Week 8 Practical Exam

Due 15 Sep by 10:00 **Points** 100 **Available** 15 Sep at 9:00 - 15 Sep at 11:00 about 2 hours

Weighting and Due Date

- Marks for this practical exam contribute 5% of the overall course mark.
- All marks will be awarded automatically by the web submission system.
- **Due date: 10am Tuesday 15 September 2020.**
- **Late penalties:** late submissions will receive a mark of 0.
- **Core Body of Knowledge (CBOK) Areas:** abstraction, design, hardware and software, data and information, and programming.

Exam Setup

Note: this exam assumes that you have already created directories for every assignment, workshop, project, and exam in your svn repository, as described on the Startup Files for Workshops and Assignments page.

1. If required, checkout a working copy of the **exam1-primary** directory from your svn repository.
2. Change directory to the working copy of the **exam1-primary** directory.
3. Copy the newest zip file attached below into the updates sub-directory and add it to svn.
4. Run the following command to place the exam's startup files in the correct locations:

```
% make install
```

5. svn add the **tokeniser.cpp** file to your svn repository and perform an svn commit.

```
    svn add tokeniser.cpp
    svn commit -m exam1
```

6. Make a submission to the web submission system (https://cs.adelaide.edu.au/services/websubmission) *assignment* named: ***Practical Exam 1***

Exam Description

This exam is based on [Workshop 05 - Writing a Tokeniser]().

You are required to complete the implementation of the C++ file **tokeniser.cpp** that is used to compile the program **tokeniser**. You will complete the implementation of a function, ***next_token()***, that will read text character by character using the function ***nextch(),*** and return the next recognised token in the input. The tokens that must be recognised will be shown in the output of the web submission system as you complete each of the 5 stages of the exam. The tokeniser interface is specified in the file **includes/tokeniser.h**.

Testing

The startup files have the same structure as used in the workshops and programming assignments. We have provided a main function in the file **main.cpp** that can be used to compile and test your ***next_token()*** function by simply running the command:

```
make
```

Initially the tests sub-directory contains a single test which is an empty input file that should produce an empty output file.

Each stage of the exam has a set of tokens that must be recognised including a particular example. The example will show you the token kind and spelling for a particular test input and commands you can use to create your own test input and output files for the example. If you wish, you are then able to test your work against this example on your own computer.

If you fail an example test, the test script will give you some detail of what your next_token() function is supposed to have done as well as some details of how it was tested. There are usually up to 257 separate tests run over each example. The results of up to 10 failed tests are then shown in a small table where each row shows the individual errors that were detected for each input.

Once you have successfully passed the example test for a given stage, the test script will automatically show you the next few stages. You do not need to complete a stage to move on, you just need to return the right token for the given example. You do not need to complete the stages in order.

Note: If your program passes all the tests on your computer it does not mean that your program is correct. It just means that it appears to passed a limited number of tests on your computer. If an example test passes on your computer but fails on the web submission system, you have a bug in your program that you need to fix. Remember that the order in which function arguments and operands in expressions are evaluated can differ from one C++ compiler to the next. If your program fails in this way it is incorrect.

Note: Your program may not compile on the web submission system if you used functions that were defined in include files automatically included on your computer but not on the web submission system. If your program does not compile on the web submission system, it is incorrect. If you do not remember what you changed that made your program not compile, you can use svn diff to see what you have changes were made between commits to svn.

Note: If your code causes the web submission test program to exit unexpectedly you may not see any feedback from the test program and you will receive 0 marks. Your program is incorrect and it is up to you to work out what you are doing wrong. You may find svn diff useful in this situation too.

Marks

Each stage of the exam has a set of randomly generated tests that are exhaustively tested in addition to the example you are shown. You do not get to see the generated tests but you do get some feedback on why they may have failed. A count is kept of how many tests called nextch() the correct number of times, how many tests returned the correct token kind and how many tests returned the correct token spelling. The mark for a particular stage is simply the percentage of all the tests that were passed.

The maximum mark available is 100. The marks awarded for each stage progressively decrease from 20 to 12.

Frequent Web Submissions

Remember to make frequent submissions to the web submission system

- a web submission is the only way to discover the next stage of the exam,
- late submissions receive a mark of 0.

Startup Files

The following zip file must be placed in the updates sub-directory and added to svn. When make is run, the zip file in the updates directory is used to update the startup files. Any files you are required to edit will not be updated but, a copy of the latest version of those files will be placed in the sub-directory originals.

- **exam1p-20200831-155825.zip**