In this assignment, you will implement a simple C library for fixed-point arithmetic.

1. Getting started

Extract fp.zip , which contains the skeleton code for the assignment.

2. Fixed-point arithmetic

You are probably at least somewhat familiar with integer and floating-point numeric data types. Integer types are exact, but cannot represent fractions. Floating-point types can represent fractions, but are inexact, since numeric operations can introduce rounding errors. Machine-level integer and floating point numeric types are finite, since they have a fixed number of bits of data in their representations, and so only allow a finite set of possible values.

A fixed-point numeric data type is a way of representing values with a fractional part using a purely integer representation. The idea is that a fixed-point value is a fixed-length string of n digits, with the first j digits being the whole part of the representation, and the last k digits being the fractional part of the representation, with n = j + k.

For example, consider a base-10 fixed-point data type where j = 5 and k = 3. The numeric value 79.25 would be represented as 00079.250. From an implementation standpoint, the position of the decimal point does not need to be explicitly represented, since it is the same for all values that belong to the data type. (Hence the term “fixed point”.) So, we can really think of the string of digits for this value being 00079250, with the knowledge that the last three digits belong to the fractional part of the value.

The key advantage of fixed-point representations is that numeric operations can be done using purely integer arithmetic. For example, using the base-10 fixed-point type described above, the sum
79.25 + 1014.999
Could be computed by the integer operation
79250 + 1014999 = 1094249
which yields the fixed point value
1094.249

3. Binary fixed point representations

Any base can be used in a fixed-point representation, and it is natural to use a binary (base-2) representation when implementing a fixed-point data type in a computer program. Let’s say that a binary fixed-point representation uses 8 bits (base-2 “digits”) for the whole part and 8 bits for the fractional part. The base-10 value 79.25 would be represented as
1001111.01000000
Note that in the fractional part “.01000000”, the “1” digit is in the fourth’s place. More generally, in a base-2 “decimal-like” fraction, the first bit after the decimal point is the half’s place (2-1), the second bit is the fourth’s place (2-2), the third bit is the eighth’s place (2-3), etc.

4. The Fixedpoint data type

In the header file fixedpoint.h you will see the following definition:

```
typedef struct {
   // TODO: add fields
} Fixedpoint;
```

An instance of Fixedpoint is a base-2 fixed point value, in which both the whole part and fractional part of the representaion are 64 bits in size. Fixedpoint values can be negative or non-negative.

Note that the Fixedpoint type has by-value semantics. It can be copied by value, passed by value, and returned by value. The only function requiring dynamic memory allocation is fixedpoint_format_as_hex, which returns a dynamically allocated character string.

5. Your task

Your main task for this assignment is to implement the following functions:

```
Fixedpoint fixedpoint_create(uint64_t whole);
Fixedpoint fixedpoint_create2(uint64_t whole, uint64_t frac);
Fixedpoint fixedpoint_create_from_hex(const char *hex);
uint64_t fixedpoint_whole_part(Fixedpoint val);
uint64_t fixedpoint_frac_part(Fixedpoint val);
Fixedpoint fixedpoint_add(Fixedpoint left, Fixedpoint right);
Fixedpoint fixedpoint_sub(Fixedpoint left, Fixedpoint right);
Fixedpoint fixedpoint_negate(Fixedpoint val);
Fixedpoint fixedpoint_halve(Fixedpoint val);
Fixedpoint fixedpoint_double(Fixedpoint val);
int fixedpoint_compare(Fixedpoint left, Fixedpoint right);
int fixedpoint_is_zero(Fixedpoint val);
int fixedpoint_is_err(Fixedpoint val);
int fixedpoint_is_neg(Fixedpoint val);
int fixedpoint_is_overflow_neg(Fixedpoint val);
int fixedpoint_is_overflow_pos(Fixedpoint val);
int fixedpoint_is_underflow_neg(Fixedpoint val);
int fixedpoint_is_underflow_pos(Fixedpoint val);
int fixedpoint_is_valid(Fixedpoint val);
char *fixedpoint_format_as_hex(Fixedpoint val);
```

In the fixedpoint.h header file you will see a detailed comment describing the required behavior for each function. You should implement the functions in fixedpoint.cpp to implement the required behavior.

Extremely important requirement

In your implementation of the Fixedpoint data type and its functions, you may not use any primitive data type with more than 64 bits in its representation. For example, the gcc compiler has an __int128 data type with a 128 bit representation. Using this type, or any other primitive type with more than 64 bits in its

representation, is not allowed.
We recommend that you use two uint64_t values in the representation of Fixedpoint, one to represent the whole part, and one to represent the fractional part. Note that you will also need to store a "tag" that keeps track of whether the Fixedpoint value is valid/non-negative, valid/negative, an error value, an positive or negative overflow value, or a positive or negative underflow value.

Hex string representation
You will note that the fixedpoint_create_from_hex and fixedpoint_format_as_hex functions work with "hex string" representations of Fixedpoint values. Hex means "hexadecimal", or base-16. In addition to the numerals 0-9, hexadecimal uses the letters a-f (or A-F) to represent values 10 through 15. Hexadecimal notation is convenient for representing binary values because each hexadecimal "digit" represents exactly 4 bits.

6. Testing

The skeleton project comes with a source file named fixedpoint_tests.cpp. This program has some unit tests for the Fixedpoint type and its functions. The unit tests use the google test framework. If you've used unit testing frameworks such as JUnit, it should be fairly straightforward. You could add additional tests of your own.
To set up the gtest with code::blosks, you could refer to: 在Codeblocks下配置GoogleTest单元测试工具.pdf

7. Submission

Submit the revised fixedpoint.h, fixedpoint.cpp, fixedpoint_test.cpp source file with testing snap-screen.