**PEACE**: **P**arallel **E**nvironment for **A**ssembly and **C**lustering of Gene **E**xpressions

A cDNA Clustering Tool

Programming Style



Copyright (c) Miami University, Oxford, OHIO.

All rights reserved.

# Copyright & License

PEACE is free software: you can redistribute it and/or modify it undergplv3-127x51.png the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

PEACE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with PEACE. If not, see <<http://www.gnu.org/licenses/>>.

Miami University makes no representations or warranties about the suitability of the software, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. Miami University shall not be liable for any damages suffered by licensee as a result of using, result of using, modifying or distributing this software or its derivatives.

By using or copying this Software, Licensee agrees to abide by the intellectual property laws, and all other applicable laws of the U.S., and the terms of GNU General Public License (version 3).

# Contents

[Copyright & License 2](#_Toc252385928)

[Contents 3](#_Toc252385929)

[Introduction 4](#_Toc252385930)

[Conventions 5](#_Toc252385931)

# Introduction

With several different programmers from diverse backgrounds actively contributing code to the PEACE project in several different languages, I have noticed a conglomeration of divergent coding conventions and styles. In themselves, the styles are fine. However, I propose that we adopt some standards or conventions to streamline our sources properly.

When a project tries to adhere to common standards a few good things happen:

* Programmers can go into any code and figure out what's going on.
* New people can get up to speed quickly.
* People new to C++ / Java are spared the need to develop a personal style and defend it to the death.
* People new to C++ / Java are spared making the same mistakes over and over again.
* People make fewer mistakes in consistent environments.
* Programmers have a common enemy ☺

It helps if the standard annoys everyone in some way so everyone feels they are on the same playing field. The proposal here is from my general experiences and has evolved over many projects, from several different companies, and literally a total of many weeks spent debating over them. It is no particular person's style -- but feel free to call it a RAOdy (pronounced rowdy) style and is certainly open to amendments.

|  |  |
| --- | --- |
|  | A quote from <http://www.possibility.com/Cpp/CppCodingStandard.html>: The experience of many projects leads to the conclusion that using coding standards makes the project goes smoother. Are standards necessary for success? Of course not. But they help, and we need all the help we can get! Be honest, most arguments against a particular standard come from the ego. Few decisions in a reasonable standard really can be said to be technically deficient, just matters of taste. So be flexible, control the ego a bit, and remember any project is fundamentally a team effort. |

# Conventions

I would like to propose the following conventions for both C++ and Java. I am not listing some of the standard conventions that all programmers generally adhere. I am only listing those that I have observed to need attention:

1. Use meaningful class and variable names. Class names are preferably nouns while method names are preferably verbs.
2. Define and restrict variables to the scope in which they are used. Tighter scopes are better for readability and troubleshooting. Needless to add it greatly aids performance.
3. Avoid hanging open braces unless you are defining a separate inner scope. In almost all cases, open "{" braces should appear on the same line as the definition of the scope they begin, including: classes/interfaces, structs, methods, if-else statements, loops, etc. Here are a few examples of good style:

|  |  |  |
| --- | --- | --- |
| public class JavaCode {  public void doIt() {  for(int i = 0; (i < 10); i++) {  if (i % 2 == 0) {  // Even  } else {  // Odd  }  }  }  } |  | class CppCode {  public:  void doIt();  }; |
|  |
|  | void  CppCode::doIt() {  for(int i = 0; (i < 10); i++) {  if (i % 2 == 0) {  // Even  } else {  // Odd  }  }  } |

1. All scopes must be visually (it could be spaces or a tab) indented by 4 spaces.
2. In C++ prefer references over pointers as suggested below:

|  |  |  |
| --- | --- | --- |
| void badMethod(std::string\* ptr); |  | void goodMethod(std::string& ptr); |

1. In C++ methods and parameters must be defined const if they cannot be modified or if methods do not have side effects.

|  |  |  |
| --- | --- | --- |
| int badMethod(int i, int j) {  return i \* this.k / j;  } |  | int goodMethod(**const** int i, **const** int j) **const** {  return i \* k / j;  } |