INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

C language 2

Labs 6 and 7
S5 EII

2020-2021

INSA

Contents

LAB 6

Multimedia application : model with an list

Objectives : *(1h30)*

— implement a list module ;

— use dynamic allocation with lists ;

— improve your software engeenering by discovering a software design that enables a better reuse of the code through a concrete application.

Requirement :

— read the course document on lists ;

— read the whole practical subject ;

— write the Makefile of the project before the practical work session

6.1 Context

During this lab and the next one, you will code a part of a basic multimedia player, handling photos and videos. Figure 6.1 shows the graphic interface proposed to the user.

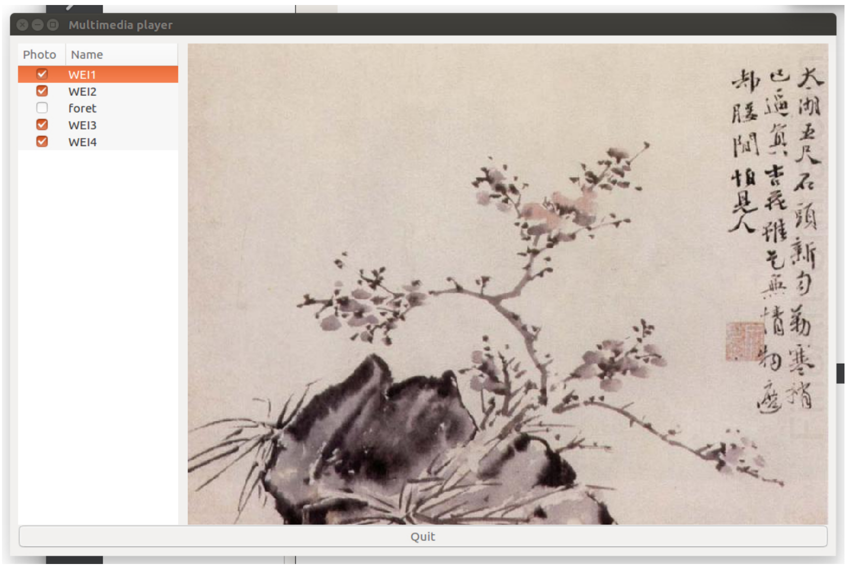

FIGURE 6.1 – Graphic interface of the multimedia player

The user can select a multimedia object in the proposed list on the left side, either a photo or a video. If it is a photo, its content is displayed on the right side. If it is a video, an external application is launched to play it. The user can change the name of the selected multimedia object.
When developing such an application, you have to handle several parts :

— the model : it is the data manipulated by the application. For example, we have to deal with here with a set of multimedia objects. We have to propose structures to represent and handle photos and videos, and to choose a container for them (array, list,...). **It will be your work**.

— the view : this is what the user sees and with what he/she interacts (windows, buttons,...). It is called an HMI Human Machine Interface). The code of the view of the player will be provided.

For the code to be at most maintainable, reusable, bug free, etc., those parts have te be coded in the most independent way possible. For example, the HMI (Human-Machine-Interface) part has been developed using GTK3+, tools to provide graphical interfaces in C language among other things. If the code handling the view and the set of multimedia objects are separated, it will be easy to change those tools when they will be upgraded/obsolete without changing the code for the set of multimedia objects. Similarly, when changing the way to represent the set of multimedia objects, the code handling the HMI will remain unchanged.

Your work for this session will be to provide the structures to handle a list of photos and videos. In the next session, you will link it with the provided GTK3+ code to display it in an HMI. You will also study the benefit of a software architecture separating the model and the view coding.

As a consequence, the code for this lab will be divided in two parts :

— the module "multimedia-object" that handles the multimedia objects (how to define a photo and a video),

— the module "listMO" that handles the set of multimedia objects with a list.

Each part will be developed progressively, as proposed hereafter.

6.2 Multimedia player : data structures

Let's focus on the data structures.

Question 6.1 : Download the provided project and open it with CLion. Check you can build the program and add a Makefile application configuration to run it.

6.2.A Module multimedia-object

Multimedia objects are identified by a name. In order to display it, we need the path to the resource. The creation date of the multimedia object is also stored.

Question 6.2 : Read the provided file "multimedia-object.h" to understand the structures defined to represent this problem.

Question 6.3 : Given the specifications of the player application, we need two functions on a multimedia object : initialize its fields and change its name :

— Code the function *fillMO* that initializes the fields of a multimedia object given as a parameter.

```
void fillMO(MultimediaObject *o, char * name, char * path, unsigned int
    day, unsigned int month, unsigned int year, TyeMultimediaObject type);
```

— Propose a function *changeNameMO* that changes the name of a multimedia object. What prototype do you propose ? Ask the teacher to valid your declaration before coding the function.

