Some adventures with compile time evaluations

FortranCon 2021

Mohd Furquan
mohdfurquan1@zhcet.ac.in
mfurquan@iitk.ac.in

ALIGARH MUSLIM UNIVERSITY

IIT KANPUR
Indian Institute of Technology Kanpur

compile time evaluation *aka* const expr

- Speed-up: calculate once, use ever after!
  - A fraction of computations in a complex program can be evaluated at compile time.
- Forces referential transparency: no state at compile time!
  - Closer to functional paradigm
  - Easier debugging and proving correctness
  - Thread safety
- Language support

  “they (*languages*) require constant expressions to be written in an impoverished language with minimal support from the type system; this is tedious and error-prone.”

  Reis & Stroustrup (2010)*

*General constant expressions for system programming languages, 2010 ACM symposium of Applied Computing.

Two approaches:
Rosetta code's C++ examples

Templates/Macro style

```cpp
#include <iostream>

template<int i> struct Fac
{
    static const int result = i * Fac<i-1>::result;
};

template<> struct Fac<1>
{
    static const int result = 1;
};

int main()
{
    std::cout << "10! = " << Fac<10>::result << "\
n";
    return 0;
}
```

Function composition style

```cpp
#include <stdio.h>

constexpr int factorial(int n) {
    return n ? (n * factorial(n - 1)) : 1;
}

constexpr int f10 = factorial(10);

int main() {
    printf("%d\n", f10);
    return 0;
}
```

Poor man's approach:
Rosetta code's Fortran example


```fortran
program test

  implicit none
  integer,parameter :: t = 10*9*8*7*6*5*4*3*2  !computed at compile time

  write(*,*) t  !write the value the console.

end program test
```

Legit., but useless!

Is it so bad for Fortran?

Founder's* vision:

Def Innerproduct
≡ (Insert +)∘(ApplyToAll ×)∘Transpose

Or, in abbreviated form:

Def IP ≡ (/+)∘(α×)∘Trans.

*Backus, John, Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. ACM Turing Award Lecture, 1977.

IP:<<1,2,3>, <6,5,4>> =
Definition of IP ⇒ (/+)∘(α×)∘Trans: <<1,2,3>, <6,5,4>>
Effect of composition, ∘ ⇒ (/+):((α×):(Trans:
<<1,2,3>, <6,5,4>>))
Applying Transpose ⇒ (/+):((α×): <<1,6>, <2,5>, <3,4>>)
Effect of ApplyToAll, α ⇒ (/+): <×: <1,6>, ×: <2,5>, ×: <3,4>>
Applying × ⇒ (/+): <6,10,12>
Effect of Insert, / ⇒ +: <6, +: <10,12>>
Applying + ⇒ +: <6,22>
Applying + again ⇒ 28

Translating into Modern Fortran:


```
program InnerProduct
  implicit none
  integer,parameter :: input=[[1,2,3],[6,5,4]]

  write(*,*) SUM(PRODUCT(input,2))  !should be computed at compile time
end program InnerProduct
```

Approach:

- Default constructor
- Composition of intrinsic functions

Improved Rosetta code's example:

```
program factorial
  implicit none

  integer :: i
  integer,parameter :: N, list = [(i, i = 1,N)]

  write(*,*) PRODUCT(list)  !should be computed at compile time
end program factorial
```

Not as general as C++ constexpr functions, useful still!

