

NJOY-2001

Robert E. MacFarlane
Nuclear Theory – Group T-16
Los Alamos National Laboratory

NJOY Users' Group -- Aix-en-Provence -- May 2001

Outline

- NJOY 2001 design and development
 - NJOY 99 upgraded to Fortran 90/95
- Status of NJOY 99
 - 99.50 works for a large number of cases
- Software Quality Engineering
 - Formalized processes may be important

NJOY 2001 Design

- Use NJOY 99 as the base
 - Don't make many method changes to allow for easier testing of the results with the new code.
- Use a modern subset of F90/95 features to improve maintainability and transportability.
 - Modules
 - Allocatable arrays
 - No implicits
 - Modernized coding style

NJOY 2001 Development

- Main development work used Solaris f90.
- Cross checked using Absoft and Portland Group compilers on linux and Absoft and Lahey compilers on DOS.
- Additional compilation and code analysis performed on SGI using FLINT and Purify.
- Results compared to NJOY 99 using diff.

Using modules in NJOY 2001

- To embody NJOY calculational modules like RECONR or BROADR.
- To break up big modules (ACER) into parts.
- To encapsulate utility routines into sensible collections.
- To provide common sets of constants.
- To eliminate COMMON blocks.
- To break the compilation up into reasonably sized parts.

A Computational Module

```
module reconm
  ! Module to provide reconr for NJOY2000
  use locale
  implicit none
  private
  public reconr

  ! global variables for reconr

  integer::nin,nout
  real(kr)::zain,awin
  ...
  real(kr),dimension(:),allocatable::dict
  integer,dimension(:),allocatable::mfs,mts,ncs
  real(kr),dimension(:),allocatable::sunr
```

contains

```
subroutine reconr
  !-----
  !
  ! Reconstruct pointwise cross sections
  !
```

Incapsulating Utility Routines

```
module endf
  ! Routines for handling ENDF formats.
  use locale
  implicit none
  private
  public contio,listio,tab1io,tab2io,moreio
  public tpidio,hdatio,dictio
  public tosend,tofend,tomend,totend
  public asend,afend,amend,atend
  public skip6,findf
  public terp1,terpa,intega
  ...
  integer,public::npage,iverf
  real(kr),public::c1h,c2h
  integer,public::l1h,l2h,n1h,n2h,math,mfh,mth,nsh,nsp,nsc
  real(kr),public::thr6
  integer::nah
  real(kr)::ah(6)
  ...
contains

  subroutine contio(nin,nout,nscr,a,nb,nw)
```

Common Set of Constants

```
module physics
  ! Provides pi and physics constants taken from CODATA
  ! as given on the NIST site; namely, bk (Boltzmann's constant),
  ! amassn (the neutron mass in amu), amu (the amu value itself),
  ! hbar (Planck's constant), ev (the conversion to eV), and
  ! clight (the speed of light).
  use locale ! provides kr
  implicit none
  real(kr),parameter,public::pi=3.14159265358979e0_kr
  real(kr),parameter,public::bk=8.617385e-5_kr
  real(kr),parameter,public::amassn=1.008664904e0_kr
  real(kr),parameter,public::amu=1.6605402e-24_kr
  real(kr),parameter,public::hbar=1.05457266e-27_kr
  real(kr),parameter,public::ev=1.60217733e-12_kr
  real(kr),parameter,public::clight=2.99792458e10_kr
end module physics
```

Breaking Up Compilation

```
# make file for NJOY2000 using Sun F90 compiler
F90 = f90
F90FLAGS = -g
njoy: locale.o version.o mainio.o physics.o util.o endf.o mathm.o \
    reconr.o broadr.o unresr.o heatr.o thermr.o \
    groupr.o gaminr.o errorr.o covr.o \
    moder.o dtfr.o ccccr.o matxsr.o resxsr.o \
    acer.o acefc.o aceth.o acepa.o acepn.o acedo.o acecm.o \
    powr.o wimsr.o plotr.o viewr.o graph.o mixr.o purr.o leapr.o gaspr.o
    ${F90} ${F90FLAGS} locale.o version.o util.o endf.o mathm.o \
    reconr.o broadr.o unresr.o heatr.o thermr.o \
    groupr.o gaminr.o errorr.o covr.o \
    moder.o dtfr.o ccccr.o matxsr.o resxsr.o \
    acer.o acefc.o aceth.o acepa.o acepn.o acedo.o acecm.o \
    powr.o wimsr.o plotr.o viewr.o graph.o mixr.o purr.o leapr.o ...
    njoy.f90 -o xnjoy
locale.o: locale.f90
    ${F90} ${F90FLAGS} -c locale.f90
version.o: version.f90
    ${F90} ${F90FLAGS} -c version.f90
mainio.o: mainio.f90
    ${F90} ${F90FLAGS} -c mainio.f90
```

Allocatable Arrays

- Replaces old home-brew dynamic storage allocation scheme.
- More transparent without pointers.
- Allows for variables to be typed consistently for procedure parameters, thus allowing for better automatic checking for improper uses of the parameters.
- Eases some storage restrictions and introduces others.

