Examples of input for testing your functional calculator

You can (and probably should) devise some test cases of your own. Keep them simple.

Some of the ones below have deliberate errors, some will not run properly even if they compile properly.

After successfully using

CMAKE CalcPVM

you can test any little program called

Something.FUN

by issuing the command

Test Something

(no parameter required)

```
// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example that can be handled by Calc.atg and by CalcPVM  EG00.FUN

        a = 1 + 2;
        b = 4 * (3 + 6);
        c = a + b;
        write(a, b, c);

// You should be generate code EG00.COD on the lines of the following
// (a,b,c etc have to be converted to numeric offsets by the compiler).

//       LDC  1
//       LDC  2
//       ADD
//       STL  a
//
//       LDC  4
//       LDC  3
//       LDC  6
//       ADD
//       MUL
//       STL  b
//
//       LDL  a
//       LDL  b
//       ADD
//       STL  c
//
//       LDL  a
//       PRNI
//       LDL  b
//       PRNI
//       LDL  c
//       PRNI
//       HALT

// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example of a trivial function   EG01.FUN

        // function definition

        Double(x) returns 2 * x;

        // no "client code" however

// You should be able to generate code EG01.COD which corresponds
// to the following (x has to be converted to a numeric offset).

//  Double LDC   2
//         LDL   x
//         MUL
//         STL   0   ; 0 is the offset of the slot in the frame
//         RET       ; header where the return value is deposited
//
// You may need to generate some other code, even in this simple case
// - think about it carefully!
```

```
// ====================================================================

$C+ // generate .cod file for checking manually
// Simple example with a single function   EG02.FUN

        // function definition

        Sqr(x) returns x * x;

        // trial call

        read(x);
        a = Sqr(x);
        write(x, " squared = ", a);

// You should generate code EG02.COD which incorporates the following
// (a, x etc have to be converted to numeric offsets by the compiler).

//       // function definition
//
//  Sqr  LDL   x
//       LDL   x
//       MUL
//       STL   0
//       RET
//
//       // trial call
//
//       LDA   x
//       INPI
//
//       FHDR
//       LDL   x
//       CALL  Sqr
//       STL   a
//
//       LDL   x
//       PRNI
//       PRNS  " squared = "
//       LDL   A
//       PRNI
//       HALT

// ====================================================================

$C+ // generate .cod file for checking manually
// Simple example with a single simple function  EG03.FUN

        // function definition

        Sqr(x) returns x * x;

        // trial call

        read(x);
        write(x, " squared = ", Sqr(x));

// You should be able to work out what sort of code this generates!

// ====================================================================

$C+ // generate .cod file for checking manually
// Simple example with a function of 3 parameters  EG04.FUN

        // function definition

        Sum(x, y, z) returns x + y + z;

        // trial call

        read("Supply three numbers ", x, y, z);
        write("They add up to", Sum(x, y, z));
```

```
// ======================================================================

$C+ // generate .cod file for checking manually
// A function called with an expression as an argument  EGO5.FUN

        // function definition

        Sqr(x) returns x * x;

        // trial call

        read("Supply a number ", x);
        write("The square of 2 *", x, " is", Sqr(2 * x));

// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example with two functions  EGO6.FUN

        // function definitions

        Sqr(x) returns x * x;
        Sum(x, y) returns x + y;

        // trial call

        read("Supply two numbers ", x, y);
        z = Sum(x, y);
        writeLine("Their sum is", z);
        writeLine("The square of their sum is", Sqr(z));

// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example with two functions (different order)  EGO7.FUN

        // function definitions

        Sum(x, y) returns x + y;
        Sqr(x) returns x * x;

        // trial call

        read("Supply two numbers ", x, y);
        z = Sum(x, y);
        writeLine("Their sum is", z);
        writeLine("The square of their sum is", Sqr(a));

// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example with two functions compounded  EGO8.FUN

        // function definitions

        Sum(x, y) returns x + y;

        Sqr(x) returns x * x;

        // trial call

        read("Supply two numbers ", x, y);
        write("The square of their sum is", Sqr(Sum(x, y)));

// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example with two functions, one calling the other  EGO9.FUN

        // function definitions

        Sqr(x) returns x * x;

