ENGG 1110 Problem Solving By Programming 2017-2018 Second Term
Faculty of Engineering
The Chinese University of Hong Kong

Due date on BB: 17 April 2018 (Tue) Project Phase 3 Full mark: 100 pts [30% of Project]
Late cut-off: 24 April 2018 (Tue) Late penalty: 20 pts [6% of Project]

Presentation at Lab: 18 April 2018 (Wed) Presentation: 100 pts [10% of Project]

Course Registration Information System Phase 3 (curegis3)

Aim: 1. Complete the system.
2. Practise file I/O and data processing.
3. Practise using arrays and structures.

Background:

This phase completes the whole system. We are going to implement the back-end data processing engine that performs file operations. Assume that there are already a list of courses and a list of grades stored in two given files, our program shall read the files, process the given data and generate reports. In terms of programming technique, we shall define and call C functions to process data stored in arrays and structures.

Requirements:

Design and implement the application in C.

- Firstly, create a new Code::Blocks Console application C project **`Phase3`**.
- Secondly, copy our given code into your main source file. Assimilate the given code into your own work.
- Lastly, check your header comment block showing your name, SID, declaration on academic honesty, etc. Make sure you have proper header file inclusion lines.

The program shall perform the following tasks:

a) Display a banner message.

b) Open, read and store course information into a defined data structure. Filename is **`courses.txt`** and the file format is defined following this example:

```
ENGG1110 3 Problem Solving by Programming
CSCI1130 3 Introduction to Programming Using Java
CSCI2040 2 Introduction to Programming Using Python
ENGG2602 1 Engineering Practicum
MATH1110 3 Elementary Mathematics
UGFX7788 4 Perspectives in University General Education
CSCI2100 3 Data Structures
PTHE1000 1 Putonghua I
ENGG1000 1 IT Foundation
END
```

Each line contains three pieces of information separated by spaces: 8-char (4 capital letters

plus 4 digits) course code, a single digit representing number of credit units, and the course name that spans until the end of line. Note that there may be spaces in the course names and each course name shall be less than 80 characters. You shall use **`fgets()`** to read the course names. The last line in the file shall be **`END`**. There are **at most 20 courses**.

c) Open, read and store grade information into a defined data structure. Filename is **`grades.txt`** and the file format is defined following this example:

```
1156493117 ENGG1000 B
1156172841 ENGG1000 B+
1156568410 CSCI2100 D
1156873296 ENGG1000 C
1156627958 UGFX7788 C-
...
1156050264 UGFX7788 B-
1156627958 ENGG1000 A-
1156709121 CSCI2100 B-
1156050264 ENGG1110 C+
1156937468 PTHE1000 A
...
1156493117 CSCI2040 B-
1156221557 PTHE1000 F
1156389229 ENGG1000 B
END
```

Each line contains three pieces of information separated by spaces: 10-digit SID, 8-char course code, and a course grade before the end of line. The last line in the file shall be **`END`**. There are **at most 100 grade records**. A student may take **at most 9 unique courses**.

d) Generate a master class-lists file named **`classlists.txt`** in the following format. Here is an example:

```
2 students in CSCI1130
1156389229 C+
1156627958 C+
4 students in CSCI2040
1156221557 F
1156493117 B-
1156568410 C
1156709121 C-
...
3 students in PTHE1000
1156050264 C-
1156221557 F
1156937468 A
4 students in UGFX7788
1156050264 B-
1156172841 B
1156389229 B+
1156627958 C-
```

It lists the courses in ascending order. Each block shows the number of students in a course as well as the SID in ascending order, together with the student's grade separated by a space.

e) Generate a master transcripts file named **`transcripts.txt`** in the following format.

Here is an example:

```
1156050264 took 4 courses
ENGG1000 D+ x1 [IT Foundation]
ENGG1110 C+ x3 [Problem Solving by Programming]
PTHE1000 C- x1 [Putonghua I]
UGFX7788 B- x4 [Perspectives in University General Education]
GPA = 2.30
1156172841 took 5 courses
ENGG1000 B+ x1 [IT Foundation]
ENGG1110 A x3 [Problem Solving by Programming]
ENGG2602 F x1 [Engineering Practicum]
MATH1110 B+ x3 [Elementary Mathematics]
UGFX7788 B x4 [Perspectives in University General Education]
GPA = 3.10
...
1156937468 took 3 courses
ENGG1000 B- x1 [IT Foundation]
ENGG2602 C- x1 [Engineering Practicum]
PTHE1000 A x1 [Putonghua I]
GPA = 2.80
```

It lists the SID in ascending order. Each block shows the number of courses taken and the transcript of a student, with each line showing:

1. Course code (in ascending order)
2. Grade
3. Number of units (x-units)
4. Course name in a pair of square brackets [ ]

It also records the GPA (with 2 decimal places) of each student by calculating a weighted-average of the grade points with the units.

f) In case of any error during the operations, print **"Error!"** and terminate the program.

