Introduction to S-Lang

John E. Davis

davis@aluche.mit.edu

MIT/CXC

Outline

- Basic Data Types
- Binary and Unary Operators
- Variables
- Conditional and Looping Statements
- Functions
- Working with Arrays
- Examples

Basic Data Types

- Signed and unsigned versions char, short, int, and long integer types
- Single and double precision floating point types
- A double precision complex type
- Strings
- User-defined Structures
- Multi-dimensional Arrays
- Associative Arrays (hashes)

Binary Operators

- **Arithmetic Operators:** `+`, `-`, `*`, `/`, `^`, `mod`
- **Relational Operators:** `>`, `>=`, `<`, `<=`, `==`, `!=`
- **Logical Operators:** `and`, `or`
- **Bitwise Operators:** `&`, `|`, `shl`, `shr`, `xor`

All the above operators work on an element-by-element basis, e.g., $Z = X < Y \Rightarrow Z_i = X_i < Y_i$.

- **Inner product Operator**: #

  $$Z = X\#Y \Rightarrow Z_{ij\ldots m} = X_{ij\ldots k}Y_{k\ldots m}$$

Variables

Generally speaking, variables must be declared before use:

```
variable x;
variable tstart, tstop;
variable energies = [0.03:12:0.0146];
```

Some programs (**isis**, **sherpa**, etc) do not require variables to be declared when used at the command line prompt.

```
    sherpa> energies=[0.03:12.0:0.0146]
```

Conditional Statements

```
if (x < 0) x = -x;

if (y < 0)
  y = -x;
else
  y = 2*y;

if (sin(x) > cos(x))
  x = x + PI/4;
else if (x < tan (x))
  x = x - PI/3;
else
  x = y;
```

Looping Constructs

Execute the `do_something` function 10 times.

```
for (i=0; i<10; i=i+1)
  do_something ();
```

```
i = 0;
while (i < 10)
 {
   do_something ();
   i++;
 }
```

```
i = 0;
do {
  do_something ();
  i++;
}
while (i != 10);
```

Looping Constructs

```
i = 0;
forever {
  i++;
  if (i == 10)
    break;
  do_something ()
}


loop (10) do_something ();
```

Functions

```
define hypot (x, y)
{
   return sqrt (x^2 + y^2);
}
```

Recursive Functions

```
define factorial ();% forward declaration
define factorial (n)
{
   if (n < 2)
     return 1;
   return n * factorial (n-1);
}
```

Like variables, functions must be declared before use.

Functions Returning Multiple Values

```
define quadratic_formula (a, b, c)
{
   variable disc = b^2 - 4*a*c;
   variable alpha = -0.5*b;
   variable beta = sqrt (abs (disc));
   if (disc < 0)
     beta *= 1i;

   return alpha + beta, alpha - beta;
}
```

Working with Arrays

Traditional Method:

```
x = Double_Type[20];
for (i=0; i<20; i++)
  x[i] = sin (2*PI*i/20.0);
```

Working with Arrays

Traditional Method:

```
x = Double_Type[20];
for (i=0; i<20; i++)
  x[i] = sin (2*PI*i/20.0);
```

S-Lang:

```
x = sin ((2*PI/20.0)*[0:19]);
```

Working with Arrays

Consider the “clipping” operation:

```
for (i=0; i<20; i++)
{
   if (x[i] < 0)
     x[i] = 0;
}
```

Working with Arrays

Consider the “clipping” operation:

```
for (i=0; i<20; i++)
{
   if (x[i] < 0)
     x[i] = 0;
}
```

S-Lang:

```
x[where(x < 0)] = 0;
```

Working with Arrays

How **`x[where(x < 0)] = 0`** works:

1. **`x<0`** tests each element of **`x`** to produce an array of 0s and 1s.

