

A Summary of Coding Standards

Redeemer University College

Computer Science Department

September 2005

Dr. D. Schuurman

1 Introduction

The purpose of this document is to provide students with a basic outline of the coding standards for Computer Science courses at Redeemer University College. These rules are by no means final, but will serve as basis for writing code submitted for assignments and will be used to evaluate marks for documentation and programming style.

2 Motivation

Coding Standards are important for a number of reasons. This reasons include:

1. Much of the cost of a piece of software goes to maintenance
2. Often programs are maintained by others besides the original author
3. Coding standards improve the readability of software making it easier to understand (and mark)

3 Programming Style

The basic principles of good coding practice include [1]:

simplicity keep programs short and manageable

clarity ensure code is easy to understand for both people and machines

generality creating programs that work well in a broad range of situations

4 Braces and Nested Constructs

Make nested constructs easy for your readers to understand by indenting and using braces where appropriate. When you have an `if-else` statement nested in another `if` statement, always put braces around the `if-else`. Thus, never write like this:

```
if (x > y)
  if (y > z)
    win();
  else
    lose();
```

always like this:

```
if (x > y)
{
  if (y > z)
    win();
  else
    lose();
}
```

If you have an `if` statement nested inside of an `else` statement you may write `else if` on one line, like this,

```
if (x > y)
  win();
else if (y > z)
  lose();
```

4.1 Automatic Formatting

Some editors (such as X-Emacs) will automatically indent and format your program source files but many editors do not. To ensure that the format of your

program source files comply with these guidelines, you may use the `indent` program to “beautify” your source code. This program will automatically adjust the spacing and indentation of your source file as required to ensure consistent indentation. To use this program type:

```
indent -bli0 -i3 <source-file>
```

where `<source-file>` is the name of your source file.

5 Naming

Use descriptive names for variables so that use of the variables is clear to the reader. There are some exceptions to this rule. Some types of variable names are used so frequently in programming such that longer variable names are not necessary. These types of variables include the use of `i` and `j` for loop counters, `p` for pointers, and `s` for strings.

Method and variable names should begin with lowercase letters. Use underscores or capital letters in the middle of names to increase readability of compound identifiers. For example: use `dollarsPerHour` or `dollars_per_hour` instead of `dollarsperhour`. All constants should be written in uppercase. For example: `PI` and `LITRES_PER_GALLON`.

In the words of Donald Knuth:

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. [2]

5.1 Arrays

Constants should be used when declaring the size of arrays rather than using literals. For example:

```
const int CAPACITY = 10;
int intArray [CAPACITY];
```

Using constants enables the source code to be more flexible and more easily changed as needed.

6 Equations and Parenthesis

Use explicit equations, and add parenthesis wherever necessary; do not rely on the compiler to execute the order of precedence. Thus, never write like this:

```
x = y - z * a + b;
```

Rather, be explicit like this:

```
x = ( y - ( z * a ) ) + b;
```

7 Comments

Good programs contain numerous comments. Comments help the reader of a program by describing important details or by pointing out salient features. Comments should not restate the obvious nor should comments contradict the code. Comments should not be used to clarify bad code. Bad code should be rewritten.

7.1 File Headers

All submitted programs require a header with the following information:

```
Title:          Title of the program
Name:           Your name here please
Date:           The date
Description:    What is the function of this program?
```

7.2 Function Headers

All functions or methods should include a header summarizing the details of the function along with a list of all input and output parameters. For example:

```
/* FUNCTION int f(int a, int b)
   Description: This function takes 2 integer
               numbers and returns their sum.
   INPUT PARAMETERS: Two Integers (A and B)
   OUTPUT PARAMETERS: One Integer (Sum)
*/
int f(int a, int b)
{
    return (a+b);
}
```

7.3 Variable Declarations

Variable names should normally be chosen so as to self-document the purpose of a variable. However, if the variable name is not sufficient to communicate the purpose of the variable, then add a comment indicating the purpose of each variable.

8 Object-Orientated Programs

For object-orientated programs, classes should also include a comment header at the top. Each method should also be preceded by a comment. The order for a class declaration should be as follows:

- Class comments
- Class name
- Field declarations
- Constructor(s)
- Methods

All Class names should begin with a Capital letter. Method and field names should begin with a lowercase letter. Fields should normally be private and any changes to fields should be handled by special methods. All fields should be properly initialized in the constructor(s).

9 Python Programming

Whitespace is significant in Python programs, and it is recommended that you use 4 spaces per indentation level. Never mix tabs and spaces. It is also recommended to use spaces around arithmetic operators and comparison operators for clarity. Write docstrings for all public modules, functions, classes, and methods. Constants should be written in all capital letters with underscores separating words. Examples include `MAX_OVERFLOW` and `TOTAL`.

10 PHP Programming

Web script programming has become more common and one of the popular languages that has caught on is PHP. When programming in PHP, there are several recommended guidelines that you should follow.

One style guideline deals with strings that contain variables. These variables can be enclosed with braces when using PHP as follows:

```
$message = "There are {$num_students} students.";
```

PHP code may also include many HTML tags. When using HTML tags, use single quotes around HTML tag attributes and SQL query items, and double quotes in PHP code. For example:

```
<td width='450'>  
$sql = "UPDATE courses SET prof = '{$prof}' WHERE dept = '{CS}';"
```

As with other languages, use uppercase for constants. It is also recommended to use uppercase for reserved SQL words and lowercase for table and column names.

Also, the formatting of PHP code is similar to C-code including increasing the indentation with each opening curly brace and decreasing the indent when a closing tag is reached.

11 XHTML and CSS Format Guidelines

Your XHTML markup files and CSS files should use proper formatting as well in order to make them easier to read. Use indenting to show the start and end of different block sections.

Make sure you're using the "alt" attribute in your image tags. The alt attribute provides alternative text that is displayed if your image is unavailable. This is also helpful to improve the accessibility of your web pages.

Specify the DOCTYPE (DTD) at the top of your page and ensure that all your webpages pass the W3C validator service. Include a link to the validator service at the bottom of your webpage by including the following links and icons:

```
<a href="http://validator.w3.org/check/referer">
  
</a>
<a href="http://jigsaw.w3.org/css-validator/check/referer">
  
</a>
```

12 SQL Statements

Structured Query Language or SQL is a language used to interact with a relational database system and is maintained as an ISO standard.

When programming using SQL ensure that the SQL keywords in each SQL statement appear in uppercase.

13 Summary

Good programming practice emphasizes *simplicity*, *clarity*, and *generality*. Good programming skills are guided by common sense and developed with experience. Using good programming style guidelines will help improve the readability of your code and make it easier to understand and debug.

References

- [1] Brian W. Kernighan and Rob Pike, *The Practice of Programming*, Addison-Wesley, 1999.
- [2] Donald Knuth, *Literate Programming*, CSLI Lecture Notes, 1992.