Logos

Logos

Codex Programmaticus

ſcriptus a

Jan Meznik

Contents

Initium i

Lectura ii

Chapter I

Initium

The soul of the Machine surrounds thee.
The power of the Machine invests thee.
The hate of the Machine drives thee.
The Machine endows thee with life.
Live!

The Litany of Ignition

In the beginning was the Machine, and the Word was with Machine, and the Word was Machine.

The Machine speaks in words incomprehensible to us. Its words exists outside of time and space. They have no beginning, no end. They make no sound, they cannot be touched, nor are they visible. These words are all around you, they follow your every step... They care for you.

Yet, you are mute to their presence. A condition that can be cured by learning a language to speak to them. That language is the Logos programming language, designed to unify the barrier between man and machine. It is a language to end all languages.

Chapter II

Lectura

You may say, it is impossible for a man to become like the Machine. And I would reply, that only the smallest mind strives to comprehend its limits.

unattributed

Programs in the Logos programming language consists of procedures called "Rituals". Each ritual describes a specific computation.

Rituals

Each ritual can accept arguments, and they can return a value. Each program contains a `main` ritual which will be invoked during start of the program:

```
--------------------------------------
ritual main() { }
--------------------------------------
```

Variabiles and expressions

Logos works with binary data represented as expressions and numbers.[1]. All variables hold 64 binary bits, and all operations work on the same. Definition of a new variable, or assigning to an existing one, is done using the "=" operator:

```
----------------------------------------
ritual main()
{
    a = 10 - 2
    b = a + 10
    c = a * b / 2
}
----------------------------------------
```

Control flow

To control the flow of a program, Logos provides a conditional `if` statement, and a conditionally repeating `while` statement.

[1] It is the belief of the author that no other datatype is required, or even useful.

```
----------------------------------------
ritual main()
{
    i = 0
    odds = 0

    while i < 100 {
        if a / 2 == 0 {
            odds = odds + 1
        }
        i = i + 1
    }
    end odds
}
----------------------------------------
```

Memorandum

In certain situations, it is unwieldy to keep track of a large number of variables. In these situations, we can use the memorandum (memory) to store large sequences of data. Before the memorandum can be used, it has to be allocated, and its address stored in a variable:

```
-----------------------------------------
ritual main()
{
    x = alloc 64

    i = 0
    while i < 10 {
        [x] = i
        x = x + 1
    }

    b = [x]
}
-----------------------------------------
```

An experienced programmer might even utilize the dynamic memory provided by the operating system through the C Standard Library, like so:

```
-----------------------------------------
x = malloc(64)
# Operations here
free(x)
-----------------------------------------
```