   ```
   test = x < 0;
   ```

2. The **`where`** function returns a list of indices that indicates *where* its argument has non-zero elements.

   ```
   i = where (test);
   ```

3. The value of **`x`** at each of the indices is set to 0.

   ```
   x[i] = 0;
   ```

Example: Bit Manipulations

```
define status_bits_histogram (evt_file)
{
   variable status
     = fits_read_col (evt_file, "status");
   status = status [where (status)];

   variable hist = Int_Type[32];
   for (i = 0; i < 32; i++)
     hist[i] = length(where(status&(1 shl i)));

   return hist;
}
```

Example: A Status Bits Tool

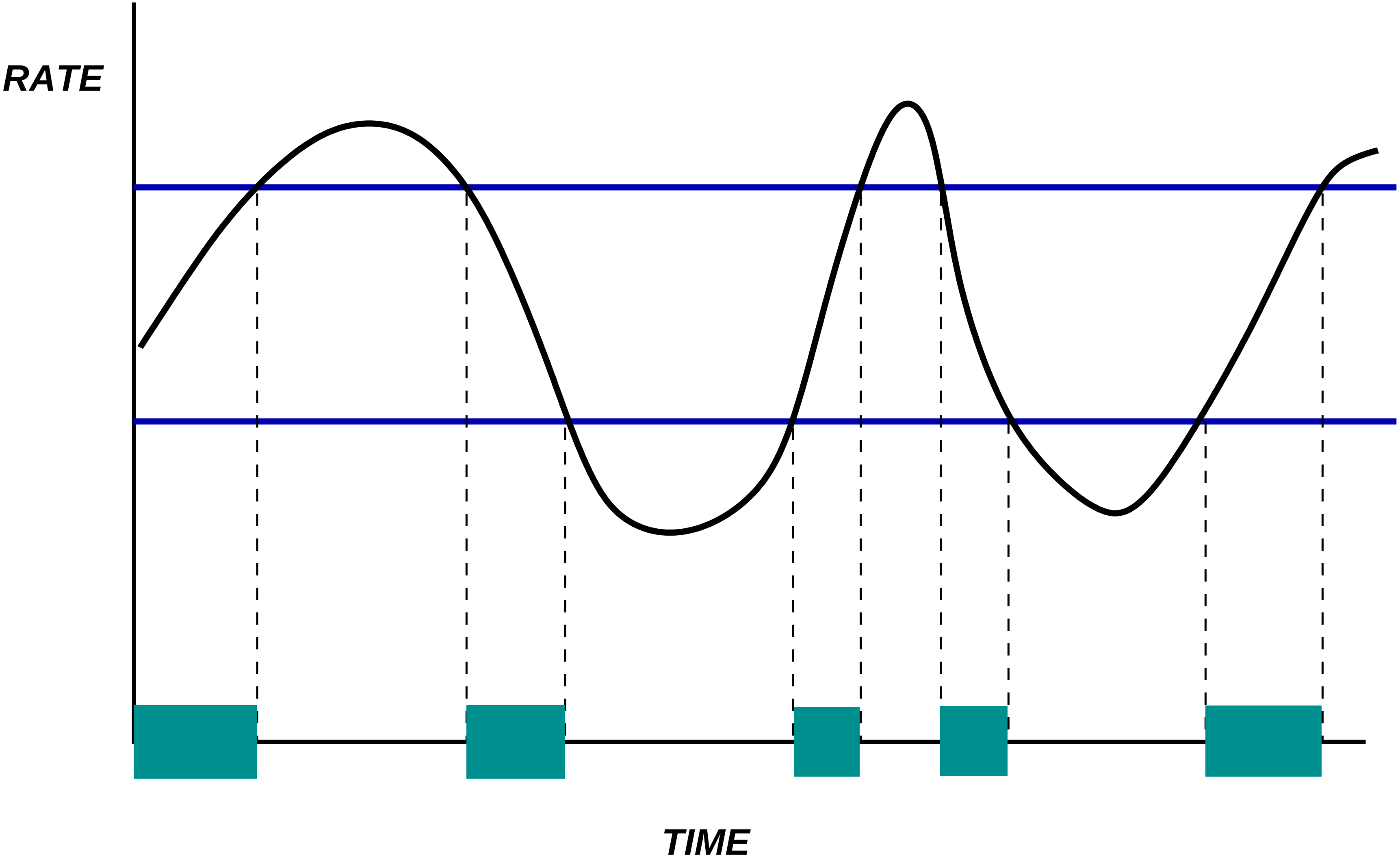
```
#!/usr/bin/env isis-script
if (__argc != 2) {
   vmessage("Usage: %s: evt-file\n",__argv[0]);
   exit (1);
}
variable file = __argv[1];
define status_bits_histogram (evt_file) {...}
variable hist = status_bits_histogram (file);
for (i = 0; i < 32; i++)
  if (hist[i])
    vmessage ("Bit %02d: %d\n", i, hist[i]);
exit (0);
```

