F P A R S E R − F O R T R A N

by Angelo Graziosi

I N T R O D U C T I O N

This document contains a few examples of fortran programs "using" the Function Parser (FParser) library.

It is a C++ library which we have interfaced in fortran exploiting the new Fortran >= 2003 standards.

We started using function parser since the end of the 1980s. Indeed at that time there was, on accademia environments, a parser written in Pascal called FONCTION (or Fonction ptr?, we don’t remember...).

Successively, we rewrote it in C/C++ and used it exensively in many programs.

A few years ago, we found FunctionParser (http://warp.povusers.org/FunctionParser) and interfaced it. Since it is written in C++, we needed to interface it first in C an then in fortran.

To test it, we wrote a simple test program, fparser_test.f90, which, in the initial comment, explains as the parser is built.

fparser_dp.f90 and fparser_cd.f90 contain the modules which interface real and complex functions. cwrapper_fparser.cc is the C interface to the parser.

With these modules, we have also written some BGI applications (see for example dynamics2d.f90, in the document about BGI−Fortran on this site).

This document has been created using EMACS (and some "friends" tools like ps2pdf, pdftk etc..).

```
!
! Fortran Interface to the Function Parser Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD
!
!   cd ~/work
!   wget http://warp.povusers.org/FunctionParser/fparser4.5.1.zip
!   mkdir fparser−4.5.1
!   bsdtar −xvof fparser4.5.1.zip −C fparser−4.5.1
!   apack fparser−4.5.1−src.tar.xz fparser−4.5.1
!   mv fparser−4.5.1−src.tar.xz ports−packages/fparser
!   cd fparser−4.5.1
!
!   g++[−mp−4.9] −Wall −O2 −DFP_SUPPORT_FLOAT_TYPE [−DFP_USE_STRTOLD] \
!     −DFP_SUPPORT_LONG_DOUBLE_TYPE \
!     −DFP_SUPPORT_LONG_INT_TYPE −DFP_SUPPORT_COMPLEX_DOUBLE_TYPE \
!     −DFP_SUPPORT_COMPLEX_FLOAT_TYPE −DFP_SUPPORT_COMPLEX_LONG_DOUBLE_TYPE \
!     −DFP_USE_THREAD_SAFE_EVAL −DFP_USE_THREAD_SAFE_EVAL_WITH_ALLOCA \
!     −c {fparser.cc,fpoptimizer.cc}
!
!   mv *.o ~/programming/fparser−fortran/
!
! It seems that on OSX 10.9 (Mavericks), with g++−mp−4.9 (MacPorts),
! it accepts also DFP_USE_STRTOLD.
!
!   cd ~/programming/fparser−fortran
!
!   g++[−mp−4.9] −Wall −O2 −I ~/work/fparser−4.5.1 −c cwrapper_fparser.cc
!
!   ar rcs libFParser.a fparser.o fpoptimizer.o cwrapper_fparser.o
!
!   mkdir −p ~/programming/lib/$PLATFORM
!   mv libFParser.a ~/programming/lib/$PLATFORM
!
! being PLATFORM: EMPTY (GNU/Linux), msys2/mingw32/mingw64 (MSYS2)
!
!   rm *.o
!
!   gfortran[−mp−4.9] −Wall −O3 −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     fparser_dp.f90 fparser_cd.f90 fparser_test.f90 \
!     −L ~/programming/lib/$PLATFORM −lFParser −lstdc++ −o fparser_test[.out]
!


!
! The functions could be defined also in the "contains" section of the main...
!
module adding_functions
  use kind_consts, only: DP
  implicit none
  private

  public :: sqr_d, sqr_c

contains
  function sqr_d(p) result (s)
    real(DP), intent(in) :: p(*)
    real(DP) :: s
    s = p(1)*p(1)
  end function sqr_d

  function sqr_c(p) result (s)
    complex(DP), intent(in) :: p(*)
    complex(DP) :: s
    s = p(1)*p(1)
  end function sqr_c
end module adding_functions

program fparser_test
  use kind_consts, only: DP
  use get_data, only: get, MAXLEN
  use adding_functions
```

```fortran
  implicit none
  call test_fx()
  call test_fz()
contains
  subroutine test_fx()
    use fparser_dp
    character(len=MAXLEN) :: fcn_str = 'x*x'
    type(FunctionParser_type) fparser, fp2
    integer :: res, use_degrees = 0, ammount
    real(DP) :: vals(2), minx = −5.0_DP, maxx = 5.0_DP, stp = 1.0_DP

    call NewParser(fparser)
    call NewParser(fp2)

    if (AddConstant(fparser,'pi',3.1415926535897932_DP) <= 0) then
       write(*,*) "AddConstant() error..."
       stop
    end if

    if (AddUnit(fparser,'cm',100.0_DP) <= 0) then
       write(*,*) "AddUnit() error..."
       stop
    end if

    if (AddFunction(fparser,'sqr',sqr_d,1) <= 0) then
       write(*,*) "AddFunction() error (Fortran function)..."
       stop
    end if

    do
       call get("Degrees (0 = radians,1 = degrees):",use_degrees)
       call get("f(x) =",fcn_str)

       if (use_degrees > 0) then
          res = Parse(fparser,fcn_str,'x',1)
       else
          res = Parse(fparser,fcn_str,'x')
       end if

       if (res < 0) exit

       write(*,'(A)') 'f(x) = '//trim(fcn_str)
       write(*,'(A)') repeat(' ',res+7)//'^'
       !
       ! Remember : ErrorMsg() is an array of characters...
       !
       write(*,*) ErrorMsg(fparser)
       write(*,*) 'Error type: ',GetParseErrorType(fparser)
       write(*,*)
    end do

    call Optimize(fparser)

    if (AddFunction(fp2,'phi',fparser) <= 0) then
       write(*,*) "AddFunction() error (fparser function)..."
       stop
    end if

    do
       call get("Degrees (0 = radians,1 = degrees):",use_degrees)
       call get("F(x,y) =",fcn_str)

       if (use_degrees > 0) then
          res = Parse(fp2,fcn_str,'x,y',1)
       else
          res = Parse(fp2,fcn_str,'x,y')
       end if

       if (res < 0) exit

       write(*,'(A)') 'F(x,y) = '//trim(fcn_str)
       write(*,'(A)') repeat(' ',res+9)//'^'
       !
       ! Remember : ErrorMsg() is an array of characters...
