```
c===========================================================
c     Solves 1-dimensional wave equation
c
c        u_tt = u_xx     0 <= x <= 1   u(0,t) = u(1,t) = 0
c
c     using second-order finite difference techniques.
c===========================================================
      program         wave1d

      implicit        none

      integer         iargc,       i4arg
      real*8          r8arg

      integer         maxn
      parameter     ( maxn = 2**15 + 1 )

c-----------------------------------------------------------
c     Storage for grid functions:
c
c     u(maxn,2)   Difference solution (two time levels)
c     lm(maxn,3)  Initial left-moving profile (lm(1:n,1))
c                 and first (lm(1:n,2)) and second
c                 (lm(1:n,3)) derivatives.
c     rm(maxn,3)  Initial right-moving profile (rm(1:n,1))
c                 and first (rm(1:n,2)) and second
c                 (rm(1:n,3)) derivatives.
c     x(maxn)     Difference mesh
c-----------------------------------------------------------
      real*8          u(maxn,2),   lm(maxn,3),   rm(maxn,3),
     &                x(maxn)

c-----------------------------------------------------------
c     Command-line arguments
c-----------------------------------------------------------
      integer         level,       ncross,       olevel
      real*8          lambda,      alm,          arm
```

```
c-----------------------------------------------------------
c     Locals.  Note the use of the integers 'n', 'np1'
c     and 'nm1' as "pointers"---indices for the second
c     dimension (time level) of u.  'np1' and 'nm1'
c     will always have the same value (1 or 2); in effect
c     "aliasing" the n + 1 and n - 1 storage.  'n' will
c     always have the complementary value (2 or 1).
c     Time step advancement can then be effected (with
c     no copying of data) by simply "swapping" the 'n' and
c     'np1/nm1' values.
c-----------------------------------------------------------
      integer        nt,          nx,
     &               n,           np1,          nm1,
     &               j,           it,           ofreq
      real*8         dx,          dt,           lamsq,
     &               t

c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .ne. 6 ) go to 900
      level  = i4arg(1,-1)
      if( level .lt. 1 ) go to 900
      lambda = r8arg(2,-1.0d0)
      if( lambda .lt. 0.0d0 ) go to 900
      ncross = i4arg(3,-1)
      if( ncross .lt. 1 ) go to 900
      alm    = r8arg(4,0.5d0)
      arm    = r8arg(5,0.5d0)

      olevel = i4arg(6,level)
      if( olevel .gt. level .or. olevel .lt. 1 ) olevel = level
      ofreq = 2 ** (level - olevel)

c-----------------------------------------------------------
c     Set up finite-difference mesh.
c-----------------------------------------------------------
      nx = 2 ** level + 1
      dx = 1.0d0 / (nx - 1)
      dt = lambda * dx
```

```fortran
c-----------------------------------------------------------
c     'nt' is the total number of time steps, including
c     the intial time.
c-----------------------------------------------------------
      nt = ncross * (nx - 1) / lambda + 1.5d0
      do j = 1 , nx
         x(j) = (j - 1) * dx
      end do
c-----------------------------------------------------------
c     Define initial left-moving and right-moving pulses and
c     derivatives.
c-----------------------------------------------------------
      call dvgaussian(lm(1,1),lm(1,2),lm(1,3),x,nx,
     &                alm,0.5d0,0.1d0)
      call dvgaussian(rm(1,1),rm(1,2),rm(1,3),x,nx,
     &                arm,0.5d0,0.1d0)
c-----------------------------------------------------------
c     Define t=0, t=dt data.
