```cpp
//Neccessary libraries, where cassert for assert() and cctype for isalpha() and toupper()
#include "KeyProvider.h"
#include <cctype>
#include <cassert>

/*
* The function preprocessString will take in a string and returns a new string
* (Exceptions for Non-alphabetic characters) that contains the word in uppercase
* @param aString (const std::string&): The given string
* @return std::string: New string in uppercase and no non-alphabetic characters
*/
std::string KeyProvider::preprocessString(const std::string& aString) noexcept
{
    //Declare a string for storing characters
    std::string result;
    //Loop through each characters on the given string
    for (const char c : aString)
    {
        if (std::isalpha(c))
            //If that character is alphabetic -> push_back that characters
            //(In uppercase) to the result string
            result.push_back(std::toupper(c));
    }
    //Return the string
    return result;
}

/*
* Constructor of KeyProvider class, which will also be used for comptuing keyword sequnce
* based on the input text. In addition, this function must guarantee that the size of fKeys
* must match with the inputted phrase
* @param aKeyword (const std::string&): The keyword
* @param aSource (const std::string&): The source phrase
* @return void
*/
KeyProvider::KeyProvider(const std::string& aKeyword, const std::string& aSource) noexcept
    : fIndex(0)
{
    //Uppercase the keyword and the source phrase
    std::string processedKeyword = preprocessString(aKeyword);
    std::string processedSource = preprocessString(aSource);
```

```cpp
    //Allocate more spaces in fKeys based on the size of processed source phrase
    fKeys.reserve(processedSource.size());

    //Get the size of processed keyword and source phrase
    size_t keywordLength = processedKeyword.size();
    size_t sourceLength = processedSource.size();

    //Here, 'i' will be the iterator for the source phrase
    //While 'j' will be the iterator for the keyword phrase
    size_t j = 0;
    for (size_t i = 0; i < sourceLength; i++)
    {
        //Loop through sourceLength.size() - 1 times, append the keyword in the length
        //of source phrase. For example, source: Me is Simon, keyword: abc -> fKeys: ABCABCABC
        if (j >= keywordLength)
            j = 0;
        fKeys.push_back(processedKeyword[j]);
        j++;
    }

    // Ensures fKeys is properly sized
    assert(fKeys.size() == sourceLength);
}

/*
* Get the keyword character where the iterator is pointed on
* @param None
* @return char: The keyword character where the iterator is pointed on
*/
char KeyProvider::operator*() const noexcept
{
    assert(fIndex < fKeys.size()); // Prevent out-of-bounds access
    // The character at the fIndex
    return fKeys.at(fIndex);
}

/*
* Advance the iterator to one index and return the updated iterator
* @param None
* @return KeyProvider&: the updated iterator
*/
KeyProvider& KeyProvider::operator++() noexcept
{
    //Advance one index
    ++fIndex;
    //Return the updated iterator
```

```
 88      return *this;
 89  }
 90
 91  /*
 92  * Advance the iterator to one index and return the old iterator
 93  * @param None
 94  * @return KeyProvider&: the updated iterator
 95  */
 96  KeyProvider KeyProvider::operator++(int) noexcept
 97  {
 98      //Make a copy of old iterator
 99      KeyProvider temp = *this;
100      //Using operator++() and return the old iterator
101      ++(*this);
102      return temp;
103  }
104
105  /*
106  * Return the equality of the underlying collection and the position of
       this
107  * object and other KeyProvider object
108  * @param aOther (const KeyProvider&): Other KeyProvider object
109  * @return bool: The equality between objects
110  */
111  bool KeyProvider::operator==(const KeyProvider& aOther) const noexcept
112  {
113      return fIndex == aOther.fIndex && fKeys == aOther.fKeys;
114  }
115
116  /*
117  * Return the in-equality of the underlying collection and the position of
       this
118  * object and other KeyProvider object
119  * @param aOther (const KeyProvider&): Other KeyProvider object
120  * @return bool: The in-equality between objects
121  */
122  bool KeyProvider::operator!=(const KeyProvider& aOther) const noexcept
123  {
124      return !(*this == aOther);
125  }
126
127  /*
128  * Return a copy of 'this' iterator object positioned in the first index of
       the string
129  * @param None
130  * @return KeyProvider: A copy of 'this' iterator at the first index
131  */
132  KeyProvider KeyProvider::begin() const noexcept
133  {
```

```
134     //Get a copy and set the index as 0
135     KeyProvider temp = *this;
136     temp.fIndex = 0;
137     //Return the copy
138     return temp;
139 }
140 
141 /*
142 * Return a copy of 'this' iterator object positioned in the last index of the string
143 * @param None
144 * @return KeyProvider: A copy of 'this' iterator at the last index
145 */
146 KeyProvider KeyProvider::end() const noexcept
147 {
148     //Get a copy and set the index as the size of the keyword
149     KeyProvider temp = *this;
150     temp.fIndex = fKeys.size();
151     //Return the copy
152     return temp;
153 }
154 
```