Remember that when copying strings, you should use functions that avoid buffer overflows.

Question 6.4 : Add a test of your module in the function called *test_MO* based on the unit test functions already used in lab 2 (files *test.h/.c*). Run the tests.

6.2.B Module listMO

To handle the set of multimedia objects, you will use a singly-linked list for this lab.

Given the specifications of the player application, we need these functions on a multimedia object :

```
 /*group 1: create a list and basic operations*/
void initList(List * l);
int isEmpty(List * l);
int insertFirst(List * l, char * name, char * path, int day, int month,
    int year, TypeMultimediaObject type);

int isFirst(List * l);
int isLast(List * l);
int isOutOfList(List * l);
char * getCurrentName (List * l);
char * getCurrentPath (List * l);
Date getCurrentDate (List * l);
TypeMultimediaObject getCurrentType (List * l);

void setOnFirst(List * l);
void setOnLast(List * l);
void setOnNext (List * l);

void printList(List * l);
int nbElement(List * l);

/*group 2 : find an element*/
int find(List * l, char* name);

/*group 3 : suppress a given an element*/
int deleteValue(List *l, char * name);

/*group 4 : suppress all information in the list */
void freeList(List * n);
```

Question 6.5 : Add the provided files "listMO.h/.c" to your project and the file "main_testlist.c" that will run the unit tests on the list module. Update the Makefile to add this program and add a Makefile application to run the program.

Question 6.6 : There are two *static* functions *newNodeList* and *freeNodeList* provided in "listMO.c". Wat do they do ? Why are they *static* ?

When you will code the functions of the list module, read the documentation provided the the "doc" directory, by opening "index.html" from where you can access to the documentation of each function.

Question 6.7 : Code the functions of the group 1. It should work when running the unit tests, correct your code otherwise.

Question 6.8 : Code the function of the group 2.

Question 6.9 : Add the unit tests of the function of the group 2 in the function *test_list*. It should work when running the unit tests, correct the code of the function otherwise.

Question 6.10 : Code the function of the group 3. It is advised to code intermediate functions :

— *deleteFirst*, the one that deletes the first element of the list

— *deleteCurrent*, the one the deletes the current element of the list

Those two functions should be used only by the functions implemented in the file "listMO.c", therefore these two functions should be *static*.

Question 6.11 : Add the unit tests of functions of the group 3 in the function *test_list*. It should work when running the unit tests, correct the code of the functions otherwise.

Question 6.12 : Code the function of the group 4.

Question 6.13 : Add the unit tests of the function of the group 4 in the function *test_list*. It should work when running the unit tests, correct the code of the function otherwise.

LAB 7

Multimedia application : model and view

Objectives : (*1h30*)

- use a list module ;
- link with a third part library ;
- improve your software engeenering by implementing a software design that enables a better reuse of the code through a concrete application.

Requirement :

- read the course document on lists ;
- read the whole practical subject ;
- write the Makefile of the project before the practical work session

From the previous lab, you have a list module to handle multimedia objects. The player presented in the previous lab requires to implement some functionnalities. You will do do it by using the functions of the list module. Then, the HMI code will be provided and the link to the GTK lib will be presented to see how to connect the view and the model.

7.1 Module managerMO-list

To handle the set of multimedia objects in the application, we need further information. As the contents of this container could be loaded from a file, it is useful to know whereas the list is modified or not, to know if it is necessary to save it, for example when quiting the application. As a consequence, you need a structure *ManagerMO* that stores the list and a MyBooleen that indicates whereas the array has been modified since its last importation from a file.

Question 7.1 : Add the provided files "commun.h" and "managerMO-SDD-list.h" to your project. There is this structure *ManagerMO* that meets the desired specifications :

```
typedef struct{
    listMO data;     /*!< list of objects*/
    MyBoolean modified;   /*!< flag set to 1 if data was modified*/
}ManagerMO;
```

Question 7.2 : As far as the functionalities are concerned, you need to provide functions for the HMI so it can access and modify this set without details about the implementation. Add the provided file "managerMO.h" to your project. For the basic functionalities, add a new file "managerMO-list.c" and code in it the following functions (use the functions yo have coded of the list module) :

```c
/*Init a manager to be empty*/
void initManagerMO(ManagerMO* g);

/* Suppress everything in the manager and leave it empty*/
void freeManagerMO(ManagerMO* g);

/*Add an object in the manager. Return 1 if everything is ok, 0 otherwise.*/
int addMO(ManagerMO* g,char * name, char * path, int day, int month, int year,
    TypeMultimediaObject type);

/*Remove an object from the manager. Return 1 if everything is ok, 0
    otherwise.*/
int removeMO(ManagerMO* g,char * name);
```

