BİL 102 – Computer Programming
HW 10 + 11

Last Submission Date: May 25, 2015 – 14:00pm

For Questions about this homework see T.A. Evren Çifçi
(e-mail: ecifci@bilmuh.gyte.edu.tr Office No: 123 – Robot Control Lab)

In this homework, you will implement a simple program to process some appointment data of some patients. All appointments start at the beginning of an hour. Basically, your code will:

- Read some appointment records from a **records file** (binary file) into a dynamically allocated array (Part1),
- Build a linked-list holding the valid appointment records in this array (Part 2),
- Handle the following operations on the link-list :
  - Adding patient history, taken from the **patients file** (text file), to each appointment in the list (Part 2),
  - Deleting a list of canceled appointments, taken from a **delete file** (text file) (Part 2),
  - Deallocating the dynamic data (Part 2),
  - Making an independent copy of the list (Part 3).

Inputs:

Records File: A binary file to hold all appointment requests. The first data of this file is the number of records in the file and the other data are the records of type Appointment_t. The default name for this file is "**Records.bin**". An example is as follows:

4	Appointment_t	Appointment_t	Appointment_t	Appointment_t
	app_id:8 patient_id:2 hour:11	app_id:3 patient_id:5 hour:19	app_id:11 patient_id:2 hour:14	app_id:203 patient_id:20 hour:11

Patients File: An XML text file to hold information about all patients. XML is standard aiming to make a text file easy readable by both human and computer. In an XML file, data is stored in some tags. A tag start and end with a special sequence as follows:

<tag_name>Data inside the tag</tag_name>

Tags can also be nested as follows:

<a>

Data for b field of record a

<c>Data for c field of record a</c>

All whitespace characters between tag definitions are ignored. A tag may be empty (include no data) as Fatma Zeki's history in the example below.

Patients file has the following tags:

Records: All data about all patients exist in this tag

Patient: All data about a patient are in this tag

ID: id number of a patient is stored in this tag

Name: Name of a patient is stored in this tag.

History: Medical history about a patient is stored here

An example is as follows:

```
<Records>
	<Patient>
		<ID>5</ID>
		<Name>Ali Veli</Name>
		<History>On 10 12 2012 applied with an headache, diagnosed with flue, purposed to use Majezik and rest. On 12 8 2014 applied with headache. Nothing done.</History>
	</Patient>
	<Patient>
		<ID>2</ID>
		<Name>Fatma Zeki</Name>
		<History></History>
	</Patient>
                                        ….....
</Records>
```

The default file name for patients file is “**Patients.xml**”.

Delete File: A text file in which the appointment id's of canceled appointments are stored. Each entry exist in a separate line in the following example:

```
10
5
2
```

The default file name for delete file is “**Delete.txt**”.

Your implementation should never expect any inputs from console.

Outputs:

Readable Records file: An XML text file to hold the contents of some Appointment_t objects. This file should have the following tags:

Size: Number of records exists here

Records: All data about all appointments exist in this tag

Appointment: All information about an appointment exists here

Variables of Appointment_t objects: All variables should exist as tags.

Ex: First 2 records in the example of records file above should be stored as follows:

```
<Size>2</Size>
<Records>
	<Appointment>
		<app_id>8</app_id>
		<patient_id>2</patient_id>
		<hour>11 </hour>
	</Appointment>
	<Appointment>
		<app_id>3</app_id>
		<patient_id>5</patient_id>
		<hour>19</hour>
	</Appointment>
</Records>
```

The order of tags should be the same as it is in the example above and there should be no other data except the above defined tags. The default file name is "**Records.xml**".

Accepted Appointments File: A csv text file holding the data of an appointment including the personal data of the patient (all data of a node_t type variable except the 'next' field). Csv is a standard accepted by spreadsheet editors (such as Excel, LibreOffice Calculator, etc.) in which data for each row exists in a separate line and columns are delimited by a special character -use ';' in your implementation. Use the column names: no, id, patient_id, name, history and hour

Ex: The records for the 2 appointments stored in the example of readable records file are:

```
no;id;patient_id;name;history;hour
1;8;2;Fatma Zeki;;11
2;3;5;Ali Veli;On 10 12 2012 applied with an headache, diagnosed with flue, purposed to use Majezik and rest. On 12 8 2014 applied with headache. Nothing done.;19
```

The default file name is “**Appointments.csv**”. ';' character is not allowed to be used in the history of patients.

Parameters File:A text file to hold the used file names and working hours such that each entry should be in a separate line in the following order: Records file name, patients file name, delete file name, readable records file, accepted appointments file, parameters file, work start hour, work end hour. There should be nothing else in the file (explanations etc.). The default file name is “**Parameters.txt**”.

Data Structures:

You will define and use the following data structures:

Appointment_t
app_id: int patient_id: int hour: int

Working_hours_t
start: int end: int

Files_t
records_file_n: char* patients_file_n: char* delete_file_n: char* readable_records_file_n: char* accepted_appo_file_n: char* parameters_file_n: char*

node_t
app_id: int patient_id: int name: char[50] history: char* hour: int next: node_t*

Appointment_t: Holds data for an appointment request; app_id is a unique number for an appointment, person_id is an id for the patient (information for patients exist in the patients file), hour is the hour of the time an appointment starts.

Files_t: Holds pointers showing data file names. Note that these are only pointers, i.e., no storage are assigned to them and they should be used to show the real data (Do NOT threat them as if they were arrays and copy file names into where they point!!). Do not assign dynamically allocated storage to them, instead, use them to point already stored file names as described in the 4th item of implementation part.

