Good Programming Practices

Although over time we develop our own individual programming style, there are good programming practices that I believe to be important. Certainly, code that is to be used in the real-world tends to be more tactfully designed; however, irrespective of the environment or situation for which a piece of software is contrived, good source code is well readable by humans and is as much self-explaining as possible. Aside from that:

- *Provide an informative header.* This should include your name, the date of initial creation, the current date (if you are modifying an existing program), class abbreviation and assignment name/number[1], and a brief description of the program.
- *Comment liberally.* Place informative comments above functions describing what they do, above blocks of code summarizing their process, etc. It is good practice to comment anything that is not self-evident to someone (other than the author) looking at the code.
- *Indent.* A new level of indentation should be used at each level of nesting. A single space or several spaces is not a proper indentation. As a matter of fact, I prefer to use the cool `tab` key on my keyboard which automatically indents *at least* four spaces. Be consistent; indenting four spaces for a statement and five spaces for the next statement at the same level illustrates a clear lack of attention to detail.
- *Clearly define blocks.* Where to put those curly braces? I suggest placing them even with the left side of the block they are signifying.
- *Use spaces.* In particular, use them around operators.
- *Wrap long lines.* Typically, this should occur at column 75-85.
- *Use informative identifiers.* Sure, i is fine for a loop iterator, but it is not fine for storing some statistical average that is eventually displayed to the user, for example. Identifiers should communicate the named object's purpose to the user. Furthermore, be consistent with naming conventions. The variables `horse_face`, `horseface`, `horseFace`, and `HorseFace` are all fine; just pick a style and be consistent.
- *Use globals sparingly.* If at all possible, do not use any; instead, pass variables as function arguments.
- *Consolidate literals.* Use named constants instead and refer to literals symbolically.
- *Use functions.* Reuse your code. There is no sense in using a copy-and-paste philosophy that is not well maintanable.
- *Strive for clarity and simplicity.* Enough said.

[1]Clearly, this information implies an academic environment

I have also found some interesting articles online that further echo and elaborate on these good programming practices:

- http://www.kmoser.com/articles/Good_Programming_Practices.php
- http://www.kmoser.com/articles/Formatting_Your_Code.php
- http://www.kmoser.com/articles/How_to_Become_a_Great_Programmer.php

An now for an example that illustrates most of the above (in C++):

```
/*****************************************************************
 * Jean Gourd
 * CSC102
 * Assignment #1: The Cleverly-Worded Assignment
 *
 * 2007-08-21 (last updated 2007-08-22)
 *
 * This program attempts to illustrate to the user several important
 * (and good) programming practices; it does nothing else.
 *****************************************************************/
#include <iostream>
using namespace std;

// constants
static const int MAX_RECORDS = 5;

// function prototypes
void print_num_records(int&);
int foolishness();

/*********
 * MAIN
 *********/
int main()
{
	int num_records = 0;

	print_num_records(num_records);

	return 0;
}

/**************
 * FUNCTIONS
 **************/
```

```
// prints the number of records in the database
void print_num_records(int& num_records)
{
        num_records = foolishness();

        cout << "There are currently " << num_records << " records in "
             << "the database.\n";
}

// does something absolutely foolish
int foolishness()
{
        int num_records = 0;

        for (int i=0; i<MAX_RECORDS; i++)
                num_records += (i % MAX_RECORDS);

        return num_records;
}
```