SE 2A04 Fall 1999

Coding Style

Instructor: W. M. Farmer

Revised: November 15, 1999

General Recommendations

- Be consistent
- Choose clarity before efficiency
- Express the structure of the software’s design in the software’s code
- Follow the conventions of the programming language being used

Keep the Code Simple

- Write procedures that fit on one screen
- Put at most one programming statement on a line
- Keep the following measures low:
  - Loop nesting level
  - Conditional nesting level
  - Number of local variables in a procedure
- Avoid control structures that radically change state
  - Exits, gotos, state jumps, self-modifying code
- Avoid nonstandard language features

Naming Programming Entities

- Naming is an important but difficult task
- One should employ a naming convention
  - Names should be short and descriptive
  - The more global the entity, the more descriptive the name should be
  - The more local, the shorter the name can be
- A name may include:
  - Type of entity or return value
  - Name of module
- Words in a name can be separated by underscores, hyphens, and case changes, but avoid using spaces

Formatting Code

- Use formatting to display the structure of the code
  - Indentation to display subordinate relationships between code
  - Alignment to identify blocks of code
  - Blank lines to separate blocks of code
- Write fully bracketed code to facilitate maintenance
- Write code in tabular form whenever possible
- Avoid “wrap-around” code

Scope of Variables

- Make the scope of variables as narrow as possible
  - Avoid global variables
- A wide-scoped variable is:
  - Harder to maintain because its instances may appear far apart from each other
  - More easily corrupted because its data can be modified by diverse procedures
- Decrease the scope of a variable by introducing procedures for accessing the variable

Procedures

- Use a convention for naming and ordering parameters
- Make explicit and carefully control any side-effects
  - Keep the use of side-effects to a minimum
- Make the scope of procedures as narrow as possible
- Any code fragment used more than once should be made into a procedure
  - Make procedures powerful
  - Use simple procedures to invoke powerful procedures in special ways

Code Documentation

- Components:
  - Specification of what the code is required to do
  - Pseudocode description of what the code does
  - Commented code
  - Proof that code’s behavior satisfies its specification
  - Mapping of code specification back to the design
- Several approaches:
  - Generate documentation from code files
  - Generate code from documentation files
  - Generate documentation and code from common files

Commenting Code

- Begin every system file with:
  - Copyright statement
  - Authors
  - Revision date
  - Description of contents
- Comment:
  - Each variable declaration
  - Each procedure definition
  - Loops and larger blocks of code
  - Anything that is not obvious
- Avoid excessive comments in procedure bodies
  - **Write code so that what it does is obvious**

Recursion

- Recursion can make code easier to describe, write, and prove correct
- Prove correctness using induction
- Simultaneous recursion is useful for defining a set of interrelated entities
- Sloppy uses of recursion can lead to total confusion
- In some cases, recursion may be highly inefficient
  - Use tail recursion in a programming language that executes tail recursive calls in constant space

Error Messages

- Make error messages as informative as possible
  - Indicate where in the code the error occurred
  - Describe the situation that caused the error
- “Throw” lower-level errors to appropriate higher-level code
- Write error messages for both the user and the developer

Conclusion

- Use an effective coding style
- Continuously look for ways of making your code:
  - Simpler
  - More powerful
  - Better documented
- Make the structure of the software explicit