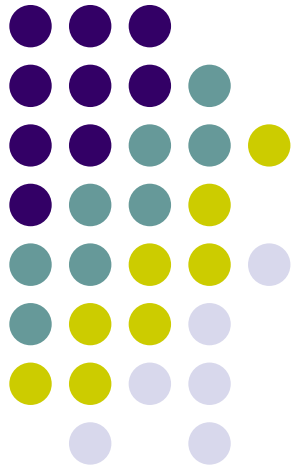
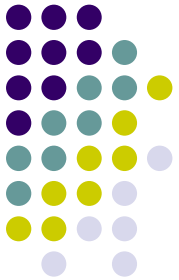
Code Structure

Xiaoqiang Wang

Florida State University

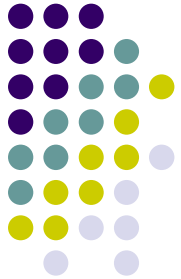


Why?



- Maintainability
- Ease of understanding
- Easier to establish collaborative efforts
- people working on the same code
- Ease of debugging
- Ease of upgrading

Requirements



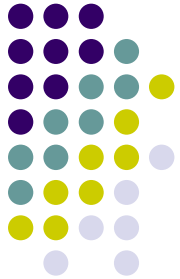
- Choice of variable names
- Choice of method names
- Code structure
  - indentation
  - spacing
  - block identification
- Proper commenting
- Proper documentation

Articles

- Useful strategies for commenting code
  - http://particletree.com/features/successful-strategies-for-commenting-code/

- Programming: C Structure and Style
  - http://en.wikibooks.org/wiki/C_Programming/Structure_and_style

Good Habits to Acquire



- For each project, keep a README file with information on requirements (libraries, include files, jar files, etc.) compilation, linking, execution
- INSTALL: how to install the software (if not obvious) and run it. Include any special requirements.
- Keep a TODO file which lists what remains to be done
- Keep a NOTES file that lists anything related to the project underway, list any bugs as they are found, and what is done to fix them.

Indentation (loops)

```
for (int j=0; j < 10; j++) {
    float c = 3 + c;
    d = c * sin(d);
}
```

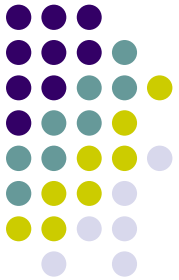
is better than

```
for (int j=0; j < 10; j++) {
float c = 3 + c;
d = c * sin(d);
}
```

Spacing

```
a = 3 + v;
c = a * 2;

for (int j=0; j < 3; j++) {
    for (k=5; k >= -2; k--) {     c *= 2;
    }
}
```

No Spacing or Indentation



```
a = 3 + v;
c = a * 2;
for (j=0; j < 3; j++) {
for (k=5; k >= -2; k--) {
c *= 2;
}
}
```

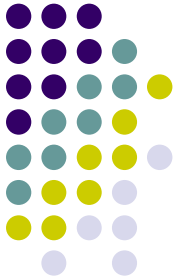
Alignment

```
for (int j=0; j < 10; j+=5) {
        a /= 3.;
   c += sin(3.*a);
        }
```

... versus

```
for (int j=0; j < 10; j+=5) {
   a /= 3.;
   c += sin(3.*a);
}
```

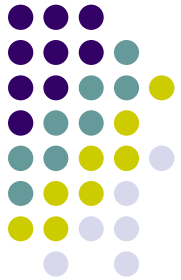
Delineation of Blocks



When a file has multiple subroutines, one should separate them by comment lines so that one see them at a glance.

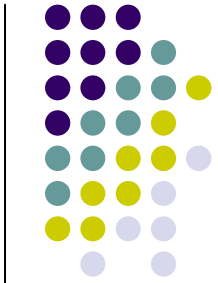
```
!---------------
subroutine sub1(...)....
end subroutine sub1
!---------------
subroutine sub2(…)..
end subroutine sub2
!---------------
```

Good Code



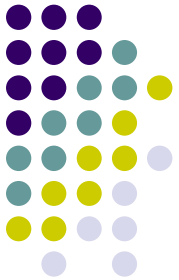
- Look at program from afar:
  - *can structure (not the function) be determined (even if type is too small to read?)*
- Model with very small font Should be able to distinguish
  - Comments
  - Subroutine/Method blocks
  - Loop and conditional blocks

Editors



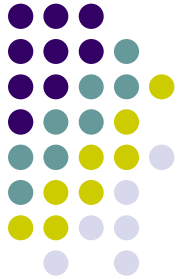
- Some editors can displays c++, Java, Fortran 90 in color (Vim Vi Improved)
  - helps distinguish
    - variables
    - methods
    - key words
  - Editing is easier
  - However:
    - Different editors use color differently
    - E.g.: keywords might be different colors

Poor C++



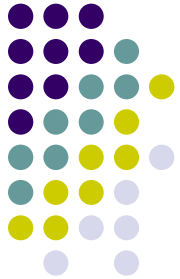
- Multiple statements per line:
  - float a=3; for (int j=2; j < 5; j++) { a=a+3;float b=3+a; }
  - poor variables
  - no structure
  - no comments
- Example of obfuscated code ...

