French Azerbaijani University
Arash Habibi
Friday June the 25th 2021

Data Structure and Algorithms 1

Practical Exam Retake

The functions written in the beginning may be helpful for the following questions. Otherwise, the questions are independant. The number of points for each question is an approximate indication and it is likely to be modified. If a function contains a compilation error, it will be counted as 0. Otherwise, for each question, a fraction of the points is related to :

- correctness of the algorithm ;
- the fact that the function runs and produces the expected results ;
- its readability ;
- its robustness ;
- its reliability ;
- its expandability ;
- its efficiency.

For readability, most of the functions described in this document are already documented in the question itself. There is no need to document those. Any function that you may find useful to write must be documented.

For robustness, invalid data should (at least) be identified as preconditions in the function documentation.

Even if you work on windows, I expect that your work be delivered with a *makefile*.

General idea

This practical exam has no connection with the /emphQwirkle project, but only with linked lists of points. In the *Moodle* page of *DSA1*, in the *Evaluations* section, you will find an archive file called *CC2_retake_toolkit.tgz*. It contains the following files :

- `Point.h`
- `Point.c`
- `PointList.h`

`Point.h` and `Point.c` define the *Point* type and a few functions dealing with individual points. The `PointList.h` header file contains the prototypes of the functions that you must write. You will have to create and write the following files :

- `PointList.c`
- `main.c` (in order to test your functions)
- `makefile`

1. Write a function with prototype

   ```
   PointList* PL_new(void);
   ```

   with no parameters, and which returns a pointer to an empty list of points.

2. Write a function with prototype

   ```
   PointList* PL_add(PointList *plist, Point p);
   ```

   which has two parameters : `plist` : a pointer to a point list, and `p` : a point. The return value is a pointer to a point list containing the same elements as `plist`, but with a newly allocated element, with a point value of `p` added at the beginning of the list.

3. Write a function with prototype :

```
void PL_print(PointList *plist, char *label);
```

This functions has two parameters `plist` which represents a list of points and `label` : a character string. This function loops through the elements of `plist` and, for each element, prints on the terminal :

— the index of the element (beginning with 0) ;

— the coordinates of the element ;

— the address of the element ;

— the address of the next element ;

and, at the end of the list, the total number of elements in `plist`. For example the following code :

```
plist = PL_new();
plist = PL_add(plist, P_new(1,2,3));
plist = PL_add(plist, P_new(4,5,6));
PL_print(plist, "two");
```

should make an output similar to :

```
---------- point list two -----------
0 (4.000000 5.000000 6.000000) 0x7fb881405a00 0x7fb8814059e0
1 (1.000000 2.000000 3.000000) 0x7fb8814059e0 0x0
2 elements.
```

4. Write a function with prototype :

```
PointList *PL_nOrigins(int n);
```

which has one parameter `n` which should be a positive integer. The return value is a point list which should be empty if `n` is negative or zero. In that case, an error message should also be printed. Otherwise, the return value should represent a point list composed of `n` elements, all with zero coordinates. For example the following code :

```
PointList *plist = PL_nOrigins(5);
PL_print(plist, "zero");
```

should produce a new linked list with 5 elements as shown below :

```
---------- Print point list zero -----------
0 (0.000000 0.000000 0.000000) 0x7faf53c05a60 0x7faf53c05a40
1 (0.000000 0.000000 0.000000) 0x7faf53c05a40 0x7faf53c05a20
2 (0.000000 0.000000 0.000000) 0x7faf53c05a20 0x7faf53c05a00
3 (0.000000 0.000000 0.000000) 0x7faf53c05a00 0x7faf53c059e0
4 (0.000000 0.000000 0.000000) 0x7faf53c059e0 0x0
5 elements.
```

5. Write a function with prototype :

```
PointList *PL_last(PointList *plist);
```

which has one parameter `plist` which represents a point list. This point list should not be empty. If it is, the return value should be `NULL`, with an error message. If `plist` is not empty, the return value should be the address of the last element of `plist`.

6. Write a function with prototype :

```
PointList *PL_append(PointList *plist, Point p);
```

which has two parameters `plist` : a point list and `p` : a point. This function has the same behavious as the `PL_add` function, except that it adds a new element at the end of the list, instead of adding it at the beginning.

7. Write a function with prototype :

```
PointList *PL_index(PointList *plist, int ind)
```

which has two parameters `plist` : a point list and `ind` : an integer. The precondition is that `ind` should be positive (or zero) and should be smaller that the number of elements in the list. If it is not, a message error should be printed and the return value should be `NULL`. If the value of `ind` is valid, then the return value should be the address of the element of `plist` whose index is `ind`. For example the following code :

```
PointList *plist = PL_new();
plist = PL_append(plist, P_new(1,2,3));
plist = PL_append(plist, P_new(4,5,6));
PL_print(plist, "two");
printf("Address of element index 1 : %p\n", PL_index(plist, 1));
```

should make an output similar to :

```
---------- Print point list two -----------
0 (1.000000 2.000000 3.000000) 0x7ff2974059e0 0x7ff297405a00
1 (4.000000 5.000000 6.000000) 0x7ff297405a00 0x0
2 elements.
Address of element index 1 : 0x7ff297405a00
```

8. Write a function with prototype :

```
PointList* PL_setPoint(PointList *plist, int ind, Point p);
```

which has three parameters `plist` : a point list, `ind` : an integer and `p` : a point. The `ind` should have the same properties as in the previous function. If it does not, then an error message should be printed and the return value should be equal to `plist` unchanged. If the value of `ind` is valid, then the function should modify the element of `plist` whose index is `ind` and set it to `p`. For example if, after the previous code, we add :

```
*plist = PL_setPoint(plist, 1, P_new(50,60,70));
```

then the result would have been :

```
---------- Print point list two -----------
0 (1.000000 2.000000 3.000000) 0x7ff2974059e0 0x7ff297405a00
1 (50.000000 60.000000 70.000000) 0x7ff297405a00 0x0
2 elements.
```

At the end of the exam, please make an archive (zip file for example) containing :

— `Point.h` and `Point.c`,

— `PointList.h` and `PointList.c`,

— `main.c` and `makefile`,

Turn in this zip file in the file deposit on Moodle in the *Evaluation* section.