       !
       write(*,*) ErrorMsg(fp2)
       write(*,*) 'Error type: ',GetParseErrorType(fp2)
       write(*,*)
```

```
   end do

   call get("min x:",minx)
   call get("max x:",maxx)
   call get("step:",stp)

   vals(1) = minx
   do while(vals(1) <= maxx)
      write(*,*) 'f(',vals(1),') = ',Eval(fparser,vals)
      res = EvalError(fparser)
      if (res > 0) then
         write(*,*) 'Eval() error: ',res
         stop
      end if
      vals(1) = vals(1)+stp
   end do

   vals(1) = minx
   vals(2) = 1.0_DP
   do while(vals(1) <= maxx)
      write(*,*) 'F(',vals(1),',',vals(2),') = ',Eval(fp2,vals)
      res = EvalError(fp2)
      if (res > 0) then
         write(*,*) 'Eval() error: ',res
         stop
      end if
      vals(1) = vals(1)+stp
   end do

   call set_epsilon(1E−7_DP)
   write(*,*) 'current eps= ',get_epsilon()

   write(*,*) 'freeing memory...'
   call DeleteParser(fp2)
   call DeleteParser(fparser)

   write(*,*) 'NOW TESTING setDelimiterChar...'
   write(*,*)

   call NewParser(fparser)

   call setDelimiterChar(fparser,'}')

   res = Parse(fparser,'(y*y)+1 }','y')

   ! res = Parse(fparser,'(y*y)+1','y')
   ! vals(1) = 2.0_DP
   ! write(*,*) 'f(y) = ', Eval(fparser,vals)

   write(*,*) 'The } is at position ', res

   call DeleteParser(fparser)

   !
   ! Quick tests
   !
   call NewParser(fparser)

   res = AddConstant(fparser,'pi',3.1415926535897932_DP)
   res = ParseAndDeduceVariables(fparser,'x*sin(pi/2)')

   vals(1) = 2.71828_DP
   write(*,*) 'fun() = ',Eval(fparser,vals)

   call DeleteParser(fparser)

   call NewParser(fparser)

   res = AddConstant(fparser,'pi',3.1415926535897932_DP)
   res = ParseAndDeduceVariables(fparser,'x*sin(pi/2)',ammount)

   vals(1) = 2.71828_DP
   write(*,*) 'fun() = ',Eval(fparser,vals)
   write(*,*) 'ammount = ',ammount

   call DeleteParser(fparser)

   call NewParser(fparser)
```

```fortran
    res = AddConstant(fparser,'pi',180.0_DP)
    res = ParseAndDeduceVariables(fparser,'x*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
    call NewParser(fparser)

    res = AddConstant(fparser,'pi',180.0_DP)
    res = ParseAndDeduceVariables(fparser,'x*y*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    vals(2) = 2.0_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
    call NewParser(fparser)

    res = AddConstant(fparser,'pi',180.0_DP)
    res = ParseAndDeduceVariables(fparser,'phy*chi*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    vals(2) = 3.0_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
  end subroutine test_fx
  subroutine test_fz()
    use fparser_cd
    character(len=MAXLEN) :: fcn_str = 'z*z'
    type(FunctionParser_cd_type) fparser, fp2
    integer :: res, use_degrees = 0, ammount
    complex(DP) :: vals(2)
    complex(DP), parameter :: JJ = (0,1)
    real(DP) :: minx = −5.0_DP, maxx = 5.0_DP, stp = 1.0_DP, x, y = 1.0_DP

    call NewParser(fparser)
    call NewParser(fp2)

    if (AddConstant(fparser,'j',JJ) <= 0) then
       write(*,*) "AddConstant() error (j complex)..."
       stop
    end if

    if (AddConstant(fparser,'pi',(3.1415926535897932_DP,0.0_DP)) <= 0) then
       write(*,*) "AddConstant() error (pi complex)..."
       stop
    end if

    if (AddUnit(fparser,'cm',(100.0_DP,100.0_DP)) <= 0) then
       write(*,*) "AddUnit() error (complex)..."
       stop
    end if

    if (AddFunction(fparser,'sqr',sqr_c,1) <= 0) then
       write(*,*) "AddFunction() error (complex Fortran function)..."
       stop
    end if

    if (RemoveIdentifier(fparser,'pi') <= 0) then
       write(*,*) "RemoveIdentifier() error (fparser complex function)..."
       stop
    end if

    do
       call get("Degrees (0 = radians,1 = degrees):",use_degrees)
       call get("f(z) =",fcn_str)

       if (use_degrees > 0) then
          res = Parse(fparser,fcn_str,'z',1)
       else
          res = Parse(fparser,fcn_str,'z')
       end if
```

```
      if (res < 0) exit

      write(*,'(A)') 'f(z) = '//trim(fcn_str)
      write(*,'(A)') repeat(' ',res+7)//'^'
      !
      ! Remember : ErrorMsg() is an array of characters...
      !
      write(*,*) ErrorMsg(fparser)
      write(*,*) 'Error type: ',GetParseErrorType(fparser)
      write(*,*)
   end do

   call Optimize(fparser)

   if (AddFunction(fp2,'phi',fparser) <= 0) then
      write(*,*) "AddFunction() error (fparser complex function)..."
      stop
   end if

   do
      call get("Degrees (0 = radians,1 = degrees):",use_degrees)
      call get("F(z,w) =",fcn_str)

      if (use_degrees > 0) then
         res = Parse(fp2,fcn_str,'z,w',1)
      else
         res = Parse(fp2,fcn_str,'z,w')
      end if

      if (res < 0) exit

      write(*,'(A)') 'F(z,w) = '//trim(fcn_str)
      write(*,'(A)') repeat(' ',res+9)//'^'
      !
      ! Remember : ErrorMsg() is an array of characters...
