```
if( m .eq. 0 ) go to 40
do 30 i = 1,m
   dtemp = dx(i)
   dx(i) = dy(i)
   dy(i) = dtemp
30 continue
   if( n .lt. 3 ) return
40 mp1 = m + 1
   do 50 i = mp1,n,3
   dtemp = dx(i)
   dx(i) = dy(i)
   dy(i) = dtemp
   dtemp = dx(i + 1)
   dx(i + 1) = dy(i + 1)
   dy(i + 1) = dtemp
   dtemp = dx(i + 2)
   dx(i + 2) = dy(i + 2)
   dy(i + 2) = dtemp
50 continue
```

```
short l;
int j;
double *a;

l=*(short *)arg[0]; /* Array length */
a=(double *)arg[1]; /* Array */
for(j=0; j<min(NRHOX, l); j++)
{
  a[j]=XNA_eos[j];
}
return ((char *)NULL);
```

```
if il eq linbeg then begin      ;true: first segment
   jint = [nw - 1]              ;end of intensity segment
   wint = w_ex(0:nw-1)
   sintl = (Transpose(sint_ex))(0:nw-1,*) ;trim to valid length
   sint = sintl
endif else begin                ;else: later segment
   jint = [jint, max(jint) + nw];trim and append to list
   wint = [wint, w_ex(0:nw-1) ]
   sintl = (Transpose(sint_ex))(0:nw-1,*)
   sint = [sint, sintl]
endelse
sme.cintb(*,il) = cintb         ;save blue continuum
sme.cintr(*,il) = cintr         ;save red continuum
```

