Assignment 3 jack_codegen()

jack_codegen()

Description

You must complete the implementation of the **jcodegen** program in the file **codegen.cpp**.

The program reads an XML representation of an abstract syntax tree of a Jack class from standard input and writes a Hack Virtual Machine (VM) code implementation of the class to standard output. It uses the functions described in **j-ast.h** to traverse an abstract syntax tree and the functions in **iobuffer.h** to write VM code to standard output. The main function is responsible for calling the **jn_parse_xml()** function and passing the result to the **jack_codegen()** function. The **jn_parse_xml()** function is responsible for reading an XML representation of the abstract syntax tree from standard input. All output must be written using the appropriate **write_to_** functions described in **iobuffer.h**.

Compiling and Running jack_codegen

When the **Makefile** attempts to compile the program **jcodegen**, it will use the file **codegen.cpp**, any other **.cpp** files it can find whose names start **codegen-** 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 **jcodegen** and our own class **xyz** that we want to use with all of our programs, we would name the extra files, **codegen-abc.cpp** and **shared-xyz.cpp** respectively with matching **codegen-abc.h** and **shared-xyz.h** include files.

The program can be compiled using the command:

```
% make jcodegen
```

The suite of provided tests can be run using the command:

```
% make test-jcodegen
```

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 copy-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-directories **bin**, **includes** or **lib**. These will be replaced during testing by the web submission system.

jack_codegen()

The final stage of writing a compiler is to generate a new representation of the program in the target language using the information discovered during parsing. The purpose of the **jack_codegen()** function is to take the output of the **jack_parser()** function and translate it into the equivalent VM language. Since Chapter 11 gave many translation examples and you have example outputs, we don't provide specific guidelines on how to develop the code generation features of the **jack_codegen()** function.

We strongly suggest that you test your function with small examples first so that you can independently test the functionality you are implementing. You can also construct small example Jack programs and see what VM code the **JackCompiler** produces. This is a good way to work out how best to generate code for specific cases. In the example outputs you have been given, the files with names ending **.Cvm** contain VM code.

Notes:

- All output must be written using the functions in **iobuffer.h**, remember to call print_output().
- All output must be written with one VM command per line.
- VM labels are local to their function, you can reuse the same name in more than one function.
- Use the label names **WHILE_EXP** and **WHILE_END** for while loops. Append a unique number for each while loop in a function starting from 0. Restart the count in each new VM function.
- Use the label names **IF_TRUE**, **IF_FALSE** and **IF_END** for if statements. Append a unique number for each if statement in a function starting from 0. Restart the count in each new VM function.
- String literals must be created using the **String.new()** constructor and then adding each character using the **String.appendChar()**method one at a time.
- If an error occurs the program must immediately call exit(0) and have not produced any output.
- During testing you may output error messages and other log messages using the functions in **iobuffer.h**.