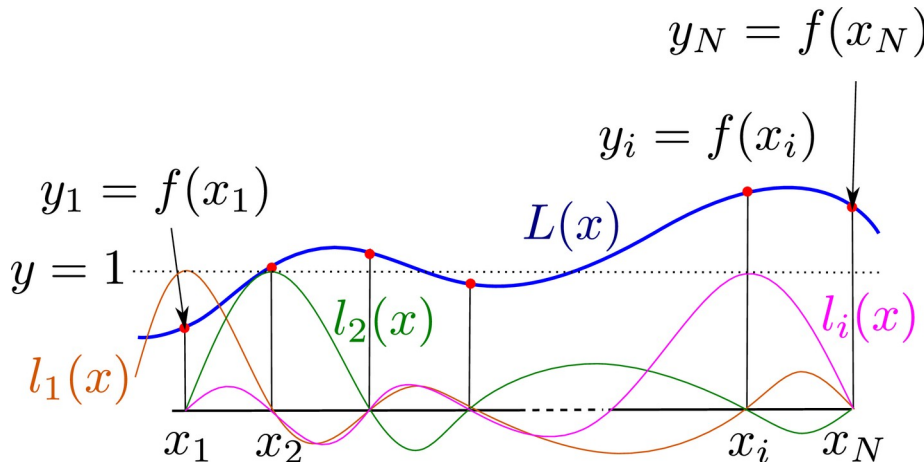
A complex example:
Lagrange polynomials & derivatives

FortranCon 2021

Lagrange basis:

$$l_i(x) = \prod_{\substack{k=1 \\ k\neq i}}^{k=N} \left(\frac{x - x_k}{x_i - x_k} \right)$$

$$l_i'(x) = \frac{\displaystyle\sum_{j=1}^{N} \left(\prod_{\substack{k=1 \\ k\neq i,j}}^{N} (x - x_k) \right)}{\displaystyle\prod_{\substack{k=1 \\ k\neq i}}^{N} (x_i - x_k)} = \left(\sum_{\substack{k=1 \\ k\neq i}}^{N} \frac{1}{x - x_k} \right) l_i(x)$$



Lagrange polynomial:

$$L(x) = \sum_{i=1}^{N} y_i l_i(x)$$

Task: Evaluate l_i and l_i' at selected M values of x, using N points (x_i's)

Selecting x from table:
Choosing quadrature rule

Table of 1D quadrature points:

```
!  ADJUSTABLE PARAMETER
   integer       ,parameter          :: M
```

```
!  1D Gaussian quadrature
   integer       ,parameter,private :: maxint = 6
   real(kind=rp),parameter,private ::                    &
      xg(maxint) = [                                     & ! coordinates
         0._rp,                                          & ! 1 point
         -1._rp/sqrt(3._rp),1._rp/sqrt(3._rp),           & ! 2 point
         -sqrt(3._rp/5._rp),0._rp,sqrt(3._rp/5._rp)]       ! 3 point


!  selecting quadrature points
    integer       ,parameter          ::                 &
      qi    = (M - 1)*M/2 + 1,                           &
      qf    = (M + 1)*M/2
```

Approach:

- Use parameters as ‘keys’ and ‘levers’.

- Declare private parameters for intermediate calculations.

Required N values of x: xg(qi:qf)

Computing $l_i(x)$ via x_i & x_k:
SPREAD all over!


```
!  ADJUSTABLE PARAMETER
   integer,parameter                  :: N
```

```
   integer           ,private        :: i, j, k, p, q, r

   logical,parameter,private,dimension(M,N,N) ::                        &
      mq = spread(reshape([((i/=j, i = 1,N), j = 1,N)],                 &
                          [N,N]),1,M)

   real(kind=rp),parameter,private,dimension(N) ::                      &
      xn    = [(-1+2*i/(N-1), i=0,N-1)]

   real(kind=rp),parameter,private,dimension(nqd,nen,nen) ::            &
      x     = spread(spread(xg(qi:qf),2,N),3,N),                       &
      xk    = spread(spread(xn,1,N),1,M),                              &
      xi    = spread(spread(xn,1,M),3,N)

   real(kind=rp),parameter          ,dimension(M,N) ::                  &
      li    = product(x  - xk,3,mq)/product(xi - xk,3,mq)
```

Approach:

- Replace loops by vector operations.

- Use SPREAD to generate unrolled loops.

$$l_i(x) = \prod_{\substack{k=1 \\ k \neq i}}^{k=N} \left(\frac{x - x_k}{x_i - x_k} \right)$$

Computing l_i'(x):
TWO approaches!

