General notes about pointers, memory, and strings

- You can use pointers to “return” multiple values from a function
    - Pointers: data types that store memory addresses
    - int *a; → variable “a” stores a memory address of an integer
        - Equivalent to int* a. Note that the star is a part of the data type declaration: (int *)a
    - Two operators: Address of, and dereferencing
        - &a → The & operator acts on a, and returns the address of a
        - *b → The * operator dereferences b, and returns whatever is stored at memory address b
        - Don’t confuse the operator * with the * in the pointer data type. It’s confusing because it’s the same character both used in the context of memory, but they’re actually wholly distinct! One is an operator, and the other is part of a data type.
    - To allow your function to write to variables passed in as arguments, you want to pass in the pointer to those variables.
        - In particular, if you want your function to write to a string variable, then you want to pass in char** (memory address of a string)
        - Example of “returning” multiple values:

```
// Takes in two integers, and "returns" the sum and product of the
integers
void sumProd(int *sum, int *prod, int a, int b) {
      *sum = a + b;
      *prod = a * b;
}

// Usage
int main() {
      int s, p;
      sumProd(&s, &p, 4, 5);
      printf("Sum: %d\n", s); // prints 9
      printf("Product: %d\n", p); // prints 20
}
```

- Memory handling: Rule of thumb is every time you malloc, you must have a corresponding free before you exit the program.
    - Note that some functions (like stdrup()) call malloc internally, so you need to free the result.
- String/array processing
    - Useful functions: getline (usage: see spec Note 2), asprintf(), strlen(), strcmp(), strncmp(), strstr(), strchr(), strcat(), strcpy(), strncpy(), strdup(), and strndup()
        - *Please make use of google + man pages before asking a ULA how to use!*
        - Getline - See Note 2

        - Note that strncpy() doesn't copy the null terminator if copying n bytes from the middle of the string.
        - Strdup, strndup, asprintf allocate memory that you later have to free.
    - Null terminators are important! Keep track of them carefully.

Strwrs

- Strwrs: Takes in an input string, performs string substitutions according to rules, and outputs substituted string
- Vocabulary
    - String substitution rule - "subRule" - *For each rule*, how should we substitute the strings? Just once? Rescan from the beginning? Start looking after the replaced string? Start looking from the replaced string? - Relevant flags are q, n, r, s
    - Rule order rule - "ordRule" - In what order do we apply the rules? Reapply current rule? Start from the first rule? Apply rules in sequence? - Relevant flags are 'Q', 'N', 'R', 'S'
        - Note that if a rule application is unsuccessful, then the rule is to try the next rule
        - If there are no more rules to apply, then the processing step is over. Print out resultant string.
    - "Successful" rule application - application of rule resulted in *at least one* string substitution
    - "Unsuccessful" rule application - application of rule resulted in zero string substitutions
    - "Rule applications" always start from the beginning of the line. Each rule application tries as many substitutions as possible, as governed by "subRule"
        - With the exception of 'q', which does at most one substitution
- High level description of strwrs:
    - Every time you apply a rule, you start from the beginning of the line.
    - Suppose you have N rules, and that you are currently applying rule $1 \leq i \leq N$.
        - First, try applying the rule i by searching from left to right. If you are unable to apply it at all, then you're done applying the rule. Application of the rule was a failure.
        - If you are able to apply it once, the application of the rule was a success. But you're not necessarily done applying that rule:
            - If rule 'q', then you've finished applying the rule.
            - If rule 'n', 'r', or 's', you apply the rule again, starting from index determined by 'n', 'r', and 's' rules. Repeat until no more match.
                - 'n' means start looking after the replaced string
                - 'r' means start looking from the first character of the replaced string
                - 's' means start from the beginning of the line
            - Takeaway: apply lower-case flags to EACH rule before considering upper-case flags
    - If the application of the rule was a failure, then apply the next rule (i + 1), starting from beginning of the line.

