Programming Assignment 8

In the **Programming/Programming08** folder of your class repository on GitHub, create a **main.cpp** and **util.h** file. In each file, include: **`iostream, string, cmath, stdexcept`**. In **main.cpp** also include **`"util.h"`**. The **util.h** file should have a header guard with the following content in a namespace called **dshw**:

```
const string DIGITS = "0123456789ABCDEF";
const string OCTALS[8] = {"000","001","010","011","100","101","110","111"};
const string HEXADECIMALS[16] = {"0000","0001","0010","0011","0100","0101","0110","0111",
                                 "1000","1001","1010","1011","1100","1101","1110","1111"};
```

From **Programming06**:

```
int getValue(char);
void validate(const string&, int); // Changed return type to void. Throw exception instead.
string pad(const string&, int, int);
string trim(const string&, int);
```

From **Programming07**:

```
long long toDecimal(const string&, int); // Changed return type from int to long long.
string add(const string&, const string&, int);
string onesComplement(const string&, int);
string twosComplement(const string&, int);
string subtract(const string&, const string&, int);
string toBase(long long, int); // Changed first parameter from int to long long.
```

Task 1: (2 points)

Create a function called **arithmeticShiftRight()** that takes a constant string reference parameter and two integer parameters. The string parameter represents a signed binary, octal, or hexadecimal number. The second integer parameter represents the base of the number. Return a string that represents the result of the string number arithmetically shifted to right by the number of positions indicated by the first integer parameter. Note: an arithmetic right shift prepends a sign digit to the left of the number on every shift. Ensure the string is valid for the base, and the base is either 2, 8, or 16. If the string or base is invalid, throw an exception.

Example:
```
arithmeticShiftRight("01010", 1, 2) => "00101"
arithmeticShiftRight("1001", 1, 2) => "1100"
arithmeticShiftRight("234751", 1, 8) => "023475"
arithmeticShiftRight("54627", 1, 8) => "75462"
arithmeticShiftRight("AA1A", 1, 16) => "FAA1"
arithmeticShiftRight("45A7", 1, 16) => "045A"
```

Task 2: (1 point)

Create a function called **logicalShiftLeft()** that takes a constant string reference parameter and two integer parameters. The string parameter represents a signed binary, octal, or hexadecimal number. The second integer parameter represents the base of the number. Return a string that represents the result of the string number logically shifted to left by the number of positions indicated by the first integer parameter. Note: a logical left shift appends a 0 to the right of the number on every shift. Ensure the string is valid for the base, and the base is either 2, 8, or 16. If the string or base is invalid, throw an exception.

Example:
```
logicalShiftLeft("01010", 1, 2) => "10100"
logicalShiftLeft("1001", 1, 2) => "0010"
logicalShiftLeft("234751", 1, 8) => "347510"
logicalShiftLeft("54627", 1, 8) => "46270"
logicalShiftLeft("AA1A", 1, 16) => "AA1A0"
logicalShiftLeft("45A7", 1, 16) => "5A70"
```

Task 3: (3 points)

Create a function called **toBinary()** that takes a constant string reference parameter and an integer parameter. The string parameter represents a signed octal, or hexadecimal number. The integer parameter represents the base of the string number. Return the octal or hexadecimal string number converted to a binary string number. Ensure the string is valid for the base, and the base is either 2, 8 or 16. If the string or base is invalid, throw an exception.

Example: `toBinary("2025",8) => "010000010101"`
`toBinary("954F",16) => "1001010101001111"`

Task 4: (4 points)

Create a function called **fromBinary()** that takes a constant string reference parameter and an integer parameter. The string parameter represents a signed binary number. The integer parameter represents the base to which the string parameter should be converted to. Return the octal or hexadecimal conversion. Ensure the string is valid for a binary number and the base parameter is either 8 or 16. If the string or base is invalid, throw an exception.

Example: `fromBinary("110001100111",8) => "6147"`
`fromBinary("1101011111000000",16) => "D7C0"`

Task 5: (5 points)

Create a function called **multiply()** that takes two constant string reference parameters and an integer parameter. The string parameters represent signed binary, octal or hexadecimal numbers. The integer parameter represents the base of the numbers. Return a string that represents the signed product of the two string parameters. Ensure the string parameters are valid for the base, the base is either 2, 8, or 16, and the string parameters are of the same base. If the base is not 2, convert the string numbers to binary. At the end of the program, convert the product back to the base. Tip: After making Q and M the same bit length, extend (pad) the two numbers by one.

Example: `multiply("01010","11110",2) => "111111101100"`
`multiply("23400751","73476230",8) => "7765034155130"`
`multiply("AA1A","989ACD0B",16) => "0000022B17ED7211E"`

Task 6: (5 points)

Create a function called **divide()** that takes two constant string reference parameters and an integer parameter. The string parameters represent signed binary, octal or hexadecimal numbers. The integer parameter represents the base of the numbers. Return a string that represents the signed quotient and remainder of the first string divided by the second. The return string should be in the following format: **"remainder quotient"** Ensure the string parameters are valid for the base, the base is either 2, 8, or 16, and the string parameters are of the same base. If the base is not 2, convert the string numbers to binary. At the end of the program, convert the quotient(Q) and remainder(A) back to the base indicated by the integer parameter. Tip: After making Q's bit length greater than or equal to the bit length of M, extend (pad) the two numbers by one.

Example: `divide("01010","11110",2) => "000000 111011"`
`divide("234751","734230",8) => "0016111 7777774"`
`divide("AA1A","989ACD0B",16) => "FFFFFAA1A 000000000"`
`divide("989ACD0B","AA1A",16) => "FEF49 000013425"`

NOTE: There is a space between the remainder and quotient.
Green = quotient, Red = Remainder