Programming Style

- You will be expected to write a structured program that is well documented

Clarity in Programming

- write code in logical order, even when order of operations does not matter
- name things informatively
- Indent
- Be consistent about naming conventions for everything, within programs, names of programs, names of directories, etc.

Why Rules?

- to minimize the occurrence of programming errors
- to help the programmer understand his/her own programs
- to help other programmers understand them

Rule: Don't break a rule unless there is a good reason.

Rule: Always document what you do, *as you do it*.

Rule: Documentation should be concise but comprehensive.

- Obvious or self-explanatory steps do not need explanation. For example, the comment at the end of the following line of code is useless:
- `x <- x + 1  # Increment x by 1`
- Short-cuts, tricks, and complicated manoeuvers (must always be carefully documented) (and in many cases should be rewritten anyway)

Rule: Directories should contain README files.

- The README file should briefly describe the purpose of the directory, information of relevance to the entire directory, etc.
- It doesn't need to explain the contents of every file in a directory.
- Usually putting documentation in the individual files is a better strategy.
- Sometimes, however, it is useful to give an overview of what various files do.

Rule: Always put comments in programs

- Perhaps the most important comments are the ones at the top of a program or function that give their purpose, date, and author.

Documentation

- Program should be self docmentating as far as possible.
- State purpose, author, and date at the begininng

Document as you go

Within general code

e.g.

```
whale.preg.glm<-function(data=whaledat){
      # Whale Pregancy GLMI
      # Colien Minto, Jan 28, 2006
   glm(calves ~ chlA + temp, family=poisson()))
}
```

Rule: Always put in comments at the beginning of a program or function to explain what it does, and how that relates to its name.

- The second part of this rule is crucial. A program's name should reflect its purpose. If it proves difficult to explain the connection between a program's name and its purpose then the name is inadequate.

Rule: All the arguments and results for each subroutine must be described.

- Input and output of a subroutine should always be documented.

Rule: Don't embed data in programs.

- Generally it is best to put data in data files rather than embedding data in programs. This makes it easier to modify the data, allows the data to be used by other programs, and makes proper documentation of the data more likely.

Rule: Never use magic numbers.

- Magic numbers are values embedded in programs rather than obtained from data files or passed-in as parameters.
- The use of magic numbers is a bad practice because it makes programs hard to understand (e.g. ``Why is 37 added on here?'') and hard to modify (e.g., ``How do I change the number of years?'').
- In S-PLUS/R, default *arguments* can be used to avoid magic numbers altogether.
- In other languages, careful declaration of parameters and constants can solve the problem.
- Often, the use of magic numbers is a reflection of other shortcomings -- e.g. assuming that there will be a certain number of values in a data file, rather than calculating the number of values automatically (which is what should always be done).

Rule: Isolate calculation routines from figure-making routines.

- Some programs calculate values. Others produce nice figures.
- Try not to mix these two purposes in a single program.
- Often considerable effort is devoted to making figures ``pretty''.
- This has nothing to do with the results being displayed, and the results certainly don't need to be recalculated!
- In S-PLUS/R, *permanent objects* can be used to store the results of a calculation and then *default arguments* can be used to import these results into programs that produce figures.

Rule: Produce final drafts of figures the first time.

- High-quality figures are easily produced in R.
- It is important that data points and axis numbers be large enough for publication.
- In general, axis labels should have the first letter of each word capitalized, with units in parentheses, e.g. Spawner Biomass (thousand tonnes).
- However, some journals require you to use the sentence style (only the first letter of the first word is capitalized).

Rule: Indent for clarity

Always indent loops, and subloops.
Use 4 spaces.

Rule: Indent for clarity

```
TwoSamTTest <- function(y1, y2) {
    # Two Sample T Test,
    # Coilin Minto Jan 20, 2006
    n1 <- length(y1); n2 <- length(y2) # obtain n values
    yb1 <- mean(y1); yb2 <- mean(y2) # obtain means
    s1 <- var(y1); s2 <- var(y2) # obtain variance
    s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2) # standard error
    tst <- (yb1 - yb2)/sqrt(s*(1/n1 + 1/n2)) #t statistic
    tst
}
```

Indent

```
model {
    for (j in 1:J){
        y[j] ~ dnorm (theta[j], tau.y[j])
        theta[j] ~ dnorm (mu, tau.theta)
        tau.y[j] <- pow(sigma.y[j], -2)
     }
    mu ~ dnorm (0.0, 1.0E-6)
    tau.theta <- pow(sigma.theta, -2)
    sigma.theta ~ dunif (0, 1000)
}
```

```
plotyield <- function(n=50, amax=20,aknife=4,p=1,psplot=T)
{
# plots a equilibrium spawner Biomass and yield for for scallops
# RAM, Aug 2, 1998

if(psplot){

        ps()
        par(oma = c(5, 3, 3, 3))
        par(mar = c(4, 4, 5, 2))
        par(las = 1)
}

f<-seq(0,0.9,length=n)
y<-rep(0,n)
j <- 0
for (e2 in f) {
         j <- j + 1
         y[j] <-  yield(fmax=e2,iratio=0.0,aknife=aknife,amax=amax,p=p)
  }

plot(f,y,type="l",lty=1,xlab="",ylab="Yield (grams per recruit)"  )

if(psplot){
```

Rule: Use editors that “speak” the programming language.

- Many editors “understand” a programming language, and will help you find errors.
- Examples:
- Winedit
- Emacs speaks statistics
- Gvim