# Chapter - 3
# Style

# Quality

Quality is designed in, not tested in.

— Dave Packard

# Style

- Style is the most important part of programming.
- Style is what separates the gems from the junk.
- It is what separates the programming artist from the butcher.
- The Mona Lisa and a paint-by-numbers picture are both paintings.
- What separates the two is ***Style***.

# Maintaining Maintenance

The average number of lines of code in a typical application has skyrocketed from 23,000 in 1980 to 1.2 million in 1990, according to a recent survey of managers attending the 1990 Annual Meeting and Conference of the Software Maintenance Association. At the same time, system age has risen from 4.75 to 9.40 years. Fortunately, the number of people devoted to maintaining them has made a comparable jump from 0.41 to 19.4.

What's worse, 74% of the managers surveyed at the 1990 Annual Meeting and Conference of the Software Maintenance Association reported that they "have systems in their department that have to be maintained by specific individuals because no one else understands them."

— Software Maintenance News, February 1991

# Comments

A program serves two masters.

- Code tells the computer what to do.
- Comments describe what the program does to the poor programmer who has to maintain it.

There are two types of comments in C++.

```
// Comments that begin with double-slash
// and go to the end of line

/* Comments that start with slash/star */
/*and go to star/slash */

/*
 * The second version can be used
 * for multi-line comments
 */
```

# Hello World

```
#include <iostream>
int main()
{
    std::cout << "Hello World\n";
    return (0);
}
```

What's missing from this program?

# Hello Again

```
/*************************************************
 * hello -- program to print out "Hello World".  *
 *          Not an especially earth-shattering program. *
 *                                               *
 * Author: Steve Oualline                        *
 *                                               *
 * Purpose: Demonstration of a simple program    *
 *                                               *
 * Usage:                                        *
 *          Run the program and the message appears  *
 *************************************************/
#include <iostream>
int main(){
    // Tell the world hello
    std::cout << "Hello World\n";
    return (0);
}
```

# Beginning Comments

- Heading
- Author
- Purpose
- Usage
- References
- File Formats
- Restrictions
- Revision History
- Error Handling
- Notes
- Anything else that’s useful

# Oualline's Law Of Documentation

90% of the time the documentation is lost.

Out of the remaining 10%, 9% of the time the revision of the documentation is different from the revision of the program and therefore completely useless.

The 1% of the time you actually have documentation and the correct revision of the documentation, it will be written in Japanese.

# Boxing with VI

Edit the file *.exrc* and add:

```
:abbr #b /**********************************************
:abbr #e **********************************************/
```

To create a top box, type
`#b<return>`

To create a box bottom
`#e<return>`

# Text-Setting

```
/*************************************************************
 *************************************************************
 ******** WARNING: This is an example of a          *******
 ******** warning message that grabs the            *******
 ******** attention of the program.                 *******
 *************************************************************
 *************************************************************/


/*------------> Another, less important warning<--------*/


/*>>>>>>>>>>>> Major section header <<<<<<<<<<<<< */


/*************************************************************
 * We use boxed comments in this book to denote the        *
 * beginning of a section or program                       *
 *************************************************************/


/*-----------------------------------------------------------*\
 * This is another way of drawing boxes                      *
\*-----------------------------------------------------------*/
```

# More Text Setting

```
/*
 * This is the beginning of a section
 * ^^^^ ^^ ^^^ ^^^^^^^^^^ ^^ ^ ^^^^^^^
 *
 * In the paragraph that follows we explain what
 * the section does and how it works.
 */


/*
 * A medium level comment explaining the next
 * dozen or so lines of code. Even though we don't have
 * the bold typeface we can **emphasize** words.
 */


/* A simple comment explaining the next line */
```

# Variables

Use long names (but not too long).

```
int p,q,s;                  // Wrong
int account_number;         // Right
```

Always comment your variable declarations

```
int    name_count;  // The number of
                    // names in the list
```

Units are important.

```
int    length; // Length of the widget
```

Is length, mm, cm, miles, light-years or microns? The answer's important.

The following comes from a real program written by Steve Oualline:

```
/*********************************************
 * Note: I have no idea what the input units are, nor   *
 * do I have any idea what the output units are,        *
 * but I have discovered that if I divide by 3          *
 * the plots look about the right size.                 *
 *********************************************/
```

# KISS (Keep it Simple, Stupid)

Which is more valuable?

