1. (8 points) Write a function-header comment for this `read_digits` function, describing what it does.

- Read decimal digits from the standard input until a nondigit is reached,
- or end-of-file,
- using “next” as the lookahead character.
- If there are no digits before the next nondigit, return 0.
- If there are < n digits, store them in a as a null-terminated string,
  and return 1.
- If there are ≥ n digits, store the first n of them in a,
  and return -1.

Commentary (not part of the answer): Since this function interacts with the global variable `next`, this really must be mentioned in the function-header comment.

And I did NOT explain how it’s done, only what is done.

2. (8 points) Write a function-header comment for function `read_while`.

- Read characters satisfying the predicate f from the standard input until a non-f character is reached,
- using “next” as the lookahead character.
- If there are no f characters before the first non-f, return 0.
- If there are < n such characters, store them in a as a null-terminated string,
  and return 1.
- If there are ≥ n such characters, store the first n of them in a,
  and return -1.
- If n == 0, then a is permitted to be NULL.

Commentary (not part of the answer): The specification that “if n == 0 then a can be NULL” is required by at least one client of this function, as you can see in main.

And I did NOT explain how it’s done, only what is done. In that spirit, I did not mention calls to `read_while_entry` and `read_while_exit`.

3. (5 points) Write a function-header comment for function `main`.

- Read four nonnegative integers (separated by whitespace) from the standard input.
- If the input does not begin with 4 integers separated by whitespace, exit with status 1.
- If any of the integers is > 999999999, exit with status 1.
- Otherwise, print the total on the standard output.
- This program fails with undefined behavior if the total is ≥ 2^{31}.

Lose points for: explaining how it’s done instead of what is done.

4. (4 points) True or false: Some input string (on stdin) can cause the assertion at line 34 to fail.
   If true, show the input.
   False.

5. (4 points) True or False: There is an input to this program that will cause signed integer overflow at line 21.
   If true, show the input.
   False.
6. (4 points) True or False: There is an input to this program that will cause signed integer overflow at line 36. If true, show the input.
   False.

7. (4 points) True or False: There is an input to this program that will cause signed integer overflow at line 47. If true, show the input.
   True. 999999999 999999999 999999999 999999999

8. (6 points) Write the function read_while_entry that contains function-entry assertions for read_while. It should be as strong (detect as many erroneous conditions) as you can reasonably make it.
   ```c
   void read_while_entry(int (*f)(int), char a[], int n) {
     assert (n>=0);
     assert (a!=NULL || n==0);
     assert (f!=NULL);
   }
   ```

9. (4 points) Write a main function, as simple as possible, that calls read_while with an erroneous condition that is not caught by your read_while_entry assertions. (Your main function doesn’t have to do anything useful; it doesn’t have to match in any way the behavior of my main function.)
   There are several possible answers. Here are two.
   ```c
   int main(void) {
     char buffer[1];
     return read_while (isdigit, buffer, 1000);
   }
   ```
   ```c
   int myfun(int x) {int b[10]; return b[x]; }
   int main(void) {
     char buffer[100];
     return read_while (myfun, buffer, 100);
   }
   ```

10. (8 points) Write the function read_while_exit that contains function-exit assertions for read_while. It should be as strong as you can reasonably make it.
    ```c
    void read_while_exit(int (*f)(int), char a[], int n, int i, int status) {
      int j;
      assert (!f(next));
      if (status== -1) {
        assert (i==n);
        for (j=0; j<n; j++)
          assert (f(a[j]));
      }
      else if (status==1) {
        assert (i<n);
        assert (strlen(a) == i); /* or: assert(0 <= i && a[i]=0); */
        for (j=0; j<i; j++)
          assert (f(a[j]));
      } else {
        assert (0);
      }
    }
    ```