Extreme case: Obfuscated code



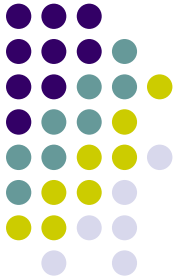
- #include <stdio.h> main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_, main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13? main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\ ;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \ ){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \ ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \ }'+}##(!!/") :t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1) :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
- Will print the verses of the **12 days of christmas.** To compile: gcc verse.c

Reasons to Obfuscate



- Code security
- Code efficiency (smaller memory)
- Avoid reverse engineering (to figure out source code from the object code)
- Sometimes reduces code size
  - classes and other code structure to simplify the compiler's job often make the source code and assembly code larger

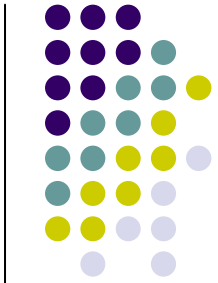
Why not to Obfuscate



- Debugging is a nightmare
  - Structure of the code is lost
  - Significance of variables is lost
  - Significance of code blocks is lost

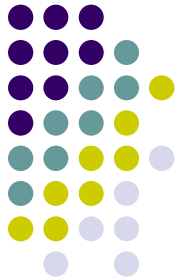
- If one needs to obfuscate,
  - Create a well-written program
  - Use an existing tool to obfuscate

Better C++



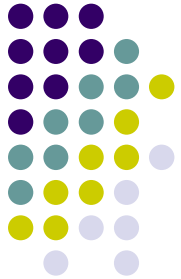
- Using specialized tools, the original intent of the programmer can be deduced
- However, it is best to write good code right away
- ***In this course, we will write well-structured code, easy to read, understand and maintain.***

Automatic Structurers



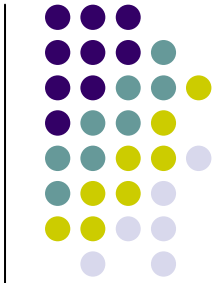
- Structurers are also called Beautifiers
- Take a code with poor structure and transform it into a code with
  - Indents
  - Spacing
    - proper alignments
- Structurers cannot change variables names since context is difficult to determine

Directory Structure



- Usually, a program is composed of a main file and many auxiliary files
- How to place these files in directories for better comprehension
- Illustrations
  - Java
    - http://java.sun.com/blueprints/code/projectconventions.html (complex example: read for homework)

Commenting Code

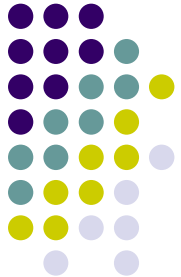


- Choose variables names that are descriptive:
  - function computeFourierTransform()
- Inline commenting
- Comment each function:
  - function summary
  - parameter description
  - return value

Inline Commenting

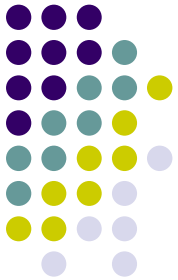
```
// Summary: Compute the 1D Fourier transform of
// the input array ain() of length n, and output the
// the result in array aout() of length n

// Parameters:
// ain() : 1D input array
// aout() : 1D output array
// n :

function computeFourierTransform(float* ain, int n,
float* aout) {
    ...
}
```

Class/Page Commenting



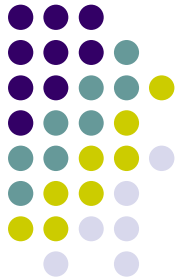
- Good programming practice often requires that there be a single class, struct, module, or more generally, concept, per file.
- This file should have global documentation with information such as author, title of class, description of class, when it was created and when it was modified.
- Tools exist to create these headers from basic information.
  - svn, cvs

Good Habitss



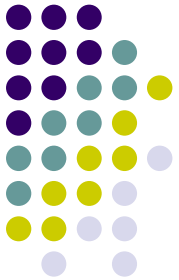
- Add a blank line before comments
- Do not give trivial comments
  - for (i=0; i < 10; i++) { // loop to 10
- Align comments
  - var MAX_USERS = 2 //all players
  - var MAX_POINTS = 100000 //needed to win game
- Comment while coding!!!
  - This is extremely important since one almost never returns to finished code to comment it!!

