Documentation and Making Source Code Readable*

Kenneth Leroy Busbee

Abstract

A source code listing and discussion of several items that make source code easier to read and maintain.

1 General Discussion

We are going to consider a simple program that might be used for testing a compiler to make sure that it is installed correctly.

Example 1: Compiler_Test.cpp source code

```
//*****************************************************
// Filename: Compiler_Test.cpp
// Purpose:  Average the ages of two people
// Author:   Ken Busbee; © Kenneth Leroy Busbee
// Date:     Jan 5, 2009
// Comment:  Main idea is to be able to
//           debug and run a program on your compiler.
//******************************************************

// Headers and Other Technical Items

#include <iostream>
using namespace std;

// Function Prototypes

void pause(void);

// Variables
```

*Version 1.5: Mar 16, 2010 11:30 am -0500
†http://creativecommons.org/licenses/by/2.0/

```
int     age1;
int     age2;
double  answer;

//*****************************************************
// main
//*****************************************************

int main(void)
  {
  // Input
  cout << "\nEnter the age of the first person --->: ";
  cin >> age1;
  cout << "\nEnter the age of the second person -->: ";
  cin >> age2;

  // Process
  answer = (age1 + age2) / 2.0;

  // Output
  cout << "\nThe average of their ages is -------->: ";
  cout << answer;

  pause();
  return 0;
  }

//*****************************************************
// pause
//*****************************************************

void pause(void)
  {
  cout << "\n\n";
  system("PAUSE");
  cout << "\n\n";
  return;
  }

//*****************************************************
// End of Program
//*****************************************************
```

Within the programming industry there is a desire to make software programs easy to maintain. The desire centers in money. Simply put, it costs less money to maintain a well written program. One important aspect of program maintenance is making source code listings clear and as easy to read as possible. To that end we will consider the following:

1. Documentation
2. Vertical Alignment

3. Appropriate use of Comments
4. Banners for Functions
5. Block Markers on Lines by Themselves
6. Indent Block Markers
7. Meaningful Identifier Names Consistently Typed
8. Appropriate use of Typedef

The above items are not needed in order for the source code to compile. Technically the compiler does not read the source code the way humans read the source code. But that is exactly the point; the desire is to make the source code easier for humans to read. You should not be confused between what is possible (technically will compile) and what is ok (acceptable good programming practice that leads to readable code). Let's cover each item in more detail.

1.1 Documentation

Documentation is usually placed at the top of the program using several comment lines. The amount of information would vary based on the requirements or standards of the company who is paying its employees or independent contractors to write the code. Notice the indication of revision dates.

1.2 Vertical Alignment

You see this within the documentation area. All of the items are aligned up within the same column. This vertical alignment occurs again when the variables are defined. When declaring variable or constants many textbooks put several items on one line; like this:

Example 2: Common Textbook Defining of Variables

```
float  length, width, height, price_gal_paint, total_area, total_cost;
int  coverage_gal_paint, total_gal_paint;
```

However common this is in textbooks, it would generally not be acceptable to standards used in most companies. You should declare each item on its own line; like this:

Example 3: Proper Defining of Variables with Vertical Alignment

```
float  length;
float  width;
float  height;
float  price_gal_paint;
int    coverage_gal_paint;
float  total_area;
int    total_gal_paint;
float  total_cost;
```

This method of using one item per line is more readable by humans. It is quicker to find an identifier name, because you can read the list vertically faster than searching horizontally. Some programmers list them in alphabetic order, especially when the number of variables exceeds about twenty.

The lines of code inside either function are also aligned vertically and indented two spaces from the left. The indentation helps set the block off visually.

1.3 Appropriate use of Comments

You can see through the source code short little comments that describe an area or section. Note the use of input, processing and output which are part of the IPO concept within the program design.

1.4 Banners for Functions

Note the use of comments in the form of a banner before each function.

Example 4: Comments as a Banner

```
//*****************************************************
// main
//*****************************************************
```

The function name is placed with two lines of asterisks. It makes it extremely easy to find each function definition because you don't have to read the functions to see where the one ends and the next one begins. You can quickly read the function names within the banners.

1.5 Block Markers on Lines by Themselves

Within many languages there is a method to identify a group of programming statements as a unit. With C++ the functions use a set of symbols, the braces {}, to identify a block of code, sometimes referred to as a compound statement. Braces are used in other aspects of programs, but for now we will look at this simple example. These braces have a tendency to cause problems, especially when they don't have a proper opening brace associated with a proper closing brace. To solve that problem many programmers simply put a brace on a line by itself and make sure the opening brace and closing brace are in the same vertical column.

1.6 Indent Block Markers

A block of code associated with a function or with a control structure is indented two or three spaces. When blocks of code are nested each nesting is indented two or three spaces. In our example above the blocks of code for the function definitions are indented two spaces.

1.7 Meaningful Identifier Names Consistently Typed

As the name implies "identifier names" should clearly identify who (or what) you are talking about. Calling you spouse "Snooky" may be meaningful to only you. Others might need to see her full name (Jane Mary Smith) to appropriately identify who you are talking about. The same concept in programming is true. Variables, constants, functions, typedefs and other items should use meaningful identifier names. Additionally, those names should be typed consistently in terms of upper and lower case as they are used in the program. Don't define a variable as: Pig and then type it later on in your program as: pig.

1.8 Appropriate use of Typedef

Many programming languages have a command that allows for the creation of an identifier name that represents a data type. The new identifier name is described or connected to a real data type. This feature is not demonstrated in the code above and is often a confusing concept. It is a powerful way to help document a program so that it is meaningful, but is often used by more experienced programmers.

2 Definitions

Definition 1: documentation
A method of preserving information useful to others in understanding an information system or part thereof.

Definition 2: vertical alignment
A method of listing items vertically so that they are easier to read quickly.

Definition 3: comments
Information inserted into a source code file for documentation of the program.

Definition 4: banners
A set of comment lines used to help separate the functions and other sections of a program.

Definition 5: braces
Used to identify a block of code in C++.

Definition 6: indention
A method used to make sections of source code more visible.

Definition 7: meaningful
A rule that says identifier names must be easily understood by another reading the source code.

Definition 8: consistent
A rule that says to type identifier names in upper and lower case consistently throughout your source code.