# 1 Types

## 1.1 Primitive Data Types

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

### 1.1.1 Numeric Data Types

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

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

- `float`
  Floats are single precision floating points.

### 1.1.2 Accumulators

In addition to basic numeric data types, there also exists one numeric data type for accumulators that are to be used inside the Stitch loops. It is:

- `int_ap`

It is equivalent to its counterpart, `int`, in the sense that it could potentially be used outside Stitch loops, and would behave as a normal `int`. However, this usage is discouraged to prevent confusion on which variables are accumulators and which ones are regular numerical data types. The `_ap` are abbreviation is for additive (plus) accumulator (`_ap`). At the moment, accumulators are limited to arrays of size 4.

### 1.1.3 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.

### 1.1.4 Arrays

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

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

There are two ways to declare an array:

```
<type> arrayName[size];
<type> arrayName[size] = {value-0,value-1,...,value-(size-1)};
```

The first declaration creates an array of size `size`, which has to be an int literal, and the values of the cells are undefined until you manually change them. The second declaration will initialize an array with the values passed to it, and the length of the set of initial arguments must match the size of the array.

You can declare an array with either the [size] by itself or with the {initial elems}. So the following are invalid array declarations in Stitch:

```
<type> arrayName[];
<type> arrayName[] = {value-0,value-1,value-2};
```

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

```
arrayName[index]
```

### 1.1.5 Matrices

Matrices are two-dimensional arrays, and are declared in a very similar fashion to their one-dimensional counterparts:

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

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

The size parameters are also not optional, and must match the dimensions of the initialized matrix.

```
int d[3][3] = { {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 row is {4,6,5}.

If the size parameters are included, but the number of elements initialized does not match, this is invalid behavior and will not compile.
An example:

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

Will not work.

*Stitch will not catch array bounds exceptions at compile time, but at runtime.*

### 1.2 String Literals

Stitch will support string literals. String literals cannot be assigned to a variable. However, they can be used inside `print`(), `error()` and file I/O statements.

### 1.3 Casting

Stitch does not support casting of any of its data types. Therefore, for any binary operators, the types of the operands must match.

## 2 Lexical Conventions

### 2.1 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 alphabetic character (unlike C, a leading underscore is not permitted), 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.

### 2.2 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 meaning are escaped by a backslash, and then surrounded by single quotes. The following characters must be escaped as such:
        - `'\\'` - 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.

## 2.3 Whitespace

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

## 2.4 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.

## 2.5 Punctuation

- single quote - `‘`
    - used to encapsulate a char literal
- double quote - `“`
    - used to encapsulate string literals
- parentheses - `()`
    - function arguments
    - conditional predicates
    - expression precedence
- square brackets - `[]`
    - array access
    - array declaration
- curly braces - `{}`
    - array declaration, function definitions, block statements
- comma - `,`
    - function parameter separation
    - array literal separation
- semicolon - `;`
    - end of statement
- colon - :
    - end of Stitch declaration

## 2.6 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 |
| - | Subtraction |
| % | Mod |

Assignment, Negation, and Equivalence Operators:

| | |
|---|---|
| = | Assignment |
| == | Equivalence |
| ! | Negation |
| != | Non-Equivalence |

Logical Operators:

| | |
|---|---|
| && | Logical AND |
| \|\| | Logical OR |

Comparison Operators:

| | |
|---|---|
| > | Greater Than |
| < | Less Than |
| >= | Greater Than or Equal To |
| =< | Less Than or Equal To |

## 2.7 Operator Precedence

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

## 2.8 Keywords

- `if(condition)`
- `else`

- `while(condition)`
- `for(assignment; condition; expression)`
- `stitch` *`variable`* `from` *`startRange`* `to` *`endRange`* `by` *`stepsize`* `:`
- `break`
- `return`
- `void`
- `main(expression, expression)`

# 3 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 which 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.

# 4 Syntax

## 4.1 Program Structure

The overall syntax of Stitch is very similar to C's syntax, with some minor differences, especially 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 defined outside of the `main()` function. All other statements will not be run.

Variables cannot be declared outside of the `main()` function, thus global variables do not exist in the Stitch language. Also, since there is no concept of pointers in Stitch, the generic structure of the `main()` function in C

```
int main(int argc, char **argv)
```

would not work because of the `char **`. However, normal formal arguments still work, such as the `int argc` component above, but they aren't useful for main because Stitch has no stdin.

## 4.2 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.

### 4.2.1 Assignment

Assignment is done using the `'='` symbol. The value of the expression on the right hand side is stored in the variable on the left hand side. The syntax for assignment is as follows:

```
variable = value;
```

```
arrayName[index] = value;
```

### 4.2.2 Arithmetic

Arithmetic operators are plus `+`, minus `-`, multiplication `*`, division `/`, and modulus `%`. The of arithmetic operators can only be expressions of type int or float. The evaluated value is of the same type. For the `+` and `-` operators, there must be spaces between the operands and the operator. The syntax for the plus operator is shown below for guidance. The same is not true for the rest of the binary operators. Because of this, it's highly suggested that there be spaces for all binary operators, not just addition and subtraction, for consistency.

```
operand1 + operand2
```

### 4.2.3 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 must match. It is not possible to compare `int`s and `float`s, for example. The return type of a comparison is always int, and the value returned is either 0 (false) or nonzero (true).

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

```
arrayName1 == arrayName2;           //syntax error
```

### 4.2.4 Logical

Logical operators are AND `&&`, and OR `||`. The operands of logical operators must have type int, and the return value is of type int and has values 0 or 1.

## 4.3 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.

### 4.3.1 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`.

### 4.3.2 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.

### 4.3.3 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.

#### 4.3.4 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. Every non-void function, including `main`, must have a return of the proper type.

#### 4.3.5 Functions

A function statement calls a function and returns a value if the called function has a `return` statement. The return type must be present for a function declaration. If nothing is to be returned from the function, then the return type should be `void`. The syntax for a function definition is the following:

```
returnType functionName(formal_argument1, formal_argument2, ...)
{
      statements
      optional return statement
}
```

# 5 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.

## 5.1 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 write(File, array)` - write the data held in `array` to the file specified by `File`. 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(File, array)` - read data from the file specified by `File` 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.

- `FILE open_r(string_literal)` - opens a file for reading at the path specified in the `string_literal`. The file is opened in "r+" mode behind the scene in C. Returns a file descriptor.

- `FILE open_w(string_literal)` - opens a file for writing at the path specified in the `stirng_literal`. The file is opened in "w+" mode behind the scene in C. Returns a file descriptor. Calling both `open_r()` and `open_w()` on the same file name is undefined.

- `void print(expression)` - prints the specified `expression` to stdout. Functions cannot be called from within the `print()` function.

- `void error(expression)` - prints the specified `expression` to stderr.

## 5.2 Miscellaneous Functions

Stitch also provides the `exit()` function meant to aid the programmer.

- `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().