      !
      write(*,*) ErrorMsg(fp2)
      write(*,*) 'Error type: ',GetParseErrorType(fp2)
      write(*,*)
   end do

   call get("min x:",minx)
   call get("max x:",maxx)
   call get("y:",y)
   call get("step:",stp)

   x = minx
   do while(x <= maxx)
      vals(1) = x+JJ*y
      write(*,*) 'f(',vals(1),') = ',Eval(fparser,vals)
      res = EvalError(fparser)
      if (res > 0) then
         write(*,*) 'Eval() error: ',res
         stop
      end if
      x = x+stp
   end do

   x = minx
   vals(2) = 1.0_DP+JJ*1.0_DP
   do while(x <= maxx)
      vals(1) = x+JJ*y
      write(*,*) 'F(',vals(1),',',vals(2),') = ',Eval(fp2,vals)
      res = EvalError(fp2)
      if (res > 0) then
         write(*,*) 'Eval() error: ',res
         stop
      end if
      x = x+stp
   end do

   call set_epsilon(1E−7_DP*(1.0_DP+JJ))
   write(*,*) 'current eps= ',get_epsilon()

   write(*,*) 'freeing memory...'
   call DeleteParser(fp2)
   call DeleteParser(fparser)
```

```fortran
    write(*,*) 'NOW TESTING setDelimiterChar...'
    write(*,*)

    call NewParser(fparser)

    call setDelimiterChar(fparser,'}')

    res = Parse(fparser,'(y*y)+1 }','y')

    ! res = Parse(fparser,'(y*y)+1','y')
    ! vals(1) = 2.0_DP+JJ*0.0_DP
    ! write(*,*) 'f(y) = ', Eval(fparser,vals)

    write(*,*) 'The } is at position ', res

    call DeleteParser(fparser)

    !
    ! Quick tests
    !
    call NewParser(fparser)

    res = AddConstant(fparser,'pi',3.1415926535897932_DP+JJ*0)
    res = ParseAndDeduceVariables(fparser,'z*sin(pi/2)')

    vals(1) = 2.71828_DP+JJ*0
    write(*,*) 'fun() = ',Eval(fparser,vals)

    call DeleteParser(fparser)

    call NewParser(fparser)

    res = AddConstant(fparser,'pi',(3.1415926535897932_DP,0.0_DP))
    res = ParseAndDeduceVariables(fparser,'z*sin(pi/2)',ammount)

    vals(1) = 2.71828_DP
    write(*,*) 'fun() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)

    call NewParser(fparser)

    res = AddConstant(fparser,'pi',(180.0_DP,0.0_DP))
    res = ParseAndDeduceVariables(fparser,'z*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
    call NewParser(fparser)

    res = AddConstant(fparser,'pi',180.0_DP+JJ*0)
    res = ParseAndDeduceVariables(fparser,'x*y*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    vals(2) = 2.0_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
    call NewParser(fparser)

    res = AddConstant(fparser,'pi',180.0_DP+JJ*0)
    res = ParseAndDeduceVariables(fparser,'phy*chi*sin(pi/2)',ammount,1)

    vals(1) = 2.71828_DP
    vals(2) = 3.0_DP
    write(*,*) 'fun_deg() = ',Eval(fparser,vals)
    write(*,*) 'ammount = ',ammount

    call DeleteParser(fparser)
  end subroutine test_fz
end program fparser_test
```

```fortran
!
! Fortran Interface to the Function Parser Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! The simplest interface (real dp) to the
! Function Parser Library for C++ (http://warp.povusers.org/FunctionParser)
!
! References
!
!   http://fortranwiki.org/fortran/show/Fortran+and+Cpp+objects
!   http://fortranwiki.org/fortran/show/Generating+C+Interfaces
!
! See also:
!
!   http://fortranwiki.org/fortran/show/c_interface_module
!
! N.B.
!
!   Since these module is an interface to C/C++ routines, a few routines
!   handling characters (string) follow the "C rules".
!   So Parse() when used in associacition with setDelimiterChar() will
!   return position in C style, i.e. starting from 0 (zero)...
!

module fparser_dp
  use kind_consts, only: DP
  use, intrinsic :: iso_c_binding, only: C_INT, C_PTR, C_NULL_PTR, &
       C_CHAR, C_DOUBLE, C_NULL_CHAR, C_SIZE_T, C_FUNPTR, &
       c_funloc!, c_f_pointer
  implicit none
  private

  type FunctionParser_type
     private
     type(C_PTR) :: object = C_NULL_PTR
  end type FunctionParser_type

  interface

     function C_FunctionParser__new() result(this) &
          bind(C,name='FunctionParser__new')
       import :: C_PTR
       type(C_PTR) :: this
     end function C_FunctionParser__new

     function C_FunctionParser__Parse(this,fcn,vars,useDegrees) result(Parse) &
          bind(C,name='FunctionParser__Parse')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: Parse
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: fcn(*), vars(*)
       integer(C_INT), value :: useDegrees
     end function C_FunctionParser__Parse

     subroutine C_FunctionParser__setDelimiterChar(this,c) &
          bind(C,name='FunctionParser__setDelimiterChar')
       import :: C_CHAR, C_PTR
       type(C_PTR), value :: this
       character(C_CHAR), value :: c
     end subroutine C_FunctionParser__setDelimiterChar

     function C_static_FunctionParser__epsilon() result(epsilon) &
          bind(C,name='static_FunctionParser__epsilon')
       import :: C_DOUBLE
       real(C_DOUBLE) :: epsilon
     end function C_static_FunctionParser__epsilon

     subroutine C_static_FunctionParser__setEpsilon(e) &
          bind(C,name='static_FunctionParser__setEpsilon')
       import :: C_DOUBLE
       real(C_DOUBLE), value :: e
     end subroutine C_static_FunctionParser__setEpsilon
```

```fortran
function C_FunctionParser__ErrorMsg(this) result(ErrorMsg) &
     bind(C,name='FunctionParser__ErrorMsg')
  import :: C_PTR
  type(C_PTR) :: ErrorMsg
  type(C_PTR), value :: this
end function C_FunctionParser__ErrorMsg

function C_FunctionParser__GetParseErrorType(this) &
     result(GetParseErrorType) &
     bind(C,name='FunctionParser__GetParseErrorType')
  import :: C_INT, C_PTR
  integer(C_INT) :: GetParseErrorType
  type(C_PTR), value :: this
end function C_FunctionParser__GetParseErrorType

function C_FunctionParser__Eval(this,vars) result(Eval) &
     bind(C,name='FunctionParser__Eval')
  import :: C_DOUBLE, C_PTR
  real(C_DOUBLE) :: Eval
  type(C_PTR), value :: this
  real(C_DOUBLE), intent(in) :: vars(*)
