URL: http://cxc.harvard.edu/ciao3.4/slang−overview.html
Last modified: December 2006

AHELP for CIAO 3.4

overview

Context: slang

Jump to: Description Examples See Also

Synopsis

Overview of the S−Lang programing language

Description

A full description of S−Lang and its uses can be found at http://www.s−lang.org/, and the CIAO documents page at http://cxc.harvard.edu/ciao/documents.html contains several manuals on using S−Lang. Much of the following is taken from the S−Lang documentation.

Language Features

The syntax of S−Lang resembles C, with a few postcript−like features. The language features both global and local variables, branching and looping constructs, user−defined functions, structures, datatypes, and arrays. In addition, there is limited support for pointer types. The concise array syntax rivals that of commercial array−based numerical computing environments, and the dynamic nature of data−type allocation removes much of the worry of memory management.

Data Types and Operators

The language provides built−in support for string, integer (signed and unsigned long and short), double precision floating point, and double precision complex numbers. In addition, it supports user defined structure types, multi−dimensional array types, and associative arrays. To facilitate the construction of sophisticated data structures such as linked lists and trees, a `reference' type was added to the language. The reference type provides much of the same flexibility as pointers in other languages.

The language provides standard arithmetic operations such as addition, subtraction, multiplication, and division. It also provides support for modulo arithmetic as well as operations at the bit level, e.g., exclusive−or. Any binary or unary operator may be extended to work with any data type. For example, the addition operator (+) has been extended to work between string types to permit string concatenation.

The binary and unary operators work transparently with array types. For example, if a and b are arrays, then a + b produces an array whose elements are the result of element by element addition of a and b. This permits one to do vector operations without explicitly looping over the array indices.

Statements and Functions

The S−Lang language supports several types of looping constructs and conditional statements. The looping constructs include while, do...while, for, forever, loop, foreach, and _for. The conditional statements include if, if−then−else, and !if.

User defined functions may be defined to return zero, one, or more values. Functions that return zero values are similar to procedures in languages such as PASCAL. The local variables of a function are always created on a stack allowing one to create recursive functions. Parameters to a function are always passed by value and never by reference. However, the language supports a reference data type that allows one to simulate pass by reference.

Unlike many interpreted languages, S−Lang allows functions to be dynamically loaded (function autoloading). It also provides constructs specifically designed for error handling and recovery as well as debugging aids (e.g., function tracebacks).

Functions and variables may be declared as private belonging to a namespace associated with the compilation unit that defines the function or variable. The ideas behind the namespace implementation stems from the C language and should be quite familiar to any one familiar with C.

Run−Time Library

Functions that compose the S−Lang run−time library are called intrinsics. Examples of S−Lang intrinsic functions available to every S−Lang application include string manipulation functions such as strcat, strchop, and strcmp. The S−Lang library also provides mathematical functions such as sin, cos, and tan; however, not all applications enable the use of these intrinsics. Most applications embedding the languages will also provide a set of intrinsic functions; CIAO 3.0 provides a number of modules − such as varmm, sherpa, isis, guide, chips, caldb, paramio, region, group, stackio, xpa, and pixlib − which provide such functions.

Input/Output

The language supports C−like stdio input/output functions such as fopen, fgets, fputs, and fclose. In addition it provides two functions, message and error, for writing to the standard output device and standard error. The Varmm library provides a number of routines for accessing astronomical datasets.

Since S−Lang is embedded into Sherpa and ChIPS, you can try the following commands in eother ChIPS or Sherpa. ISIS is written in S−Lang, so is also available to try out these examples.

Example 1

```
variable x, y, z;
x = 3;
y = sin (5.6);
z = "I think, therefore I am.";
```

S−Lang is different from many other interpreted languages since variables and functions must be declared before they are used (by using the variable statement as shown in the first line). However you don't need to define the type of the variable when it is declared, since the data type is determined upon assignment; after the above, x contains an integer, y a double, and z a string. It is also possible to change the type of a variable:

```
  x = "x was an integer, but now is a string";
```

Note that variable names are case sensitive in S−Lang. Further discussion of variable use in S−Lang can be found by using "ahelp slang variables".

Example 2

```
import("varmm");
variable X = [0:2*PI:0.01];
variable Y = 20 * sin (X);
print(X);
writeascii(stdout, X[[10:12]], Y[[10:12]]);
```

The first line creates an array of doubles between 0 and 2*PI with a step size of 0.01 (PI is a pre−defined constant in S−Lang). The second line creates another array, where each element is 20 times the sine of the corresponding element in the X array. Note that S−Lang is case sensitive with names. The last two lines use Varmm functions (see "ahelp varmm") to display the contents of the S−Lang variables. If the above were saved to the file test.sl (the suffix is not required), then it can be run using:

```
  unix% slsh test.sl
  Double_Type[629]
  0.1     1.99667
  0.11    2.19557
  0.12    2.39424
```

The slsh program takes the supplied file name and executes its contents as S−Lang code; it has been added to CIAO as of version 3.0. Previously you would have had to say something like "chips −−batch −−slscript test.sl" to execute the code.

The first output line is from the print() command and shows that X is a one−dimensional array of doubles containing 629 elements. The remaining lines are the output of the writeascii() function; rather than print out all the whole array we have restricted our attention to just the 11th to 13th elements (as in C, S−Lang starts counting at an array index of 0) by using the [[10:12]] syntax.

Example 3

```
unix% chips

Welcome to ChIPS, version CIAO3.0
Copyright (C) 1999−2003, Smithsonian Astrophysical Observatory

chips> x1 = [0:2*PI:0.01]
chips> y1 = 20 * sin (x1)
chips> () = curve(x1,y1)
chips> symbol red
```

Here we use the S−lang interpreter directly from ChIPS. This example is discussed in more detail in "ahelp slang ciao".

See Also

calibration
    caldb
chandra
    coords, guide, isis, level, pileup, times
chips
    chips, chips_eval
concept
    autoname, parameter, stack, subspace
dm

dm, dmbinning, dmcols, dmfiltering, dmimages, dmimfiltering, dmintro, dmopt, dmregions, dmsyntax

gui

gui

modules

paramio, pixlib, stackio, varmm

sherpa

sherpa_eval

slang

math, slang, tips, variables

tools

ascii2fits

The Chandra X−Ray Center (CXC) is operated for NASA by the
Smithsonian Astrophysical Observatory.
60 Garden Street, Cambridge, MA 02138 USA.

URL:
http://cxc.harvard.edu/ciao3.4/slang−overview.html
Last modified: December 2006