TRIP programming

M. Gastineau
gastineau@imcce.fr
Observatoire de Paris - IMCCE - CNRS
Astronomie et Systèmes Dynamiques
77, avenue Denfert Rochereau
75014 PARIS

October 29, 2014

Scripts

- TRIP provides a full programming language that enables you to write a series of TRIP statements into a file.
- TRIP has a case sensitive interpreter and can receive your commands from the keyboard.
- Or you write your program using your preferred text editor and save it with the extension .t.
- The script file are loaded and executed using the include statement.
- TRIP looks for the script files in the current directory and then in the directory specified by the variable _path if they are not found.
- The C comments /* .... */ can be used in the script file.
- The single line C++ comments // .... can be used in the script file.
- The @@ statement resumes the TRIP session, loads and executes the last script.

Functions

- The macro statement defines a function.
- The arguments are given by value.The arguments could be modified in a function if they are given by reference using the brackets [] around the argument.
- A variable number of arguments is specified by .... They are retrieved using the array macro_optargs.
- The returned value is specified by the return statement.
- The functions are called using the operator %.
- Private variables could be defined inside the function using private.

source

```
macro mymean[T, mysum,...]
{
 private m;
 mysum = sum(T);
 m = mysum/size(T);
 if (size(macro_optargs)==1) then { m=m*macro_optargs[1]; }:
 return m;
};
v=1,100;
m3 = %mymean[v,[s], 3];
```

Private variables

- Variables may be private to a file, a macro or a loop.
- Private variables of a file still exist even if the execution of the file is finished and are visible from any function of this file.
- The private _ALL; statement lets that all next implicit variables are private to the current control flow (file or macro).
- Following the private _ALL; statement, the public x,...; statement defines the global variables which are visible from any statement.

source

```
macro func[x,y] {
    private u,v;
    ....
};
macro func2[x,y] {
    private _ALL;
    global a, b;
    ....
};
```

Flow control

- TRIP supports four flow control statements
  - for loops.
  - while loops.
  - sum loops.
  - if-then-else statements.
  - switch-case-else statements.
- The stop statement interrupts the execution of a loop statement.

if-then-else

The if-then-else statement looks like

source

```
if (condition) then
{


  statements


};
```

or

source

```
if (condition) then
{
  statements
}
else
{
  statements
};
```

- If the boolean condition is true, then the statement block following the then is executed. Otherwise, the execution continues in the optional else block.
- It is possible to combine several conditions by using else if. Only the statements following the first condition that is evaluated as true will be executed.

Switch

The switch statements are used to perform one of several possible sets of operations. They are intended to replace nested if statements depending on the same operation.

source

```
n = 5;
value1  = 1;
value2  = -1;
value3  = 2;
switch (n)
{
 case value1 : { msg "n is value1";  };
 case 0, value2, value3 : { msg "n is value2 or value3";  };
 else { msg "n isn't value1/2/3 or 0"; }
};
```

output

n = 5
value1 = 1
value2 = -1
value3 = 2
n isn't value1/2/3 or 0

Note: Unlike C, TRIP doesn't need 'break' in each case statement and it accepts strings as value of the switch-case statements.

For

- The for statement repeats a definite number of times the statements.
- The stop statements lets you exit early from the loop. In nested loops, stop terminates from the innermost loop only.
- The default step is 1 .
- Private variables could be defined inside the loop using private.

source

```
for j=1 to n {
 for k=n to 1 step -1 {
    A[j]= 1/(j+k)$
 }:
};
```

Note: Unlike C, TRIP can't use the variable i or I as a loop variable.
In most cases, one can replace nested loops with efficient vector manipulation.

Sum

- The sum statement repeats a definite number of times the statements and performs the summation of the return statement.
- The s = sum j=... to .... { return ...$ }; statement is equivalent to perform s=0$ for j=... to .... { s=s+...$ };
- The stop statements lets you exit early from the loop. In nested loops, stop terminates from the innermost loop only.
- The default step is 1 .
- Private variables could be defined inside the loop using private.

source

```
s = sum k=1 to 4 {
      n= mod(k,3)$
      return (1+x)**n;
};
```

output

```
s(x) =
4
+ 4*x
+ 1*x**2
```

While

- The while statement repeats an indefinite number of times the statements.
- The boolean condition is evaluated before the block is executed if the condition is true. This repeats until the condition becomes false.
- The stop statements let you exit early from the loop. In nested loops, stop terminates from the innermost loop only.
- Private variables could be defined inside the loop using private.

source

```
n=5;
while (A[j]<=n) do {
    A[j] = n-A[j]$
};
```

Note: Unlike C, TRIP can't use the variable i or I as a loop variable.

Structures

- The declaration specifies a list of items. The items can be private.
- The functions can be declared as a member of the structure and access to the private items.
- The object self is the value of the structure during the execution of the function.

source

```
struct point2D {
        x, y;
        private r;
};

macro point2D@Hypot { return sqrt(x^2+y^2); };

macro point2D@Init[vx, vy] {
        x = vx$ y = vy$
        r = %Hypot$
        afftab(self);
};

struct point2D pt1, pt2;
afftab(pt1);
pt1@x = 3;
pt1@y = 4;
pt2%Init[5,6];
```

Parallel computing

- Several OpenMP directives are supported in the macro statements.
- If a statement in parallel region prevents the parallel execution, TRIP executes this region in sequential and emits a warning message.
- OpenMP directives are comments which are started by !trip :
- //!trip omp parallel for

  The iterations of the next loop 'for' are executed in parallel by the threads. The implicit barrier is executed at the end of the loop.

source

```
dim A[1:100]$
/*!trip omp parallel for */
for j=1 to 100 { A[j]=(j+x+y+z)**50$ };
```