```
module types
  integer, parameter::dp=selected_real_kind(15) !15 significant digits
end module types
module globals
  use types
  real(dp)::a0,a1,a2,a3,a4
end module globals

program rootfinder
  use types
  use globals
  implicit none
  integer::exitflag,maxit,numit
  real(dp)::epsi1,epsi2,r,r1,r2
  character (len=1)::ans
  character (len= 3)::fmt="(a)"
  interface
    subroutine secant(xnew,xold,xolder,f,epsi1,epsi2,maxit,numit,exitflag)
      use types
      integer, intent(in)::maxit
      integer, intent(out)::exitflag,numit
      real(dp), intent(out)::xnew
      real(dp), intent(inout)::xold,xolder
      real(dp), intent(in)::epsi1,epsi2
      interface
        function f(x)
          use types
          real(dp), intent(in)::x
          real(dp)::f
        end function f
      end interface
    end subroutine secant

    subroutine newton(xnew,xold,f,fp,epsi1,epsi2,maxit,numit,exitflag)
      use types
      integer, intent(in)::maxit
      integer, intent(out)::exitflag,numit
      real(dp), intent(out)::xnew
      real(dp), intent(inout)::xold
      real(dp), intent(in)::epsi1,epsi2
      interface
        function f(x)
          use types
          real(dp), intent(in)::x
          real(dp)::f
        end function f
        function fp(x)
          use types
          real(dp), intent(in)::x
          real(dp)::fp
        end function fp
      end interface
    end subroutine newton

    function poly(x)
      use types
      real(dp), intent(in)::x
      real(dp)::poly
    end function poly

    function polyp(x)
      use types
      real(dp), intent(in)::x
      real(dp)::polyp
    end function polyp

  end interface

  write(*,fmt,advance="no")"Enter 4th order polynomial coefficients (a0-a4): "
```

```
  read(*,*)a0,a1,a2,a3,a4
  write(*,fmt,advance="no")"Epsilon for establishing root found? "
  read(*,*)epsi1
  write(*,fmt,advance="no")"Epsilon for establishing slow progress? "
  read(*,*)epsi2
  write(*,fmt,advance="no")"Maximum number of iterations? "
  read(*,*)maxit
  write(*,fmt,advance="no")"Secant (s) or Newton's (n) method? "
  read(*,fmt)ans
  if(ans=="s")then
    write(*,fmt,advance="no")"Enter 2 initial root estimates:  "
    read(*,*)r1,r2
    call secant(r,r1,r2,poly,epsi1,epsi2,maxit,numit,exitflag)
  else
    write(*,fmt,advance="no")"Enter an initial root estimate:  "
    read(*,*)r1
    call newton(r,r1,poly,polyp,epsi1,epsi2,maxit,numit,exitflag)
  end if
  select case (exitflag)
    case (1)
      write(*,*)"Root found within tolerance ",epsi1
      write(*,*)"   Root: ",r
      write(*,*)"   Number of iterations: ",numit
    case (2)
      write(*,*)"Slow progress tolerance (",epsi2,") reached"
      write(*,*)"   Root estimate at exit: ",r
      write(*,*)"   Number of iterations: ",numit
    case (3)
      write(*,*)"Maximum number of iterations reached (",maxit,")"
      write(*,*)"   Root estimate at exit: ",r
    case (4)
      write(*,*)"Divergence suspected, iterations halted"
      write(*,*)"   Root estimate at exit: ",r
  end select
end program rootfinder


subroutine newton(xnew,xold,f,fp,epsi1,epsi2,maxit,numit,exitflag)
  use types
  implicit none
  integer, intent(in)::maxit
  integer, intent(out)::exitflag,numit
  real(dp), intent(out)::xnew
  real(dp), intent(inout)::xold
  real(dp), intent(in)::epsi1,epsi2

  !Local variables
  real(dp)::xolder ! place holder for old root estimate
  real(dp)::fxold,fpxold,fxnew  ! placeholders for function values

  !Interface to the subprogram used to evaluate the function in question and
  ! it's derivative
  interface
    function f(x)
      use types
      real(dp)::f
      real(dp), intent(in)::x
    end function f
    function fp(x)