end function C_FunctionParser__Eval

function C_FunctionParser__EvalError(this) &
     result(EvalError) &
     bind(C,name='FunctionParser__EvalError')
  import :: C_INT, C_PTR
  integer(C_INT) :: EvalError
  type(C_PTR), value :: this
end function C_FunctionParser__EvalError

subroutine C_FunctionParser__Optimize(this) &
     bind(C,name='FunctionParser__Optimize')
  import :: C_PTR
  type(C_PTR), value :: this
end subroutine C_FunctionParser__Optimize

function C_FunctionParser__AddConstant(this,name,val) result(AddConstant) &
     bind(C,name='FunctionParser__AddConstant')
  import :: C_CHAR, C_DOUBLE, C_INT, C_PTR
  integer(C_INT) :: AddConstant
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  real(C_DOUBLE), value :: val
end function C_FunctionParser__AddConstant

function C_FunctionParser__AddUnit(this,name,val) result(AddUnit) &
     bind(C,name='FunctionParser__AddUnit')
  import :: C_CHAR, C_DOUBLE, C_INT, C_PTR
  integer(C_INT) :: AddUnit
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  real(C_DOUBLE), value :: val
end function C_FunctionParser__AddUnit

function C_FunctionParser__AddFunction(this,name,functionPtr, &
     paramsAmount) result(AddFunction) &
     bind(C,name='FunctionParser__AddFunction')
  import :: C_CHAR, C_FUNPTR, C_INT, C_PTR
  integer(C_INT) :: AddFunction
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  type(C_FUNPTR), value :: functionPtr
  integer(C_INT), value :: paramsAmount
end function C_FunctionParser__AddFunction

function C_FunctionParser__AddFunction2(this,name,fp) &
     result(AddFunction2) &
     bind(C,name='FunctionParser__AddFunction2')
  import :: C_CHAR, C_INT, C_PTR
  integer(C_INT) :: AddFunction2
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  type(C_PTR), value :: fp
end function C_FunctionParser__AddFunction2

function C_FunctionParser__RemoveIdentifier(this,name) &
     result(RemoveIdentifier) &
```

```
          bind(C,name='FunctionParser__RemoveIdentifier')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: RemoveIdentifier
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: name(*)
     end function C_FunctionParser__RemoveIdentifier

     function C_FunctionParser__ParseAndDeduceVariables(this,fcn, &
          amountOfVariablesFound,useDegrees) result(ParseAndDeduceVariables) &
          bind(C,name='FunctionParser__ParseAndDeduceVariables')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: ParseAndDeduceVariables
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: fcn(*)
       integer(C_INT), intent(out) :: amountOfVariablesFound
       integer(C_INT), value :: useDegrees
     end function C_FunctionParser__ParseAndDeduceVariables

     subroutine C_FunctionParser__delete(this) &
          bind(C,name='FunctionParser__delete')
       import :: C_PTR
       type(C_PTR), value :: this
     end subroutine C_FunctionParser__delete

  end interface

  interface NewParser
     module procedure FunctionParser__new
  end interface NewParser

  interface Parse
     module procedure FunctionParser__Parse
  end interface Parse

  interface setDelimiterChar
     module procedure FunctionParser__setDelimiterChar
  end interface setDelimiterChar

  interface get_epsilon
     module procedure static_FunctionParser__epsilon
  end interface get_epsilon

  interface set_epsilon
     module procedure static_FunctionParser__setEpsilon
  end interface set_epsilon

  interface ErrorMsg
     module procedure FunctionParser__ErrorMsg
  end interface ErrorMsg

  interface GetParseErrorType
     module procedure FunctionParser__GetParseErrorType
  end interface GetParseErrorType

  interface Eval
     module procedure FunctionParser__Eval
  end interface Eval

  interface EvalError
     module procedure FunctionParser__EvalError
  end interface EvalError

  interface Optimize
     module procedure FunctionParser__Optimize
  end interface Optimize

  interface AddConstant
     module procedure FunctionParser__AddConstant
  end interface AddConstant

  interface AddUnit
     module procedure FunctionParser__AddUnit
  end interface AddUnit

  interface AddFunction
     module procedure FunctionParser__AddFunction,FunctionParser__AddFunction2
  end interface AddFunction
  interface RemoveIdentifier
     module procedure FunctionParser__RemoveIdentifier
  end interface RemoveIdentifier

  interface ParseAndDeduceVariables
     module procedure FunctionParser__ParseAndDeduceVariables
  end interface ParseAndDeduceVariables

  interface DeleteParser
     module procedure FunctionParser__delete
  end interface DeleteParser

  public :: AddConstant, AddFunction, AddUnit, DeleteParser, ErrorMsg, Eval, &
       EvalError, FunctionParser_type, get_epsilon, GetParseErrorType, &
       NewParser, Optimize, Parse, ParseAndDeduceVariables, RemoveIdentifier, &
       setDelimiterChar, set_epsilon

contains

  ! Fortran wrapper routines to interface C wrappers
  subroutine FunctionParser__new(this)
    type(FunctionParser_type), intent(out) :: this
    this%object = C_FunctionParser__new()
  end subroutine FunctionParser__new

  function FunctionParser__Parse(this,fcn,vars,useDegrees) result(Parse)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: fcn, vars
    integer, intent(in), optional :: useDegrees
    integer :: Parse
    integer :: degrees = 0

    if (present(useDegrees)) degrees = useDegrees

    Parse = C_FunctionParser__Parse(this%object,trim(fcn)//C_NULL_CHAR, &
         trim(vars)//C_NULL_CHAR,degrees)
  end function FunctionParser__Parse

  subroutine FunctionParser__setDelimiterChar(this,c)
    type(FunctionParser_type), intent(in) :: this
    character(len=1), intent(in) :: c
    call C_FunctionParser__setDelimiterChar(this%object,c)
  end subroutine FunctionParser__setDelimiterChar

  function static_FunctionParser__epsilon() result(epsilon)