Scientific Programming

LECTURE 2: ELEMENTS OF PROGRAMMING LANGUAGES

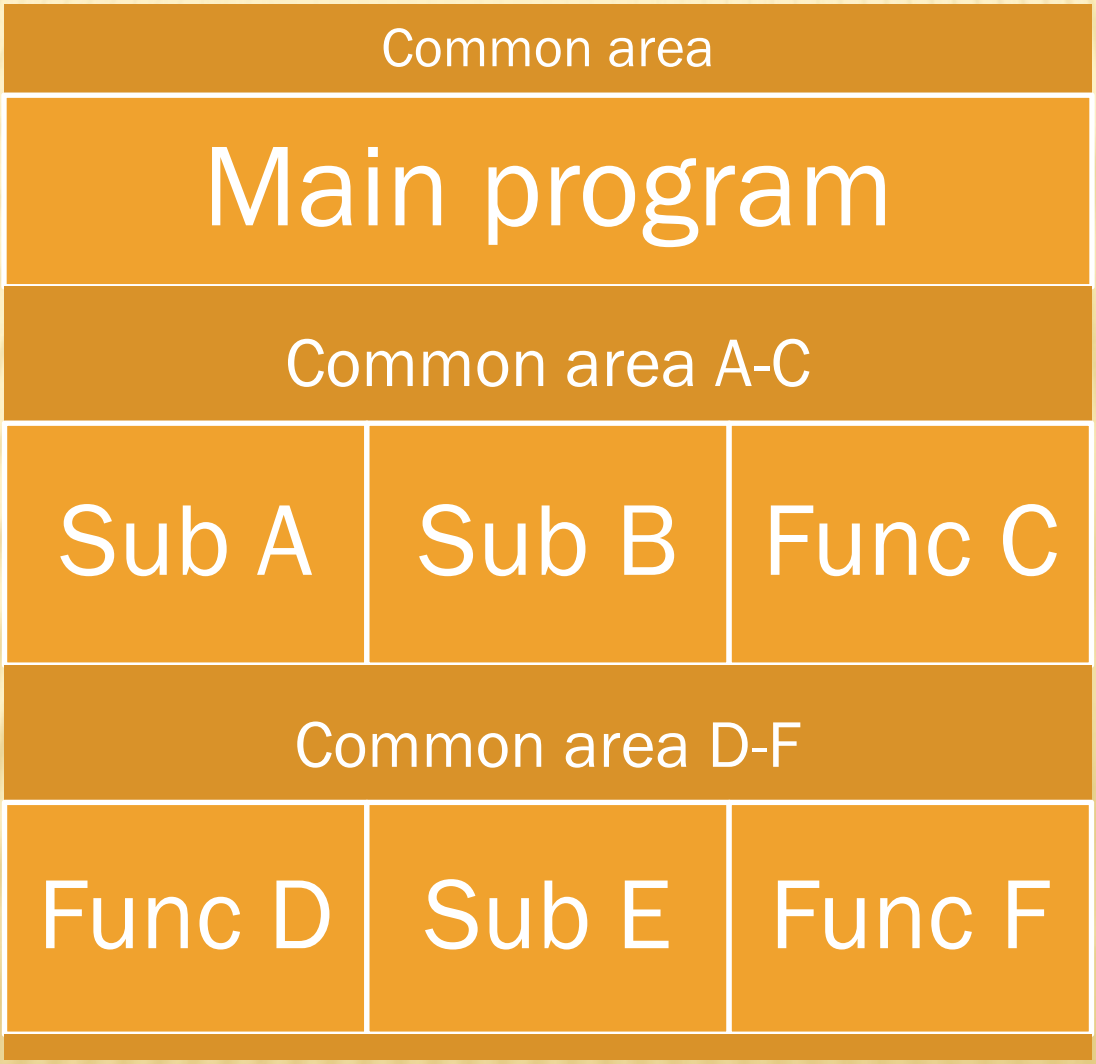
PROGRAMMING STYLE IS IMPORTANT

- Development
- Readability
- Debugging
- Optimization
- Recycling
- Evolution

PARTS OF A PROGRAM

- Main program
  What is special about the main program?
- Subroutines and Functions
  What is the difference between the two?
- Common area for some or all parts of the program
  What is this a common area?

TYPICAL LAYOUT



ELEMENTS OF A SUBROUTINE/FUNCTION

Constants

- Integer: `2, -3, -32768, 2147483647`
- Floating point: `1.0, -3.333e-37, 1.d305`
- String: `'Ab_ Cd:/e'` (Null termination or length declaration)

Optional

- Logical: `.true., .false.`
- Byte: `-1B, 255B`
- Double precision, complex etc.

ELEMENTS OF A SUBROUTINE/FUNCTION

Variable type are the same as types of constants

Variable dimensions:

- Scalars: `double a, b;`
- Arrays: `char s[10];`

In Python arrays are a derived type

Optional

- Pointers `int *i;`
- Structures
- Derived types

ELEMENTS OF A SUBROUTINE/FUNCTION

Declarations

Variables are declared at the top. Some languages allow implicit rules for scalars. All arrays and other derived types must be declared explicitly.

If your code is a long-term project use explicit declarations only!!!

By default, all local variables in a subroutine are allocated on stack. You can force heap allocation (static, save etc.)

ALLOCATING MEMORY

Several languages allow explicit memory allocation

- C, C++, of course have pointers
- FORTRAN90 has allocatable variables
- IDL and Python can create/modify variable type anytime anywhere in the code

  This operation is very useful and safe as long as the allocation is done on stack. Cleaning will happen automatically on exit from a subroutine but allocating heap is dangerous!!!
- Multi-dimensional arrays can be 1D or truly nD

PARAMETERS OF SUBROUTINES/FUNCTIONS

Parameters can be passed to subroutines either via common area of through formal parameters. **80% of bugs are introduced at this stage**!!!

Parameters can be passed “by address” (e.g. FORTRAN) or “by values”. Some languages (e.g. C) allow both.

“By value” parameters can be used inside subroutine but all modifications are lost upon return (safe)

“By address” parameters allow modifications of the values but …

PARAMETERS OF SUBROUTINES/FUNCTIONS

… you can do things.

Examples:

1. Passing a constant by address
2. Passing a different variable type than expected
3. Passing arrays is a separate story
   - In FORTRAN, IDL etc. first index of a multi-dimensional array runs fastest
   - In C it is the other way around!
   - In a subroutine you can use dimensions which are different from the ones in the caller!

