C - Compiler - Brief Documentation

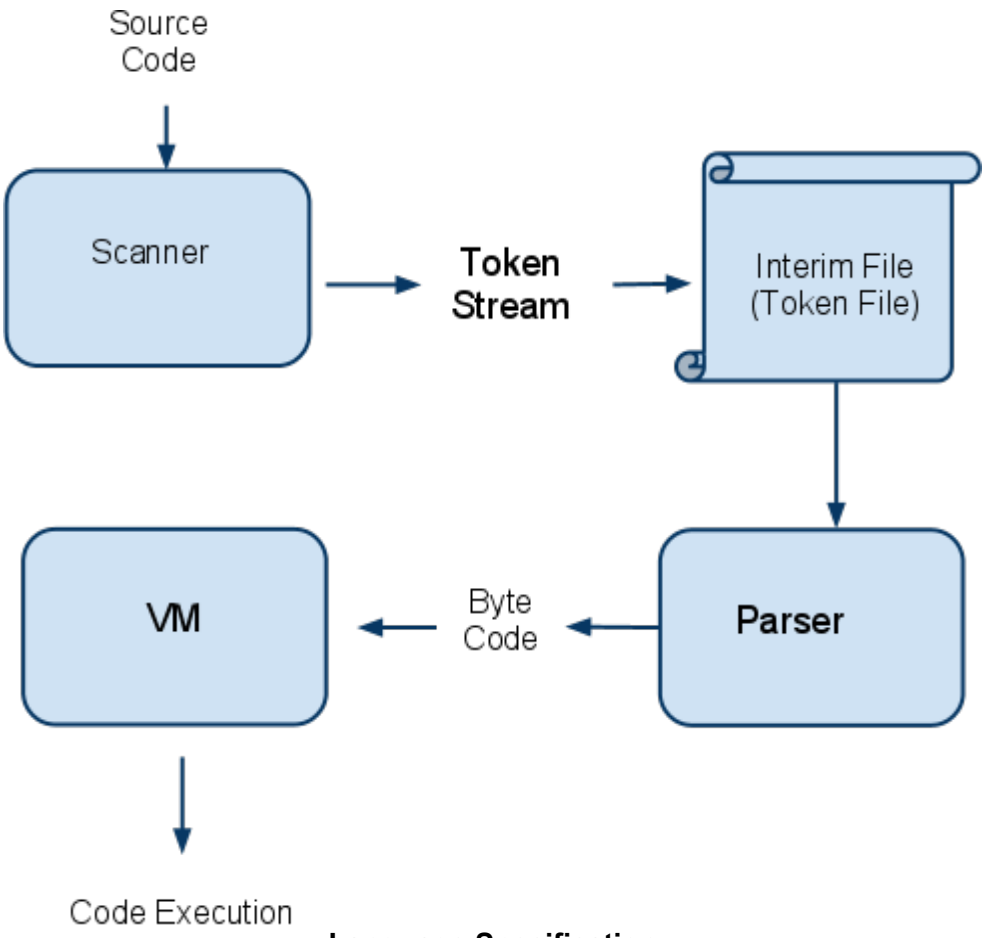
By Eudis Duran

Introduction

In this document I describe my stack based C compiler implementation. If you would like a description on how to build/run the compiler and the virtual machine, please refer to README.md. I begin by describing the general program flow. Subsequently, I describe the subset of C that I was able to complete in the given time. I briefly describe the virtual machine specification as well.

Finally, I elaborate on the source code layout, so that readers of the code can easily follow the program execution flow at their leisure.

General Program Flow



Language Specification

General

All C programs must have an entry point. The entry point must be a *main* function that returns *int*, with or without arguments.

Identifiers

Identifiers can be composed of the following character set: _, a-z, A-Z, 0-9
* Cannot begin with a digit.

Comments

Only C-style comments are supported. That is, comments should begin with /* and end with */

Pre-processor Directives

Only file inclusion functionality is supported at the moment. i.e. `#include "filename.h"`

Keyword Support

Although the scanner part of the compiler handles (or generates tokens for) all keywords in the language, the parser does not use many of these. Below is a list of all the keywords which actually are interpreted by the parser.

`int`	`float`	`char`	`if`	`else`
`void`	`while`	`goto`	`do`	`case`
`printf`	`switch`			

The switch statement does not interpret breaks in-lined. When a case is executed, it is exclusively executed; breaks occur implicitly at the end of a case.

Operator Support

Below is a list of the operators that are supported.

+	-	*	/	&&
||	<	>	<=	>=
%				

Scope Blocks

All scope blocks begin and end with curly brackets. There is no syntactic sugar for blocks with single expressions, which is commonly provided in C implementations. For instance, the following would throw and error:

```
int x;
while(1)
      x = x + 1;
```

Array Support

Array support is extremely minimal. The only fundamental type supported integer for array indexing and declarations. Character, float and integer declarations are successfully implemented, however using floats or integers in expressions will crash the system.

Only integer array support is currently available.

Virtual Machine Specification

Op-code Width: 8-bit
Operand Width: 32-bit

Instruction Format: It is a 40-bit *struct* composed of op-code + operand.

Instruction Set:

push : push value at address into stack.
pushi : push immediate value.
pop : pops the latest value off the top of the stack.
exch : exchange the two values in the stack.
dup : copy the top of the stack.

Arithmetic Instructions:

add, addf : Add two operands (int, float)
sub, subf : Subtract two operands(int, float)
mul, mulf : Multiply two operands (int, float)
div, divf : divide two operands (int, float)

Logical Instructions:

and : logical and
or : logical or
equ : returns 1 if operands are equal, 0 otherwise.
not : nots the top of the stack (logical 1 or 0).
lss : returns 1 if operand(1) is less than operand(2), else 0
gtr : returns 1 if operand(1) is greater than operand(2), else 0
leq : returns 1 if operand(1) is less than operand(2), else 0
geq : returns 1 if operand(1) is greater than operand(2), else 0.
neg : negates the top of the stack (int)
negf : negates the top of the stack (float)

Special:

cvi : Convert to integer.
cvr : Convert to float.

nop : Does nothing.

jmp : absolute jump to address
jfalse : jump if 1
jtrue : jump if 0
halt : exit(-1), stop program execution.

get :
put :

writeint :
writeintid :

writefloat :
writefloatid :

writestring:

VM Executable Format:
These segments are concatenated contiguous in file.

Begining of file … … End

Data + Code + String Data Length + Data Segment Length + Code Segment Length

Source Code Layout

Bugs

A few code caveats before the layout overview.

Labels

Labels do not have parity checking. That is, the compiler does not check if a label is paired with a goto. For instance, the following code crashes the VM:

```
goto L:
     printf("HI");
```

Error Reporting

There is very poor, if any, error reporting. The error reporting that is outputted now does not contain line/column information and other helpful things. This came about because of my decision to write the compiler in phases. The phase that produces the token stream file (scanner), loses all line/column information.

Built In Functions

I wrote the *printf* function before there was any talk of function call implementation (during spring break), and so I did my best with what I had. As a result, I introduced a bug, that even with my sincerest efforts, I have not been able to eliminate. It has to do with printing really long strings. For some reason, the printf can print these long strings, as long as there is a space between printf, and the string itself, like so:

```
printf ("really long string...");
```

However, if there is no space, the source will not build.