Sample Run: (NO user input required)

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
Course Registration Information System Phase 3
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
Reading courses.txt
Read 9 courses
Reading grades.txt
Read 40 records
Writing classlists.txt
Wrote 9 classlists
Writing transcripts.txt
Wrote 10 transcripts
Done!
```

Given Data Structures and Function Prototypes:

```
typedef struct course {
    /* basic info to be read from courses.txt */
    char code[8+1];
    int  unit;
    char name[80+10+1];     // allow 10 more dummy char as a buffer

    /* extracted info during data processing */
    int  student_count;
    int  grade_record_index[100]; // index into grade records
} course_type;


typedef struct grade_record {
    /* basic info to be read from grades.txt */
    int  sid;
    char code[8+1];
    char grade[2+1];
} grade_record_type;


typedef struct student {
    /* extracted info during data processing */
    int  sid;
    int  course_count;
    int  course_index[9];  // a student can take at most 9 courses
    double gpa;
} student_type;


// function for reading courses.txt into basic info of courses[] array of struct
// return number of courses read
int read_courses(course_type  courses[]);


// function for reading grades.txt into basic info of grades[] array of struct
// return number of grade records read
int read_grades(grade_record_type  grades[]);


// function for extracting info from grades[] array into courses[] array
// e.g. fill in courses[0].student_count and courses[0].grade_record_index[...]
//      fill in courses[1].student_count and courses[1].grade_record_index[...]
// this is just one of the many possible ways for extracting the classlists
// you may solve this problem in your own way
void extract_classlists(course_type  courses[], grade_record_type  grades[]);


// function for sorting, processing and writing to classlists.txt
void write_classlists(course_type  courses[], grade_record_type  grades[]);


// function for extracting info from grades[] and courses[] array into students[]
// e.g. fill in students[0].course_count and students[0].course_index[...]
//      fill in students[1].course_count and students[1].course_index[...]
// this is just one of the many possible ways for extracting the transcripts
// you may solve this problem in your own way
void extract_transcripts(course_type  courses[], grade_record_type  grades[],
                         student_type students[]);
```

```
// function for mapping a grade to a grade point
// return a corresponding grade point in 0.0 - 4.0
double map_grade_point(char grade[]);


// function for sorting, processing and writing to transcripts.txt
void write_transcripts(course_type  courses[], grade_record_type  grades[],
                       student_type students[]);


int main(void) {
    int           course_count;
    course_type   courses[20];          // at most 20 courses in the array

    int                grade_record_count;
    grade_record_type  grades[100];     // at most 100 grade records in the array

    int                student_count;
    student_type       students[100];   // at most 100 students!

    // call functions to complete the task
    course_count = read_courses(courses);  // for example

    return 0;
}
```

Guidance and Submission:

1. A sample Excel spreadsheet file is provided for your reference and self-generation of test data sets. Note that uniqueness constraint is NOT implemented in the spreadsheet such that you may need to manually delete or modify duplicated grade records to avoid students taking the same course more than once. Saving the worksheets **`courses`** and **`grades`** as Tab Delimited Text (.txt) files will generate test files that resemble to our requirements.

2. Use plural word form and past tense in all applicable situations to simplify the task.

3. Try printing information read from the input files to the screen to aid debugging.

4. Edit, Build (Compile), Run and Debug your program. If you do something wrong, don't panic. Double-click on the first error message. Check it, correct it and retry. Remember that a single mistake may trigger dozens of error messages. Always begin tackling the first error and conquer one at a time. Be reminded that the error message itself as well as the indicated line number may not be accurate.

5. Thoroughly Test Run your program with different input data sets such as extreme cases and boundary cases. Prepare and save your test input files in proper folder location such that your program can read them. You shall design your own test plan and test cases. Check and compare your program output carefully. You may use online diff tools to inspect your output files.

6. Locate your Code::Blocks project folder **`Phase3\`** and
   Upload and Submit two files **`Phase3.cbp`** and **`main.c`** via Assignment Collection Box on **`https://blackboard.cuhk.edu.hk`**. We will examine your Code::Blocks project to assess you work on project management, coding style and/or running more tests.

Presentation Arrangement and Details:

Course teachers will announce presentation schedule and arrangements for individual sections. Students shall attend the last lesson at the lab and present the work. Each student is allocated 8 – 10 minutes to present the work in English and to conduct Q&A with a teacher or a TA. Even if your work is incomplete OR you plan to submit late, you shall still present your work as much as possible.

Marking Scheme and Notes:

1. The submitted program should be free of any typing mistakes, compilation errors and warnings. Please make sure your submitted work can be compiled successfully on Code::Blocks, our official platform.

2. Your proper declaration statement, your name and SID, file naming, adequate comment, code indentation, coding style such as variable naming, etc. are under assessment in every programming assessments unless specified otherwise.

3. Output formatting will be taken into account, e.g., word spellings, spaces, number formats, etc.

4. The organization and structure of your work will be considered. For example, you shall define and call some functions as specified. Use also the given data structures.

5. ***Remember to do your submission on Blackboard*** by the due date. Late submission made within our grace period will be accepted with late penalty. No further late submission may be accepted.

6. If you submit multiple times, **ONLY** the content and time-stamp of the **latest** one would be counted. You may delete (i.e. take back) your attached file and re-submit. We ONLY take into account the last submission.

7. Markers will check your work vigorously.

University Guideline for Plagiarism

Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations. Details may be found at http://www.cuhk.edu.hk/policy/academichonesty/. With each assignment, students are required to submit a statement that they are aware of these policies, regulations, guidelines and procedures.