Secure Coding (CNT 4419)
Assignment II

Objective: To become acquainted with, and use, a C safe-math library.
Due Date: Sunday, November 5, 2023 at 11:59pm. No late submissions will be accepted.

Assignment Description
Complete this assignment by yourself. While doing this assignment you will need to run a C compiler. One option is to use an online C compiler such as https://paiza.io/en/languages/online-c-compiler. If you use Paiza, you may optionally create an account to save your code. Paiza has an Input tab for providing input to the program.

This assignment asks you to use the Safe Math library. An example program (*Main.c*) using this library can be found at https://paiza.io/projects/hqKdfhnoRkdSNH2GTpOOSA. This example includes sample input in the Input tab. You may fork this code (that is, add a copy to your account) using the menu next to the Run button.

If you are not using Paiza, begin by downloading the *safe-math.h* file from the repository at https://github.com/nemequ/portable-snippets/tree/master/safe-math. Then copy-paste the code shown in *Main.c* at the Paiza link above into your own *Main.c* file.

As part of the Safe Math library, you will be using the `psnip_safe_char_mul`, `psnip_safe_int_mul`, and `psnip_safe_long_mul` functions, which are declared and implemented in the *safe-math.h* file.

Below is brief documentation for the library functions you will need to use. This documentation appeared in the project's *readme* in GitHub. Note: `psnip_safe_bool`s can be used as regular bools.

- **psnip_safe_bool psnip_safe_char_mul (char* res, char a, char b)**
  Attempts to multiply chars `a` and `b`, and stores the results in the address `res`. If the operation can be completed without overflowing or underflowing, a `psnip_safe_bool` value is returned that evaluates to true. Otherwise, a `psnip_safe_bool` that evaluates to false is returned.

- **psnip_safe_bool psnip_safe_int_mul (int* res, int a, int b)**
  Attempts to multiply integers `a` and `b`, and stores the results in the address `res`. If the operation can be completed without overflowing or underflowing, a `psnip_safe_bool` value is returned that evaluates to true. Otherwise, a `psnip_safe_bool` that evaluates to false is returned.

- **psnip_safe_bool psnip_safe_long_mul (long* res, long a, long b)**
  Attempts to multiply longs `a` and `b`, and stores the results in the address `res`. If the operation can be completed without overflowing or underflowing, a `psnip_safe_bool` value is returned that evaluates to true. Otherwise, a `psnip_safe_bool` that evaluates to false is returned.

Use these functions to implement a program that inputs two strings A and B from stdin, converts A and B to longs (set to 0 if the conversion fails), multiplies A and B using the smallest single type T, and outputs the result. The eligible types T are char, int, and long, where char is "smaller" than int and int is "smaller" than long. The two possible program outputs are of the form:

1. "`A and B can be successfully multiplied as Ts, producing product C.`" (where `T` is the smallest single type that is large enough for storing A, B, and C)
2. "`A and B cannot be multiplied as longs.`"

You will want to use the constants `CHAR_MIN`, `CHAR_MAX`, `INT_MIN`, `INT_MAX`, `LONG_MIN`, and `LONG_MAX`, which are found in the standard C-header file *limits.h*. For this assignment, you should only need the *Main.c* and *safe-math.h* files, with your *Main.c* including only the *safe-math.h* and standard-library headers (e.g., *stdio.h*, *limits.h*, *errno.h*, and *stdlib.h*).

Submit to Canvas your completed *Main.c* file.

Sample Executions

Input 1:

```
1
2
```

Output 1:

```
1 and 2 can be successfully multiplied as chars, producing product 2.
```

Input 2:

```
111
222
```

Output 2:

```
111 and 222 can be successfully multiplied as ints, producing product 24642.
```

Input 3:

```
111111
222222
```

Output 3:

```
111111 and 222222 can be successfully multiplied as longs, producing product
24691308642.
```

Input 4:

```
11111111111
22222222222
```

Output 4:

```
11111111111 and 22222222222 cannot be multiplied as longs.
```

Input 5:

```
Here A is set to 0 because this line can't be converted to a long.
9
```

Output 5:

```
0 and 9 can be successfully multiplied as chars, producing product 0.
```