Chapter - 29 Programming Adages

General

- Comment, comment, comment. Put a lot of comments in your program. They tell other programmers what you did. They also tell you what you did.
- Use the “KISS” principle. (Keep it Simple, Stupid.) Clear and simple is better than complex and wonderful.
- Avoid side effects. Use ++ and -- on lines by themselves.
- Never put an assignment inside a conditional. Never put an assignment inside any other statement.
- Know the difference between = and = =. Using = for = = is a very common mistake and is difficult to find.

General

•Never do “nothing” silently.

```
// Don't program like this
for (index = 0; data[index] < key; ++index);
// Did you see the semicolon at the end of
// the last line?
```

Always put in a comment.

```
for (index = 0; data[index] < key; ++index)
        /* do nothing */;
```

•Practice coding. People involved in almost every other profession that requires some significant level of skill and creativity practice (e.g. artists, athletes). Helping others learn to program, makes good practice for you by going over what you already know, or think you know.

Design

•If you come to a choice between a relatively “quick hack” or a somewhat more involved but more flexible solution, always go for the more flexible solution. You’re more likely to reuse it or learn from it. You’re also more likely to be thankful later on when requirements shift a little and your code is ready for it.

•Never trust any user input to be what you expect. What would your program do at any given point if a cat walked across the keyboard, several times?

•Watch out for signed-unsigned conversions and over/under flow conditions.

Declarations

- Put variable declarations one per line and comment them.
- Make variable names long enough to be easily understood, but not so long that they are difficult to type in. (Two or three words is usually enough.)
- Never use default return declarations. If a function returns an integer, declare it as type **int**.

Switch Statement

- Always put a default case in a switch statement. Even if it does nothing, put it in.

```
switch (expression) {
    default:
        /* do nothing */;
        break;
}
```

- Every case in a switch should end with a **break** or `/* fall through */` statement.

Pre-processor

•Always put parentheses () around each constant expression defined by a pre-processor **#define** directive.

```
#define BOX_SIZE (3*10) /* size of the box in pixels */
```

•Put () around each argument of a parameterized macro.

```
#define SQUARE(x) ((x) * (x))
```

•Surround macros that contain complete statements with curly braces.

```
// A fatal error has occurred.  Tell user and abort
#define DIE(msg) {(void)printf(msg);exit(8);}
```

•When using the **#ifdef**/**#endif** construct for conditional compilation, put the **#define** and **#undef** statements near the top of the program and comment them.

•Whenever possible use **const** instead of **#define**.

•The use of **inline** functions is preferred over the use of parameterized macros.

Style

•A single block of code enclosed in {} should not span more than a couple of pages. Split up anything much bigger than that up into several smaller, simpler procedures.

•When your code starts to run into the right margin, it’s about time to split the procedure into several smaller simpler procedures.

•Always define a constructor, destructor and copy constructor for a class. If using the C++ defaults, “define” these routines with a comment such as:

```
class example {
    public:
        // example -- default constructor
```

Compiling

- Always create a *Makefile* so others will know how to compile your program.
- When compiling, turn on all the warning flags. You never know what the compiler might find.

The Ten Commandments for C++ Programmers

By Phin Straite

1. Thou shalt not rely on the compiler default methods for construction, destruction, copy construction, or assignment for any but the simplest of classes. Thou shalt forge these "big four" methods for any non-trivial class.
2. Thou shalt declare and define thy destructor as virtual such that others may become heir to the fruits of your labors.
3. Thou shalt not violate the "is-a" rule by abusing the inheritance mechanism for thine own twisted perversions.

The Ten Commandments for C++ Programmers

4. Thou shalt not rely on any implementation-dependent behavior of a compiler, operating system, nor hardware environment, lest your code be forever caged within that dungeon.
5. Thou shalt not augment the interface of a class at the lowest level without most prudent deliberation. Such ill-begotten practices imprison thy clients unjustly into your classes, and foment unrest when code maintenance and extension are required.
6. Thou shalt restrict thy friendship to truly worthy contemporaries. Beware, for thou art exposing thyself rudely as from a trench coat.
7. Thou shalt not abuse your implementation data by making it public or static except in the rarest of circumstances. Thy data are thine own; share it not with others.

The Ten Commandments for C++ Programmers

8. Thou shalt not suffer dangling pointers or references to be harbored within your objects. These are nefarious and precarious agents of random and wanton destruction.

9. Thou shalt make use of available class libraries as conscientiously as possible. Code reuse, not just thine own but that of your clients as well, is the holy grail of OO.

10. Thou shalt forever forswear the use of the vile `printf`/`scanf`, rather favoring the flowing streams. Cast off thy vile C cloak and partake of the wondrous fruit of flexible and extensible I/O.

Final Note

Just when you think you've discovered all the things that C++ can do to you — think again. There are still more surprises in store for you.

Question: Why does the following program think everything is two? (This inspired the last adage.)

```
#include <iostream>
int main() {
    int number;

    std::cout << "Enter a number: ";

    std::cin >> number;

    if (number =! 2)
        std::cout << "Number is not two\n";
    else
        std::cout << "Number is two\n";
    return (0);
}
```