    real(DP) :: epsilon
    epsilon = C_static_FunctionParser__epsilon()
  end function static_FunctionParser__epsilon

  subroutine static_FunctionParser__setEpsilon(e)
    real(DP), intent(in) :: e
    call C_static_FunctionParser__setEpsilon(e)
  end subroutine static_FunctionParser__setEpsilon

  ! function FunctionParser__ErrorMsg(this) result(ErrorMsg)
  !   use general_routines, only: str_len
  !   type(FunctionParser_type), intent(in) :: this
  !   type(C_PTR) :: cstr
  !   integer :: len
  !   character, pointer :: str
  !   character(len=70) :: ErrorMsg
  !   cstr = C_FunctionParser__ErrorMsg(this%object)
  !   len = str_len(cstr)
  !   call c_f_pointer(cstr,str)
  !   ErrorMsg = trim(adjustl(str(1:len)))
  ! end function FunctionParser__ErrorMsg

  function FunctionParser__ErrorMsg(this) result(ErrorMsg)
    use general_routines, only: c_f_string
    type(FunctionParser_type), intent(in) :: this
    character, dimension(:), pointer :: ErrorMsg
    ErrorMsg => c_f_string(C_FunctionParser__ErrorMsg(this%object))
  end function FunctionParser__ErrorMsg

  function FunctionParser__GetParseErrorType(this) result(GetParseErrorType)
    type(FunctionParser_type), intent(in) :: this
    integer :: GetParseErrorType
    GetParseErrorType = C_FunctionParser__GetParseErrorType(this%object)
```

```fortran
  end function FunctionParser__GetParseErrorType

  function FunctionParser__Eval(this,vars) result(Eval)
    type(FunctionParser_type), intent(in) :: this
    real(DP), intent(in) :: vars(:)
    real(DP) :: Eval
    Eval = C_FunctionParser__Eval(this%object,vars)
  end function FunctionParser__Eval

  function FunctionParser__EvalError(this) result(EvalError)
    type(FunctionParser_type), intent(in) :: this
    integer :: EvalError
    EvalError = C_FunctionParser__EvalError(this%object)
  end function FunctionParser__EvalError

  subroutine FunctionParser__Optimize(this)
    type(FunctionParser_type), intent(in) :: this
    call C_FunctionParser__Optimize(this%object)
  end subroutine FunctionParser__Optimize

  function FunctionParser__AddConstant(this,name,val) result(AddConstant)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: name
    real(DP), intent(in) :: val
    integer :: AddConstant
    AddConstant = C_FunctionParser__AddConstant(this%object, &
         trim(name)//C_NULL_CHAR,val)
  end function FunctionParser__AddConstant

  function FunctionParser__AddUnit(this,name,val) result(AddUnit)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: name
    real(DP), intent(in) :: val
    integer :: AddUnit
    AddUnit = C_FunctionParser__AddUnit(this%object, &
         trim(name)//C_NULL_CHAR,val)
  end function FunctionParser__AddUnit

  function FunctionParser__AddFunction(this,name,fcn,paramsAmount) &
       result(AddFunction)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: name

    interface
       function fcn(x) bind(C)
         use iso_c_binding, only: C_DOUBLE
         real(C_DOUBLE), intent(in) :: x(*)
         real(C_DOUBLE) :: fcn
       end function fcn
    end interface

    integer, intent(in) :: paramsAmount
    integer :: AddFunction

    AddFunction = C_FunctionParser__AddFunction(this%object, &
         trim(name)//C_NULL_CHAR,c_funloc(fcn),paramsAmount)
  end function FunctionParser__AddFunction

  function FunctionParser__AddFunction2(this,name,fp) &
       result(AddFunction2)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: name
    type(FunctionParser_type), intent(in) :: fp
    integer :: AddFunction2

    AddFunction2 = C_FunctionParser__AddFunction2(this%object, &
         trim(name)//C_NULL_CHAR,fp%object)
  end function FunctionParser__AddFunction2

  function FunctionParser__RemoveIdentifier(this,name) &
       result(RemoveIdentifier)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: name
    integer :: RemoveIdentifier

    RemoveIdentifier = C_FunctionParser__RemoveIdentifier(this%object, &
         trim(name)//C_NULL_CHAR)
  end function FunctionParser__RemoveIdentifier
  function FunctionParser__ParseAndDeduceVariables(this,fcn, &
       amountOfVariablesFound,useDegrees) result(ParseAndDeduceVariables)
    type(FunctionParser_type), intent(in) :: this
    character(len=*), intent(in) :: fcn
    integer, intent(out), optional :: amountOfVariablesFound
    integer, intent(in), optional :: useDegrees
    integer :: ParseAndDeduceVariables
    integer :: ammount = −1,degrees = 0

    if (present(amountOfVariablesFound)) ammount = amountOfVariablesFound
    if (present(useDegrees)) degrees = useDegrees

    ParseAndDeduceVariables = &
         C_FunctionParser__ParseAndDeduceVariables(this%object, &
         trim(fcn)//C_NULL_CHAR,ammount,degrees)

    if (present(amountOfVariablesFound)) amountOfVariablesFound = ammount
  end function FunctionParser__ParseAndDeduceVariables

  subroutine FunctionParser__delete(this)
    type(FunctionParser_type), intent(inout) :: this
    call C_FunctionParser__delete(this%object)
    this%object = C_NULL_PTR
  end subroutine FunctionParser__delete
end module fparser_dp
```

```
!
! Fortran Interface to the Function Parser Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! The simplest interface (complex double) to the
! Function Parser Library for C++ (http://warp.povusers.org/FunctionParser)
!
! References
!
!   http://fortranwiki.org/fortran/show/Fortran+and+Cpp+objects
!   http://fortranwiki.org/fortran/show/Generating+C+Interfaces
!
! See also:
!
!   http://fortranwiki.org/fortran/show/c_interface_module
!
! N.B.
!
!   Since these module is an interface to C/C++ routines, a few routines
!   handling characters (string) follow the "C rules".
!   So Parse() when used in associacition with setDelimiterChar() will
!   return position in C style, i.e. starting from 0 (zero)...
!

