```
character(len=20):: file
real(kind=8)     :: a,b
integer          :: istat,
                    c_read
...
istat = c_read(a,file)
write(*,*) '"a" in file'&
          //file//' is:',a
...
```

```
int c_read(double *a, char *file)
{
  char *name, *space;
  int len;
  FILE *finp;

  space=strchr(file, ' ');
  if(space==NULL) return 1;
  len = 1 + (int)(space-file);
  name=(char *)malloc(1, len);
  strncpy(name, file, len-2);
  name[len-1] = '\0';
  finp=fopen(name, "rb");
  free(name);
  if(finp==NULL) return 2;
  fread(&a,sizeof(double),1, finp);
  fclose(finp);
  return 0;
}
```


LECTURE 7: GOING MULTI-LINGUAL

WHY WOULD ANYONE WANT TO MIX LANGUAGES?

- While doing the home work you have perhaps noticed that the language you are using suites some problems better than the others
- Specific implementation of certain algorithms may only exist in certain languages
- You need to link your code with a system library (not one of those that are included with the compiler)

MAIN ISSUES WHEN MIXING LANGUAGES

- Who is the boss? (What language to use for the main program)
- Parameter passing and returning
- Names of the entry points
- Compiling
- Linking
- Libraries

WHAT LANGUAGE TO USE FOR THE MAIN PROGRAM?

- Mixing languages is worth doing when some functionality is missing.
- For your project use the language that is most suitable (or you are most comfortable with) and complement it with subroutine(s) in other language(s) is necessary
- In addition to language-specific libraries the code needs a special module that loads to code into memory and initializes it. This module is language-specific and thus linking must be done by the compiler corresponding to the main program

EXAMPLES:

1. Extract data structures from multiple binary files, extract values for structure fields with identical names, treat missing fields, create an output structure through an advanced statistical analysis a data set for each field and insert the result in to another binary file
2. Include a visualization option into a hydro-code that can be triggered by a flag.

PARAMETER PASSING: RESTRICTIONS

- FORTRAN is all about speed so all parameters that go into a FORTRAN subroutine must be represented by identical pointer.
- FORTRAN 77 functions can only return a single value.
- FORTRAN 90 functions can return anything although scalars are still returned by value while everything else – by pointer.
- This functionality is even more restrictive when FORTRAN calls a program in another language. E.g. to convert FORTRAN parameters into IDL or PYTHON objects it is often easier to write a C-wrapper.
- Special care must be taken when dealing with strings: Null-terminated, length-prefixed, structures {int len; char *s;} etc.

EXAMPLES:

- IDL calls C:

IDL

```
ErrStr = Call_External(prefix+'sme_synth.so', $
     Entry.Opacity, nDep, OpBlue, OpRed, /S_Value)
If ErrStr ne '' Then Begin
  print,'Opacity (call_external): '+ErrStr
  return
EndIf
```

C

```
extern "C" char const *SME_DLL Opacity(int n, void *arg[]);
```

- C calls FORTRAN

C

```
SPLIST=(char *)calloc(N_SPLIST, 8);
i=eqlist_(ABUND, ELEMEN+1, my_species, ION, SPINDEX,
          SPLIST, NLINES, 0, N_SPLIST, nelem, 8, 8);
```

Ftn

```
     integer function eqlist(abund,elemen,spname,ion,spindx,
&                            splist,nlines,nlist,SPLDIM,ELESIZ)
```

NAMES OF THE ENTRY POINTS

- Compilers try to extract the information about expected parameters. This is do through “mangling” of the entry point names.
- C and FORTRAN 77 do not do it but even here the names are different: an underscore can appended to the name in object file.
- Linux/Unix platforms provide a tool to find out the mangled names: nm *object or library*

```
...     T __Z6RKINTSPdiddS_S_S_IRlS_s
        U _eqlist_
```

```
int RKINTS(double *MUs, int NMU, double EPS1, double EPS2,
           double *FCBLUE, double *FCRED, double *TABLE,
           long NWSIZE, long &NWL, double *WL,
           short long_continuum);
```

PARAMETER PASSING

- Most of the programming languages pass parameters either by value or/and by address (pointers)
- A few languages incorporate options for creating/destroying/modifying variable types anywhere in the code. To achieve this variables are replaced by objects where in addition to the value creation and destruction methods are also described.
- The return value has similar options with more or less restrictions on what can be passed

COMPILING

- When mixing languages the compilation must be performed separately from the linking process
- For compilable languages the merger is done at linking:

```
cc –c –underscore c_sub.c
f77 –c main.f
f77 –o fort_and_c main.o c_sub.o –lc -lm
```

- Calling compiled subroutines from interpreters requires building relocatable libraries. This also works for compilable languages.

LINKING

- Linking is the most difficult step. This is the point where all calls must be associated with the corresponding entry points.
- Compiler has little possibility to adjust the names so you may need to use it yourself (`nm`).
- Linking: you need to load the right libraries.
- Using the same script that compiles the main program simplifies things but the libraries for other languages are still needed.
- You can still use `ld` but then all the relevant libraries and the corresponding paths must be specified.
- Keep in mind that addressing mode must be the same across all the subroutines. You cannot mix 64-bit with 32-bit.

LIBRARIES

- Compilers come with their own libraries.
- All system calls are grouped into libraries that come with the OS.
- Specific tools (e.g. math) libraries are extra
- You can make your own libraries as well. This is another way to mix languages!
- Compile individual files with a special flag making them relocatable:

```
f90 –c –fPIC sub_fort.f
g++ –c –fPIC sub_cpp.cpp
ld –shared –o libmy.so sub_fort.o sub_cpp.o \
   /opt/fortran90/lib/libF90.a -lg2c -lstdc++ \
   -lcl –lm -L/usr/lib/LAPACK -llapack -lblas
```

CONCLUSIONS

1. Consider mixing languages when you feel like programming a part in another language or when you have an existing subroutine(s) in another language.
2. Carefully select the language for the main program.
3. Read compiler documentation to understand the parameter passing conventions.
4. Check that entry point naming convention.
5. Compiler different parts first
6. When linking use the same compiler that you used for the main program add libraries for additional language(s)

HOME WORK: LINK FORTRAN AND C

FORTRAN

```
intrinsic none
integer bar,i,pos,length
length=67
do i=1,length
  pos=bar(0)
enddo
pos=bar(1)
length=167
do i=1,length
  pos=bar(0)
enddo
pos=bar(-1)
end
```

C

```
#include <stdio.h>

int bar_(int *reset)
{
  static int count=0;
  if(*reset)
  {
    printf("\n"); count=-1;
  }
  else printf("=");
  if(++count == 80)
  {
    printf("\n"); count=0;
  }
  return count;
}
```

NEXT LECTURE: PARALLEL COMPUTING

The last lecture is on Tuesday October 20th at 10:15.

Last chance to report Home Work Part II is on Wednesday October 21st at 14:15.