In this exercise we familiarize to the buffered file handling.

Excercise 4a (Buffered i/o in standard library, 1,0p)

Part A

Demonstrate the buffering of i/o-library for yourself so that you set the state of standard output (stdout) to fully buffered and give your own buffer to the library. Set the buffer size to 5 bytes for example. Then write 13 characters, one character at a time, to the display (standard output). After sending each character to the stream (using function `putchar` or `fputc`) put your program to sleep one second (using function `sleep`). Then you can see easily how characters are displayed only when the buffer is full (5 characters at a time in 5 seconds period).

Observe what happens to three last characters in the following situations:

a) the main function is terminated by `fflush(stdout)` and `return 0` in this order.
b) the main function is terminated by `exit(0)` without `fflush(stdout)` or `return 0`
c) the main function is terminated by `return 0` (without `fflush(stdout)` and `exit(0)`)

Try to find the explanation for the phenomenon you see in situations a, b and c.

Part B. (Using services of i/o-library to file descriptor)

Often we have only file descriptor available for the file or device (no name is available). This is the case for example when using sockets or pipes. It is still possible to use services of i/o-library (buffering, conversion, formatting) also in these cases.

Now you use lotto number simulator that sends seven random numbers in ascii separated with newlines. Because they are in ascii, you can read one character at a time and write them directly to the screen using write system call. Then you will see the numbers in plain text beautifully below each other. If you however, need to do some calculation with the numbers, it is not possible without conversion to number (to int in this case).

You only have file descriptor of the lotto simulator. The file descriptor is opened with the function

```
int OpenRandomGenerator();
```

Then it is possible to read that descriptor and get those seven lotto numbers (one number in two seconds).

Remark 1.The lotto number simulator is used in very similar way than chat fellow simulator.

Now you are asked to write the program that reads the seven lotto numbers from the lotto simulator descriptor and displays them to the screen. In addition to that the program counts how many lotto numbers are below 20. The result is displayed.

Hint 1. Use function `fscanf`. Then you don't need to use any separate conversion functions.

Hint 2. To use lotto simulator you need to link the object file of the lotto simulator to your executable as follows:

gcc ex4B.c RandomGenEdu.o -oex4B.exe (in Edunix)

Copy the file RandomGenEdu.o to your working directory

Extra exercise 4b. (More examination of the i/o-library, 0,25p)

Find the file /usr/include/libio.h and the definition of `struct _IO_FILE` in that file. FILE is only another name for this structure. The file libio.h is a header that is included into header file stdio.h.

Open any text file using library function `fopen` and read (`fread` or `fgetc`) something from it. After that, access the FILE structure and display the following information from that structure:

1. filedescriptor,
2. the size of the buffer and
3. values of _IO_buf_end and _IO_buf_base in hexadecimal
4. the buffering mode.

These things can be found and/or calculated in the following way:

filedescriptor is in the field _fileno

the size of the buffer is from two members as follows

_IO_buf_end - _IO_buf_base

the procedure to find out the buffering mode is as follows

There is the field _flags and mask constants
_IO_UNBUFFERED and _IO_LINE_BUF.
1) If the bit _IO_UNBUFFERED if ON in the _flags field
the buffering mode is unbuffered
2) If the bit _IO_LINE_BUF if ON the buffering mode
is line buffered
3) If both of these bits are OFF, the buffering mode is
fully buffered

Display the same information about stdout.

Remark. It is important to do one i/o operation (read or write) before members _IO_buf_end and _IO_buf_base have correct values in the FILE structure. Test with the

same file that you used in the first test but this time don't read anything from the file before accessing the FILE structure. Do you see a difference and if you do suggest a (practical) reason for the difference.