CMPUT 350 Assignment 1

Due: Wednesday Sept. 25, 22:00

Important: Read this text completely and start working on the assignment as soon as possible. Only then will you get an idea about how long it will take you to complete it.

If you worked on the assignment on your personal computer, copy your files to the undergraduate lab machine (e.g. uf13.cs.ualberta.ca) with scp and then test your code there as well before submitting it. Like the labs, we will be using this environment to test your code.

When done, submit your solution files on eClass:

```
README.txt CPoint.c String.h String.cpp Set.h Set.cpp hexagons.cpp
```

In this assignment you can't use C++ container types or functions from C++ libraries such as STL or Boost. However, you can use standard C libraries

Your submissions will be marked considering their correctness (syntax and semantics), documentation, efficiency, and consistent coding style. Add assert statements to check important pre/post-conditions.

Finally, make sure that you follow the course collaboration policy which is stated on the course webpage. File `README.txt` must be completed!

---

1. [13 marks] Given C++ code `Point.cpp`, your task is to relive the early C++ days in which only C compilers were available by transforming the C++ code into equivalent C code - line by line - inspired by the example in the lecture (Part 2, slide 4). Your C++ to C translator that runs in your brain is very basic - it can't remove useless code. The resulting code stored in `CPoint.c` has to compile using

```
gcc -Wall -Wextra -O -std=c99 CPoint.c
```

without errors nor warnings on the lab computers. To simplify marking, put comments that contain the original C++ lines in front of the C code implementing them.

Note: we are not asking you to write a general C++ to C transformer. Your task is to convert the given file into C code manually.

---

2. [12 marks] Implement a C++ string class in files `String.h` and `String.cpp` following the coding style and comment suggestions presented in the lecture.

Strings are immutable objects, like in Java, i.e., once created, Strings cannot be changed. This allows us to save memory when copying Strings by storing character sequences in shared objects that are reference counted. I.e., whenever a String is created, a shared object is allocated that contains the character sequence with an initial reference count of 1. Later, whenever a String gets copied or assigned, the count is incremented and the new copy just points to the shared object. When Strings are deleted, the reference count is decremented. Once the count reaches

0, the shared object is no longer in use and can be destroyed. In the shared objects, strings are stored as C-strings, i.e., 0-terminated char arrays.

You may add `private const` data members and `private` methods to the class definition, but no other members.

```
g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -std=c++17 ...
```

without generating errors nor warnings on the lab computers.

Make sure your implementation does not leak memory by testing it thoroughly creating, assigning, copying, and destroying thousands of Strings in `mainString.cpp` and running `valgrind`. You only submit `String.*` which don't contain test code!

---

3. [16 marks] Implement a memory efficient set class in files `Set.h` and `Set.cpp` following the coding style and comment suggestions presented in the lecture.

The sets considered here represent subsets of { `0..n-1` } and are implemented in terms of bit sequences stored in int arrays, where bit `i` encodes whether element `i` is in the set (`bit=1`) or not (`bit=0`).

For instance, for `n = 4`, integer 11 (base 10) = `1011` (base 2) represents the set 0,1,3 because bit 0, 1, and 3 are set to 1 (read from right to left - lower order bits are listed first). 2 is not in the set because its corresponding bit (third lowest) is 0.

In general, bit `k` indicates whether number `k` is in the set: if the `k`-th bit is 1, `k` is in the set. If the `k`-th bit is 0, `k` is not in the set.

What are the advantages of storing sets like this? High speed and low memory requirements. As an example consider sets of numbers between 0 and 31. They can be represented as a single integer value. E.g.

```
A={ 0, 1, 2, 8, 31 }      <=> 10000000'00000000'00000001'00000111 (base 2)
B={ 8, 17, 30, 31  }      <=> 11000000'00000010'00000001'00000000
---------------------
A intersect B = { 8, 31 } <=> 10000000'00000000'00000001'00000000
```

which happens to be the result of a bitwise and operation (`&` in C++) on the bit sequences for A and B. Nifty - and quite fast - as such operations only take a nanosecond or so on modern CPUs!

In this problem, bit sequences will be stored in int arrays with the goal of using as little memory as possible (1 bit per element). An `int` variable can hold a certain number of bits (use `static const int INT_BITS = sizeof(int)*8` in your code). Your code needs to allocate a large enough int array to store `n` bits (how many integers do you need exactly?), with bit `i = 1` indicating that element i is contained in the set.

Set functions `clear()`, `complement()`, `add(const Set &s)`, and `remove(const Set &s)` need to be implemented as efficiently as possible, which means that you can't afford to iterate through all set elements individually.

As illustrated above, implement above functions using bitwise integer operations `&`, `|`, and ∼ to test, set, and negate individual bits, as well as bit-shift operations `<<` and `>>`.

E.g. the following test checks whether bit i is set in integer y:

```
if ((1 << i) & y) ... ,
```

- (`b | c`) computes the union of two bit sets encoded as integers b and c,
- (`b & c`) computes the intersection, and
- ∼ b computes the complement set of b

etc.

Sets are sole owners of their data, i.e., data aren't shared when copying or assigning sets and therefore have to be allocated and copied. You may add private const data members and private methods to the class definition, but no other members.

Your code has to compile using

```
g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -std=c++17 ...
```

without generating errors nor warnings on the lab machines.

Make sure your implementation does not leak memory by testing it thoroughly. We suggest using valgrind. Also add assertions that ensure that sets are compatible and elements are within range `0..n-1`. See `mainSet.cpp` for some examples.

Note: You only submit `Set.h` and `Set.cpp` which doesn't contain test code.

Note on assert: In C and C++, assertions work as follows:

```
#include <cassert>
...
assert(i >= 0);   // if i < 0 the program will stop here with an error message
```

To make your program faster after convincing yourself that it is correct, checking assertions can be switched off by passing `-DNDEBUG` to the compiler.

---

4. [16 marks] Design, implement and test a point localization algorithm for hexagonal tessellations. See details in `hexagons.cpp`.

Note: There is an explanation question at the bottom of `hexagons.cpp`, make sure you answer that question!

Your code has to compile using

```
g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -std=c++17 ...
```

without generating errors nor warnings on the lab machines.