Assignment: Intro to Template MetaProgramming

If your submission generates the correct output by breaking one or more constraints in the Requirements then don't submit to the auto-grader!!! What happens if you decide to ignore this warning? A human grader will regrade your submission to zero points. Second, subverting assessments by passing something off as something else is a violation of the academic integrity policy and appropriate remedies as cited in that policy will be enforced.

Learning Outcomes

This assignment will provide you with the knowledge and practice required to develop and implement software involving:

- Practice working with variadic templates and template parameter packs.
- Practice generating sequence types with C++17 and C++11 approaches.
- Practice use of keywords such as `typename`, `using`, `decltype`, `auto`, `if constexpr`.

Requirements

- You are given a header file `index_sequence.h` with a definition of a sequence type. Study its implementation, and learn how to write types like this one.

  You are also given translation unit `main.cpp` that contains the test code. It can be compiled:

  - With pre-processor symbol `CPP11` to include `cpp11.h` header for C++11, or
  - ***Without*** pre-processor symbol `CPP11` to include `cpp17.h` header for C++17.

  Your task is to implement both headers so that each program can generate a **sequence type** consisting of powers of 2: from 2^{N} down to 2^0, where `N` is a pre-processor symbol specified in the compilation command line. For example, if the programs are compiled and run using the following commands, they must print the output below:

```
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++11 -o seq11
main.cpp -DN=6 -DCPP11
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o seq17
main.cpp -DN=6
./seq11 >  actual-output.txt
./seq17 >> actual-output.txt
```

  For `N` equal to 6, the output should be the following (take note that 64 is 2^6):

```
[C++11]
  64
  32
  16
  8
  4
  2
  1

[C++17]
```

```
  64
  32
  16
  8
  4
  2
  1
```

While implementing the code you must observe the following constraints:

C++11

- Must be implemented as `cpp11.h`.
- Must conform to the C++11 standard.
- Must be based on a single base class template and one partial specialization.
- Keyword `using` is permitted no more than 3 times.

C++17

- Must be implemented as `cpp17.h`.
- Must conform to the C++17 standard.
- Must use a single function template that includes the `if constexpr` statement.
- Must not contain keywords:
  - `class`
  - `struct`
  - `typedef`
- Keyword `using` is permitted only once.
- Must not contain multiplication (`*`) or addition (`+`) operators; consider bitwise operators.

Both implementations will be tested for various values of `N`; a complete build script used for grading will be the following:

```
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++11 -o seq11
main.cpp -DN=0 -DCPP11
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o seq17
main.cpp -DN=0

echo "#### N=0 #### " > actual-output.txt

./seq11 >> actual-output.txt
./seq17 >> actual-output.txt

g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++11 -o seq11
main.cpp -DN=6 -DCPP11
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o seq17
main.cpp -DN=6

echo "#### N=6 #### " >> actual-output.txt

./seq11 >> actual-output.txt
./seq17 >> actual-output.txt

g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++11 -o seq11
main.cpp -DN=63 -DCPP11
```

```
g++ -Wall -Werror -Wextra -Wconversion -pedantic -std=c++17 -o seq17
main.cpp -DN=63

echo "#### N=63 #### " >> actual-output.txt

./seq11 >> actual-output.txt
./seq17 >> actual-output.txt
```

Make sure that the resulting file `actual-output.txt` matches the provided file *expected-output.txt*.

Submission Details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions. **Valgrind** and documentation using **Doxygen** is required.

Submission file(s)

You are required to submit `cpp11.h` and `cpp17.h`. Without any comments you can expect each file to contain about 25 lines of code.

Automatic evaluation

1. In the course web page, click on the appropriate submission page to submit the two files.
2. Please read the following rubrics to maximize your grade:
   - Your submission will receive an F grade if your submission doesn't compile with the full suite of `g++` options.
   - F grade if your submission doesn't link to create an executable.
   - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. $A+$ grade if output of function matches correct output of auto grader.
   - A maximum of D grade if Valgrind detects even a single memory leak or error. A teaching assistance will check you submission for such errors.
   - A deduction of one letter grade for missing file-level documentation in submission files. A deduction of one letter grade for each missing function definition documentation block in each submission file. Your submission must have **one** file-level documentation block and function-level documentation blocks for ***every function you're defining***. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing or incomplete or copy-pasted (with irrelevant information from some previous assessment) block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and one documentation block is missing, your grade will be later reduced from $A+$ to $B+$. Another example: if the automatic grade gave your submission a C grade and the two documentation blocks are missing, your grade will be later reduced from C to E.