Esercizio 2

Un sistema legacy scritto C deve essere esteso con nuove funzionalità scritte in Rust. Per minimizzare le modifiche al codice esistente si è deciso di salvare i dati gestiti dal sistema legacy su file e il programma Rust li deve leggere per processarli. L'obiettivo è fare l'interfaccia di comunicazione (salvataggio dati / lettura) dei due programmi.

I dati sono dei record definiti in C in questo modo

```
typedef struct {
        int type;
        float val;
        long timestamp;
} ValueStruct;

typedef struct {
        int type;
        float val[10];
        long timestamp;
} MValueStruct;

typedef struct {
        int type;
        char message[21]; // stringa null terminated lung max 20
}  MessageStruct;

typedef struct {
        int type;
        union {
                ValueStruct val;
                MValueStruct mvals;
                MessageStruct  messages;
        };
} ExportData;
```

I dati da esportare sono di tre tipi:

- misure di vario tipo memorizzate in ValueStruct / MValueStruct
- messaggi di vario tipo memorizzati in MessageStruct.

I dati vengono poi incapsulati in una struct ExportData che può contenere un qualsiasi dato da esportare, con il tipo indicato in type (1=Value 2=MValue 3=Message)

Scrivere quindi un programma C che crei un vettore con 100 valori da esportare e li salvi in modo binaro con la seguente funzione. *fp è un file aperto precedentemente:

```
void export(ExportData *data, int n, FILE *pf) {
        fwrite(data, sizeof(ExportData), )
```

```
}
```

Creato il file scrivere in programma Rust che:

- prenda da command line il nome del file da leggere
- definisca una struttura dati idonea a contenere i dati lettti (struct CData)
- la struct abbia un metodo from_file in grado di leggere il dati da file aperto
- memorizzare i dati letti in un array di struct CData

Attenzione: vi sono sono due modi possibili per leggere il contenuto del file binario in Rust e creare degli oggetti:

- leggere i byte degli attributi uno per uno e poi convertirli nel tipo desiderato con il trait from_le_bytes / from_be_bytes
- leggere tutta la struct in un byte buffer una volta noto tipo e dimensione e poi reinterpretarla come struct Rust

Nel secondo caso si deve usare del codice unsafe, perché? Quale dei due approcci è più efficiente e perché? E possibile con tutte le struct definite o vi sono problemi?