The Human Genome Project was a landmark 13 year project to map the structure of human DNA. The project made it possible for many advances in medicine. It is also useful for other fields, such as agriculture and anthropology.

The basic process used was to extract the DNA from the cells of a person. Unfortunately, the whole DNA (about 3 billion nucleotides long) can’t be extracted in one piece. So, the DNA is copied many times, and then these copies are broken up into a very large number of small pieces (each about 100 to 500 nucleotides long). The small pieces are processed in a sequencer machine capable of detecting the strand of nucleotides. The result of processing one strand then is a sequence of letters describing the type and order of nucleotides in the strand. The strands only contain four nucleotide bases: adenine $A$, guanine $G$, cytosine $C$, and thymine $T$. So, strands are represented by a text string containing the letters AGCT.

In order to recreate the order of the full DNA strand, these pieces must be stitched together in the correct order. To do this, the ends of two incomplete DNA strands are compared to see if they match when overlapped. If so, it is possible that these DNA strands are two pieces of a larger whole. The strands are matched and merged repeatedly until an approximation of the full DNA string is reproduced.

Here is an example of two DNA strands that match when overlapped:

| Strand 1: ACGGACATAGTCATT |
| Strand 2: CATAGTCATTTCATG |
| Combined: ACGGACATAGTCATTTCATG |

**Assignment**

In this step of the assignment, you will create a class that represents the DNA strand. It will be capable of storing any size of string (using the heap), checking for overlap with other strands, and merging two strands together. You will also create a program that tests your class.

**Requirements**

- The class must be declared in `Strand.h` and implemented in `Strand.cpp`.
- A class `Strand`: Must use heap memory. May not use `std::string`, `std::vector`, or any other STL or third party storage containers.
- `Strand` must have a default constructor, a copy constructor, a destructor, and an assignment operator.
- `Strand` must have a constructor with this signature: `Strand(const char * src);` where `src` is a c-string with the value of the `Strand`. If `src` is the null pointer, then the `Strand` value must be set to have no bases.
- `Strand` accessor methods must not allow contents to be modified by external code.
- `Strand` must have a method `size_t size() const` that returns the number of bases stored in the strand.
- `Strand` must have no memory errors. This includes leaks, double deletes, writing to deleted memory, bad pointer access, etc. Use `valgrind` or other memory checkers to test it.
- `Strand` must overload the operators: `strand + strand` and `strand += strand`. The first results in a new `Strand` object whose value is the concatenation of the two inputs. The second results in the lhs operand’s value being modified by concatenating the rhs operand’s value.
- `Strand` must overload the operators: `strand == strand` and `strand < strand`. An ASCIIetical comparison is expected.
- `Strand` must have a method `Strand substrand(size_t i, size_t j) const` that returns a copy of the strand that starts at position $i$ and ends at position $j - 1$. It should behave much like Python slicing. If $i$ is equal to or greater than $j$, return an empty `Strand`.
- `Strand` must have a method `Strand merge(size_t i, const Strand &right) const`. This method makes a new strand that is the concatenation of a copy of the current strand object up to position $i - 1$, and all of the strand `right`.
- `Strand` must have a method `size_t overlap(const Strand &right) const`. This method returns an index into the current strand. The index is where the end of the current strand exactly matches the beginning of the right strand. For example, if the strands are `XYZABC` and `ZABQRST`, the overlap is at index `2`.
the return would be 2, because that is the index of ZABC within XYZABC. If there is no overlap between the two strands, then this method will return a value that is greater than or equal to the size of the current strand. For example, ABC and XYZ would return 3, because that is the size of the current strand, and there is no overlap.
- Strand can have more methods as you deem useful in implementing the required methods.
- A unit test program must be created. This program will have at least one function per method of the Strand class. Each function will then fully exercise at least one of the Strand methods. In making unit tests, you should check for all interesting input conditions possible. For example, the test of overlap should test cases with complete overlap, no overlap, lots of overlap, very little overlap, and medium size overlap.
- A Makefile must be provided for this assignment. At minimum, it must have a target unittest that will be sure your unit test program is built correctly, and then run valgrind on your unit test program.

Selected Examples

Show Off Your Work

To receive credit for this assignment, you must upload the source code (Strand.h, Strand.cpp and your unit test file(s)) and the Makefile to the Canvas submission system.

Additionally, the program must build and run. In addition to your unit tests, your Strand class will be tested by a grading unit test. Any incorrect performance or memory errors will be counted against the assignment score.