PARAMETERS OF SUBROUTINES/FUNCTIONS

Good practice:

1. Protect parameters that should not be modified inside a subroutine (FORTRAN90/95: “input”, C/C++: pass “by value”, etc. or make a local copy)
2. Double check type consistency between the caller and the subroutine
3. When passing an array always pass its declared dimensions

DERIVED DATA TYPES

- Many object-oriented languages allow complex data types: structures, unions. These are useful when the list of parameters becomes prohibitively long – just put then in a structure and pass it in.
- Many object-oriented languages allow declaration of classes: data types and operation methods. One can even overload the conventional operations (+, -, *, /). E.g. complex numbers. Classes can be based on other classes inheriting properties, common areas and methods. Remember that each class has a stack associated with it and thus context switching takes its toll.
- Structures can be very useful when dealing with sequences of heterogeneous data objects

OPERATIONS AND ELEMENTARY FUNCTIONS

- In scientific computing the useful lines look like this: `a=b*c+d`
- In practice the CPU can only do arithmetics on identical data types. If `b`, `c` and `d` are of different type they are converted to highest precision type first
- If `a` is not the same type as the RHS another conversion will be required
- Precision:
  - 4-byte floating point operations keep 7-8 decimal places
  - 8-byte floating point operations keep 13-14 decimals
  - Elementary function would typically drop another digit.

EXECUTION FLOW CONTROL

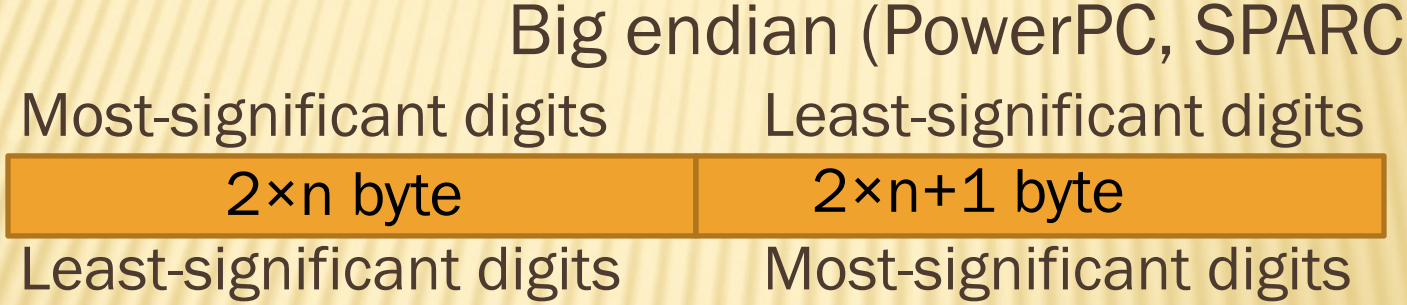
- go to
  - if ... then ... else — better for cache memory operation
  - case ... of — many if's
- repeat loop
  - do loop — better for branching prediction

INPUT AND OUTPUT

- All I/O comes in formatted and unformatted flavors
- Operations sequence:
  - Open file
  - Perform input/output
  - Close file
- Some files are open by default and cannot be closed
- The simplest is text I/O: string is read and parsed according to the types of variables (unformatted) or according to the format pattern (formatted)
- Binary I/O: variables/constants are written to a file or variables are read in the way the are stored in computer memory

INPUT AND OUTPUT: THE TRICKY PARTS

- Byte order: most of the RISC machines (Sun SPARC, PowerPC, Cell) have different byte order than e.g. Intel-based computers. 2-byte integer:



- Byte order can make file/code combination non-portable

INPUT AND OUTPUT: THE TRICKY PARTS

- Binary files (unformatted) may have an additional hidden parts. E.g. binary files created by FORTRAN consists of records bracketed by two 4-byte integers. This values are identical and specify record length in bytes. This is a left-over from the times of magnetic tapes when reading was linear and slow: having record length on both sides allowed reading forward and backward. Record length is also affected by the byte order.
- FORTRAN also has the so-called “direct access” which is different in that all the records have the same length. This way one can compute the start position of any given record.

OPENING AND POSITIONING A FILE

- In all languages file opening has quite a bit of flexibility. It creates a structure that has all the information for accessing a file. This structure is referenced with a pointer (C, C++) or a number (FORTRAN)
- When opening a file one can often specify an intended action (read, write, update) and position (start, end)
- ***Always open files that you do not intend to write into as "read only"!***

NEXT LECTURE: COMPILING AND LINKING

You will also get the home work