        SumSqr(x, y) returns Sqr(x) + Sqr(y);

        // trial call

        read("Supply two number ", x, y);
        write("The sum of their squares is", SumSqr(x, y));
```

```
// ======================================================================

$C+ // generate .cod file for checking manually
// Simple example with three functions, compounded  EG10.FUN

        // function definitions

        Sqr(x) returns x * x;

        Sum(x, y) returns x + y;

        SumSqr(x, y) returns Sum(Sqr(x), Sqr(y));

        // trial call

        read("Supply two numbers ", x, y);
        write("The sum of their squares is", SumSqr(x, y));

// ======================================================================

$C+ // generate .cod file for checking manually
// Another example with two functions  EG11.FUN

        // function definitions

        Sqr(x) returns x * x;

        Sum(x, y) returns x + y;

        Sqr(y) returns y * y;

        // trial call

        read("Supply two numbers ", x, y);
        write("The sum of their squares is", Sum(Sqr(x), Sqr(y)));

// ======================================================================

$C+ // generate .cod file for checking manually
// Example with two functions, one with no parameters  EG12.FUN

        // function definitions

        AgeNow() returns 67;

        PlusOne(x) returns x + 1;

        // trial call

        writeLine("Pat will be", PlusOne(AgeNow()), " next year");
        writeLine("Do you suppose he will live until he is",
                  PlusOne(PlusOne(PlusOne(AgeNow()))), "?");

// ======================================================================

$C+ // generate .cod file for checking manually
// Another example with two simple functions  EG13.FUN

        // function definitions

        Sum(x, y) returns + x + y;
        Product(x, y, z) returns x * y;

        // trial call

        read("Supply three numbers ", x, y, z);
        writeLine("Their sum is", Sum(x, y, z));
        writeLine("Their product is", Product(x, y));

// ======================================================================

$C+ // generate .cod file for checking manually
// Yet another example with two simple functions  EG14.FUN

        // function definitions

        Sum(x, y, z) returns x + y;
        Product(x, y, z) returns  x * y * z;

        // trial call

        read("Supply three numbers ", x, y, z);
        writeLine("Their sum is", Sun(x, y, z));
        writeLine("Their product is", Product());
```

```
// ====================================================================

$C+ // generate .cod file for checking manually
// Another example with two simple functions  EG15.FUN

        // function definitions

        Double(x) returns 2 * x;
        Triple(x) returns x + Double(x);

        // trial call

        read("Supply a number ", x);
        writeLine("Double ", x , " is", Double(x));
        writeLine("Triple ", x , " is", Triple(x));
        writeLine("\nAn assertion that Double(x) is greater than Triple(x) is",
                  Double(x) > Triple(x));
        writeLine("\nDo you suppose that holds for any or all x?");
        writeLine("If not, why not, and can you prove it?");

// ====================================================================

$C+ // generate .cod file for checking manually
// Errors - example with two misnamed functions  EG16.FUN

        // function definitions

        double(x) returns 2 * x;
        triple(x) returns x + touble(x);

        // trial call

        read("Supply a number ", x);
        writeLine("Double ", x , " is", Double(x));
        writeLine("Triple ", x , " is", Triple(x));
        writeLine("\nAn assertion that Double(x) is greater than Triple(x) is",
                  Double(x) > Triple(x));
        writeLine("\nDo you suppose that holds for any or all x?");
        writeLine("If not, why not, and can you prove it?");

// ====================================================================

$C+ // generate .cod file for checking manually
// Errors - Wrong brackets in parameter lists   EG17.FUN

        // function definitions

        Sqr[x) returns x * x;

        Sqr(x) returns x * x;

        Sum(x, y, z) returns x + y + z;

        Product(x, y, z) returns x * y * z;

        // trial call

        read ("Supply three numbers ", x, y, z);
        writeLine("x =", x, ", y =" , y , ", z =", z);
        writeLine("x squared =", Sqr(x),
                  ", Sum =" , Sum{x, y, z),
                  ", Product =", Product(x, y, z));

// ====================================================================

$C+ // generate .cod file for checking manually
// Examples of bad errors EG24.FUN

        // function definition

        NextInSequence(10, 11) returns 12;

        Bad returns 5;

        Good(x) returns -x;

        Worst{ x ) returns Good(y);

        Horrible = 12;

        // client code
         writeLine("how did I get here?");
```