The SIGI Programming Language

Short introduction

This is a verbatim copy of the SIGI reference. The author hopes that this short (*albeit concise*) reference manual will serve as a proper introduction to the capabilities of the SIGI Linear Programming Language.

Introduction to SIGI

If you compare SIGI to other languages, you'll immediately see that there is, infact, a huge difference. SIGI does not resemble the major high-level programming languages (*C, C++, BASIC, PASCAL*) and does not try to provide a massive standard library to cloud its purpose. SIGI is a very small language, with no operation codes larger than 1 character. Instead, it prides on **elegant simplicity**. SIGI also seperates itself from other programming languages through the alternative way of handling memory as data, instead of actual, named variables or constants, making SIGI very portable.

SIGI concepts

- Internal memory handling (no tight connection to the operating system)
- Linear memory array, in which, all code is operated
- No user-definable functions, SIGI programs depend on a small base of standard functions
- No named variables, only the memory array

SIGI Operation Codes

Operation codes, or *opcodes*, are the set of characters, SIGI maps its default functions on top of. The standard SIGI opcodes are:

Character	**Function**
+	Add one to the current cell
-	Subtract one from the current cell
*	Add ten to the current cell
_	Subtract ten from the current cell
:	Add 100 to the current cell
;	Subtract 100 from the current cell
>	Go to the next cell
<	Go to the previous cell
p	Print the current cells ASCII value
c	Print the current cells integer value
a	Replace the value of the current cell with the one of the succeeding character
0	Reset the cells value to 0

In addition to the simple function opcodes, SIGI supplies a looping construct, that loops a given piece of code *n* number of times (*n being the value of the next cell*), which looks like:

```
( opcodes )
```

There is also a stream processing opcode, which processes input (through pipes) until end-of-file has been reached. Input is presented one character at a time. Syntax:

```
{ opcodes }
```

An example SIGI program

This program prints out the whole ASCII table (both the integer ID and the actual character):

```
>:***--<
n(+c>>a p0<<pn)n
```

SIGI memory concepts

In SIGI, all data is stored in the linear memory array, with each element (*called a cell*) sized 4 bytes. In the default SIGI interpreter, there are 8192 cells of memory, effectively weighing in at 32 kB of RAM.

The SIGI parser starts out with all values at 0, and the current location at zero:

Cell:	**0**	1	2	3	4
Value:	**0**	0	0	0	0

Now, by typing in `*` in our SIGI file, the memory array suddenly looks like this:

Cell:	**0**	1	2	3	4
Value:	**10**	0	0	0	0

We can now go a couple of cells forward: `>>` will do just that. We will also want to set the value of the cell to 'S' (without the quotes). This is done with `aS`. The result is now:

Cell:	0	1	**2**	3	4
Value:	0	0	**83**	0	0

By now, you'll probably be asking "*Wait a minute, I thought we assigned the value S to the cell*". We actually didn't. What we assigned is the **ASCII** value of the character S, which happens to be 83. All data in SIGI is represented as numbers. We can print out the number value of the current cell with the easy and convenient command `c`. But we can also print out the ASCII (character) value with the command `p`. It's all up to you.

To view the whole ASCII table, run `sigi examples/ascii_table` *in the SIGI source directory*

SIGI and personal cryptography

If you want an easy and convenient way of applying simple cryptography, you can write your own SIGI program, which you can distribute to your friends (along with the SIGI interpreter, of course). The simplest way of applying this (*not a very secure algorithm, mind you*) is writing something like:

```
{*++p0}
```

for the encoder part and

```
{_--p0}
```

for the decoder part. You can subsequently send your decoding application to all your friends (*with an unique addition/subtraction set of opcodes, rather than the default presented to you here*) and you'll be able to easily add atleast a bit to the default e-mail security.

The SIGI interpreter

The SIGI interpreter is a rather special piece of work. Rather than being written using a specialized tool (lex/yacc), the SIGI interpreter is written in the ANSI C language. This allows for extreme portability and *easy access to the memory* through the standard library routines.

The standard interpreter supplies:

32 kilobytes of readily accessible memory (8092 pages)
A portable, bug-free implementation of the SIGI programming language
A particularily unrestrictive license (BSD 3-clause)

If you intend to write a new SIGI interpreter, the reference code is a good place to start. Although not extensively commented, the code is readable enough for intermediately skilled C (*and hopefully C++*) programmers.

The SIGI documentation and reference interpreter is written by David H. Kristensen and licensed under the BSD license. This document was typeset with **groff** on an Apple iBook G4. The official SIGI project page is: http://project.departure.dk/sigi.html