module fparser_cd
  use kind_consts, only: DP
  use, intrinsic :: iso_c_binding, only: C_INT, C_PTR, C_NULL_PTR, &
       C_CHAR, C_DOUBLE_COMPLEX, C_NULL_CHAR, C_SIZE_T, C_FUNPTR, &
       c_funloc!, c_f_pointer
  implicit none
  private

  type FunctionParser_cd_type
     private
     type(C_PTR) :: object = C_NULL_PTR
  end type FunctionParser_cd_type

  interface

     function C_FunctionParser_cd__new() result(this) &
          bind(C,name='FunctionParser_cd__new')
       import :: C_PTR
       type(C_PTR) :: this
     end function C_FunctionParser_cd__new

     function C_FunctionParser_cd__Parse(this,fcn,vars,useDegrees) &
          result(Parse) bind(C,name='FunctionParser_cd__Parse')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: Parse
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: fcn(*), vars(*)
       integer(C_INT), value :: useDegrees
     end function C_FunctionParser_cd__Parse

     subroutine C_FunctionParser_cd__setDelimiterChar(this,c) &
          bind(C,name='FunctionParser_cd__setDelimiterChar')
       import :: C_CHAR, C_PTR
       type(C_PTR), value :: this
       character(C_CHAR), value :: c
     end subroutine C_FunctionParser_cd__setDelimiterChar

     function C_static_FunctionParser_cd__epsilon() result(epsilon) &
          bind(C,name='static_FunctionParser_cd__epsilon')
       import :: C_DOUBLE_COMPLEX
       complex(C_DOUBLE_COMPLEX) :: epsilon
     end function C_static_FunctionParser_cd__epsilon

     subroutine C_static_FunctionParser_cd__setEpsilon(e) &
          bind(C,name='static_FunctionParser_cd__setEpsilon')
       import :: C_DOUBLE_COMPLEX
       complex(C_DOUBLE_COMPLEX), value :: e
     end subroutine C_static_FunctionParser_cd__setEpsilon
```

```
function C_FunctionParser_cd__ErrorMsg(this) result(ErrorMsg) &
     bind(C,name='FunctionParser_cd__ErrorMsg')
  import :: C_PTR
  type(C_PTR) :: ErrorMsg
  type(C_PTR), value :: this
end function C_FunctionParser_cd__ErrorMsg

function C_FunctionParser_cd__GetParseErrorType(this) &
     result(GetParseErrorType) &
     bind(C,name='FunctionParser_cd__GetParseErrorType')
  import :: C_INT, C_PTR
  integer(C_INT) :: GetParseErrorType
  type(C_PTR), value :: this
end function C_FunctionParser_cd__GetParseErrorType

function C_FunctionParser_cd__Eval(this,vars) result(Eval) &
     bind(C,name='FunctionParser_cd__Eval')
  import :: C_DOUBLE_COMPLEX, C_PTR
  complex(C_DOUBLE_COMPLEX) :: Eval
  type(C_PTR), value :: this
  complex(C_DOUBLE_COMPLEX), intent(in) :: vars(*)
end function C_FunctionParser_cd__Eval

function C_FunctionParser_cd__EvalError(this) &
     result(EvalError) &
     bind(C,name='FunctionParser_cd__EvalError')
  import :: C_INT, C_PTR
  integer(C_INT) :: EvalError
  type(C_PTR), value :: this
end function C_FunctionParser_cd__EvalError

subroutine C_FunctionParser_cd__Optimize(this) &
     bind(C,name='FunctionParser_cd__Optimize')
  import :: C_PTR
  type(C_PTR), value :: this
end subroutine C_FunctionParser_cd__Optimize

function C_FunctionParser_cd__AddConstant(this,name,val) &
     result(AddConstant) bind(C,name='FunctionParser_cd__AddConstant')
  import :: C_CHAR, C_DOUBLE_COMPLEX, C_INT, C_PTR
  integer(C_INT) :: AddConstant
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  complex(C_DOUBLE_COMPLEX), value :: val
end function C_FunctionParser_cd__AddConstant

function C_FunctionParser_cd__AddUnit(this,name,val) result(AddUnit) &
     bind(C,name='FunctionParser_cd__AddUnit')
  import :: C_CHAR, C_DOUBLE_COMPLEX, C_INT, C_PTR
  integer(C_INT) :: AddUnit
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  complex(C_DOUBLE_COMPLEX), value :: val
end function C_FunctionParser_cd__AddUnit

function C_FunctionParser_cd__AddFunction(this,name,functionPtr, &
     paramsAmount) result(AddFunction) &
     bind(C,name='FunctionParser_cd__AddFunction')
  import :: C_CHAR, C_FUNPTR, C_INT, C_PTR
  integer(C_INT) :: AddFunction
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  type(C_FUNPTR), value :: functionPtr
  integer(C_INT), value :: paramsAmount
end function C_FunctionParser_cd__AddFunction

function C_FunctionParser_cd__AddFunction2(this,name,fp) &
     result(AddFunction2) &
     bind(C,name='FunctionParser_cd__AddFunction2')
  import :: C_CHAR, C_INT, C_PTR
  integer(C_INT) :: AddFunction2
  type(C_PTR), value :: this
  character(C_CHAR), intent(in) :: name(*)
  type(C_PTR), value :: fp
end function C_FunctionParser_cd__AddFunction2

function C_FunctionParser_cd__RemoveIdentifier(this,name) &
     result(RemoveIdentifier) &
```

```
          bind(C,name='FunctionParser_cd__RemoveIdentifier')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: RemoveIdentifier
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: name(*)
     end function C_FunctionParser_cd__RemoveIdentifier

     function C_FunctionParser_cd__ParseAndDeduceVariables(this,fcn, &
          amountOfVariablesFound,useDegrees) result(ParseAndDeduceVariables) &
          bind(C,name='FunctionParser_cd__ParseAndDeduceVariables')
       import :: C_CHAR, C_INT, C_PTR
       integer(C_INT) :: ParseAndDeduceVariables
       type(C_PTR), value :: this
       character(C_CHAR), intent(in) :: fcn(*)
       integer(C_INT), intent(out) :: amountOfVariablesFound
       integer(C_INT), value :: useDegrees
     end function C_FunctionParser_cd__ParseAndDeduceVariables

     subroutine C_FunctionParser_cd__delete(this) &
          bind(C,name='FunctionParser_cd__delete')
       import :: C_PTR
       type(C_PTR), value :: this
     end subroutine C_FunctionParser_cd__delete

  end interface

  interface NewParser
     module procedure FunctionParser_cd__new
  end interface NewParser

  interface Parse
     module procedure FunctionParser_cd__Parse
  end interface Parse

  interface setDelimiterChar
     module procedure FunctionParser_cd__setDelimiterChar
  end interface setDelimiterChar

  interface get_epsilon
     module procedure static_FunctionParser_cd__epsilon
  end interface get_epsilon

  interface set_epsilon
     module procedure static_FunctionParser_cd__setEpsilon
  end interface set_epsilon

  interface ErrorMsg
     module procedure FunctionParser_cd__ErrorMsg
  end interface ErrorMsg

  interface GetParseErrorType
     module procedure FunctionParser_cd__GetParseErrorType
  end interface GetParseErrorType

  interface Eval
     module procedure FunctionParser_cd__Eval
  end interface Eval

  interface EvalError
     module procedure FunctionParser_cd__EvalError
