Design Document

Introduction

This software document is prepared to explain CMPE230 PROJECT #2. First I will explain the parsetrans part, then I'll pass to the filestats.

For both programs, I will first try to explain my main code and the algorithms I used. While I'm trying to explain the code, I will explain what kind of data structures I used, too. After that, I will explain my subroutines.

Parsetrans.pl

The program first gets the command-line arguments by getfiles() subroutine. After that, it opens the given required, complementary and hss files to get the lecture lists. It simply finds correct places to read, then using regular expressions, it pushes the lecture code to the corresponding array. The name of the arrays I used here are @reqlist,@hsslist and @cclist. These arrays will be used to find the correct place of the taken course. While getting the required lectures, the program counts the hss and the cc occurrences in the list, but it does not push them to the array. Later in the program I will use these numbers to understand whether the student has taken enough complimentary and hss courses to graduate or not. Taking the hss list was quite simple, nothing extraordinary. Taking the complementary courses list was a bit complicated than the others, because in this part, the courses may be given in groups and there are some exceptions from the groups. The code should handle all of them. My algorithm is simple to take complementary courses correctly. First the program searches the line which consists lectures, then it pushes all elements in a temporary array. After that, it pushes all elements of this array to @cclist up to the “except” statement. While it is pushing, if an element have – in it, which means between, the program pushes all the elements starting from the smaller to larger. Entries coming after the except statement should not be in the @cclist, so it erases the corresponding entries from @cclist. The space complexity of this algorithm may be a bit large than some other approaches, but even if the @cclist array consists of 400 elements, the memory space taken for the array will be nothing at all. Also, the algorithm was easy to write.

The second part of the program, actually the main part, will start now. The program opens the transcript file with HTML::TokeParser. All information needed is given in the table, which has the attribute “ID=Table3”. So, program reads the file up to there. If it couldn't find Table3, which means the transcript file is over, the program passes to the third part, which prints the output. I will now explain what program does after finding the correct table. The program uses getlectures subroutine after finding the correct place. I will explain getlectures after finishing the explanation of the main code. This subroutine simply returns the array, which contains the lecture information in it. After taking the lecture information, the program checks if the lecture was in the repeat list. Then, it pushes the lecture code if the lecture is being repeated, i.e there exists a “R“ in the information array. After that, the lecture code is compared with the @reqlist, @hsslist and @cclist if it is not withdrawn, failed, repeated or the course has not been graded yet. Of course, first the lecture is compared with the @reqlist, and if it is not found, then it is compared with the @hsslist, so on. If the lecture is found somewhere in the arrays, it is pushed into the corresponding array. If the lecture cannot be found anywhere, it is recorded as other. This procedure is continued up to the end of the file. Here, I used @requiredcourses, @hsscourses, @cccourses and @othercourses to record the type of the lecture. After classifying all of the lectures, the program checks if the student has taken enough courses to graduate. This is done by checking all of the @requiredlist elements occur in the @requiredcourses array. Also @hsscourses and @cccourses arrays have to consist enough number of elements. This enough number had been recorded in the $cccount and $hsscount variables while the program was taking the required courses from the file. If student did not take enough courses, the code of the course is pushed into @nottaken array.

The third part of the program prints lists. It does not have to be explained, it is simply prints all elements of the corresponding arrays.

Subroutines

getlectures

This subroutine takes all information about a lecture. The information can consist of lecture name, lecture code, credits, grade; withdraw, fail or repeat status and if it is repeated, the code of the repeated lecture.

We know that given information is in table with the attribute ID=Table3. So, the code scans all tokens until it reaches end of the table "</table>" tag. It scans all of the tokens like this: First it deletes all whitespace characters from the token, then it deletes the &nbsp; special character if it exists. If some characters remained, these must be the information. So, it is pushed to the @lectures array. At the end of the subroutine, this array is returned.

getfiles

This subroutine takes the given command-line arguments. It scans the pattern -t, -r, -c, -h, --h or –help. Only last two pattern is extra. It shows the usage and arguments of the program. Anything other than these will be understood as a mistake. The string coming after first four arguments is pushed to the @filelist array and the array is returned at the end of the subroutine. Of course there is an order in the @filelist. This order is as follows: transcript_file, required_list_file, complimentary_list_file, hss_list_file.

