Programming with Persistent Data Assignment 2016

Dr. Pierpaolo Dondio

(15% of Final Exam marks. Submission Date: Friday 6th May)

The assignment is divided into 3 exercises. You are required to submit the C code , the compiled exe code and a quick 1 page description (for each exercise) of how you solved each problem.

1. Scramble a File (4 marks)

You are required to create two utilities, scramble.c and unscramble.c, that are called by command line specifying the name of the file to scramble/unscramble. For example:

```
C:> scramble myfile.txt
```

```
C:> unscramble myfile.txt.sbl
```

The scramble utility reads a file (any type of file) and generates a new file with the same name as the input file but with the extension .sbl added to the end of the file (keep the previous extension, for istance picture.bmp becomes picture.bmp.sbl). The .sbl file is generated by swapping bytes two by two. For instance, the 1st byte is swapped with the 2nd, the 3rd with the 4th and so on..

For instance if the input file is

byte 1	byte 2	byte 3	byte 4	…	byte n-1	byte n

The output file will be:

byte 2	byte 1	byte 4	byte 3	…	byte n	byte n-1

If the input file has an odd number of bytes the last one will not be swapped.

The *scramble* utility encrypts a file so that it cannot be read anymore by the right application.

The utility *unscrumble* does the reverse task: it takes a scrambled file .sbl and recreate the original file by removing the .sbl extension from the name and reswapping back the bytes.

Try the program with any type of file and check if, by scrambling and unscrambling a file, the resulting file is the same as the original starting file.

2. Soccer League (6 marks)

The file results.txt contains the results of a fanstasy league with 20 european national teams. Each line of the text file contains the result of a single match in the following format:

```
team1 team2 goals1 goals2
```

Example

```
Ireland England 2 1

Italy France 2 2

Austria Poland 1 3
```

…

the terms are separated by a space character.

You are required to:

a. Create a structure *st_match* that can holds the result of a single match (team1, team2, goals1, goals2), and save the data stored in *results.txt* into a binary file of structures *st_match* called *results.bin*
You need to read all the text file and save the data into the binary file using the structure you have defined.

b. Create a program *team_stats.c* that asks the user to enter the name of a team from the keyboard. If the team is valid (one of the 20 teams taking part to the league), the program shows the following statistics (for example, if the team entered was `Ireland`):

```
Team: Ireland

Match Played: 20

Win: 10

Draw: 6

Loss: 4

Goals Scored: 43

Goals Conceided: 31
```

Note that the numbers shown are just an example, they do not correspond to the data stored in the file.
Important: you are required to read the data from the binary file results.bin created at point a.

c. Create a program *table.c* that displays the league table. For each team you are required to show only the total points (3 points for a win, 1 for a draw, 0 for a loss).
[OPTIONAL] Two additional marks if the league is sorted in descending order

(Note that a,b,c are all 2 marks each)

3. Read MP3 Information (5 marks)

You are required to create an utility called *read_mp3.c* that is called by command line specifying the name of the file to analyze. For example:

```
C:> read_mp3 mysong.mp3
```

The utility is supposed to get some metadata information from the mp3 file by reading the MP3 audio tag stored in the file, and display the following information on the screen.

```
Title of the song

Author of the song (artist)

Album

Year

Genre
```

Some of the information could not be available. In that case an "Unknown" value should be displayed.

We use only MP3 files that contain a valid MP3 audio tag ID3 version 1.

The mp3 audio tag ID3v1 format is displayed in the next page.

You need to read the audio tag at the end of the file, save into the right structure in memory and analyse the data. In order to check if the mp3 file contains a correct audio tag, check it the first 3 bytes contain the letter 'T' , 'A' , 'G' . If not, display an error message "The file does not contain a valid audio tag" and terminate the program. If the audio tag is valid, you are required to display Title, Artist, Album, Year and Genre.

MPEG Audio Tag ID3v1 Documentation

The TAG is used to describe the MPEG Audio file. It contains information about artist, title, album, publishing year and genre. There is some extra space for comments. It is exactly 128 bytes long and is located at very end of the audio data. You can get it by reading the last 128 bytes of the MPEG audio file.

```
AAABBBBB BBBBBBBB BBBBBBBB BBBBBBBB
BCCCCCCC CCCCCCCC CCCCCCCC CCCCCCCD
DDDDDDDD DDDDDDDD DDDDDDDD DDDDDEEE
EFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFG
```

Sign	Length (bytes)	Position (bytes)	Description
A	3	(0-2)	Tag identification. Must contain 'TAG' if tag exists and is correct.
B	30	(3-32)	Title
C	30	(33-62)	Artist
D	30	(63-92)	Album
E	4	(93-96)	Year
F	30	(97-126)	Comment
G	1	(127)	Genre

There is a small change proposed in **ID3v1.1** structure. The last byte of the Comment field may be used to specify the track number of a song in an album. It should contain a null character (ASCII 0) if the information is unknown.

Genre is a numeric field which may have one of the following values:

0	'Blues'	20	'Alternative'	40	'AlternRock'	60	'Top 40'
1	'Classic Rock'	21	'Ska'	41	'Bass'	61	'Christian Rap'
2	'Country'	22	'Death Metal'	42	'Soul'	62	'Pop/Funk'
3	'Dance'	23	'Pranks'	43	'Punk'	63	'Jungle'
4	'Disco'	24	'Soundtrack'	44	'Space'	64	'Native American'
5	'Funk'	25	'Euro-Techno'	45	'Meditative'	65	'Cabaret'
6	'Grunge'	26	'Ambient'	46	'Instrumental Pop'	66	'New Wave'
7	'Hip-Hop'	27	'Trip-Hop'	47	'Instrumental Rock'	67	'Psychadelic'
8	'Jazz'	28	'Vocal'	48	'Ethnic'	68	'Rave'
9	'Metal'	29	'Jazz+Funk'	49	'Gothic'	69	'Showtunes'
10	'New Age'	30	'Fusion'	50	'Darkwave'	70	'Trailer'
11	'Oldies'	31	'Trance'	51	'Techno-Industrial'	71	'Lo-Fi'
12	'Other'	32	'Classical'	52	'Electronic'	72	'Tribal'
13	'Pop'	33	'Instrumental'	53	'Pop-Folk'	73	'Acid Punk'
14	'R&B'	34	'Acid'	54	'Eurodance'	74	'Acid Jazz'
15	'Rap'	35	'House'	55	'Dream'	75	'Polka'
16	'Reggae'	36	'Game'	56	'Southern Rock'	76	'Retro'
17	'Rock'	37	'Sound Clip'	57	'Comedy'	77	'Musical'
18	'Techno'	38	'Gospel'	58	'Cult'	78	'Rock & Roll'
19	'Industrial'	39	'Noise'	59	'Gangsta'	79	'Hard Rock'
80	'Folk'	92	'Progressive Rock'	104	'Chamber Music'	116	'Ballad'

81	'Folk-Rock'	93	'Psychedelic Rock'	105	'Sonata'	117	'Poweer Ballad'
82	'National Folk'	94	'Symphonic Rock'	106	'Symphony'	118	'Rhytmic Soul'
83	'Swing'	95	'Slow Rock'	107	'Booty Brass'	119	'Freestyle'
84	'Fast Fusion'	96	'Big Band'	108	'Primus'	120	'Duet'
85	'Bebob'	97	'Chorus'	109	'Porn Groove'	121	'Punk Rock'
86	'Latin'	98	'Easy Listening'	110	'Satire'	122	'Drum Solo'
87	'Revival'	99	'Acoustic'	111	'Slow Jam'	123	'A Capela'
88	'Celtic'	100	'Humour'	112	'Club'	124	'Euro-House'
89	'Bluegrass'	101	'Speech'	113	'Tango'	125	'Dance Hall'
90	'Avantgarde'	102	'Chanson'	114	'Samba'		
91	'Gothic Rock'	103	'Opera'	115	'Folklore'		

Any other value should be considered as 'Unknown'