ECE26400 Programming Assignment #1

This is **NOT** a programming exercise. It is a programming assignment. In this programming assignment, you will implement one function:

1. `my_strtol`, a function that is similar to the function `strtol` in standard library, `stdlib`.

The main learning goals are:

1. How to "iterate" over a string, which is an array of characters terminated with the NULL character `'\0'`.
2. How to convert a string into a number in different bases.
3. How to set an error number when the base is invalid, or when there is an overflow or underflow.
4. How to read through a manual page (for `strtol`, in this case) and design test cases to thoroughly understand a function (`strtol`, in this case).
5. How to test your own implementation.

This assignment is related to PE02. If you have a meaningful submission for PE02, your PE02 score may be improved by your PA01 score, i.e., final PE02 score = max(PE02 score, PA01 score).

1 Getting started

You should unzip on eceprog.ecn.purdue.edu the zip file `pa01_files.zip` using the following command:

```
unzip pa01_files.zip
```

The zip file `pa01_files.zip` contains a folder, named `PA01`, and three files within the folder:

1. `answer01.c`: This is the file that you would hand in after modification. It has the skeleton of the functions in it. This is the only file you have to include in the zip file to be submitted through Brightspace.
2. `answer01.h`: This is a "header" file and it contains a declaration and a brief description of the functions you will be writing for this exercise. **You should not make changes to** `answer01.h`**.**
3. `pa01.c`: You should use this file to write testing code that runs the functions in `answer01.c`, in order to ensure their correctness.

To get started, read this document in its entirety. You will be writing code in the `answer01.c` file.

You will also write code in the `pa01.c` file to test the function written in the `answer01.c` file. `pa01.c` is meant for your testing of `answer01.c`. It should not be submitted.

It is most important that you read the manual page of `strtol`. You can use the command

```
man strtol
```

to access the manual page on a linux machine. You can also use "man strtol" as the search string on an internet browser to access the manual page. Your understanding of the `strtol` function is crucial to your successful implementation of `my_strtol` in this programming assignment.

2 Function you have to write

You have some experience with strings from PE02. In a sense, this assignment is an extension of PE02. You should read the description of PE02 again.

You have to write a single function: `my_strtol`.

```
long int my_strtol(char const *nptr, char **endptr, int base)
```

The main differences between `str_to_long_int` (from PE02) and `my_strtol` are:

1. If the third parameter of `my_strtol` is 0, the conversion can be performed in base 8 (octal), base 16 (hexadecimal), or base 10 (decimal). It depends on the first or the first two characters of the input string after the optional +/- sign.

   - If the first character is `'0'` (and the second character is not `'x'` or `'X'`), the conversion should be done in base 8.
   - If the first two characters are `"0x"` or `"0X"`, the conversion should be done in base 16.
   - Otherwise, the base is 10.

   In general, you want to figure out the correct base, the location at which the conversion starts, and the location at which the conversion fails.

   In the file `pa01.c`, we use the string `" +0X044"` in the test function. In addition, you should try other strings, for example, `" +0x"`, `" 0 "`, `" -09 "`, or `" +0XG4"`, in the test function. The test function will print the converted value, the address of the location at which the conversion fails, the remaining substring (starting at the location at which the conversion fails), and the `errno`.

2. If the third parameter of `my_strtol` is 16, `my_strtol` also accepts an input string that is prefixed with `"0x"` or `"0X"`. You want to figure out the location at which the conversion starts and the location at which the conversion fails.

3. You may have to save the location (of the input string) at which the conversion fails. If the second parameter passed into `my_strtol` is not `NULL`, the address of that location (at which the conversion fails) has to be saved in `*endptr`, the location pointed to by the address in the second parameter.

   - if the string is an invalid representation for that particular base, the conversion fails at the first character. Consider for example, `" +x "`, a string with leading and trailing white spaces, the conversion is considered to have failed at the first leading white space for base 2, 3, ..., or 33.
   - For the same string, the conversion is considered to have failed at the first trailing white space for base 34, 35, or 36

   It is important that you try to call `strtol` with different input strings and bases to figure out what should be stored in `*endptr` when `endptr` is not `NULL`.

As in `str_to_long_int`, the variable `errno` should be set to `EINVAL` if the base is invalid. It should be set to `ERANGE` if the conversion of the input string exceeds the range.

The preceding string examples may not have covered all differences and the all the details. You should always refer to the manual page of `strtol` for more details and try out the `strtol` function on many different strings on eceprog.ecn.purdue.edu extensively. Here are more examples you can try: `" ++ "`,

" - ", " +x ", " -0xAaG ", " 0xFFFFFFFFFFFFFFFFAAA " and " -0XFFFFFFFFFFFFFFFFAAA ". In particular, these examples will show you how `strtol` handles strings that are not valid or strings will cause overflow or underflow.

2.0.1 `NULL` address

When the second parameter `endptr` provided to `my_strtol` is not `NULL`, you have to store the address of the location of the character at which the conversion fails. `NULL` is numerically equivalent to 0. It is a special word that is defined to be `((void *)0)` in `stddef.h`. Typically, we do not include `stddef.h` since we almost always include `stdlib.h`, which includes `stddef.h`. Since we are not allowed to include `stdlib.h` in this assignment, the file `answer01.c` includes `stddef.h`.

2.0.2 Typecasting

The second parameter `endptr` of `my_strtol` is of type `char **`. Therefore, the type of `*endptr` is `char *`. This does not match the type of the address of the location of the character at which the conversion fails, which is `char const *`. For example, the compiler will complain with a warning if your program performs the following assignment:

```
*endptr = nptr;
```

You have to perform a typecasting such as the following to remove the warning:

```
*endptr = (char *)nptr;
```

2.1 Printing, helper functions and macros

All debugging, error, or log statements in `answer01.c` should be printed to `stderr`. We do not expect any messages to be printed to `stdout`.

You may define your own helper functions in `pa01.c` and `answer01.c`. You may view those test functions provided in `pa01.c` as helper functions. All these functions are declared and defined as static functions. In other words, the scope of these functions are only within the files that they are found. Declaring and defining these functions as static eliminate the conflicts in function names. You should not modify any of the `.h` files. You may reuse any helper functions you have created in PE02.

You are also allowed to reuse the `char_to_int` function as a static helper function. However, please note that the `INV_SYMBOL` is not defined in `PA01`. If you plan to use that, you have to define it explicitly in `answer01.c`. **Do not define `INV_SYMBOL` in `answer01.h` since you are not submitting `answer01.h`.**

You also should not use macros that start with `T_` (Upper case T and underscore) in their names. We will use such macros in the evaluation of your submissions. If you use such macros, we may not be able to evaluate your submission properly.

3 Compiling your program

We use the same flags introduced in PE01 to compile the program for debugging purpose.

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -g -pedantic -fstack-protector-strong \
    --param ssp-buffer-size=1 pa01.c answer01.c -o pa01
```

When we evaluate your program, we also use the optimization flag `-O3` (uppercase letter `O` and number three) instead of the `-g` flag as follows:

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -O3 -pedantic -fstack-protector-strong \
    --param ssp-buffer-size=1 pa01.c answer01.c -o pa01
```

You are recommended to copy the `Makefile` from PE01 and modify it appropriately for PA01.

4 Running and testing your program

It is your responsibility to test the function implemented in `answer01.c` and ensure that it works in a similar way as `strtol`, in terms of the returned value of the function, the update of the contents at the location whose address is the second argument of the function (if the second argument is not a `NULL` address), and the `errno`.

How should you test your implementation? You have to understand the `strtol` function, which is a black box to you. However, the manual page, and by trying out `strtol` function with different input string should allow you to understand the `strtol` function well enough to complete this assignment successfully.

Note that you are not allowed to call `strtol` in your function, as we do not allow you to include `stdlib.h` in `answer01.c`. If that function shows up in `answer01.c`, even if it is commented, you will receive ZERO for this assignment.

However, you are allowed to use `strtol` in your test functions in `pa01.c` to verify that your function works correctly.

You are recommended to change the functions in `pa01.c` so that it is more convenient for you to test multiple strings at the same time. For example, you can rewrite the program to allow the `main` function to accept different test strings as input arguments. You can also modify the `test_my_strtol` functions to accept input argument(s).

Moreover, you should check for memory errors/issues of your program using `valgrind`.

5 Submission

You must submit a zip file called `PA01.zip`, which contains one file:

1. `answer01.c`

Assuming that you are in the folder that contains `answer01.c`, use the following command to zip your file:

```
zip PA01.zip answer01.c
```

Make sure that you name the zip file as `PA01.zip`. Moreover, the zip file should not contain any folders. Submit `PA01.zip` through Brightspace. You may submit as many versions as you like. Brightspace will keep only the most recent submission, and we will grade only the final submission.

6 Grading

Your `answer01.c` should not include `stdlib.h` and it should not call `strtol`. If your submission has any occurrence of `stdlib.h` or `strtol`, your submission will receive a 0 grade, even if such an occurrence appears in a comment.

All debugging messages should be printed to `stderr`. We do not expect any output to be printed to `stdout`. It is important that if the instructor has a working version of `pa01.c`, it should be compilable with your `answer01.c` to produce an executable. For evaluation purpose, we will use different combinations of your submitted `.c` files and our `.c` files to generate different executables. If a particular combination does not allow an executable to be generated, you do not get any credit for the function(s) that the executable is supposed to evaluate. We use both `-g` and `-O3` flags (separately) to compile your submission. Your submission is evaluated only when compilation using each of the flags is successful.

Be aware that we set a time-limit for each test case based on the size of the test case. If your program does not complete its execution before the time limit for a test case, it is deemed to have failed the test case. We will not announce the time-limits that we will use. They will be very reasonable.

`my_strtol` returns the converted long integer, sets the `errno`, and if required, "returns" the location of the input string at which the conversion stops. The converted long integer accounts for 70%, the `errno` accounts for 10%, and the location at which the conversion stops accounts for 20%.

The occurrence of any memory issues (memory errors or memory leaks flagged in a `valgrind` report) will result in 50-point penalty.

7 A few points to remember

Your `answer01.c` should not include `stdlib.h` and it should not call `strtol`. If your submission has any occurrence of `stdlib.h` or `strtol`, your submission will receive a 0 grade, even if such an occurrence appears in a comment.

All debugging messages should be printed to `stderr`. We do not expect any output to be printed to `stdout`.

You are not submitting `answer01.h`. Therefore, you should not make changes to `answer01.h`.

You can declare and define additional static functions that you have to use in `pa01.c` and `answer01.c`.

You should not use macros that start with `T_`.

Grading of programming exercises and assignments is performed on machines with similar setup as eceprog.ecn.purdue.edu. You should perform testing of your work on eceprog.ecn.purdue.edu before submission. Correct output on your computer does not translate into correct output on eceprog.ecn.purdue.edu.