# Documentation and Code Style

CS 1044

# Motivation

- Writing good computer programs is **not** just about getting the correct results
- **Documenting your code** and **making it readable** makes it easy for another reader to understand, such as another team member
- It also helps **you** if you need to go back and look at your own code after a month, or a year

# Commenting

- C++ lets us insert **free text** comments into our code
- Comments are **ignored** by the compiler – they are solely for the programmer's benefit

# Single-Line Comments

- Single-line comments are preceded by //
- Everything from the // to the end of the line is ignored

```
// A comment can be on its own line:
// The following line applies sales tax.
double total = subtotal * 1.05;

int x = 5;   // Can also be at end of a line
```

# Block Comments

- Block comments can start with `/*` and end with `*/`
- These can be on one line, part of a line, or span multiple lines

```
/* The following line of code is the one that
   applies the sales tax to the subtotal */
double total = subtotal * 1.05;

double x = 4.0 /* don't do this */ + y;
```

# Internal Commenting

- Make judicious use of internal commenting inside your functions
- Every variable should include a brief comment describing its purpose, if not obvious from the name
- Potentially “tricky” algorithms, expressions, or blocks of code should be preceded by a comment that explains them

# Useless Comments

- Comments should explain **why** your code does something, not **how**
- Your code itself already says **how**
- Example of a useless comment:

```
int x = 5;  // sets x to be equal to 5
```

# Documenting Functions

- Every function you define in your program should start with a comment describing:
  - What does the function do? What is its **purpose**?
  - What are each of its **parameters**?
  - If it is not a `void` function, what does it **return**?

# Function Comments

```
//
// Computes the sum and average of two numbers.
//
// a the first number
// b the second number
// avg a reference to a variable that will
//     receive the average of a and b
// returns the sum of a and b
//
int sum_and_average(int a, int b, double& avg);
```

# Function Comments

```
//
// Computes the sum and average of two numbers.
//
// a the first number
// b the second number
// avg a reference to an int that will
//     receive the average of a and b
// returns the sum of a and b
//
int sum_and_average(int a, int b, double& avg);
```

The first paragraph should be a brief description of the function – its purpose, and/or how it behaves

# Function Comments

Every parameter should be listed, followed by the name of the parameter, followed by a description

```
//
// Computes the sum and average of two numbers.
//
// a the first number
// b the second number
// avg a reference to a variable that will
//     receive the average of a and b
// returns the sum of a and b
//
int sum_and_average(int a, int b, double& avg);
```

# Function Comments

```
//
// Computes the sum and average of two numbers.
//
// a the first number
// b the second number
// avg a reference to a variable that will
//     receive the average of a and b
// returns the sum of a and b
//
int sum_and_average(int a, int b, double& avg);
```

If the function is non-void, you must finally have a description of the value the function returns

# Other Points of Style

- Commenting is only one facet of code style
- Remember that C++ ignores most whitespace in your program
- So, the following is a valid, but ugly, program

```
#include <iostream>
using namespace     std;int main(      ){cout<<
"hello"                                <<
endl;return
                                                0;}
```

# Curly Braces

- Curly braces for a block should **line up vertically**
- This symmetry makes it easier to see where blocks of code begin and end

- **Good**

```
if (x)
{
    something();
}
```

- **Not as good**

```
if (x) {
    something();
}
```

# Curly Braces

- Always use curly braces for `if`/`else`/`for`/`while` blocks
- Even if only one statement follows them
- Code is less error prone and easier to modify this way

- **Good**

```
if (x)
{
    something();
}
```

- **Not as good**

```
if (x)
    something();
```

# Indentation

- Indent your code to show its structure
- Use curly braces as a guide – each pair represents a new indentation level

```
int main()
{
    bool x = true;

    if (x)
    {
        foo();
    }
}
```

# Other Use of Spaces

- Use a single space between `if`/`else`/`for`/`while` and the following parenthesis
- Put a single space around operators like `+`, `-`, `*`, `/`, `&&`, `||`, etc.
- No spaces immediately inside parentheses, or after function names

- **Good**

```
if (x)
{
    foo(x + 9);
}
```

- **Not as good**

```
if( x )
{
    foo (x+9);
}
```

# Naming Conventions

- Variables, functions, and data types should be lowercase with words separated by underscores, or "`camelCase`"
- Constants should be all `UPPERCASE`

```
int sum;
double average(...);
int group_size;
int groupSize;

const double TAX_RATE =
    1.05;
```