char array)` - prints the specified character array to stderr.

6.2 Miscellaneous Functions

Stitch also provides a small set of functions meant to aid the programmer.

- `int sizeof(expression)` - returns the size, in bytes, of the evaluated expression.

- `int lengthof(array)` - returns the number of elements in the specified array. If a non-array is passed to `lengthof()` as an argument, `lengthof()` returns 1.

- `exit(int)` - if called from the main body of the program, this exits the program with a code of int. If called in a stitch loop, `exit()` will exit all threads, as well as the main program. A wrapper for the C function exit().

- `cut(int)` - if called from the main body of the program, this exits the program with a code of int. If called in a stitch loop, `cut()` will exit only the thread in which it is called, leaving the rest running. This is a wrapper for the pthread_exit() function in the POSIX API.