CS Basics - Exercises
Control Structures in C

C. Grothoff and E. Benoist

Fall Term 2018-19

1 The three types of `char`

The `getchar()` function returns an `int`. Why is that? Consider the following program:

```
#include <stdio.h>
#define TYPE char
int main()
{
  int ret = getchar ();
  TYPE c = (TYPE) ret;
  fprintf (stdout,
           "%d\n",
           (EOF == c));
  return 0;
}
```

The C Standard says:

"Three types of `char` are specified: `signed`, plain, and `unsigned`. A plain char may be represented as either `signed` or `unsigned`, depending upon the implementation, as in prior practice. The type `signed char` was introduced to make available a one-byte signed integer type on those systems which implement plain char as `unsigned`. For reasons of symmetry, the keyword `signed` is allowed as part of the type name of other integral types."

Change the code:

- Define the TYPE as `signed char`
- Define the TYPE as `unsigned char`

What happens? Why? What does this mean for the original program? Also, compile the original program with `-funsigned-char` or `-fsigned-char`!

2 Fun with `scanf`

Consider the following program:

```
#include <stdio.h>
int main()
{
  char x[30];
  char y[30];

  scanf ("%[^\n]", x);
  scanf ("%[^\n]", y);
  fprintf (stdout,
           "%s - %s\n",
           x,
           y);
  return 0;
}
```

What happens with a (benign) input of a line of less than 30 characters? Why?
The `*` can be used in a format string to suppress assignment. Consider:

```
#include <stdio.h>
int main()
{
  int x;

  scanf ("%*c%d", &x);
  fprintf (stdout,
           "%d\n",
           x);
  return 0;
}
```

Use this as inspiration to fix the original program!

3 Fun with `sscanf`

Consider the following program:

```
#include <stdio.h>
int main()
{
  char input[] = "1a";
  unsigned int x;
```

```
  if (1 == sscanf (input, "%u", x))
    fprintf (stdout,
             "Got a number %u\n",
             x);
  return 0;
}
```

What happens? Why? Modify your code to reject inputs that include anything except for a number!

4 Prime time

Complete the following program to print the first n prime numbers, where n is entered interactively. Do not worry about efficiency, simply test if `i` is divisible by any number $j \in [2, i)$.

```
#include <stdio.h>
#include <stdbool.h>
int main()
{
  unsigned int n;

  if (1 != scanf (                ))
    return 1;
  for (unsigned int i=2;    ;i++)
  {
    bool i_is_prime = true;









    if (i_is_prime)




  }
  return 0;
}
```

5 Interactive program

5.1 Factorization

Exercise 1 *Write an interactive program that asks for a number and writes on the standard output, the positive factors of the number. If the given number is negative, the first factor must be* `-1`.

Example

```
user1@mymachine > ./factors
Type a number: 24
Factors: 2 * 2 * 2 * 3
```

Method

You will increment a number (the divisor) divide your number by the divisor. If the remainder is 0, take the divisor as a factor and change the number to be the result of the division.

Modification

Modify your program such that it asks for new numbers to factorize, as long as the number is not 0. If the number is 0, the program terminates.

5.2 Greatest common divisor

Exercise 2 *Write a program asking for two numbers and computing their greatest common divisor (see Wikipedia* `http://en.wikipedia.org/wiki/Greatest_common_divisor` *for more details, if you do not remember your 7th grade mathematics). Since you have a method for factorizing the numbers, you could use it.*

6 Text conversion

Exercise 3 *Write a C program that parses the file* `resources/test.txt` *(located in the Git repository) from* `stdin` *(using* `scanf()`*). Your C program then should write (using* `printf()`*) the file* `resources/result.txt` *to* `stdout`*. For parsing you can use expressions of the form* `"%[^xy]"` *which match anything that is not* `x` *or* `y`*.*