

Exercise 05

I'm not proud of this exercise. In it, you get to take a perfectly good program for generating prime numbers (kind of like the one we looked at in class) and turn into a horrible mess of spaghetti code. This will involve some over-use of the **goto** statement, not the kind of thing you'd normally do in the real world of software development. Maybe you can get it out of your system on this exercise and never want to use goto again.

You'll find a couple of implementation files, and some sample inputs and expected outputs on the course homepage, or you can download them using the following curl commands. Copy these into your own directory to get started.

```
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/primes.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/ugly.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/input-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/expected-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/input-2.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise05/expected-2.txt
```

The primes.c program is a working program, coded like you'd normally code in C, with structured looping statements, and using a function to help simplify the code. This program is just an example; you don't need to change it. Your job is to complete the ugly.c program. This one will do the same job as primes.c, but it will do it in a substantially uglier way. Here's what you need to do.

- After reading the input, the primes.c program has a for loop that checks each value in the given range to see if it's prime. In your ugly.c program, write a loop that does the same thing, but using if and goto statements instead of primes. There's a section below giving some hints about how to do this.
- On each iteration of its loop, the main() function in primes.c calls a function called isPrime(), to check if the current value is prime. It's nice to have functions like this to help organize our code, but we can get by without them if we're willing to write some more, ugly code. Let's do that.

In your ugly.c program put the logic from the body of isPrime() right inside your main function. When you want to check a value to see if it's prime, you'll just execute the code to look for a factor, rather than calling a separate function to do it.

This is called, *inlining*, where you replace a call to a function with a copy of the function's body. We'll see it again later in the semester

- Finally, replace the for loop from isPrime() with looping code made from if and goto statements.

If you've done it right, your ugly.c program shouldn't have any for loops, while loops or do/while loops, and it should only have a main() function, with all the code inside it. It's going to be ugly, with a lot of labels to specify the destinations of goto statements. I used five different labels in my solution.

Looping With Goto

The goto statement is a low-level way of altering the flow of control in your program. It's more like how looping is done in the underlying assembly language. That's probably how it got into C in the first place. The language has much more human-readable ways of organizing the code, but goto is still available, if you need some really unconventional flow-of-control behavior, or if you just don't other people to be able to read your code (or to like you).

Goto statements always jump to a label elsewhere in the same function. A label is just an identifier with a colon after it, like this one:

```
someLabel:
```

You can put labels between any statements in a function. Then, elsewhere in the function, either before or after the label, you can tell execution to jump to the label with a goto, like:

```
goto someLabel;
```

There are lots of different ways to make a loop with goto statements. Let's consider one approach. Here's a basic for loop:

```
for ( int i = 0; i < 10; i++ ) {  
    printf( "%d\n", i );  
}
```

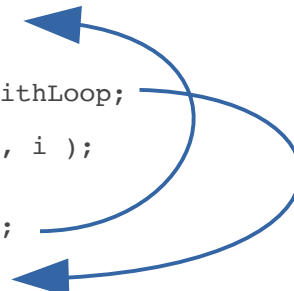
This loop sets the value of variable, i, then it tests at the top of the loop. If the condition isn't true, it exits the loop immediately. We could simulate this with a goto as follows. We test the condition with an if statement and use a goto to jump to the end of the loop if the condition is false:

```
int i = 0;  
if ( i >= 10 )  
    goto doneWithLoop;  
  
// ... body of loop, increment i, go back to the top ...
```

```
doneWithLoop:
```

This takes care of getting out of the loop when we need to. We just need to add code for the body of the loop. Then, at the bottom of the loop, we can increment the loop control variable and go back to the top to re-run the test, like in the following:

```
int i = 0;  
topOfLoop:  
    if ( i >= 10 )  
        goto doneWithLoop;  
  
    printf( "%d\n", i );  
  
    i++;  
    goto topOfLoop;  
  
doneWithLoop:
```

A diagram illustrating the loop structure. Two blue arrows originate from the 'goto topOfLoop;' statement and point back to the 'topOfLoop:' label. Another blue arrow originates from the 'goto doneWithLoop;' statement and points to the 'doneWithLoop:' label.

At the bottom of the loop, we increment the loop control variable, then jump back up to the top, right before the test. There are other ways to write this loop using goto. Feel free to use approaches in your solution, if they make more sense to you.

Sample Execution

Once your program is ready, you should be able to run it as follows. Here, we're just typing in the two values from the first input file, input-1.txt

```
$ ./ugly  
10 60  
11  
13  
17  
19  
23  
29  
31  
37
```

41
43
47
53
59

Or, you can have your program read from one of the sample input files, then check its output against the example output file provided.

```
$ ./ugly < input-2.txt > output.txt  
$ diff output.txt expected-2.txt
```

Submitting your Solution

When you're done, check your program to make sure you don't use any looping constructs (no for loops, no while loops. ... no do/while loops), and that all your code is inside main(). Consider how you'd never really want write code like this in practice. Then, with an appropriate sense of regret, submit the source for your ugly.c program to the exercise_05 assignment in Moodle.