Assignment 3 jack_pretty()

jack_pretty() function

Description

You must complete the implementation of the **jack_pretty()** function in the file **pretty.cpp**. The function **jack_pretty()** is used by a program **jpretty** that reads an XML representation of an abstract syntax tree of a Jack class from standard input and writes the equivalent Jack source code to standard formatted to a specific coding standard. The **jack_pretty()** function uses the functions described in **csdocument.h** to traverse an abstract syntax tree and the functions in **iobuffer.h** to write Jack source code to standard output. The precompiled main program is responsible for passing the abstract syntax tree to the **jack_pretty()** function. All output must be written to the iostream **buffer** instead of **cout**.

Compiling and Running jpretty

When the **Makefile** attempts to compile the program **jpretty**, it will use the file **pretty.cpp**, any other **.cpp** files it can find whose names start **pretty-** and any **.cpp** files it can find whose names start with **shared-**. For example, if we have our own class **abc** that we want to use when implementing **jack_pretty()** and our own class **xyz** that we want to use with all of our functions, we would name the extra files, **pretty-abc.cpp** and **shared-xyz.cpp** respectively with matching **pretty-abc.h** and **shared-xyz.h** include files.
The program can be compiled using the command:
% make pretty
The suite of provided tests can be run using the command:
% make test-pretty
The test scripts do not show the program outputs, just passed or failed, but they do show you the commands being used to run each test. You can cut-paste these commands if you want to run a particular test yourself and see all of the output.
Note: Do **not** modify the provided **Makefile** or the sub-directory **includes** or the sub-directory **lib**. These will be replaced during testing by the web submission system.

jack_pretty()

Not every program is laid out to suit individual taste and some programming styles may be difficult for others to read. The purpose of the **jpretty** program is to take an abstract syntax tree for a Jack class and output the equivalent Jack source code using a consistent set of coding standard rules. The specific rules that the **jpretty** program must implement are as follows:

- The indentation level starts at 0.
- When a line is output, it must start with 4 spaces for each level of indentation.
- Left curly brackets "**{**" must be on their own line and the level of indentation is incremented after the line is output.

- Right curly brackets "**}**" must be on their own line and the level of indentation count is decremented before the line is output
- If a subroutine contains local variable declarations, there must be an empty line after the last variable declaration.
- All static, field and local variable declarations must be displayed one variable per line, in their order in the abstract syntax tree.
- If a class contains subroutine declarations and either static or field variable declarations, there must be an empty line after the last variable declaration.
- All subroutine declarations in a class, except the last, must be followed by an empty line.
- Every if statement and while statement must be followed by a blank line unless they are immediately followed by "**}**".
- Unless otherwise noted, all tokens within a line must be separated by a single space character.
- The must be no whitespace characters after the last token on a line.
- Commas "**,**" and dots "**.**" do not have any white space before or after them.
- Left round brackets "**(**" do not have any white space before them unless the previous token is **if**, **while**, **=**, or an infix operator.
- Left round brackets "**(**" do not have any white space after them.
- Right round brackets "**)**" do not have any white space before them.
- Unary operators do not have whitespace after them.
- Unary operators do not have whitespace before them unless the previous token is **=**, or an infix operator.
- Semi-colons ";" always terminate a line.
- A new line of output can only start if required by a previous rule.

The example outputs for the **jpretty** program are named ***.Pjack**.

Notes:

- All output must be written using the functions in **iobuffer.h**, remember to call output_buffer().
- No error messages may be output.
- If an error occurs the **jpretty** program must immediately call exit(0) and produce no output.
- During testing you may write error messages and other log messages to **cerr**. These must be removed before you submit your work.