# Report of the exercise 8.3

## Version A

We need to implement a user menu with the following options

* *R n:* read the data of a student number n (ID = n);
* *W n:* write the data of a student number n (ID = n);
* *E:* end program.

In version A of the program, the file, whose name is specified in command line, has to read using file pointers.

First, we use the *CreateFile()* function that returns, on success, an HANDLE to a file object and we specify the parameters in the following way:

hIn = CreateFile(argv[1], GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

**0** corresponds to the file sharing mode and means that it cannot be shared, **NULL**, which is a pointer to SECURITY\_ATTRIBUTES structure, remember that everyone has full control of a newly created file, **OPEN\_EXISTING** specifies the type of opening the file, in that case fail if the file does *not* exist, **FILE\_ATTRIBUTE\_NORMAL** refers to the attributes of the file handle and if it is set like this it means no other attributes are set, **NULL** represents an handle of an open GENERIC\_READ file but, generally, is set to NULL.

Then there is the while(1) loop in which we take the command using the \_fgetts(cmd, sizeof(cmd)+1, stdin);

Next there is the switch case based on the command inserted:

* *R:* we use SetFilePointer(hIn, NULL, NULL, FILE\_BEGIN); because we need to position the pointer at the beginning of the file and then we must read the file:  
  while (ReadFile(hIn, &stud, sizeof(student), &nr\_bytes, NULL) && nr\_bytes > 0)   
  Where *hIn* is the file handle, *stud* is the memory buffer to receive the input data, *sizeof(stud)* is the number of bytes we expect to read, in fact we are reading one line at time, *nr\_bytes* is the actual number of bytes transferred and it is the parameter that we use to remain in that while loop because when it is equal to 0 it means that we reach the end of file and finally *NULL*, pointer to OVERLAPPED structure. While the condition is verified we check if the value saved in the structure stud.id is equal to the one introduced by command line; if it is found then all the line corresponding to the id is printed out otherwise, when the condition nr\_bytes == 0 is verified, so we have searched in the whole file, we print out an error message.
* *W:* this is like the previous case but we also introduce a counter, *i,* initially set to 0. We search if there is the id that we put by command line, if it is present we overwrite the record already present in the file otherwise we increment i and we search in the whole file. If the id is not present inside the file we add a new record specifying all the fields: \_tscanf("%d %ld %s %s %d", &stud.id, &stud.register\_number, &stud.name, &stud.surname, &stud.mark), very useful since allow us to specify the format of each parameter and then we store it with the: if (WriteFile(hIn, &stud, sizeof(student), &wr\_bytes, NULL) && wr\_bytes == sizeof(student)) which has the same structure of the ReadFile but instead of nr\_bytes now we have wr\_bytes which is the actual number of bytes transferred, if 0 indicates end of file.
* *E:* here we have simply to exit from the program so we close the handle opened at the beginning even if Windows automatically will close all open handles on exit but it is good practice to close handles before terminating. If you insert a command that is not specified in the switch case it will be printed on the screen a message: “Command not supported”.

## Version B

This time, overlapped data structures have to be used to read, instead of file pointers. The first thing that we do is declare the overlapped structure:   
OVERLAPPED ov = { 0, 0, 0, 0, NULL }; //Internal, InternalHigh, Offset, OffsetHigh, hEvent  
where *Internal* and *InternalHigh* are reserved, *Offset* and *OffsetHigh* are the offset of the file pointer from the initial byte and *hEvent* is a field used for the asynchronous I/O and must be NULL.   
With respect to version A, there are a few changes:

* *R:* we use a counter set to 0, i, and a variable nRead initially set to 1; then there is a while condition: if nRead > 0 we have to update the reference in the following way:

filePos.QuadPart = i\*sizeof(student);

ov.Offset = filePos.LowPart;

ov.OffsetHigh = filePos.HighPart;  
where filePos is defined as LARGE\_INTEGER. In filePos.QuadPart we have 64 bit integer at 0x1000, ov.Offset the lower 32 bits of the 64 bit integer at 0x1000 and in ov.OffsetHigh the upper 32 bits of the 64 bit integer at 0x1004. Then, we call:  
ReadFile(hIn, &stud, sizeof(student), &nRead, &ov);   
specifying the last parameter that refers to the overlapped structure.

* *W:* like the R case but we have to pay attention when we want to overwrite a record in the file because we have to set the FilePointer at the beginning of the record in order to overwrite it correctly and for that operation is used the counter i necessary to point to the correct line:   
  SetFilePointer(hIn, i\*sizeof(student), NULL, FILE\_BEGIN);
* *E:* same as Version A command *E*.

## Version C

In that case we want to lock each record before reading or writing and release the same record as soon as the operation has been performed. Let’s see what change respect the case shown in the exercise version B:

* *R:* after the initialization phase we acquire the lock in the following way:   
  if (LockFileEx(hIn, LOCKFILE\_EXCLUSIVE\_LOCK, 0, sizeof(student), 0, &ov))   
  we perform the ReadFile() function and then we release the lock:   
  if(UnlockFileEx(hIn, 0, sizeof(student), 0, &ov)).
* *W:* like in the R case, before and after each operation of reading and writing we acquire and release the lock.
* *E:* same as Version A command *E*.