CSC10002 – Advanced Programming

Lab 04
Binary File IO

Cảm ơn thầy Trần Duy Quang đã cung cấp template cho môn học

Department of Software Engineering-FIT-VNU-HCMUS

1

Notes

Create a single solution/folder to store your source code in a week.

Then, create a project/sub-folder to store your source code of each assignment.

The source code in an assignment should have at least 3 files:

- A header file (.h): struct definition, function prototypes/definition.
- A source file (.cpp): function implementation.
- Another source file (.cpp): named YourID_Ex01.cpp, main function. Replace 01 by id of an assignment.

Make sure your source code was built correctly. Use many test cases to check your code before submitting to Moodle.

Name of your submission, for example: **18125001_W01_07.zip**

2

Content

In this lab, we will review the following topics:

- How to read / write a binary file?

3 Assignments

A: YY: 01 problem / assignment.

H: YY: 05 problems / assignments.

Command line arguments requirement is optional.

3.1 Copy file

Use command prompt argument and binary file IO techniques to create a MYCOPYFILE program.

User will enter:

```
MYCOPYFILE -s path_of_source_file -d path_of_destination
```

The destination file will have same name as the source file.

For example:

```
MYCOPYFILE -s D:/film.mkv -d D:/Level1/Level2/Level3
```

3.2 Split file

Use command prompt argument and binary file IO techniques to create a MYSPLITFILE program.

User will enter:

```
MYSPLITFILE -s path_of_source_file -d path_of_destination -numpart x
```

x is a positive integer number, number of splitted parts.

OR

```
MYSPLITFILE -s path_of_source_file -d path_of_destination -sizeapart x
```

y is a positive integer number (in bytes), size of a splitted part.

For example:

```
MYSPLITFILE -s D:/film.mkv -d D:/Level1/Level2/Level3 -numpart 3
```

OR

```
MYSPLITFILE -s D:/film.mkv -d D:/Level1/Level2/Level3 -sizeapart 1000000
```

Names of splitted parts are: film.mkv.part01, film.mkv.part02, film.mkv.part03…

3.3 Merge file

Use command prompt argument and binary file IO techniques to create a MYMERGEFILE program.

User will enter:

```
MYMERGEFILE -s path_of_part01 -d path_of_destination
```

The program will find all *.part01, *.part02, *.part03… files and merge into a single file.

The program will print errors if there is any part does not exist.

For example:

```
MYMERGEFILE -s D:/Level1/Level2/Level3/film.mkv.part01 -d D:/NewLevel
```

3.4 Polynomial

Question 4 (2.5 points)

A binary file POLY.BIN stores a number of polynomials in single indeterminate x as follow:

	Byte Order	Size	Description
Polynomial #0	00:	2 bytes 8 bytes ... 8 bytes	$<n_0>$: a 2-byte integer, number of terms of polynomial #0. -$<deg_0><coef_0>$: a pair of 4-byte floating numbers, degree and coefficient of monomial #0 in polynomial #0. ... $<deg_{n0-1}><coef_{n0-1}>$: degree and coefficient of monomial #n_0-1 in polynomial #0.
Polynomial #1	$2+n_0*8$:	2 bytes 8 bytes ... 8 bytes	$<n_1>$: number of terms of polynomial #1. $<deg_0><coef_0>$: degree and coefficient of monomial #0 in polynomial #1 ... $<deg_{n1-1}><coef_{n1-1}>$: degree and coefficient of monomial #n_1-1 in polynomial #1
...	...	...	...

a) Write function to read from file POLY.BIN all polynomials into memory (RAM).

b) Write function write to file POLY_LARGEST.BIN in the same format of POLY.BIN polynomials which have the largest degree.

3.5 TAR File

Question 5 (2 points)

The TAR file is an archive file format commonly used in LINUX systems. It stores files in sequence. Each file stored in the TAR file consists of header block and data block. Header block is a 512-byte block, containing metadata about the stored file (file name, file size, ...). Data block starts immediately after header block, containing actual data of the stored file. Data block are padded with zero bytes to make it multiple of 512 bytes. At the end of the TAR file, there are two 512-byte empty blocks filled with zero bytes to mark the end.

```
[Header block of file 1]
[Data   block of file 1]
...
[Header block of file N]
[Data   block of file N]
[Empty  block           ]
[Empty  block           ]
```

There are two notable fields in header block:

- File name: byte offset 0, byte size 100, null-terminated string representing file name padded with zero at the end.
- File size: byte offset 124, byte size 12, null-terminated string representing file size in octal padded with zero at the beginning.

Example:

```
file name test.txt stored as: test.txt\0\0\0...
file size 426 bytes (decimal) => 652 (octal) stored as:
```

'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'6'	'5'	'2'	'\0'

a) Write function `long long fileSizeToDecimal(char fileSizeField[12])` to convert file size field to decimal number.

b) Write function `void printFiles(const char *tarfile)` to print to screen file name and file size of each stored file in a TAR file, using function written in a).