sBasic -- yet another simple programming language

Contents

sBasic Usage

sBasic is designed for freely programmable embedded applications that do not allow scripting languages like JS, Lua or Python.

One should not write applications with it, in contrary it is designed to write extensions for micro controller projects.

sBasic Language Specification

Whitespace

The following whitespace characters will be converted to a single blank (`0x20`): `\t\r` and multiple blanks.

The newline character (`\n`) is a special character however multiple newline characters will be interpreted as one.

A statement always ends with a newline character. Tokens are separated by blanks.

Variables and Arithmetic Operations

Variables have data types. Usually they will be integers, but there might be other application specific datatypes.

Variables have to be declared explicitly using the scheme:

```
<datatype> <variablename> '=' <value>
```

Note that variable names must start with an alphabetic character. Variable names can contain after the first character all of the following characters:

```
a..zA..Z0..9_
```

Datatype names must follow the same conventions as the variable name convention.

Note: Variables are always global.

Here is an example for valid variable declarations:

```
int A = 2
int B = 3
int A_plus_B = 0
```

Arithmetic commands are application specific, for integer usually `+ - * /` are defined. Arithmetic commands can be grouped by using parenthesis `( )`.

`*` and `/` will be executed before `+` and `-`, the inner of parenthesis will be executed before the outer and by default commands will be executed in the order they appear.

Conditional Statements

sBasic knows the if-else-structure:

```
IF <expr>
THEN
        <statement>
        {<statement>}
[ELSE
        <statement>
        {<statement>}]
FI
```

Loops

sBasic knows the while-loop:

```
WHILE <expr>
DO
        <statement>
        {<statement>}
DONE
```

Routines

Because in sBasic variables are always global there is no need for functions. Instead Routines are used to group code.

Routines can be defined defined at any point of the code but the first statement. However when the program execution reaches a Routine this will result in a stack-underflow. The compiler will therefore check that a routine cannot be reached without a call.

Any routine must therefore not be the first statement and must follow either a `CALL` or the `HALT` command.

A routine is defined like this:

```
ROUTINE <routine_name>
        <statement>
        {<statement>}
END
```

The name of a Routine must follow the conventions for variable names.

Routines can be executed by calling them:

```
CALL <routine_name>
```

Special Commands

There is one built-in special command:

```
STOP
```

that will cause the bytecode interpreter to halt.

Comments

Comments start with a `#` at the beginning of the line and span over the entire line. The `#` sign might follow one (or more) blanks.

Strings

There are two types of strings in sBasic: single line strings and multi line strings.

A single line string starts and ends with a `"` character and must not span over several lines. The following conventions for strings are used: `` ` `` is treated as character `` ` ``, `\\` is treated as a single `\`, `\n,r,t` is the newline, carriage return, tab character. Any other combination of `\` and a character is a syntax error.

A multi line string starts and ends with a `` ` `` character and can span over several lines. Every newline character in the string will be treated as such. `"` is treated as character `"`.

sBasic Application Extensions

When it comes to real world applications the sBasic language should be extended with Application Specific Commands (ASC). Those provide a way to directly access the assembly language controlling the bytecode interpreter.