Implementation:

1. Modes: In your implementation, you will supply 2 mode of operations:
    - Debug Mode: In this mode your code will print some basic debug information (size and content of arrays, etc.)
    - Normal Mode: only operational information will be printed.
2. Implementation Files: For each part, you will provide 2 library files (header and surce file) and a test file including function main to test only the functions specified in that part. **You are NOT allowed to copy the contents of files of different parts –Code replication is not permitted** ; when necessary you should include instead. For part1, you should submit one more source file which has a main() function and no other function and produces a records file. Therefore, you are expected to submit the files in the 4th item of the “General” part at the end.
3. Default File Names: Default file names should be defined (and assigned) globally as constant character pointers.

4. Command-line Arguments: your implement should accept 8 types of optional command-line arguments:
    - Records File Name: Name of the records files, if not sent the default value is used. Argument descriptor: -r
    - Patients File Name: Name of the patients file, if not sent the default value is used. Argument descriptor: -p

- Delete File Name: Name of the delete file, if not sent the default value is used. Argument descriptor: -d
    - Readable Records File Name: Name of the readable records file, if not sent the default value is used. Argument descriptor: -x
    - Accepted Appointments File Name: Name of the accepted appointments, if not sent the default value is used. Argument descriptor: -c
    - Parameters File Name: Name of the parameters file, if not sent the default value is used. Argument descriptor: -t
    - Start Work: Starting hour of work, if not sent the default value (defined 9 as macro) is used. Argument descriptor: -s
    - End Work: End work hour, if not sent the default value (defined 17 as macro) is used. Argument descriptor: -e

    Arguments are optional and if they are used, they should come after the argument descriptors. For example, assuming that the name of the executable is “exec”:

    - ./exec : program runs with default values
    - ./exec -s 10: work starts at 10 a.m., other values are default
    - ./exec -p People.txt -r Records.dat -e 16: People.txt is used as patients file name, Records.dat as records file name, 16 as Work end and other values are default.

    '-' character is not allowed to be used in file names.

5. Make your implementation as described in the parts below.

1. **(70 Pts)** Write a program to produce a records file as described in the implementation part, implement and test the following functions:

- **Appointment_t* getRequests(const Files_t* files, int* size)** :reads all appointments in the records file into a dynamically allocated fully-filled array and returns the array.
- **void write_appointments(Appointment_t appointments[], int size, const Files_t* files)**: writes all appointments in the input array to readable records file in the described format.
- **void get_main_arguments(int argc, char *argv[], Working_hours_t* hours, Files_t* files)**: takes the arguments of main() as input parameter and ,using globally defined defaults when necessary, returns used input file names and working hours as output parameters.
- **void print_parameters(const Files_t* files, const Working_hours_t* hours)**: Writes file names and working hours to Parameters file in the described format.

2. **(100 Pts)** Implement and test the following functions:

- **node_t* build_ll(Appointment_t appointmens[], int size, const Working_hours_t* hours)** : considers all appointments in the array, builds and returns a linked-list including all appointment requests in the order of their time, except:
  - those for out of working hours
  - those for an already reserved time (2nd and other requests for the same time)

  An empty string should be assigned to name field and a NULL pointer to history field of each record.
- **void write_accepted_app(node_t* head, const Files_t* files)**: write all appointments in the list into the accepted appointment file in the described format. Note that this function should work properly both before and after assigning the name and the history fields of appointments.
- **void add_personal_data(node_t* head, const Files_t* files)**: takes personal data from the patients file and adds the corresponding name and history information to each appointment. Note that the name field has its allocated storage while the history field does not have. So, at first, required amount of dynamic allocation should be assigned to history field.
- **int delete_appointments(node_t** head, const Files_t* files)**:deletes all records in the delete file from the linked list and returns the number of appointments deleted.
- **void free_list(node_t* head)**: frees all dynamically allocated memory in the list. Should be able to work properly both before and after assigning the name and the history fields of appointments.

3. **(30 Pts)**Implement and test the following function

- **node_t* make_independent_copy_ll(node_t* head)**: makes an independent copy of the list given. Test your implementation following the steps below:
    - Create a list/ Use an existing list.
    - Make an independent copy of the list by calling make_independent_copy().
    - Change the history field of some elements of the copy list and show that the original list remains unchanged.

  Note that the test procedure in this part is strict. In the case of not applying this procedure, this part is not graded.

General:

1. Obey honor code principles.
2. Obey coding convention.
3. Read your homework carefully and follow the directives about the I/O format (data file names, file formats, etc.) and submission format strictly. Violating any of these directives will be penalized.
4. Your submission should include the following file and NOTHING MORE (no data files, object files, etc):

    HW10_<student_name>_<studentSurname>_<student number>_Test1.c
    HW10_<student_name>_<studentSurname>_<student number>_Part1.h
    HW10_<student_name>_<studentSurname>_<student number>_Part1.c
    HW10_<student_name>_<studentSurname>_<student number>_ProduceRecordFile.c
    HW10_<student_name>_<studentSurname>_<student number>_Test2.c
    HW10_<student_name>_<studentSurname>_<student number>_Part2.h
    HW10_<student_name>_<studentSurname>_<student number>_Part2.c
    HW10_<student_name>_<studentSurname>_<student number>_Test3.c
    HW10_<student_name>_<studentSurname>_<student number>_Part3.h
    HW10_<student_name>_<studentSurname>_<student number>_Part3.c
5. Do not use non-English characters in any part of your homework (in body, **file name**, etc.).
6. Deliver the printout of your work **until the last submission date**.