Allocating and Deallocating

```
subroutine groupr
!-----
!  
! compute self-shielded group-averaged cross sections
!  
...
real(kr),dimension(:, :, :),allocatable::ans
real(kr),dimension(:, :),allocatable::ff
...
call init(ng2g,nlg,nz)
allocate(ans(nlg,nz,ng2g))
ng2=ng2g
nl=nlg
allocate(ff(nlg,ng2g))
...
call panel(elo,enext,ans,ff,nl,nz,ng2,ig2lo,nlg,ng2g)
...
deallocate(ans)
deallocate(ff)
```

Using in Subroutines

```
subroutine panel(elo,ehi,ans,ff,nl,nz,ng,iglo,nlg,ng2)
...
use util ! provides error,mess
! externals
integer::nl,nz,ng,iglo,nlg,ng2g
real(kr)::elo,ehi,ans(nlg,nz,ng2g),ff(nlg,ng2g)
! internals
...
do iz=1,nz
  do il=1,nl
    ...
    do ig=1,ng1
      if (ff(il,ig).ne.zero) then
        igt=ig1+ig-iglo+1
        ...
        if (igt.gt.1) then
          ans(il,iz,igt)=ans(il,iz,igt)+rr*ff(il,ig)
          if (igt.gt.ng) ng=igt
        endif
      endif
    enddo
  enddo
enddo
```

Take Care With Constants

- Implicit type conversions
 - Bad: $x=0.5*(a+b)$
 - Good: $x=(a+b)/2$
- Comparisons
 - Bad: `if (x.eq.0.) then`
 - Good: `if (x.eq.zero) then`
 - Where `real(kr),parameter::zero=0`

More Care With Constants

- Parameters of procedures
 - o Bad: call `getsig(0.,thresh,idis,sig,nl,nz)`
 - o Good: `e=0`
 - o Then: call `getsig(e,thresh,idis,sig,nl,nz)`

No More DATA Statements

```
real(kr),dimension(10),parameter::qp10=(/ &
-1.e0_kr,-.9195339082e0_kr,-.7387738651e0_kr,&
-.4779249498e0_kr,-.1652789577e0_kr,.1652789577e0_kr,&
.4779249498e0_kr,.7387738651e0_kr,.9195339082e0_kr,1.e0_kr/)
real(kr),dimension(10),parameter::qw10=(/ &
.0222222222e0_kr,.1333059908e0_kr,.2248893420e0_kr,&
.2920426836e0_kr,.3275397612e0_kr,.3275397612e0_kr,&
.2920426836e0_kr,.2248893420e0_kr,.1333059908e0_kr,&
.0222222222e0_kr/)
real(kr),parameter::rndoff=1.000002e0_kr
real(kr),parameter::delta=0.999995e0_kr
real(kr),parameter::emax=1.e10_kr
real(kr),parameter::small=1.e-10_kr
real(kr),parameter::zero=0.e0_kr
real(kr),parameter::smin=1.e-9_kr
```

Remaining Style Problems

- Should be using the INTENT qualifier for procedure parameters.
- Some procedures are still too long and some blocks have too many levels of indentation.
- There are still too many statement numbers.
- The free format coding lines cause some problems with our traditional use of UPD for revision control.

Development of NJOY99

- NJOY99 was developed during the summer of 1999 based on NJOY97.
- The main goal was to incorporate cleanly a large number of changes needed for high-energy data (150 MeV), outgoing and incident charged particles, and photonuclear data for the upcoming MCNP4C.
- A large set of files based on ENDF/B-VI.5 was released for internal testing in October 1999.

Release of NJOY99

- The official release of NJOY99 took place early in January of 2000 after some last minute changes in the photonuclear part.
- Unfortunately, we didn't get any local testing done before MCNP4C became available outside and we began to get reports of problems.
- We worked through the summer replicating problems reported by others, finding others ourselves, and fixing them up one by one.

Testing of NJOY99

- Using NJOY 99.20, we did some extensive testing in September 2000 of library production for MCNP4C and MCNP4B with internal consistency checks, test runs in MCNP, and plots:
 - ENDF/B-VI.5 complete, with multiple temperatures
 - A large subset of JENDL-3.2
 - Some JEF-2.2 and preliminary JEFF materials.
- Performed a selection of benchmark tests with good results.