FortranCon 2021

Approach 1:

- Simple, no new definition is required.
- Zero denominator will cause problem!

```
real(kind=rp),parameter         ,dimension(M,N) ::                  &
   li_x  = sum(1./(x  - xk),3,mq)*li
```

$$l_i'(x) = \left(\sum_{\substack{k=1 \\ k \neq i}}^{N} \frac{1}{x - x_k} \right) l_i(x)$$

Approach 2:

- Requires definition of an additional mask.
- Complex computation.

```
logical       ,parameter,private,dimension(M,N,N,N) ::              &
   cq = spread(reshape([(((i/=j.AND.(i-j/=k.AND.i-j/=k-N),          &
                          i = 1,N), j = 1,N), k = 1,N-1)],          &
                       [N,N,N-1]),1,M)

real(kind=rp),parameter         ,dimension(M,N) ::                  &
   li_x = sum(product(spread(x  - xk,4,N-1),3,cq),3)                &
        /     product(       xi - xk,3,mq)
```

$$l_i'(x) = \frac{\sum_{j=1}^{N} \left(\prod_{\substack{k=1 \\ k \neq i,j}}^{N} (x - x_k) \right)}{\prod_{\substack{k=1 \\ k \neq i}}^{N} (x_i - x_k)}$$

Multidimensional polynomials:
via tensor products!

$$m_{ijk}(x,y,z) = l_i(x)l_j(y)l_k(z)$$

```
real(kind=rp),parameter,dimension(M**3,N**3) ::                    &
   mijk   = reshape([((((((li(i,p)*li(j,q)*li(k,r),                &
                    i = 1,M), j = 1,M), k = 1,M),                  &
                    p = 1,N), q = 1,N), r = 1,N)],                 &
                    [M**3,N**3])
```

$$m_{ijk_{,x}}(x,y,z) = l_i'(x)l_j(y)l_k(z)$$

```
real(kind=rp),parameter,dimension(3,N**3,M**3) ::                  &
   mijk_x = reshape([((((((li_x(i,p)*li  (j,q)*li  (k,r),          &
                          li  (i,p)*li_x(j,q)*li  (k,r),           &
                          li  (i,p)*li  (j,q)*li_x(k,r),           &
                    p = 1,N), q = 1,N), r = 1,N),                  &
                    i = 1,M), j = 1,M), k = 1,M)],                 &
                    [3,N**3,M**3]
```

Approach:

- Implied loops and RESHAPE to accomplish tensor products.

Thank you!

Questions?

FortranCon 2021

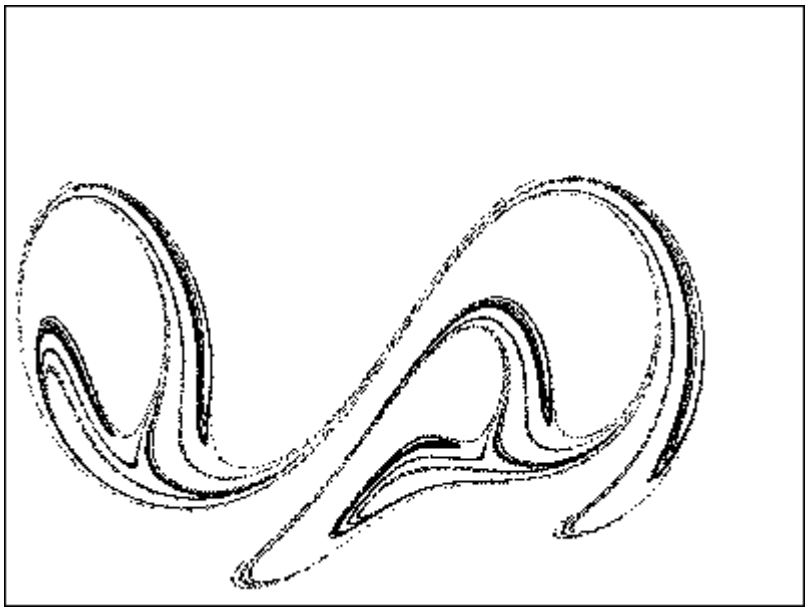


Image credit: Piet Jonas