CMPUT 350 Lab 1 Prep Problems

Note: We expect you to test your programs. One simple way of doing this in case your program reads input from stdin is to prepare text files — say `inp1.txt`, `inp2.txt`, ... — containing test case inputs and then use

```
./a.out < inp1.txt
./a.out < inp2.txt
...
```

---

1. Write a C program `e1c.c` that reads integers from `stdin` until the end of input is reached, and prints the sum of all encountered numbers to stdout - using `scanf`/`printf`. In case of invalid inputs, your program needs to write "illegal input" to `stderr` (using `fprintf(stderr, ...)`) and exit. Test your program with several inputs.

Hint: when using `scanf`, a return value different from the number of expected items indicates that an invalid input or the end of the input was encountered (see `man 3 scanf`). To distinguish the latter cases use `feof(stdin)` (see `man 3 feof`).

---

2. Similar to problem 1, write C++ program `e2.cpp` that reads integers from `stdin` until the end of input is reached, and prints the sum of all encountered numbers to stdout — using `std::cin/std::cout`. In case of invalid inputs, your program needs to write "illegal input" to `stderr` and exit. Test your program with several inputs.

Hint: After reading using `std::cin`, you can check whether the input succeeded like so:

```
cin >> ...;
if (!std::cin) {
    // something isn't right
    if (std::cin.eof()) {
        // end of file/input reached
    } else {
        // bad input
    }
} else {
    // reading succeeded
}
```

This works because there is an implicit conversion from `istream` to `bool`. Thus, this also works:

```
int i;
while (std::cin >> i) {
    // reading succeeded
    // ...
}
// something isn't right ...
```

---

3. Write C++ program `e3.cpp`, using `std::cin/std::cout/new/delete`, that reads integers (in decimal notation with white-space (space,newline,tab) as delimiter) from `stdin` and prints them in reverse order with one space in between them when the end of input is reached. You may assume that the input contains at most 10,000,000 integers. If not, your program needs to write "input too big" to `stderr` and exit. If an illegal input is encountered, "illegal input" needs to be printed to `stderr` and your program must exit at that point. Test your program with different inputs and ensure that it doesn't leak memory using valgrind.

Hints:

- Runtime stacks are usually quite small (a few megabytes). This means that large arrays can't be allocated on the stack.
- Input stream `operator >>` skips over white-space characters.