  end interface EvalError

  interface Optimize
     module procedure FunctionParser_cd__Optimize
  end interface Optimize

  interface AddConstant
     module procedure FunctionParser_cd__AddConstant
  end interface AddConstant

  interface AddUnit
     module procedure FunctionParser_cd__AddUnit
  end interface AddUnit

  interface AddFunction
     module procedure FunctionParser_cd__AddFunction, &
          FunctionParser_cd__AddFunction2
  end interface AddFunction
```

```fortran
  interface RemoveIdentifier
     module procedure FunctionParser_cd__RemoveIdentifier
  end interface RemoveIdentifier

  interface ParseAndDeduceVariables
     module procedure FunctionParser_cd__ParseAndDeduceVariables
  end interface ParseAndDeduceVariables

  interface DeleteParser
     module procedure FunctionParser_cd__delete
  end interface DeleteParser

  public:: AddConstant, AddFunction, AddUnit, DeleteParser, ErrorMsg, Eval, &
       EvalError, FunctionParser_cd_type, get_epsilon, GetParseErrorType, &
       NewParser, Optimize, Parse, ParseAndDeduceVariables, RemoveIdentifier, &
       setDelimiterChar, set_epsilon

contains

  ! Fortran wrapper routines to interface C wrappers
  subroutine FunctionParser_cd__new(this)
    type(FunctionParser_cd_type), intent(out) :: this
    this%object = C_FunctionParser_cd__new()
  end subroutine FunctionParser_cd__new

  function FunctionParser_cd__Parse(this,fcn,vars,useDegrees) result(Parse)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: fcn, vars
    integer, intent(in), optional :: useDegrees
    integer :: Parse
    integer :: degrees = 0

    if (present(useDegrees)) degrees = useDegrees

    Parse = C_FunctionParser_cd__Parse(this%object,trim(fcn)//C_NULL_CHAR, &
         trim(vars)//C_NULL_CHAR,degrees)
  end function FunctionParser_cd__Parse

  subroutine FunctionParser_cd__setDelimiterChar(this,c)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=1), intent(in) :: c
    call C_FunctionParser_cd__setDelimiterChar(this%object,c)
  end subroutine FunctionParser_cd__setDelimiterChar

  function static_FunctionParser_cd__epsilon() result(epsilon)
    complex(DP) :: epsilon
    epsilon = C_static_FunctionParser_cd__epsilon()
  end function static_FunctionParser_cd__epsilon

  subroutine static_FunctionParser_cd__setEpsilon(e)
    complex(DP), intent(in) :: e
    call C_static_FunctionParser_cd__setEpsilon(e)
  end subroutine static_FunctionParser_cd__setEpsilon

  ! function FunctionParser_cd__ErrorMsg(this) result(ErrorMsg)
  !   use general_routines, only: str_len
  !   type(FunctionParser_cd_type), intent(in) :: this
  !   type(C_PTR) :: cstr
  !   integer :: len
  !   character, pointer :: str
  !   character(len=70) :: ErrorMsg
  !   cstr = C_FunctionParser_cd__ErrorMsg(this%object)
  !   len = str_len(cstr)
  !   call c_f_pointer(cstr,str)
  !   ErrorMsg = trim(adjustl(str(1:len)))
  ! end function FunctionParser_cd__ErrorMsg

  function FunctionParser_cd__ErrorMsg(this) result(ErrorMsg)
    use general_routines, only: c_f_string
    type(FunctionParser_cd_type), intent(in) :: this
    character, dimension(:), pointer :: ErrorMsg
    ErrorMsg => c_f_string(C_FunctionParser_cd__ErrorMsg(this%object))
  end function FunctionParser_cd__ErrorMsg

  function FunctionParser_cd__GetParseErrorType(this) result(GetParseErrorType)
    type(FunctionParser_cd_type), intent(in) :: this
    integer :: GetParseErrorType
    GetParseErrorType = C_FunctionParser_cd__GetParseErrorType(this%object)
  end function FunctionParser_cd__GetParseErrorType

  function FunctionParser_cd__Eval(this,vars) result(Eval)
    type(FunctionParser_cd_type), intent(in) :: this
    complex(DP), intent(in) :: vars(:)
    complex(DP) :: Eval
    Eval = C_FunctionParser_cd__Eval(this%object,vars)
  end function FunctionParser_cd__Eval

  function FunctionParser_cd__EvalError(this) result(EvalError)
    type(FunctionParser_cd_type), intent(in) :: this
    integer :: EvalError
    EvalError = C_FunctionParser_cd__EvalError(this%object)
  end function FunctionParser_cd__EvalError

  subroutine FunctionParser_cd__Optimize(this)
    type(FunctionParser_cd_type), intent(in) :: this
    call C_FunctionParser_cd__Optimize(this%object)
  end subroutine FunctionParser_cd__Optimize

  function FunctionParser_cd__AddConstant(this,name,val) result(AddConstant)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: name
    complex(DP), intent(in) :: val
    integer :: AddConstant
    AddConstant = C_FunctionParser_cd__AddConstant(this%object, &
         trim(name)//C_NULL_CHAR,val)
  end function FunctionParser_cd__AddConstant

  function FunctionParser_cd__AddUnit(this,name,val) result(AddUnit)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: name
    complex(DP), intent(in) :: val
    integer :: AddUnit
    AddUnit = C_FunctionParser_cd__AddUnit(this%object, &
         trim(name)//C_NULL_CHAR,val)
  end function FunctionParser_cd__AddUnit

  function FunctionParser_cd__AddFunction(this,name,fcn,paramsAmount) &
       result(AddFunction)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: name

    interface
       function fcn(z) bind(C)
         use iso_c_binding, only: C_DOUBLE_COMPLEX
         complex(C_DOUBLE_COMPLEX), intent(in) :: z(*)
         complex(C_DOUBLE_COMPLEX) :: fcn
       end function fcn
    end interface

    integer, intent(in) :: paramsAmount
    integer :: AddFunction

    AddFunction = C_FunctionParser_cd__AddFunction(this%object, &
         trim(name)//C_NULL_CHAR,c_funloc(fcn),paramsAmount)
  end function FunctionParser_cd__AddFunction

  function FunctionParser_cd__AddFunction2(this,name,fp) &
       result(AddFunction2)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: name
    type(FunctionParser_cd_type), intent(in) :: fp
    integer :: AddFunction2

    AddFunction2 = C_FunctionParser_cd__AddFunction2(this%object, &
         trim(name)//C_NULL_CHAR,fp%object)
  end function FunctionParser_cd__AddFunction2

  function FunctionParser_cd__RemoveIdentifier(this,name) &
       result(RemoveIdentifier)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: name
    integer :: RemoveIdentifier

    RemoveIdentifier = C_FunctionParser_cd__RemoveIdentifier(this%object, &
         trim(name)//C_NULL_CHAR)
  end function FunctionParser_cd__RemoveIdentifier

  function FunctionParser_cd__ParseAndDeduceVariables(this,fcn, &
       amountOfVariablesFound,useDegrees) result(ParseAndDeduceVariables)
    type(FunctionParser_cd_type), intent(in) :: this
