COMP 4959: Lab 2

Submit a source file named `cut.rs`. Maximum score: 12

Implement a Rust program to print selected parts of lines read from standard input.

We call our program `cut`. It is invoked with a command-line argument specifying the ranges of characters or fields to print. The argument starts with `-c` or `-f` followed by one or more ranges, separated by commas. Each range is one of:

`N`	`N`'th character/field, counted from 1
`N:`	from `N`'th character/field, to end of line
`N:M`	from `N`'th to `M`'th (included) character/field
`:M`	from first to `M`'th (included) character/field

where `M` and `N` are positive integers (and `N` cannot exceed `M` in the case of `N:M`).

The invocation of the program is invalid if the argument does not start with `-c` or `-f`, or if a range is invalid, or if successive ranges are not separated by a comma. Note also that the program must be invoked with exactly one argument and that there must be at least one range. In case the program is incorrectly invoked, it simply exits, after printing a message.

Two valid commands (note: `$` is the command prompt):

```
$ ./cut -c2,:3,10,5:7,13:  < data.txt
$ ./cut -f:3,5,8 < data.txt
```

For both examples, the file `data.txt` is read from standard input via I/O redirection. For the first command, characters to print are specified by `2,:3,10,5:7,13:`. Characters in a line are numbered starting from 1. `2` and `10` specify characters 2 and 10 (second and tenth characters). `:3` specifies (the range of) characters 1, 2 and 3; `5:7` specifies characters 5, 6, and 7; `13:` specifies all characters starting from character 13. Characters selected for printing are written in the same order they are read, and is written exactly once. In our example, for each line, characters 1, 2, 3, 5, 6, 7, 10, 13 and later characters are printed in that order. Note that the line terminator (e.g. the newline character) in each input line is always printed.

The second command is similar to the first one except that the ranges specify fields to prints. Fields are separated by tabs. The first field is the text before the first tab, the second field the text after the first tab but before the second tab, and so on. The ranges in `:3,5,8` specify fields 1, 2, 3, 5 and 8. Note that in the output, the fields are also separated by tabs. As in the character case, the line terminator of each input line is always printed.

For the implementation, define an `enum` type (named `Range`) that consists of the 4 types of ranges shown above. (Note: This is different from the `Range` trait in the standard library.) For this `Range`, provide functions to parse a string into a range and to test whether a range contains a specific number:

```
impl Range {
  fn parse(s: &str) -> Option<Range> { ... }
  fn contains(&self, n: usize) -> bool { ... }
  // ...
}
```

You can then build functions that parse the command-line argument and use it to figure out which characters or fields to print.

As an example, if the content of `data.txt` is as follows

```
homer simpson 25 # bad at nuclear engineering
ned flanders 99 ! high score
monty burns 65
```

then the output of `./cut -c2,:3,10,5:7,13:  < data.txt` is:

```
homr spn 25 # bad at nuclear engineering
nedflae 99 ! high score
mony bn65
```

For a second example, assume that each line has 4 fields separated by tabs (shown with a tab width of 8) in the following `data.csv` file

```
1       Burns, Montgomery (Monty)       A66666666       **Registered**
2       Simpson, Bartholomew (Bart)     A12345678       **Registered**
3       Simpson, Homer  A87654321       **Registered**
```

then the output of `./cut -f2,3 < data.csv` is as follows

```
Burns, Montgomery (Monty)       A66666666
Simpson, Bartholomew (Bart)     A12345678
Simpson, Homer  A87654321
```