Assignment 3 jack_optimiser_r()

jack_optimiser_r()

Description

You must complete the implementation of the **jopt-r** program in the file **optimiser-r.cpp**.

The program reads an XML representation of an abstract syntax tree of a Jack class from standard input, using **jn_parse_xml()** and writes an optimised version to standard output, using **jn_print_as_xml()**. The **jack_optimiser_r()** function uses the functions described in **j-ast.h** to traverse the abstract syntax tree and construct an optimised copy that does not include any redundant code

Compiling and Running jopt-r

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

The program can be compiled using the command:

```
% make jopt-r
```

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

```
% make test-jopt-r
```

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_optimiser_r()

One advantage of having a parser produce an abstract syntax tree is that optimisations can be applied to the abstract syntax tree before code generation. The purpose of the **jack_optimiser_r()** function is to make a copy of an abstract

syntax tree and where possible eliminate redundant code. That is, if part of the abstract syntax tree represents code that can never be executed, it is eliminated.

The following cases of redundant code must be eliminated by the optimiser while performing a depth first copy of the original abstract syntax tree.

If Statements

An **ifStatement** node consists of three parts, an **expression** node for the condition, a **statements** node for the then part and a **statements** node for the else part. There will not be a second **statements** node if there is no else part.

If the **expression** node contains a single **term** node containing a **keywordConstant** node that has value **true**, the **statement** node containing the **ifStatement** node is replaced by the all of the **statement** nodes that are children of the **statements** node for the then part.

If the **expression** node contains a single **term** node containing a **keywordConstant** node that has value **false**, the **statement** node containing the **ifStatement** node is replaced by all of the **statement** nodes that are children of the **statements** node for the else part. If there is no else part the **statement** node containing the **ifStatement** node is not copied.

While Statements

A **whileStatement** node consists of two parts, an **expression** node for the condition, and a **statements** node for the loop body. If the **expression** node contains a single **term** node containing a **keywordConstant** node that has value **false**, the **statement** node containing the **whileStatement** node is not copied.

Return Statements

When a return statement is executed the current function terminates and the following code in the enclosing block of statements will never be executed. Therefore, when a **returnStatement** node is found whilst copying a **statements** node, no more of children of that **statements** node are copied.

If an **ifStatement** node has a **statements** node for the then part and the else, an both **statements** nodes contain a **returnStatement**node, the parent of the **ifStatement** node is considered to be a **returnStatement** node by its parent **statements** node.

If a **whileStatement**'s **expression** node contains a single **term** node containing a **keywordConstant** node that has value **true**, and while copying the **statements** node a **returnStatement** node is found, the parent of the **whileStatement** node is considered to be a **returnStatement** node by its parent **statements** node.

Notes:

- All output must be written using the functions in **iobuffer.h**, remember to call print_output().
- 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**.