c-----------------------------------------------------------
      n    = 1
      np1  = 2
      nm1  = 2
      u(1,nm1)  = 0.0d0
      u(1,n  )  = 0.0d0
      do j = 2 , nx - 1
         u(j,nm1) = lm(j,1) + rm(j,1)
         u(j,n)   = u(j,nm1) + dt * (lm(j,2) - rm(j,2)) +
     &              0.5d0 * dt * dt * (lm(j,3) + rm(j,3))
      end do
      u(nx,nm1) = 0.0d0
      u(nx,n  ) = 0.0d0
c-----------------------------------------------------------
c     Output t=0 and (posssibly) t=dt data
c-----------------------------------------------------------
      call gnuout(u(1,nm1),x,nx,0.0d0,ofreq)
      t = dt
      if( ofreq .eq. 1 ) then
         call gnuout(u(1,nm1),x,nx,t,ofreq)
      end if
```

```
c-----------------------------------------------------------
c     Main evolution loop.  Note time indexing from 2 to
c     nt - 1 to ensure that mod(it,ofreq) works properly.
c
c     Alternatively
c
c     do it = 3 , nt
c        if( mod(it-1,ofreq) ... ) then
c-----------------------------------------------------------
      lamsq = lambda * lambda
      do it = 2 , nt - 1
         do j = 2 , nx - 1
            u(j,np1) = 2.0d0 * u(j,n) - u(j,nm1) + lamsq * (
     &         u(j-1,n) - 2.0d0 * u(j,n) + u(j+1,n) )
         end do
         t = t + dt
c-----------------------------------------------------------
c        'gnuplot' style output every 'ofreq' steps
c-----------------------------------------------------------
         if( mod(it,ofreq) .eq. 0 ) then
            call gnuout(u(1,np1),x,nx,t,ofreq)
         end if
c-----------------------------------------------------------
c        Swap 'pointers'.
c-----------------------------------------------------------
         np1 = n
         n   = nm1
         nm1 = np1
      end do
      stop
c-----------------------------------------------------------
c     Usage.
c-----------------------------------------------------------
900   continue
         write(0,*) 'usage: wave1d <level> <dt/dx> '//
     &      '<ncross> <a left-mover> <a right-mover> '//
     &      '<olevel>'
      stop

      end
```

```fortran
c===========================================================
c     Given input x-values x_j, j = 1 , n, returns gaussian
c     profile, g_j and first two derivatives g'_j and
c     g''_j.  Gaussian profile defined by
c
c     g(x;a,x0,del) = a * exp( - ((x-x0)/del)**2 )
c
c     Returns 0.0 for all three profiles if all inputs are
c     not valid.
c===========================================================
      subroutine dvgaussian(g,dg,ddg,x,n,amp,x0,del)
         implicit        none
         integer         n
         real*8          g(n),     dg(n),    ddg(n),    x(n)
         real*8          amp,      x0,       del
         real*8          xmx0,     delm2,    mdelm2
         integer         j

         if( del .eq. 0.0d0  .or.  n .lt. 1 ) then
c-----------------------------------------------------------
c           Invalid input.
c-----------------------------------------------------------
            do j = 1 , n
               g(j)   = 0.0d0
               dg(j)  = 0.0d0
               ddg(j) = 0.0d0
            end do
         else
            delm2  = 1.0d0 / (del ** 2)
            mdelm2 = -2.0d0 * delm2
            do j = 1 , n
               xmx0 = x(j) - x0
               g(j)   = amp * exp(-delm2 * xmx0**2)
               dg(j)  = mdelm2 * xmx0 * g(j)
               ddg(j) = mdelm2 * (1.0d0 + mdelm2 * xmx0**2)
     &                         * g(j)
            end do
         end if
         return
      end
```

```fortran
c===========================================================
c     Output to standard out for subsequent plotting via
c     gnuplot.
c===========================================================
      subroutine gnuout(u,x,nx,t,stride)
         implicit       none

         integer        nx,           stride
         real*8         u(nx),        x(nx),        t

         integer        j

c-----------------------------------------------------------
c        Basic gnuplot 'splot' format for parametric
c        plotting is:
c
c        <x>    <y>    <f(x,y)>
c-----------------------------------------------------------
         do j = 1 , nx , stride
            write(*,*) t, x(j), u(j)
         end do
c-----------------------------------------------------------
c        Empty line separates rows of data.
c-----------------------------------------------------------
         write(*,*)

         return

      end
```