Example: Shifting an Array

```
% Shift the elements of an array to
% the left n times. Example: [1,2,3,4,5]
% produces [2,3,4,5,1] for n = 1
define shift (x, n)
{
   variable len = length(x);
   variable i = [0:len-1];

   % allow n to be negative and large
   n = len + n mod len;
   return x[(i + n)mod len];
}
```

Example: Filtering a lightcurve

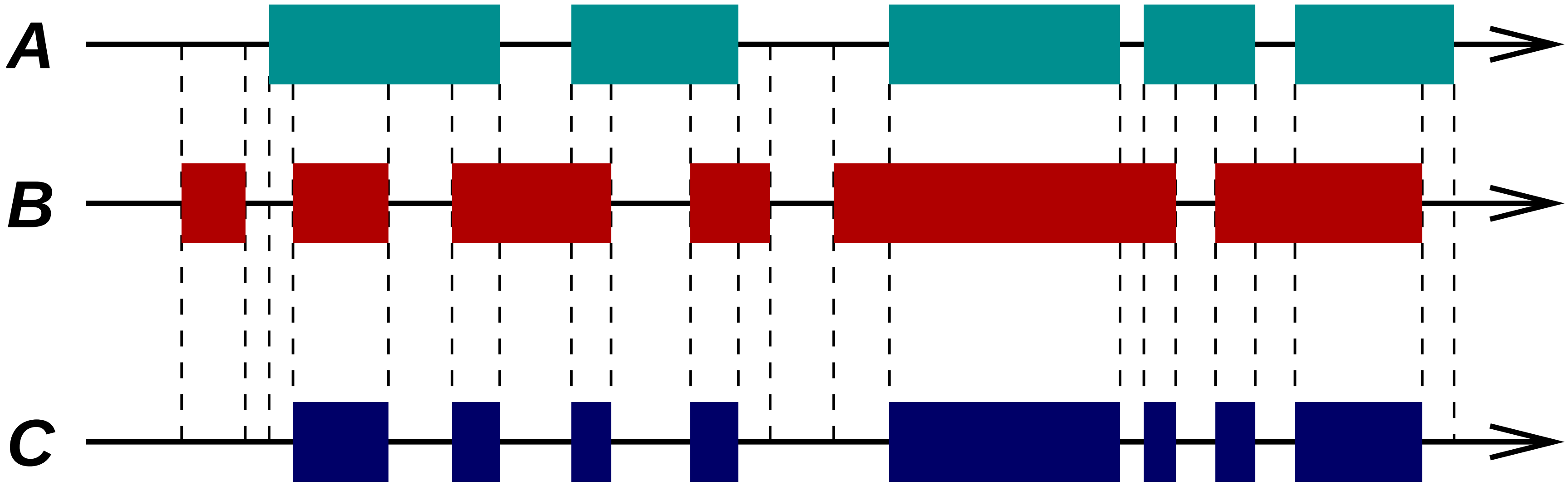


Example: Filtering a lightcurve

```
define filter_lc (time, rate, minrate, maxrate)
{
   variable g = ((rate >= minrate)
                 and (rate < maxrate));
   g = [g, 0];
   time = [time, time[-1]+(time[-1]-time[-2])];

   variable d = g - shift(g,-1);
   variable start = time[where (d == 1)];
   variable stop = time[where (d == -1)];
   return start, stop;
}
```

Example: Intersecting GTIs



$$C = A \cap B$$

Example: Cumulative Sum

```
define cumul_sum (a)
{
   variable i, n = length (a);
   variable b = Double_Type[n];
   variable s = 0.0;
   _for (0, n-1, 1)
     {
        i = ();
        s += a[i];
        b[i] = s;
     }
   return b;
}
```

Example: Structures

```
public define gti_new (start, stop)
{
   variable a = struct
     {
        start, stop
     };
   a.start = start;
   a.stop = stop;
   return a;
}
```

Example: Sorting an Array

```
static define internalize_gtis (a, b)
{
   variable t, w, n, i;

   t = [a.start, b.start, a.stop, b.stop];
   n = length (a.start) + length (b.start);
   w = ones (2*n);
   w[[n:]] = -1;

   i = array_sort (t);
   return t[i], cumul_sum(w[i]);
}
```

Example: Intersecting GTIs

```
public define gti_intersect (a, b)
{
   variable t, w, i;
   (t,w) = internalize_gtis (a, b);
   i = where (w == 2);
   return gti_new (t[i], t[i+1]);
}
```