Use comments to document design choices.
Use comments to explain why the code works the way it does.
Do not use comments to state what the code does (the code already says this).

```
// most single line comments look like this

/*
 * VERY important single line comments look like this
 */

/*
 * Multi-line comments look like this. Make them real sentences.
 * Fill them in so they look like real paragraphs.
 */
```

Every source file must begin with this ID block.
Obviously, you should replace the contents to the right of each header with your information.

```
// ---------------------------------------------------------------------------
// Name: your name
// Course-Section: CS105-section #
// Assignment: project #
// Date due: 01/01/1980
// Collaborators: names of collaborators (see collaboration policy)
// Resources: list of resources utilized
// Description: a short description of the program's purpose should go here.
//              The description may take more than one line.
// ---------------------------------------------------------------------------
```

Use a single space after all keywords unless otherwise stated in this guide.
Use a single space before and after all binary operators except the . and -> operators.
Use a single space after a comma.

Indentation is at least two spaces OR an 8 character tab. Do not mix tabs and spaces.
Use either spaced indents or tab key indents, never both in the same source file.
All indents are the same width.
★ Four space indents are preferred.

All code and comments should fit within 77 columns. Wrap lines that exceed this length.
Use a single second level indent for wrapped lines (continuation).

```
    cout << "This is a really long line that should be wrapped " <<
        "so it does not extend past column 77 of the screen. " <<
        "Note that each continued line is indented to show " <<
        "that it is a continuation of the statement. << endl;
```

Closing and opening braces always go on a separate line (Allman or BSD style).

```
void checkWeight(float weight)
{
    if (weight > MAX_WEIGHT)
    {
        oversize = true;
        cout << "Load is oversized" << endl;
    }
    else
    {
        oversize = false;
        cout << "Load meets weight constraints" << endl;
    }
}
```

Indent statement blocks inside control structures (if, while, do, etc.).
A single statement does not require braces, but may have braces if preferred.

```
    if (isControlStructure)
        cout << "Indent one level" << endl;

    if (batteryLow)
    {
        cout << "The whole statement block is indented one level" << endl;
        cout << "The braces are not indented" << endl;
    }
```

In a switch statement, the case is not indented (align with the switch keyword).
Align the colons in the same column with a single space before and after the colon.
Indent the statements in each case to the same column.

```
    switch (letter)
    {
    case 'A' : highCount++;
               break;
    case 'B' : lowCount++;
               break;
    default  : otherCount++;
    }
```

Do not add whitespace at the end of a line or at the end of a file.
A single blank line at the end of the file is acceptable, but not required.

Use a single blank line between logically distinct portions of code.

Increment and decrement operators are attached to their operand (no space).

```
frog++;
--prince;
```

Named constants are all uppercase.

```
const float CM_PER_INCH = 2.54;
```

Avoid numeric literals ("magic numbers"). Use named constants instead of numeric literals.

```
const float CM_PER_INCH = 2.54;
```

When numeric literals are unavoidable (for example a mathematical formula), include a comment to document the origin or meaning of the numeric literal.

```
// www.bmi-formula.com
bmi = (pounds / (inches * inches)) * 703;
```

Do not use a space after an opening parenthesis or a space before a closing parenthesis.
Do not use a space between parentheses.

```
splitCheck = totalBill + ((totalBill - tax) * tipPercent);
```

Use lower case for all identifiers except when otherwise specified in this style guide.

Use meaningful names for all variables, functions, and other identifiers.

Use a consistent identifier naming scheme, either camelcase or underscore separated.

```
int camelCaseNames;
 -OR-
int underscore_separated_names;
```

Do not use global variables (variables with a global scope).

Do not start identifier names with an underscore.
Many libraries use names that start with an underscore and avoiding this practice minimizes the risk of accidentally creating a name conflict in your code.

For-loops should use the conventional idiomatic style without spaces before and after operators.

```
for (int i=0; i<var; i++)
```

There should be no space between the function name and the argument list.
Function prototypes go at the top of the source file where they are defined for single file programs. Functions that are used in more than one source file are placed into separate header and implementation files.

Use a single blank line followed by a single line comment of 75 dashes before every function definition except the *main* function. The *main* function is preceded by a single blank line.

Every function except the *main* function must have a brief comment that discusses the purpose of the function, the input parameters, and outputs and/or return value.

```
//---------------------------------------------------------------------------
/* The power() function uses Ohm's law to compute the power output of the
 * amplifier in watts. The function displays the total power in watts.
 */
void power(float volts, float amperes)
{
    cout << "Watts: \t" << (volts * amperes);
}

//---------------------------------------------------------------------------
/* isAlarmTime() checks to see if the alarm time is equal to the current
 * time. The inputs are the alarm hour and alarm minute along with the
 * current hour and current minute. True is returned when they are equal and
 * false otherwise.
 */
bool isAlarmTime(int ahr, int amn, int hr, int mn)
{
    bool ringAlarm = false;

    if ((ahr == hr) && (amn == mn))
        ringAlarm = true;

    return ringAlarm;
}
```

The array subscripting operator (square-brackets operator) is attached to the variable.

```
cout << myArray[5] << endl;
```

Enumeration values are all uppercase.

```
  enum typename { ONE, TWO, THREE };
  enum class typename { ONE, TWO, THREE };
```

The reference operator is attached to the type name.
The address-of operator is attached to the operand (variable).

```
void getSize(int& length, int& width);
myPointer = &length;
```

The scope resolution operator should not be preceded or followed by whitespace.

```
std::string first_name;
std::cout << first_name << std::endl;
```

Source code contained within a single file should be in the following order:

```
ID BLOCK
includes in alphabetical order
using directives

enumerated types

structs

function prototype
function prototype
function prototype

main function

//-----------------------------------------------------------------------
/*
 *
*/
function definition

//-----------------------------------------------------------------------
/*
 *
*/
function definition
```

Source code projects that span multiple files should use the "single" layout above for the driver. Header files should be in the following order:

```
ID BLOCK
begin inclusion guard
includes in alphabetical order
using directives

enumerated types

structs

function prototype
function prototype
function prototype
end inclusion guard
```

-OR-

```
ID BLOCK
begin inclusion guard
includes
using directives

enumerated types

structs

class declaration
end inclusion guard
```

Always use inclusion guards on header files.
Inclusion guard macro names should be the header file name in all upper case with the period replaced by an underscore.

If the header file name is `coco.h`, then the inclusion guards surround the header source.

```
ID BLOCK
#ifndef COCO_H
#define COCO_H

header body

#endif /* COCO_H */
```

For pointer variable declarations the asterisk is attached to the type name.
The indirection operator is attached to the operand (variable).

```
int* myIntPointer;
squareFeet = *length + width;
cout << *(arrayPointer + i) << endl;
```

The arrow operator should not be preceded or followed by whitespace.

```
myPtr->name = "Josie";
```

Only declare one class per header file.
Always separate the class declaration and implementation (header and implementation files).
Exception: Templated classes must contain the implementation in the header file.
Never #include implementation (cpp) files.

Class implementation files should be in the following order:

```
ID BLOCK
includes in alphabetical order
using directives

//---------------------------------------------------------------------------
/*
 *
*/
function definition

//---------------------------------------------------------------------------
/*
 *
*/
function definition
```