CS 121 SI - Worksheet 12 - Miscellaneous Programming Tips

Intro:

- This worksheet, like Thursday's lecture, mostly serves to help you move closer to typing like a professional programmer.
- This worksheet also encourages you to define what programming style you want for later years to come.
- You may or may not consider this worksheet extremely boring. I understand, I took this class, although do note that it is here to improve your programming skills (and will if you follow along).

Some General Tips for C++ Programming: This is as an addition to Dr. Siska's lecture

- **WRITE PSEUDO-CODE!**
  - This is explained in more detail in a later section (i.e. "Code Creation Tips")

- **Naming Schemes:**
  - *Class names:* ProperCase
  - *Constants:* CAPS_AND_UNDERSCORES
  - *Identifiers:* Generally up to your personal preference
    - *Main Rule:* **Pick a naming scheme, stick with it**
    - *Tip:* **Underscores** are generally easier to read, but take more time to type
    - Make every identifier (be it function, variable, constant, class, etc.) **meaningful and concise**. You'll thank yourself in the long run.
      - The more used the variable, the smaller the name should be.
      - The less used the variable, the more descriptive the name.

- **Indentation:** *Tabs vs. Spaces - Which to use?*
  - Don't have more than one executable statement per line (for ease of readability)
  - Tabs vs. Spaces - Which to use?
    - Mostly up to you, but only use one
      - **Reason:** Various text editors treat spaces/tabs differently.
      - *Tip:* Change your editor's tab/indent settings to set up tabbing

- **Bracing**: *Egyptian versus In-line*
  - *Egyptian:* For the experienced, lazy, or those that really want to save a line.
    - ex:

```
void main() {
  // curly braces form an egyptian shape
}
```

  - *In-line:* For everyone. This is easier to read as indents are marked clearly.
    - ex:

```
void main()
{
  // curly braces are on the same line
}
```

- **Comments**: When do you need them?
    - Used for explaining difficult/weird portions of your code (e.g. tricky optimizations)
    - Used for creating stubs and writing pseudo-code
    - Unless requested by a prompt/project group, you generally shouldn't need to comment your work.
        - **Reasons**:
            1. Your identifiers (variables and functions) should describe their purpose concisely and concretely.
            2. Your logic should should be clear to begin with.
            3. Your pseudocode already explains your software.

- **Operator Precedence:** Best not to rely on it
    - e.g. *int y = 3*x + 1;* should be rewritten as *int y = (3*x) + 1;* for safety

- **Function Prototype/Definition Grouping:**
    - Group similar tasks together
        - e.g. Move all setters together in a class

Code Creation Process: A General Approach:

- Steps for (generally) writing software:
    1. When starting a program, **always write pseudocode first**.
        a. This will save you time in the long run (it has for me, countless times)
    2. After your pseudocode is complete (and revised), write your pseudo-code into your source code files as comments.
        a. Note that, at this point, you should not have (many) executable lines.
        b. These are generally referred to as stubs.
    3. Incrementally implement parts of the pseudo-code (until your program is made).
        a. This helps makes integrating your code a lot easier.
        b. Note that pseudo-code translation to C++ is not one-to-one; You might need more than one line of C++ code to represent your pseudo-code.
        c. This also helps make testing a whole bunch simpler to deal with.

Practice:

1. Are there any tips that you disagree with?

2. What tips would you suggest to others in terms of programming?

3. Go back to one (or more) of your programming assignments and apply the above tips.