Math 4610 Lecture Notes

How to Build a Shared Library [*]

Joe Koebbe

September 5, 2019

[*]These notes are part of an Open Resource Educational project sponsored by Utah State University

Steps for Building a Shared Library:

Shared libraries are essential for reusing code. If there are a number of routines or methods that you plan to use over and over, it makes sense to write the code once, debug the code and then keep the code in a central location. Then when the codes are needed, a code can link to the shared library and access the code. In this part of the course, we will learn to build and use shared libraries.

Setting Things Up

To complete this work, you will need Cygwin or Linux up on your own laptop or go to the Engineering Computer Lab on the third floor of the Engineering Building. Once you have a command line terminal up and running, start by making a directory in which to develop and keep. So, create a folder in the
math4610
folder. Note that this is not a repository at this point. If you do not have a folder named math4610 and then use
cd math4610 mkdir src cd src
Once these commands have created the proper structure, you can create files with code we need.
There are four codes that we can put into a shared libraries. These are
* smaceps - single precision machine epsilon computation
* dmaceps - double precision machine epsilon computation
* errabs - compute the absolute error in approximation of one number by another
* errrel - compute the relative error in approximation of one number by another
In the end, this may seem like a lot of work. But imagine having 200 routines that will be used over and over. Then this gets a bit more interesting.

The Codes We Have:

The first if we write the code in Fortran it might look like the following:

```
c
c -------------------------------------------------------------------------
c
c Author: Joe Koebbe
c         Department of Mathematics and Statistics
c         Utah State University
c
c Subroutine Name:  abserr
c
c Description: This code returns the distance or absolute error between
c              an exact value and an approximate value
c
c Input:
c
c       x - the approximate value
c       xe - the exact value
c
c Output:
c
c       eabs - the return value containing the absolute error of the two
c              input values
c
c -------------------------------------------------------------------------
c
```

```
      program old_main
c
c set up the problem
c ------------------
c
      real seps = 1.0
      integer ipow = 0
      call smaceps(seps, ipow)
      tval = 1.0 / 2.0**(ipow+1)
      print *, "Output from smaceps:"
      print *, ""
      print *, tval, ipow
c
c stop
c ----
c
      stop
      end
c
c -------------------------------------------------------------------------
c -------------------------------------------------------------------------
c
      subroutine smaceps(seps, ipow)
c
c intialize some variables for the iteration
c -------------------------------------------
c
      one = 1.0
      eps = 1.0
c
c ipow keeps track of the power of two that is determined by machine
c precision
c ---------
c
      ipow = -1
c
c set a loop that will kick out when precision is met
c ----------------------------------------------------
c
      do 1 i=1,1000
c
c add a perturbation to 1
c -----------------------
c
         appone = one + eps
c
c call the routine for absolute error
c -----------------------------------
c
         call abserr(diff, appone, one)
c
c check the difference to see if the two are the same to machine
c precision
c ---------
```

```
c
          if(diff .eq. 0.0) return
c
          seps = diff
c
c update the power and the value of the perturbation
c --------------------------------------------------
c
          eps = 0.5 * eps
          ipow = ipow + 1
c
    1 continue
c
c all done
c --------
c
      return
      end
c
c -------------------------------------------------------------------------
c -------------------------------------------------------------------------
c
      subroutine abserr(eabs, x, xe)
c
c just take the absolute value of the two numbers and set the output
c for the routine
c ---------------
c
      eabs = abs( x - xe )
c
c done
c ----
c
      return
      end
```

If you compile the code in the block above as we have alreay done in the class it will produce the desired result. So, at the command prompt, typing

```
% gfortran -o old_main old_main.f
```

will produce the executable **a.exe** and typing the command

```
% ./a.exe
```

will produce output that will give the number of powers of 2 and the machine precision value.

```
output from smaceps:
5.96046448E-08          23
```

which means single precision will give about 8 decimal digits of accuracy.

Creating Pieces and Stuffing the Object Modules into a Shared Library

The first step is to pull off the two subroutines in the code in the last section. So, there are two codes to include:

```
      subroutine smaceps(seps, ipow)
c
c intialize some variables for the iteration
c -------------------------------------------
c
      one = 1.0
      eps = 1.0
c
c ipow keeps track of the power of two that is determined by machine
c precision
c ---------
c
      ipow = -1
c
c set a loop that will kick out when precision is met
c ----------------------------------------------------
c
      do 1 i=1,1000
c
c add a perturbation to 1
c -----------------------
c
         appone = one + eps
c
c call the routine for absolute error
c -----------------------------------
c
         call abserr(diff, appone, one)
c
c check the difference to see if the two are the same to machine
c precision
c ---------
c
         if(diff .eq. 0.0) return
c
         seps = diff
c
c update the power and the value of the perturbation
c --------------------------------------------------
c
         eps = 0.5 * eps
         ipow = ipow + 1
c
    1 continue
c
c all done
c --------
c
```

```
      return
      end
```

---

and

```
      subroutine abserr(eabs, x, xe)
c
c just take the absolute value of the two numbers and set the output
c for the routine
c ---------------
c
      eabs = abs( x - xe )
c
c done
c ----
c
      return
      end
```

Also, we will need to split off the main program that should look like the following.

```
c
c -----------------------------------------------------------------------
c
c Author: Joe Koebbe
c         Department of Mathematics and Statistics
c         Utah State University
c
c Subroutine Name:  abserr
c
c Description: This code returns the distance or absolute error between
c              an exact value and an approximate value
c
c Input:
c
c       x - the approximate value
c       xe - the exact value
c
c Output:
c
c       eabs - the return value containing the absolute error of the two
c              input values
c
c -----------------------------------------------------------------------
c
      program old_main
c
c set up the problem
c ------------------
c
      real seps = 1.0
      integer ipow = 0
      call smaceps(seps, ipow)
      tval = 1.0 / 2.0**(ipow+1)
      print *, "output from smaceps:"
      print *, tval, ipow
c
c stop
c ----
c
      stop
      end
```

In this example, there will be four files:

- old_main.f - all of the code is in one file
- main.f - just the main part of the code
- smaceps.f - this file contains the code to compute the machine epsilon
- abserr.f - the routine that computes the absolute error

Craating the Object Modules for the Routines

Use the commands:

```
% gfortran -c abserr.f
% gfortran -c smaceps.f
```

to create the two files **abserr.o** and **smaceps.o**. The next step will create the library. Use

```
% ar rcv mylib.a *.o
```

Note that the asterisk is a wildcard character. The output will list the files as they are added. That is

```
a - abserr.o
a - smaceps.o
```

If you look at what is in the folder (using ls) the files will make sense. One last thing needs to be done in the library. The command is

```
ranlib mylib.a
```

which creates a random access library. This basically allows programs to link to the library more efficiently.

Testing the Library:

To test the work, we need to compile and link the main program file without the subroutines contained. This can be done using the following command

```
% gfortran main.f mylib.a
```

The result is another version of the **a.exe** file than can be executed as above.

What is Next?

At this point, you can add more object models to your library with the idea of creating a whole suite of codes.