Question 7.3 : The application will also need to know the properties of the manager. As a consequence, add the following functions in "managerMO-list.c" (use the functions yo have coded of the list module) :

```c
/*Test if the manager is modified*/
MyBoolean isModified(ManagerMO *g);

/*Provide the number of objects stored in the manangеr. */
int nbObject(ManagerMO* g);
```

Question 7.4 : The player will need to travel through the set to access the data and to change it (for example the name of an object). We have to provide functions that are generic to any container, not only list-based one.

Define the following functions (there is an example of use after the prototypes declarations) :

```c
/*init the manager such as the list current element is set on the first one*/
void start(ManagerMO *g);

/*Test if the "current" member of te list is out of list */
MyBoolean end(ManagerMO *g);

/*Increment the "current" member  to go to the next element in the list
    contained in the manager*/
void next(ManagerMO *g);

/*Change the name of the current object in the list. Return 1 if everything is
    ok, 0 otherwise. */
int changeNameCurrentMO(ManagerMO* g, char * name);

/*Provide the name of the current object in the list.*/
char * getNameCurrentMO(ManagerMO* g);

/*Test if the current object is a photo. */
MyBoolean isPhotoCurrentMO(ManagerMO* g);

/*Provide the path of the current object.  */
char * getPathCurrentMO(ManagerMO* g);

/*Provide the type of the current object.  */
TypeMultimediaObject getTypeCurrentMO(ManagerMO* g);

/*Test if an object is in the manager. After the call of this function, the
    "current" member should be positionned on it if found. Return MYTRUE if
    found, MYFALSE otherwise.*/
MyBoolean findMO(ManagerMO *g, char * name);
```

Here is an example of use to display every object name :

```
int main_console1(void)
{
    ManagerMO player;

    //....

    for(start(&player); !end(&player) ; next(&player))
        printf("%s \n",getNameCurrentMO(&player));
}
```

You can check that the HMI has no clue there is a list in the manager with such prototypes.

7.1.A Testing the manager data structure

Question 7.5 : Add the provided file "main_console1.c" with the *data* directory. The Makefile rule to build it is provided. Add a Makefile application to launch *main_console1* to obtain a first version of a multimedia player running extern applications to display the photos/videos. It should work, correct your code otherwise.

7.2 Graphical user interface

You have finished coding for this lab. Now we will see how to connect your code with provided HMI code.

Question 7.6 : Add the provided code that defines the GTK application and how to load a list of multimedia objects saved in a file.

The project can not be built yet. To use the GTK+ library, you need to tell the compiler and linker where to find the GTK+ files :

— to compile : where to find the GTK+ header files. The compiler options are provided by : **` pkg-config –cflags gtk+-3.0`** .

— to link : where to find the binary code of the called GTK+ functions. The linker options are provided by : **` pkg-config –libs gtk+-3.0`**.

With *main_gtk*, the HMI will be launched. The Makefile rule to build the program is provided. Add the Makefile application to launch it. In case of failure, ask a teacher.

7.2.A Change the model

Question 7.7 : Add the file *managerMO-array_teacher.o* in your directory (be careful with your *clean* rule). Add the provided rule *main_gtk2* to your Makefile. Add the Makefile application to launch it. As you can see, the main program is the same as previously but when defining the macro MY_USE_ARRAY, the structure used for the manager is :

```
  typedef struct{
  MultimediaObject * data; /*!< array of multimedia objects*/
  int n;  /*!< number of multimedia objects*/
  MyBoolean modified; /*!< a flag to knows if the data was modifies since it
      was loaded*/
  int current; /*!< the index of teh current multimedia object*/
  }ManagerMO;
```

The object file provided contains the implementation of the functions for the manager using an array instead of a list. As you can see, as long as the prototypes of "managerMO.h" and their specifications are met, it works. So the application can use all models that provide an implementation for the prototypes of "managerMO.h".

Conclusion : once the specifications between HMI and model are well defined, the two parts are independent. One part can be upgraded/changed without recoding the other part.

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 [0] 2 23 23 82 00
Fax +33 [0] 2 23 23 83 96

www.insa-rennes.fr

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE