In this lab, you'll be writing a program that uses pointer arithmetic to write a simple string processing function. In addition, you'll use some of C's standard library string functions along with your function to build up a more complex string.

1. Exploration

This part is just to get you comfortable with how pointers and pointer arithmetic work before you write your string processing function. You don't have to turn this program in.

Here's the skeleton of the exploration program that you'll complete. Type it in accurately! Or just copy and paste it if you're using a local text editor, I guess.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "stringy!";
    int len = strlen(str);
    char* end = str + len;

    // Here, I want you to print four things:
    // str (use the %p specifier to print its address!)
    // end
    // len
    // end - str (use %lu for this)

    char* p;
    for(p = str; p < end; p++)
        printf("%s\n", p);

    return 0;
}
```

Add the prints in the place specified, then compile and run it. Look how the end pointer is calculated, and look what address it prints out. Then see what end - str prints out.

At the end of the program you see a common kind of loop in C string processing. Instead of using str[i], we can move a pointer through the string. When we print out the string pointed to by p, you'll see something interesting going on. Think about how strings in C really work – they're a sequence of bytes with a 0 byte at the end. Try instead using %c and printing out the value at p: *p.
2. Reversing a string

Now, in a file called lab3.c, you’ll write a function that reverses a string. Its prototype looks like this:

```c
void str_reverse(char* dest, const char* source);
```

It takes two parameters. `dest` is the place where the reversed string will be placed, and `source` is the string to reverse. The `const` on that parameter prevents you from changing the data in `source`. It’s a way to prevent some mistakes.

The function should work like this:

- Declare `const char* p` that points to to the last character of `source`. Not the zero terminator, but the character right before that. Think through how to calculate this pointer by thinking about how a string like "hello" is represented.
- Make a `for` loop that decrements `p` as long as it’s greater than or equal to `source`. In this way `p` will move through the `source` string backwards.
- Inside the `for` loop, copy the ONE character pointed to by `p` into the location pointed to by `dest`, and then increment `dest`.
- Last, after the loop, put a ‘\0’ character into the location pointed to by `dest` to properly terminate the string.

Once working properly, you should be able to call it like this:

```c
char buf[50];
str_reverse(buf, "Hello, world!");
printf("%s\n", buf);
```

This should print `dlrow ,olleH`

**If nothing prints out**, you probably calculated `p` incorrectly and copied the zero-terminator character into the first character of `dest`.

3. snprintf

Now you’ll write the program that uses `str_reverse`. There are a few ways to build up strings from multiple pieces in C, but probably the most flexible way (and least likely to end in mistakes and crashes) is the `snprintf` function. It works just like `printf`, but instead of printing to the console, it puts the text into a string that you give it. You can use it like:

```c
char output[200];
snprintf(output, sizeof(output), "This is some stuff: %d, %d, %d", 3, 4, 5);
```

and now the `output` string contains "This is some stuff: 3, 4, 5".
Finish your lab3.c file by making main do the following:

- Declare an array of strings. You can do that by writing `const char* variable_name[] = { ... }`. Inside the initializer, put the following strings:
  - "pullup"
  - "plop"
  - "racecar"
  - "racecat"
  - "rats"
  - "deleveled"
- For each of the strings in the array:
  - Reverse the string into a new string variable.
  - Make another string variable about 200 characters long.
  - Check if the non-reversed and reversed strings are equal by using the `strcmp` function. This takes two strings, and if they're equal returns 0. You have to write "strcmp(....) == 0" in order to check if the strings are equal.
  - If the strings are equal, use `snprintf` to print a message that looks like this into the long string variable: "rotator" backwards is "rotator", and it is a palindrome!
  - If the strings are not equal, do the same, but say that it isn't a palindrome.
  - Use `puts` to output the long string variable. You just write `puts(variable)`.

Once your program is working (it should print six lines), redirect the output to lab3.txt as you've done previously:

`./lab3 > lab3.txt`

**What to submit**

Hand in your lab3.c and lab3.txt files.

```
tar cvf USERNAME_lab3.tar lab3.c lab3.txt
gzip USERNAME_lab3.tar
cp USERNAME_lab3.tar.gz ~jfb42/submit/449
```