Programming Project 04

This assignment is worth 30 points (3% of the course grade) and must be **completed and turned in before 11:59 on Monday, February 12th**

Assignment Overview
This assignment will give you more with functions and introduce the use of the string data type.

Background
We are going to work with programs that can identify "palindromic prime" numbers in various bases. Let's take a look.

Palindrome
You all remember palindromes, right? They are sentences that read the same forwards and backwards. The same principle can be applied to numbers. A palindromic number is a number whose value is the same read forwards and backwards. In $base_{10}$: 123454321, 151, 1. In $base_2$: 101, 111000111, 0. In $base_{16}$: 123aba321, f1f, d

Different Bases
It's obvious that whether a number is palindromic depends on the base that it is expressed in. For example: 17 base10 is not, but expressed as base2, 10001, it is. 170 base10 is not, but expressed in base16, aa, it is. We need to be able to convert to different bases to understand whether a number is palindromic.

Let's remember how numbers in a different base work. Let's look at hexadecimal (base 16). Look at the number below

16^3	16^2	16^1	16^0
a	b	1	0

Each of the symbols occupies a position in the string, and that position is a power of the base we are working with. Thus, this number represents:

$$(a*16^3)+(b*16^2)+(1*16^{)1}+(0*16^0)=(10*4096)+(11*256)+16+0=43,792$$

where the character 'a'=10 and 'b'=11, two of the characters a-f, the 6 extra characters past 9 we need to represent base 16.

We cannot represent such a number as a long (or any other builtin type) as there is no way to represent the a-f characters. Thus, we often represent such a higher base number with a **string**.

Converting to another base
Imagine you have the following global constant string defined (and you probably should). This is an example of a good global value!

```
const string the_chars = "0123456789abcdef"
```

If we work with only these symbols, then we can have, at max, a base 16 number. How to convert from a long to a string in a different base?

First, the process of conversion is pretty straight forward. Start with your long `n`:

- find the remainder of dividing `n` by the `base` you are working in (2, 10 or 16 for us)
- add the character that remainder represents to the result
- divide `n` by the `base`
- repeat as long as `n` is greater than 0

How to find the correct character? Conveniently, the index of the symbol in `the_chars` is its integer equivalent. For example, the index of `'9'` is 9 in the string. The index of `'a'` is 10, of `'f'` is 15. We can convert between the long value and its character as follows:

- to find the appropriate symbol for a long `my_l`, use it as an index `the_chars[my_l]`. Thus `the_chars[10]` is `'a'`

Prime Palindrome

You already know what a prime number is. It is a number that is only divisible by itself and 1. We are going to look for numbers in various bases that are both prime and palindromes. Note that the selection of base affects whether a number is a palindrome or not.

Project Description / Specification

Warning

Again, in this project as in the last we provide *exactly* our function specifications: the function name, its return type, its arguments and each argument's type. We will often provide you with a `main` program to include in your file (or a separate main program file when we start to develop mult-file programs). Do not change the function declarations!

function: `main`**:**

We provide in the Mimir project a starter file proj04.cpp without the functions. We provide this as a test rig, the kind you should be writing eventually for yourself. When run the main test each function individually based on input. Place your function code in the file marked by a comment. Feel free to experiment and test as you like, but remember to put the main code back to its original setup when you turn the code in. The TAs will be looking to see that the main you turn in is unchanged.

function `reverse_str`:

- argument is a single string
- return is a string

returns a new string which reverses the order of the argument string.

function: `is_palindrome`**:**

- single string argument
- bool return

returns true or false, is the string a palindrome

- uses `reverse_str`

function: `long_to_base`:

- two arguments
    - long , the long we are converting

    - long , the base we are converting the first argument to
- return is string

Converts the provided long into a string, where the string represents the long converted to the provided base.

function: `is_prime`:

- Takes a single long argument
- return is bool

Return is true/false, is the provided argument prime

function: `is_pal_prime`:

- takes a single long argument
- returns one of the following 4 strings:
    - `binary-pal-prime`
    - `decimal-pal-prime`
    - `hex-pal-prime`
    - `not-pal-prime`

Uses `is_prime`, `is_palindrome` and `long_to_base` to determine if the long argument is both prime and a palindrome in one of three bases: binary, decimal or hexadecimal

Deliverables

`proj04/proj04.cpp` -- your source code solution (*remember to include your section, the date, project number and comments in this file*).

1) Be sure to use "proj04.cpp" for the file name and to have it in a directory called proj04
2) Electronically submit a copy of the file to Mimir for testing

Discussion/Hints

1. It is possible that a number could be pal_prime in multiple bases. For example, 313 is pal_prime in decimal and in binary, but not hex. The order the Mimir tests check is: binary, decimal then hex. So for the code listed, if you submitted 313 to `is_pal_prime`, the return would be `binary-pal-prime`.