I added some default filenames to simplify the usage of the program. Maybe there exists 50 transcripts of which their complimentary file, required file and hss file are the same. These default values are:
Transcript file: student.html
Required courses file: required.txt
Complimentary courses file: cc.txt
Hss file: hss.txt

Notes:
i- The output of my program includes not only the course code, but also the name. If this wanted to be changed as only the course code, a little modification in the code should be done. That is, delete ",$lectures[1]" from the lines 182,197,213,220 and the comment mark # from the line 229.
ii- I tried to fix most of the warnings. Do not care the remaining one.

Sample Output

```
[bosisler@bosisler proje2]$ parsetrans.pl --help

--h or --help: Prints this message
-t:Transcript file
-r:Required_courses_file
-c:cc_file
-h:hss_file

Default values:

Transcript File:student.html
Required_courses_file:required.txt
cc_file:cc.txt
hss_file:hss.txt

[bosisler@bosisler proje2]$ parsetrans.pl -t deneme.html -c cc.txt -h hss.txt -r required.txt



REQUIRED COURSES

CMPE450.01 SOFTWARE ENGINEERING
MATH202.01 DIFFERENTIAL EQUATIONS
CMPE350.01 FORMAL LANGUAGES & AUTOMATA THEORY
CMPE492.01 PROJECT
EE  212.01 INTRODUCTION TO ELECTRONIC ENGINEERING
IE  306.01 SYSTEMS SIMULATION
MATH343.01 PROBABILITY
CMPE321.01 INTRODUCTION TO DATABASE SYSTEMS
CMPE300.01 ANALYSIS OF ALGORITHMS
CMPE322.01 OPERATING SYSTEMS
CMPE344.01 COMPUTER ORGANIZATION
IE  310.01 OPERATIONS RESEARCH
CMPE240.01 DIGITAL SYSTEMS
CMPE250.01 DATA STRUCTURES & ALGORITHMS
EE  210.02 INTRODUCTION TO ELECTRICAL ENGINEERING
MATH201.02 MATRIX THEORY
CMPE220.01 DISCRETE COMPUTATIONAL STRUCTURES
CMPE230.01 SYSTEMS PROGRAMMING
PHYS202.01 PHYSICS IV
CMPE160.01 INTERMEDIATE PROGRAMMING
EC  102.08 INTRO.TO ECONOMICS II
ENGG110.01 ENGINEERING GRAPHICS
MATH102.04 CALCULUS II
PHYS201.02 PHYSICS III
```

CHEM105.03 FUNDAMENTALS OF CHEMISTRY
CMPE150.01 INTRODUCTION TO COMPUTING(PASCAL,C)(CMPE
EC 101.03 INTRODUCTION TO ECONOMICS I
MATH101.03 CALCULUS I
PHYS121.01 INTRODUCTORY MECHANICS & THERMODYNAMICS

HSS COURSES

PSY 101.01 INTRODUCTION TO PSYCHOLOGY I

COMPLIMENTARY COURSES

CMPE422.01 DATABASE SYSTEMS
CMPE494.01 SP.TOP:SERVICE ORIENTDE ARCHIT.AND WEBSE
CMPE477.01 WIRELESS & MOBILE NETWORKS
CMPE471.01 INFORMATION SYSTEM SECURITY
CMPE472.01 FUNDAMENTALS OF ELECTRONIC COMMERCE
CMPE475.01 COMPUTER NETWORKS

OTHER COURSES

BM 402.01 ENGINEERING IN MEDICINE
CMPE320.01 PRINCIPLES OF PROGRAMMING LANGUAGES
HEB 101.01 ELEMENTARY HEBREW I
HTR 406.01 HIST. OF THE TURKISH REPUBLIC FOR F.S.II
HTR 405.01 HIST.OF THE TURKISH REPUBLIC FOR F.S.I
TK 316.01 ADVANCED TURKISH GRAMMAR II
PE 123.05 TENNIS I
TK 315.01 ADVANCED TURKISH GRAMMAR I

Student has to take these courses in order to graduate:

TK221Turkish
TK222Turkish
HTR311HistoryofTurkishRepublicI
CMPE360NumericalMethods
HTR312HistoryofTurkishRepublicII
CC
CC
HSS