    character(len=*), intent(in) :: fcn
    integer, intent(out), optional :: amountOfVariablesFound
    integer, intent(in), optional :: useDegrees
    integer :: ParseAndDeduceVariables
    integer :: ammount = −1,degrees = 0

    if (present(amountOfVariablesFound)) ammount = amountOfVariablesFound
    if (present(useDegrees)) degrees = useDegrees

    ParseAndDeduceVariables = &
         C_FunctionParser_cd__ParseAndDeduceVariables(this%object, &
         trim(fcn)//C_NULL_CHAR,ammount,degrees)

    if (present(amountOfVariablesFound)) amountOfVariablesFound = ammount
  end function FunctionParser_cd__ParseAndDeduceVariables

  subroutine FunctionParser_cd__delete(this)
    type(FunctionParser_cd_type), intent(inout) :: this
    call C_FunctionParser_cd__delete(this%object)
    this%object = C_NULL_PTR
  end subroutine FunctionParser_cd__delete
end module fparser_cd
```

```
//
// C Interface to the Function Parser Library
// by Angelo Graziosi  (firstname.lastnameATalice.it)
// Copyright Angelo Graziosi
//
// It is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
//
// References
//
//   http://fortranwiki.org/fortran/show/Fortran+and+Cpp+objects
//   http://fortranwiki.org/fortran/show/Generating+C+Interfaces
//
// See also:
//
//   http://fortranwiki.org/fortran/show/c_interface_module
//

#include "fparser.hh"

extern "C"
{
  //
  // C interface to FuncionParser (double)
  //
  FunctionParser *FunctionParser__new()
  {
    return new FunctionParser();
  }

  int FunctionParser__Parse(FunctionParser *This,const char *fcn,
                            const char *vars,int useDegrees)
  {
    return This->Parse(fcn,vars,useDegrees);
  }

  void FunctionParser__setDelimiterChar(FunctionParser *This,char c)
  {
    This->setDelimiterChar(c);
  }

  double static_FunctionParser__epsilon()
  {
    return FunctionParser::epsilon();
  }

  void static_FunctionParser__setEpsilon(double e)
  {
    FunctionParser::setEpsilon(e);
  }

  const char* FunctionParser__ErrorMsg(FunctionParser *This)
  {
    return This->ErrorMsg();
  }

  int FunctionParser__GetParseErrorType(FunctionParser *This)
  {
    return This->GetParseErrorType();
  }

  double FunctionParser__Eval(FunctionParser *This,const double *vars)
  {
    return This->Eval(vars);
  }

  int FunctionParser__EvalError(FunctionParser *This)
  {
    return This->EvalError();
  }

  void FunctionParser__Optimize(FunctionParser *This)
  {
    This->Optimize();
  }
```

```
int FunctionParser__AddConstant(FunctionParser *This,
                                 const char *name,double value)
{
  return This->AddConstant(name,value);
}

int FunctionParser__AddUnit(FunctionParser *This,
                            const char *name,double value)
{
  return This->AddUnit(name,value);
}

int FunctionParser__AddFunction(FunctionParser *This,const char *name,
                                double (*functionPtr)(const double*),
                                int paramsAmount)
{
  return This->AddFunction(name,functionPtr,paramsAmount);
}

int FunctionParser__AddFunction2(FunctionParser *This,const char *name,
                                FunctionParser *fp)
{
  return This->AddFunction(name,(*fp));
}

int FunctionParser__RemoveIdentifier(FunctionParser *This,const char *name)
{
  return This->RemoveIdentifier(name);
}

int
FunctionParser__ParseAndDeduceVariables(FunctionParser *This,
                                        const char *fcn,
                                        int* amountOfVariablesFound,
                                        int useDegrees)
{
  return This->ParseAndDeduceVariables(fcn,amountOfVariablesFound,
                                       useDegrees);
}

void FunctionParser__delete(FunctionParser *This)
{
  delete This;
}

//
// C interface to FuncionParser_cd (complex double)
//
FunctionParser_cd *FunctionParser_cd__new()
{
  return new FunctionParser_cd();
}

int FunctionParser_cd__Parse(FunctionParser_cd *This,const char *fcn,
                          const char *vars,int useDegrees)
{
  return This->Parse(fcn,vars,useDegrees);
}

void FunctionParser_cd__setDelimiterChar(FunctionParser_cd *This,char c)
{
  This->setDelimiterChar(c);
}

std::complex<double> static_FunctionParser_cd__epsilon()
{
  return FunctionParser_cd::epsilon();
}

void static_FunctionParser_cd__setEpsilon(std::complex<double> e)
{
  FunctionParser_cd::setEpsilon(e);
}

const char* FunctionParser_cd__ErrorMsg(FunctionParser_cd *This)
{
  return This->ErrorMsg();
}
```

```
  int FunctionParser_cd__GetParseErrorType(FunctionParser_cd *This)
  {
    return This->GetParseErrorType();
  }

  std::complex<double>
  FunctionParser_cd__Eval(FunctionParser_cd *This,
                          const std::complex<double> *vars)
  {
    return This->Eval(vars);
  }

  int FunctionParser_cd__EvalError(FunctionParser_cd *This)
  {
    return This->EvalError();
  }

  void FunctionParser_cd__Optimize(FunctionParser_cd *This)
  {
    This->Optimize();
  }

  int
  FunctionParser_cd__AddConstant(FunctionParser_cd *This,
                                 const char *name,std::complex<double> value)
  {
    return This->AddConstant(name,value);
  }

  int FunctionParser_cd__AddUnit(FunctionParser_cd *This,
                                 const char *name,std::complex<double> value)
  {
    return This->AddUnit(name,value);
  }

  int FunctionParser_cd__AddFunction(FunctionParser_cd *This,const char *name,
    std::complex<double> (*functionPtr)(const std::complex<double>*),
    int paramsAmount)
  {
    return This->AddFunction(name,functionPtr,paramsAmount);
  }

  int FunctionParser_cd__AddFunction2(FunctionParser_cd *This,const char *name,
                                      FunctionParser_cd *fp)
  {
    return This->AddFunction(name,(*fp));
  }

  int FunctionParser_cd__RemoveIdentifier(FunctionParser_cd *This,
                                          const char *name)
  {
    return This->RemoveIdentifier(name);
  }

  int
  FunctionParser_cd__ParseAndDeduceVariables(FunctionParser_cd *This,
                                          const char *fcn,
                                          int* amountOfVariablesFound,
                                          int useDegrees)
  {
    return This->ParseAndDeduceVariables(fcn,amountOfVariablesFound,
                                         useDegrees);
  }

  void FunctionParser_cd__delete(FunctionParser_cd *This)
  {
    delete This;
  }

}
```