Some Recommended Programming Principles/Practices/Habits/. . .

Lee S Koh

NOTE: The entries are not necessarily mutually exclusive (*i.e.* , they may overlap or conflict one another).
In case of conflict, strike an appropriate compromise.
Risking telling the obvious, situations quoted are just select illustrative examples.

- **Don't "make the soup too salty"**
  - Don't include "`using namespace std;`" in header files
  - Don't introduce "flexibility-reducing" newline in an outputting function
- **Least privilege -- enable/enpower/reveal/... only what's necessary, not anything more**
  (part of "Do keep on the defensive")
  - Don't unnecessarily *pass by reference*
  - Use *pass-by-value* or *pass-by-* `const`*-reference* (instead of *pass by reference*) if ***no side effect*** (on the original) is intended
- **Don't do the same thing more than once**
  - Don't keep making an *identical function call* (*i.e.* , one that ***returns the same value every time***) over and over in a loop
    - Rather make the function call outside the loop and capture the return value in a local variable (which is then used in the loop)
  - Minimize # of operations in the repetitive part of a loop construct if doing so won't sacrifice other desirables (clarity, safety, …)
    - (shifting elements of array `data` with `used` items when removing a `key`-matching item)

```
for (i = 0; i < used; ++i)
{
   if (a[i] == key)
   {
      for (j = i + 1; j < used; ++j)
         a[j - 1] = a[j];
      break;
   }
}
```

vs

```
for (i = 0; i < used; ++i)
{
   if (a[i] == key)
   {
      for (j = i; j < used - 1; ++j)
         a[j] = a[j + 1];
      break;
   }
}
```

- **Don't "throw away old TV before new TV is in hand"**
  - When resizing a dynamic array, don't free up "old" array before the "new" array is in place
- **Don't be inconsistent (contradictory) between design intent and language feature usage**
  - Appropriately include "`const`" if a member function is meant to be an ***accessor***
  - Use *pass-by-value* or *pass-by-* `const`*-reference* (instead of *pass by reference*) if ***no side effect*** (on the original) is intended
- **Don't expose/baffle client to/with implementation detail -- be client-oriented**
  - Don't include *known-only-to-implementor* detail in error-reporting messages
  - Begin item numbering with 1 instead of 0 when crafting user interface
- **Don't sacrifice efficiency unless there's something else more desirable to be gained**
  - Use *pre* -version of `++` or `--` (instead of the *post* -version) when either version will give the same outcome
  - Use *pass-by-* `const`*-reference* (instead of *pass-by-value*) when size of object involved is ***big***
  - Use *initializer* /*initialization list* (instead of *in-body assignments*) wherever possible when implementing constructor
  - Code as compactly as possible if doing so won't sacrifice other desirables (clarity, safety, …)
    - (for a `Container` class where `used` tracks the # of items)

```
bool Container::empty() const
{
   return used == 0;
}
```

vs

```
bool Container::empty() const
{
   bool answer;
   if (used == 0) answer = true;
   else answer = false;
   return answer;
}
```

- **Do look for simpler/clearer & more direct/efficient alternative(s) if the one at hand seems unnecessarily complex/awkward**
  - Avoid the more costly ***repeatedly swap*** when all that's needed is to ***repeatedly shift***
    - (shifting elements of array `data` with `used` items when removing a `key`-matching item found at index `keyIndex`)

```
for (i = keyIndex + 1; i < used; ++i)
{
   data[i - 1] = data[i];
}
```

vs

```
for (i = keyIndex; i < used - 1; ++i)
{
   hold = data[i];
   data[i] = data[i + 1];
   data[i + 1] = hold;
}
```

  - Avoid confusingly expressing underlying logic, inviting ***1-off error*** , and incurring extra computations
    - (shifting elements of array `data` with `used` items when removing a `key`-matching item found at index `keyIndex`)

```
for (i = keyIndex + 1; i < used; ++i)
{
   data[i - 1] = data[i];
}
```

vs

```
count = used - keyIndex - 1;
for (i = 0; i < count; ++i)
{
   data[keyIndex + i] = data[keyIndex + i + 1];
}
```

- **Do keep on the defensive**
  (block ways that others may abuse/misuse, take safer ways ourselves to avoid falling victim to our own shortcomings)
  - `if (10 == i) {...}` is safer than `if (i == 10) {...}`
  - Trap error conditions wherever possible and expedient
- **Do constantly apply <u>common sense</u> and check if things <u>make sense</u>.**