Note about this transcript: The owner of this transcript is a foreign student. So, he does not need to take HTR311, HTR312, TK221 and TK222. But required list is not for foreign students.

filestats.pl

The program first starts with hash definitions and initializations. These hashes are:
%filelist: key: filename value: filesize
%filepath: key:filename value: filepath
%filetype: key:filename value: filetype
%duplist: key:duplicate filename value: filesize
%duppath: key:duplicate filename value: filepath
%filesize: key:filetypes value: totalsizes
%filenum: key:filetypes value: total count

These information can be stored in a smaller number of hashes. But then the elements of the hashes will be arrays, not scalars. As a result, the implementation and understanding of the code will be really harder.

After these hashes, the program takes command-line options by getoption subroutine. It is nearly the same as the getfiles subroutine in the other program. After that, the program opens the directories and gets files and directories. This part of the code is taken from the code we wrote on the lecture. Of course, there are many additions to that code. First, the program opens the given directory. If it cannot open, it prints a warning message. This situation can occur if program tries to open some directories that the user does not have permission to open. In this situation, the final output of the program can be different than the user expects. To avoid it, the program should be used with root permissions. After opening the directory, the program gets elements in that directory. Here we must check the . and .. directories, links and non-plane files. So the program first checks the . and .. directories. If the element is one of them, it is skipped and the other element is taken. Then links are checked. The program checks links because some links can link the outer directory or the directory that program traverses at that time. So, infinite loops occur if the program enters these link locations. The program simply checks that and if the element is a link, fileoper subroutine with “other” parameter is called. Then next element is taken. The program then checks plane file condition. If a file is not plain, it is skipped. After that, the element is checked to understand if it is a directory. If it is a directory, it is pushed into the @subdirlist array. If a file skips all of these controls, it means that it is a simple file. The program then determines the type of the file by looking at the extension and then fileoper subroutine with appropriate parameter is called.

After all of these, duplicate file control starts. The program controls files with MD5 checksum which has both same name and size. If duplicates are found, it is added to %duplist and %duppath hashes. The %duppath hash takes all of the paths in a single scalar variable with a seperation mark “\n”. If a file has no duplicates, it is added to %filepath, %filelist and %filetype hashes. Then, other element is taken in the directory. After traversing all of the elements in the directory, the program pushes all subdirectories to @dirlist to traverse. This traversing procedure is continued until all elements of @dirlist is finished.

The next part is about searching the pattern and printing the results in a formatted way. If no pattern is given, program sorts the file sizes to find the largest 10 file. Most part of this is done by find_largest subroutine. It returns largest 10 files of the given hash. After that, program writes the first table. If a pattern is given, it is searches the pattern in the %filelist and %duplist hashes. Then it sorts sizes of the files again and find_largest works again. Then, the program writes the changed first table. After that, the program displays the other tables. Here, writing top10 files table is a bit complicated. Because of using the filename as a hash key, the program can store only one path. But if a file has duplicates and is in the top10 list, the program should show these. So, the program checks whether the file has duplicates or not. After that, it writes the correct output.

Subroutines

getoption

This subroutine takes command-line arguments. It checks if a relative or a fullpath is given. It returns the array of parameters. First parameter is pattern, the second one is the path.

fileoper

This subroutine simply increments the count and size of the appropriate type. It returns the type.

get_md5

It returns the MD_5 hex digest of the given file.

find_largest

This subroutine finds and returns keys which contains the given values of the given hash. It takes 3 parameters; the hash to search for, the duplicate hash and values to search for. It first checks the first hash. When it finds the given value, it pushes the key of the hash to the @largest array. Then it checks if this value is also in the duplicates. If it finds, it searches how many times does this duplicate occur. Then it pushes the key that times.

Notes: These notes can be found in the explanation part. However, they are important, so I write here again:

1) The program should be executed in root permissions to prevent possible wrong output.
2) Two files are duplicates if and only if their names, sizes and MD5 sums are equal in my program.

Sample Output

General notes: Some parts of my code may be done more efficient than this, but the specification of the project was poor and because of this I understood some parts of the project when I was implementing. So, my solutions to these new problems sometimes be fast and dirty. I didn't try to write my code most efficient (and most time consuming for me) way because it is not very important for these projects. Sorry for that.