        - If there are no more rules, you're done.
    - If the application of the rule was a success, then use 'Q', 'N', 'R', or 'S' to determine which rule to apply next.
        - 'Q' says you're done at this point.
        - 'N' says apply the next rule (i+1) (note: from the beginning of the line)
        - 'R' says apply the current rule (i) again (note: from the beginning of the line)
        - 'S' says apply rules starting from rule #1 (note: from the beginning of the line)

Examples:

strwrs -n aa a	"**aa**abb" → "aabb"
strwrs -r aa a	"**aa**abb" → "**aa**bb" → "abb"
strwrs -Qn aa a abb aab	"**aa**abbb" → "aabbb"
strwrs -Rn aa a abb aab	"**aa**abbb" | "**aa**bbb" | "**abb**b" | "a**abb**" --> "aaab"
strwrs -Nn aa a abb aab	"**aa**abbb" | "a**abb**b" → "aaabb"

- Additional features: +,-,/
    - If + is in the beginning of the pattern, it means match with the beginning of the string. If + is anywhere else, it is a literal '+' character.
    - Same with -, but with the end of the string.
    - / allows you to escape the next character. For example, it allows you to escape + in the beginning of the pattern, or - at the end of the pattern.
        - If / is anywhere else in the string, it "escapes" the next character, *unless* it is the last character of the string (in that case, there's nothing to escape, so it's just the literal '/').

Examples:

strwrs -n +aa a	**"aa**baa" → "abaa"
strwrs -n aa- a	"aab**aa"** → "aaba"
strwrs -n ++ -	"**+**-++" → "--++"
strwrs -n /++ -	"+-**++**" → "+--"

- Tips
    - Flag parsing - If total number of arguments is even, then argv[1] should be the flag
        - 'N' and 'n' are defaults
        - If flag contains multiple lowercase and uppercase rules, only the last uppercase rule and last lowercase rule are effective
        - E.g., If the flag is -nqRSrq, then the effective uppercase rule is S and the effective lowercase rule is q
            - Note that if the flag contains a letter that does not correspond to a rule (e.g., x, P), then you should print out an error message and quit
    - Note 2 gives you some boilerplate code for reading lines from stdin. Would recommend starting from there. Don't forget to #define _GNU_SOURCE.

- Note 4 gives you additional instructions for what to do when a string substitution leaves the line unchanged, which might ordinarily result in infinite rule substitution.
- Note 5 says that Fi/Ti don't contain newlines. Also, no string can "contain" null characters, since they're used to terminate strings. But the remaining 254 characters are fair game.
- What you can & can't use
    - YES: pointers, malloc, arrays, strings, etc.
    - NO: global variables, structs
- Use -v flag on the staff binary for verbose output to better understand program behavior.

Appendix: More examples

```
elp34@~$ echo "aaaabbb" | /c/cs223/Hwk3/strwrs -Qn aa a abb aab
aabbb
elp34@~$ echo "aaabbb" | /c/cs223/Hwk3/strwrs -Qn aa a abb aab
aabbb
elp34@~$ echo "aaaabbb" | /c/cs223/Hwk3/strwrs -Qq aa a abb aab
aaabbb
```

// Q means quit after ALL POSSIBLE substitutions of first successful rule (mediated by lowercase letter)

```
elp34@~$ echo "acacb" | /c/cs223/Hwk3/strwrs -Nn ac d acb e
ddb
elp34@~$ echo "acacb" | /c/cs223/Hwk3/strwrs -Nq ac d acb e
de
```

// N[nrs] means move onto the next rule after ALL POSSIBLE substitutions with the current rule, as mediated by the lowercase letter (so with q it will quit after 1 substitution of ac)

```
elp34@~$ echo "bacac" | /c/cs223/Hwk3/strwrs -Nn ac d b e
edd
```

// N means try the next rule STARTING FROM THE BEGINNING of the string

```
elp34@~$ echo "aaaabbb" | /c/cs223/Hwk3/strwrs -Rn aa a b e
aeee
```

// R means after all possible substitutions of a rule (mediated by lowercase flag), if successful, apply the SAME RULE STARTING FROM THE BEGINNING of the string

```
*** Show use of -v flag for verbose output from staff solution!!! Super helpful
elp34@~$ echo "aaaabbb" | /c/cs223/Hwk3/strwrs -v -Rn aa a b e
F=aa    T=a     POS=0   OLD=aaaabbb     NEW=aaabbb
                POS=1   OLD=aaabbb      NEW=aabbb
F=aa    T=a     POS=0   OLD=aabbb       NEW=abbb
```

F=b T=e POS=1 OLD=abbb NEW=aebb
POS=2 OLD=aebb NEW=aeeb
POS=3 OLD=aeeb NEW=aeee