1) A clear, well written, easy to read, but broken program

2) A clever complex working program.

# Precedence Rules

ANSI Standard Rules

1. () [] -> .
2. ! ~ ++ -- (type) - (unary)
   * (dereference) & (address of) sizeof
3. * (multiply) / %
4. + -
5. << >>
6. < <= > >=
7. == !=
8. & (bitwise and)
9. ^
10. |
11. &&
12. ||
13. ?:
14. = += -= etc.
15. ,

# Practical Precedence Rules

1. * (multiply) / %
2. + -

Put parentheses around everything else.

# Clever vs. Simple

Clever and very compact.

```
while ('\n' != *p++ = *q++);
```

What does this do? It takes up very little space, but don't save space, save the sanity of the people that follow you.

Simple and somewhat easier to understand.

```
while (1) {
    *destination_ptr = *source_ptr;

    ++destination_ptr;
    ++source_ptr;

    if (*(destination_ptr-1) == '\n')
        break;  // exit the loop if done
}
```

# Naming Style

Most programs follow the convention that variables are all lower case:

```
source_ptr, current_item
```

Most **#define** constants are all upper case:

```
MAX_ITEMS, SCREEN_WIDTH
```

Constants declared with the **const** C++ keyword generally follow no convention. (Unfortunately)

Some people use Upper/Lower case instead of underscores (_) in variables.

```
CurrentItemList, LargestAccount
```

# Indentation

Indentation is a religious issue. Many religious wars are waged over where to put the curly braces ({}).

Some common styles are:**The short form**

```
while (! done) {
    std::cout << "Processing\n";
    next_entry();
}


if (total <= 0) {
    std::cout << "You owe nothing\n";
    total = 0;
} else {
    std::cout << "You owe " <<
                  total << " dollars\n";
    all_totals = all_totals + total;
}
```

# Braces stand alone

```
while (! done)
{
    std::cout << "Processing\n";
    next_entry();
}


if (total <= 0)
{
   std::cout << "You owe nothing\n";
   total = 0;
}
else
{
   std::cout << "You owe $" << total << "\n";
   all_totals = all_totals + total;
}
```

# Clarity

Programs should read like a technical paper. Break your code into sentences, paragraphs, sections, chapters and books.

Example of a program with no “paragraphs.”

```
// poor programming practice
temp = box_x1;
box_x1 = box_x2;
box_x2 = temp;
temp = box_y1;
box_y1 = box_y2;
box_y2 = temp;
```

# Paragraphs add Clarity

Same program with paragraphs added in.

```
/*
 * Swap the two corners
 */


/* Swap X coordinate */
temp = box_x1;
box_x1 = box_x2;
box_x2 = temp;


/* Swap Y coordinate */
temp = box_y1;
box_y1 = box_y2;
box_y2 = temp;
```

# Simplicity

Dennis Day worked on an old radio program called the Jack Benny show. Dennis was not the brightest person in the world. The following exchange took place during a remote broadcast from San Diego:

Jack: I don't understand it, Dennis. It took you five days to get here from Los Angeles. What took you so long?

Dennis: I ran into a lot of traffic in Salt Lake city.

Jack: Dennis... Dennis... Why did you go from Los Angeles to San Diego by way of Salt Lake City?

Dennis: I wanted to avoid the traffic light in Oceanside.

The moral of this story. When you program don't go from Los Angeles to San Diego by way of Salt Lake City.

# Simplicity

- A single function should not be more than one or two pages long.
- Avoid complex logic like multiple-nested if's.
- About the time your code starts to run into the right margin, you probably should consider splitting up into smaller, simpler units.
- Did you ever read a sentence, like this one, where the author went on and on, stringing together sentence after sentence with the word "and" and didn't seem to understand the fact that several shorter sentences would do the job much better and didn't it bother you?

  C++ statements should not go on forever. Split long statements into smaller, simpler ones.
- Split large single code files into multiple smaller ones. Any file with more than about 1,500 lines of code is hard to edit and even harder to understand.
- When designing classes, try to put one class per module.

# The Golden Rule of Programming:

Make your program as clear and as simple as possible.