Selected Testing Results

Assembly	ENDF/B- VI.5	JENDL- 3.2	ENDF60
Godiva	.9966(09)	1.0006(09)	.9963(12)
Flattop- 25	1.0011(09)	.9980(10)	1.0027(13)
Jezebel	.9969(09)	.9980(08)	.9971(10)
Flattop- Pu	1.0035(10)	.9933(11)	1.0040(14)
Jezebel- 23	.9936(08)	1.0134(09)	.9931(11)
Flattop- 23	1.0026(10)	1.0061(11)	1.0034(13)
Bigten (1D)	1.0144(08)	.9939(08)	1.0073(10)
ORNL- 10	.9982(05)		.9972(05)
HEU- SOL- THERM- 032	.9974(05)		.9951(05)
ORNL- 1	.9978(08)		
HEU- SOL- THERM- 013- 1	.9976(08)		.9960(07)
L10	.9980(12)		
L7	.9988(12)		1.0034(13)
HEU- SOL- THERM- 001- 5	.9996(09)		
HEU- SOL- THERM- 001- 8	.9979(10)		
HEU- SOL- THERM- 010- 1	1.0004(12)		
HEU- SOL- THERM- 009- 1	.9992(11)		1.0054(11)
PU- SOL- THERM- 011- 2	1.0007(11)		.9998(11)
TRX- 1 (metal)	.9916(08)		.9878(08)
BAPL- 1 (oxide)	.9961(08)		.9928(08)

Recent NJOY99 Work

- We are now up to NJOY 99.50 released (and 99.52 internally) including some improvements, some fixes for problems with more modest impact than the first 20, and some more important fixes that affect isolated materials.
- We have recently done extensive testing of the IAEA compilation of photonuclear evaluations, leading to some improvements to NJOY and to a number of proposals for formatting changes in the evaluations.

Software Quality Assurance

- Software Quality Assurance (SQA), or Software Quality Engineering (SQE), is receiving increased emphasis for some NJOY uses. Modern ideas are based on successful safety management programs and require:
 - a *risk assessment* for the software project, and
 - the use of *formal processes* commensurate with the risks assigned to the project.

NJOY Risk Assessment

- Software that could affect injury to the public or is absolutely necessary for the success of a mission is clearly at a *high* risk level.
- Software that is exploratory or personal would be at a *low* risk level.
- NJOY is somewhere in between on the low end of the scale. Its results can affect codes at a higher risk level, but NJOY is insulated by the existence of additional levels of testing.

NJOY Level of SQA Formality

- The level of risk means that NJOY can use a *basic level* of formality, and the fact that it is an existing code that it is not managed by a team of programmers simplifies things further.
- One system defines the goal of the *basic level* to be "stabilize the software, prevent its degradation with time, allow it to survive losses of staff or capability, and to provide a basis to advance to higher levels of formality if needed."

Elements of NJOY SQA

- We are currently concentrating on:
 - Version control of software components
 - Issue tracking and requirements management
 - Automated build process
 - Regression testing and user-level testing
 - Application release process
 - Code documentation
 - Documentation of processes and coding practices

Version Control

- NJOY has always had a revision control process.
- UPD has been used for a number of years, but it may have problems with F90 free form code and doesn't work well for controlling documentation, make files, or testing scripts.
- We could work to extend UPD, or we could change to a more standard system like *razor*.
- The latter would make putting recent updates on the web difficult, favoring fewer more formal releases.

Issue Tracking

- We chose a web-based issue tracking system *<http://t2.lanl.gov/codes/njoy99/Issues.html>* for use with NJOY99 because of the open and international nature of NJOY.
- Issues are submitted to ryxm@lanl.gov or through the njoy@nea.fr list.
- Everybody can see the issues, help assure that issues are not forgotten, and see how issues connect to the revision control process.

More on Issues

- Issues should be coordinated with a formal *Code Requirements Document* and a process for updating the requirements. It is known that "requirements creep" is a major risk factor in software development.
- Systems like *razor* include issue tracking with automatic coupling to revision control.

Testing

- *Regression Testing* is the activity of regularly building the code and executing a series of tests to assure that changes don't degrade the performance of the code on the supported platforms.
- It is best if it is fully automated (so far difficult for NJOY).
- It is best if the test suite provides adequate coverage of the various parts of the code (the current NJOY set is deficient here).

More on Testing

- *User Acceptance Testing* is the activity of determining if the code satisfies the needs of the users. It is intended to build user confidence.
- For NJOY, this consists of running very large sets of real processing jobs to find as many special cases as possible.
- NJOY results are fed into selected application codes to check that the interfaces perform as expected and that the answers are as expected by the users.