Choice of Variable Names



- Good variable names avoids excessive commenting.
- Same for function names and class names

Function Arguments



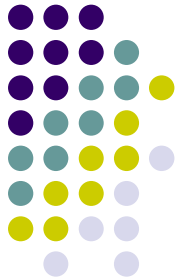
function getSpeed(stoneDistance, stoneTime)

indicates a function the gets the speed of a stone given its displacement and time of displacement.

I wish to compute the speed of a car, so I use:
getSpeed(carDistance, carTime)

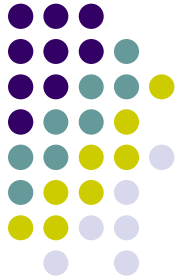
This indicates a poor choice of variable names for the function. Better would be:
function getSpeed(distance, time)

Some Conventions



- Constants are all capitalized
  - #define PI 3.1415
  - final int SIZE = 3
- method names start with lower case, but later syllables are capitalized:
  - computeFourierTransform (…..)
- class and module names start with upper case:
  - FourierTransform
- variable names:
  - fourier_transform
  - fourierTransform (popular alternative)

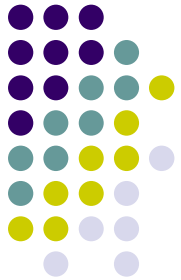
Conventions



- The above are only conventions
- They are harder to enforce in Fortran since variables with different capitalizations are the same
- These conventions make it easy to identify methods, classes, variables, etc.
- Hungarian notation (used by Microsoft):
  - int speed_i // integer
  - int* speed_ip // pointer to integer
  - int& speed_ir // pointer to reference

  Not used by most people, controversial

How to Name Variables



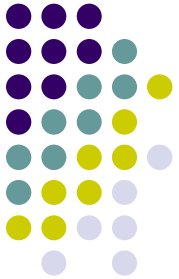
- Fortran does not distinguish between lower/upper case
  - integer :: icaps, ICAPS

    generates an error

- C++/Java distinguish capitalization
  - int icaps, ICAPS

    is accepted by the compiler

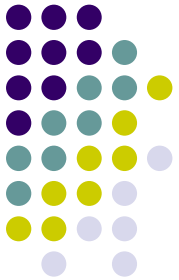
Fortran 77



Spaces are not required, so the following is legal

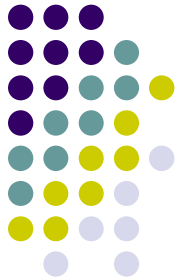
```
....
   integeri,j
   realvar
   do13i=1,10
   v=v+32
13 continue
```

Fortran 77: Quicksort



- Spaces are not necessary
- Worst example
  - Subroutines: s1, s2, ... sn
  - Variables: v1, v2, ..., vn
  - No comments No spaces

Fortran 90/95

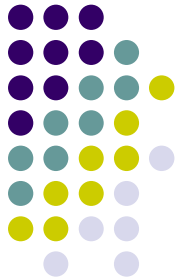


Spaces are necessary between key words (do, if, ...). The previous example becomes

```
integer i,j
real var
do i=1,10
v=v+32
enddo
```

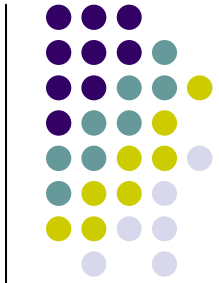
which is a little more readeable.

Better Fortran 90/95



```
integer i,j
real var

do i=1,10
   v=v+32
enddo
....
```

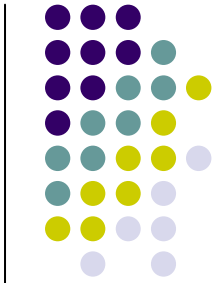
Fortran Examples



- http://people.sc.fsu.edu/~jburkardt/f_src/f_src.html
  - list of fortran source code

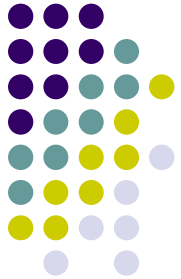
- http://people.sc.fsu.edu/~jburkardt/f_src/qshep3d/qshep3d.f90
  - interpolation code

Documentors



- doxygen: C++/Java
  - generates class structure information in addition to user comments interspersed throughout the code, following certain conventions.
- Spag (commercial Fortran restructurer and documentor)
- f90doc (free Fortran commentor)
  - http://erikdemaine.org/software/f90doc/

Good habits to acquire in this class:



- For each project, keep a README file with information on
  - requirements (libraries, include files, jar files, etc.)
  - compilation, linking, execution
  - how to install the software (if not obvious)
- Keep a TODO file which lists what remains to be done
- Keep a NOTES file that lists anything related to the project underway, list any bugs as they are found, and what is done to fix them.