AC21008: Multi-paradigm programming and data structures
Assignment 3 - Open Addressed Hash Tables in C++

This is an individual task, i.e. not team-based.
Due date: Wednesday 20th Nov, 10:00pm

Part 1 - Open Addressed Hash Table

Your task is to implement a templated open addressed hash table in C++. To get you started, skeleton `HashTable.h` and `HashNode.h` files are provided. You must implement the methods declared in the skeleton classes and you may implement additional methods.

Your code must handle failures (such as inability to allocate memory, duplicate hash keys, out of range, etc.) gracefully by throwing an appropriate exception.

Part 2 - N-gram Frequency Table

Your task is to implement a tool `ngram` written in C++ that outputs the most frequent n-grams occurring in an input file. You must use your hash table from Part 1 in order to count the number of occurrences of all n-grams in an input file. Your hash table must store the n-grams as keys and their respective count as the associated data.

An n-gram is a sequence of n items. Items can be numbers, words, characters, etc. For example, the sentence "Hello world." contains the character trigrams (3-grams) "Hel", "ell", "llo", "lo ", "o w", etc. The last character trigram in that sentence is "ld."

The same sentence contains the words "Hello" and "world". These two words are the two word unigrams (1-grams) in the sentence and "Hello world" is the only word bigram (2-gram) in the sentence.

What to implement

The **minimum functionality** that your `ngram` tool must provide is:

- Calling `./ngram` processes the file `inputfile.txt` if it exists. If the file does not exist, the program must exit with a non-zero return value and an error message.
- If `inputfile.txt` exists, `./ngram` outputs a sorted list that contains the 10 most frequent **character trigrams** in `inputfile.txt` and their frequency as a percentage with exactly 2 decimal places. The output should be to `stdout` (e.g., using `cout`) not to a file. Each frequency and n-gram must be listed on a separate line in the format "12.34:abc". Note that only the ':' character is to separate the decimal digit and n-gram.

Optionally, `ngram` may take up to the following four command line parameters:

- Parameter 1: `filename`, where `filename` is used instead of `inputfile.txt`
- Parameter 2: `N`, where `N` is the the number of items that form an N-gram, instead of 3.
- Parameter 3: `K`, where `K` is the number of top most frequent n-grams to output, instead of 10.
- Parameter 4: One of the keywords `word`, `char`, `decimal` indicating whether the n-gram items are words, characters, or decimal digits.

What you should submit

You must submit the files `HashTable.h`, `HashNode.h`, and `ngram.cpp`. We will only look at those three files and *no* other files may be submitted.

These three files should be submitted as a single zip file. From the Linux command line you should run the following command to create this zip file:

```
zip assignment3.zip HashTable.h HashNode.h ngram.cpp
```

The zip file should then be uploaded to the submission folder on MyDundee.

Make sure to pay careful attention to the submission instructions and ensure that you submit the **edited** versions of the HashTable files rather than the skeleton versions provided. **Failure to follow these instructions may impact the grade you receive**.

Marking scheme

The letter mark awarded for this assignment is determined as follows:

BF: Any of the following:

- No `HashTable.h` or no `HashNode.h` file submitted.
- You have not implemented **your own hash table** based on the provided skeleton `HashTable.h` and `HashNode.h` files.
- The implementation is not an extension of the provided skeletal classes.
- The implementation is a different data structure instead of (a part of) an open addressed hash table.

CF: HashTable.h and HashNode.h are submitted, but one or more the following issues are found:

- The tester file `test0.cpp` cannot be compiled or does not result in an executable program with "`g++ -Wall -Wextra -pedantic test0.cpp`"
- The compiled tester file `test0.cpp` crashes.

M3: The code crashes or fails on `test1.cpp`
M2: The code crashes or fails on `test2.cpp`
M1: The code crashes or fails on `test3.cpp`

If all of the preceding tests succeed you will earn `D3` or higher. Additional tests (not provided) will be run on your submitted files (including `ngram.cpp`). For each test that your submitted files pass, you will be awarded the next higher letter mark. For example, if your files do not pass any additional tests, you will get a `D3`. If your files pass 3 additional tests you will earn a `C3`. If your files pass 13 tests, you will earn an `A1`.

The additional test files are not provided, but they concern the following aspects of your code:

- 1 test: Fails if there are compiler warnings when your HashTable code is compiled with the switches "`-Wall -Wextra -pedantic`" on any of the files `test0.cpp`, through `test3.cpp`.
- 2 tests concern special cases, exceptions, and error handling (out of memory conditions, hash collisions) of your `HashTable.h` and `HashNode.h` implementations.
- 1 test concerns the quality of your hash function (in terms of uniformity of its output, preventing collisions) in `HashTable.h`
- 1 test concerns the efficiency of your code in `HashTable.h`.

To earn marks for the following tests concerning the `ngram.cpp` file, you must have a functioning `HashTable.h` and `HashNode.h` implementation as determined by the tests up to the letter mark `D3`. Additionally, your `ngram.cpp` program must provide the correct (and not hard-coded) output for the sentence "This is ngram."

- 1 test checks for compiler warnings when compiling ngram.cpp with "`-Wall -Wextra -pedantic`".
- 1 test concerns the basic functionality of `ngram.cpp`
- 2 tests: Special cases, exceptions, and error handling (out of memory conditions, hash collisions) of your `ngram.cpp` implementation.
- 1 test concerns the efficiency of your `ngram.cpp` code.
- 4 tests concern the presence and functionality of the 4 optional Parameter features of `ngram.cpp` listed above.