CSci 515 Coding Style Guidelines

Spring 2015

Overview

Learning correct style and formatting standards for written programs are an important part of become a good computer scientist and software engineer. Style and formatting guidelines are to programming as grammer, punctuation and formatting rules are to written languages. They help you to communicate the intention and meaning of your programs to both other readers (other programmers and software engineers) as well as your future self trying to read and understand code you may have written sometime in the distant past.

In general, you must follow good programming practice tips and specifications given in our Deitel textbook for this course. Failure to follow follow a good programming formatting practice as specificed in our textbook may cost you points on your lab and programming assignments. In general, if your follow these guidelines and format your code to look and conform to the style shown in your textbook, then you will not have points removed for style issues. However, the following are additional issues and guidelines that you must also follow, that may be slightly different or that we will emphasize from the Deital style formatting and guidelines.

File header documentation

All assignment files you create for this course must have a file document header included at the beginning of the source code file. We will use a pseudo docoxygen format, and ask you to provide the following information at the beginning of each of your source files submitted for assignment for this course:

```
/**
 * @author Joe Student
 * @cwid   123 45 678
 * @assg   Assignment #1
 * @date   January 20, 2015
 *
 * @description Provide a short description of the problem and the approach you
 *              took to solving the problem.
 */
```

Function header documentation

In a similar manner, all functions you write for this course should include a function header declaration and documentation using (pseudo) doxoxygen formatted comments. Almost all programming shops require that all classes and functions (including member functions) be documented in this manner where they are declared. The purpose is mainly to document the input parameters to the function, and any output results the function returns. Use the following format to document all functions you write for assignment for this course:

```
/** Check for process arrivals
 * Check the process table information to find processes that are
 * arriving.  Place all arriving processes at the end of the round
 * robin scheduling queue.
 *
 * @param currentTime An int value, the current time step of the
 *              simulation, check for processes in table arriving at
 *              this time.
 * @param processTable A pointer to a ProcessTable struct, a list of
 *              all the process information for processes we are
 *              simulating, including their arrival times.
 * @param rrQueue A STL list holding pointers to Process struct items.
 *              This is our simulated round robin queue. This parameter
 *              is passed as a reference parameter.  We will add
 *              any arriving processes to the end of this queue as a
 *              side effect of calling this function, since it is a
 *              reference parameter.
 * @returns bool True if a process arrived, false otherwise
 */
bool checkProcessArrivals(int currentTime, ProcessTable* processTable,
                          list<Process*>& rrQueue)
{
  for (int pid=0; pid < processTable->numProcesses; pid++)
  {
    Process* p = processTable->process[pid];
    if (p->arrivalTime == currentTime)
    {
      DEBUG(cout << "   Process arrives: "
                 << p->processName << endl;)
      rrQueue.push_back(p);
      return True
    }
  }

  return False
}
```

Indentation

Make sure you pay special attention to the Deital style guidelines regarding proper indentation of course code. For this course we require you to use spaces (no embedded tabs) for indentation. You are required to use 2 spaces as the unit of indentation for all code/block levels for code submitted for this class. DO NOT use hard coded (embdeded) tabs in your submitted programs.

Function and Variable Names

All functions and variables should follow camelCaseNameing convention for your programs for this course. When creating a user defined type, like a class or a struct, you should use CamelCaseNameing with the initial letter capitalized (this differentiates functions and variables from classes and user defined types). Constants, in enumerated types or otherwise, should use ALL$_{\text{CAPSUNDERSCORE}}$ convention for nameing. Make sure you choose meaningful variable, function, class and constant names for your programs, as meaningful well chosen names make your programs more readable and reduce the need for extensive comments. See Deitel guidelines for more hints on how to choose meaningful variable names for your programs.

Brace Placement for Control Blocks

For this course you are required to place (most) all braces defining a control block (like a for loop or if statement) on a separate line by themselves, indented appropriately. For example, this function has 2 levels of indentation, and all levels are consistently indented and all opening/closing braces appear on their own line for readability:

```
/** Display a matrix
 * A helper function for debugging.  Display a state matrix to
 * standard output
 *
 * @param rows The number of rows in the matrix
 * @param cols The number of cols in the matrix
 * @param m A 2 dimensional array of rows x cols integers
 */
void displayMatrix(int rows, int cols, int v[MAX_PROCESSES][MAX_RESOURCES])
{
  int r, c;

  // display column headers
  cout << "   "; // extra space over for row labels
  for (c = 0; c < cols; c++)
  {
    cout << "R" << c << " ";
  }
  cout << endl;

  // now display data in matrix
  for (r = 0; r < rows; r++)
  {
    cout << "P" << r << " ";
    for (c = 0; c < cols; c++)
    {
      cout << setw(2) << v[r][c] << " ";
    }
    cout << endl;
  }
  cout << endl;
}
```

Whitespace

Pay special attention to the programming guidelines regarding space within statements and between blocks of code and functions. For example, always put a single space before and after all binary operators (like +, -, «, etc.). Put a single space after commas (,) and semicolons (;) separating lists of parameters or declarations in functions/control blocks. But a single blank line before and after a control block inside of a function. For this course, you should place 2 blank lines between the end of each function and the beginning of the next function documentation. In general, pay attention to the formatting of whitespace in our Deitel textbook and follow the conventions shown there.