      use types
      real(dp)::fp
      real(dp), intent(in)::x
    end function fp
 end interface

  !This routine implements the Newton's method for root finding
  !Arguments and their types
  !  xnew: final root estimate (real(dp), intent out)
  !  xold: initial root estimate (real(dp), intent in)
  !  f: name of the user supplied function subprogram containing the function
  !     whose root we are trying to determine (real(dp), external)
  !  fp: name of the user supplied function subprogram containing the derivative

  !      of the function whose root we are trying to determine
  !      (real(dp), external)
  !  epsi1: convergence tolerance - root is assumed to have been found if the
  !         function value at xnew is less then epsi1 (real(dp), intent in)
  !  epsi2: slow progress tolerance - progress is assumed to be too slow to
  !         continue if xnew-xold is less than epsi2 (real(dp), intent in)
  !  maxit: maximum number of iterations allowed (integer, intent in)
  !  numit: number of iterations already performed (intent, out)
  !  exitflag: indicates exit condition from this subroutine
  !    exitflag=1 indicates solution found with tolerance epsi1
  !    exitflag=2 indicates slow progress condition
  !    exitflag=3 maximum number of iterations reached
  !    exitflag=4 divergence test failed (difference between successive
  !               root estimates increasing)



  ! code for Newton's method goes here

  return
end subroutine newton

subroutine secant(xnew,xold,xolder,f,epsi1,epsi2,maxit,numit,exitflag)
  use types
  implicit none
  integer, intent(in)::maxit
  integer, intent(out)::exitflag,numit
  real(dp), intent(out)::xnew
  real(dp), intent(inout)::xold,xolder
  real(dp), intent(in)::epsi1,epsi2

  !Local variable
  real(dp)::fxold,fxolder,fxnew  ! placeholders for function values

  !Interface to the subprogram used to evaluate the function in question
  interface
    function f(x)
      use types
      real(dp)::f
      real(dp), intent(in)::x
    end function f
  end interface

  !This routine implements the secant method for root finding
  !Arguments and their types
  !  xnew: final root estimate (real(dp), intent out)
  !  xold: initial root estimate (real(dp), intent in)
  !  xolder: another initial root estimate (real(dp), intent in)
  !  f: name of the user supplied function subprogram containing the function
  !     whose root we are trying to determine (real(dp), external)
  !  epsi1: convergence tolerance - root is assumed to have been found if the
  !         function value at xnew is less then epsi1 (real(dp), intent in)
  !  epsi2: slow progress tolerance - progress is assumed to be too slow to
  !         continue if xnew-xold is less than epsi2 (real(dp), intent in)
  !  maxit: maximum number of iterations allowed (integer, intent in)
  !  numit: number of iterations already performed (intent, out)
  !  exitflag: indicates exit condition from this subroutine
  !    exitflag=1 indicates solution found with tolerance epsi1
  !    exitflag=2 indicates slow progress condition
  !    exitflag=3 maximum number of iterations reached
  !    exitflag=4 divergence test failed (difference between successive
  !               root estimates increasing)
```

```
  ! Code for secant method goes here

  return
end subroutine secant

function poly(x)
  use types
  use globals
  implicit none

  real(dp)::poly
  real(dp), intent(in)::x

  poly=a4*x**4+a3*x**3+a2*x**2+a1*x+a0
  return
end function poly

function polyp(x)
  use types
  use globals
  implicit none

  real(dp)::polyp
  real(dp), intent(in)::x

  polyp=4d0*a4*x**3+3d0*a3*x**2+2d0*a2*x+a1
  return
end function polyp
```