Ohio State University Dept. of Astronomy

Coding Guidelines

Indent to show program structure
Indentation of structures and routines should be done. Acceptable indents range from 2-6 spaces per level.
Be Consistent! Control structures should be indented in the same way -- don't change part way through a program. Some individual preferences are fine here. Some people prefer:

```plaintext
if (. . .)
{
    then part
}
else
{
    else part
}
and others prefer:
if (. . .) {
    then part
} else {
    else part
}
Use one or the other.
```

Variable/Class/Function names should be descriptive.
Pick a variable naming style and stick with it.

Variable Scopes
Variables local to a routine should be declared at the top of the routine. I know many people like to declare variables within loops and inside the routine. Avoid this. The problem becomes tracing down where the variable was declared. The top of a function is a commonly accepted place for this and the amount of space you are saving within a block is usually insignificant.
Global variables should be avoided whenever possible -- and it is almost always possible. While there are times when global variables are a necessary evil you should really make an effort to avoid them.

Comments
Comments should clarify or provide a convenient quick reference to the code. The block comment should be indented to match the block of code being defined.
At the top of the module/file you should identify the name of the function, author, date, short description. This is helpful if somebody needs to search through a bunch of files. They know how recent it is, who was responsible, and (maybe) what it does. (Because a person is responsible does not always mean the module was produced by said person)

Example:

```/*!
\mainpage isl - Interjoined Software Logic for OSU/ISL instrument

\author Astronomy Staff, OSU Astronomy Dept. rgonzale@astronomy.ohio-state.edu)
\date 2005 May 01

\section Usage
prompt> isl - from an X window
Where: \c OSU instrument systems

\section Introduction
Interjoin: To join mutually.
...

ISL interjoins several systems into a new system by acting like a UNIX shell, initializing and starting the required OSU/ISL functions.

ISL activates the programs listed in the files /home2/isl/control/islpgm.ctl Each line in these control files is a system command to start a program.

<pre>
2005 May 01 - new application [osu]
</pre>

\todo

<ul>
</ul>
*/
```

```/*!
\file isl.c
\brief isl program creates a new system out of a task list.
*/
```
Use Appropriate Control Structures
The ‘while’ loop tests at top, the ‘for’ is useful for a set number of iterations, and the ‘do-while’ tests at bottom. Use the right structure. Pick the most appropriate one. Use ‘switch’ instead of an ‘if’ statement if possible. I know programmers hate ‘goto’ statements but I love them. It makes life easier when one is trying to develop tight code. REAL programmers don’t use ‘goto’ statements but I do, and it makes my code much more readable and reliable. Also I think that, people who say not to use them, don’t know why they are saying it. They are just mimicking what FORTRAN programmers use to say back in the sixties and seventies. Getting the ‘goto’ data punch card statement in the right place was a bear, and if you got one out of place in the deck of punch cards it would probably be the end of your routine and you would have to start all over again. There are no more punch cards so the problem of the ‘goto’ statements is nonexistent. So on our projects please feel free to use them.

Use Appropriate Data Structures
If an array is appropriate use it. Avoid using single variable names like a1, a2, a3, a4 instead of using an array. Also, if the array can be most appropriately sized at run time (dynamically allocated) do it. Static arrays are known for causing problems when data sizes exceed expected bounds (segmentation faults). If a linked list or tree would be more appropriate use it.

Use Appropriate Algorithms
Pick appropriate algorithms. For sorting a list that is known to be small a simple sort algorithm may be appropriate. If you are going to order a huge list, find an efficient algorithm. Remember how fast $n^2$, $n \log n$, and $n$ diverge.

Keep Routines Short and Use Routines Rather Than Pasting Code
Routines/functions should be short enough that a person can reasonably understand them. How short is reasonable? If you have a routine over a page you really should be thinking about whether it can be split up, but in some cases it is necessary to write a lengthy module, if it flows. This does not apply to the main program that is calling said routines and functions.
Breaking up a main program could become confusing at times (To Many Tangents Can Be Confusing) (TMTCBC).

Repeated code should become functions. I see too many times when it is cut and pasted throughout a program. If there is something wrong in that code it then needs to be fixed in all those places! If it is localized into a function it only needs to be fixed in that one spot.

Disclaimer
These suggestions are provided by research and by practitioners in industry. There are entire books on programming style and practices.