For this last week of practice 3 you need to implement:
list.c (copy list.h), p3_e2.c and copy-paste queue.h, queue.c & p3_testqueue.c, modify their names with an l at the end (queuel.h, queuel.c & p3_testqueuel.c) and make changes for a circular list in the queue. Also you will need to copy the functions you used for integers in the previous practice and add some basic ones in integer.h and integer.c.

Basically you need to:
list.c:
Note in theory you have a pointer to first, now in here we have the pointer in last
Remember you can use macros if it is easier for you
Copy typedef struct _NodeList and struct _List given.
Create the following private functions:
NodeList* nodelist_ini();
void nodelist_free(NodeList* pn, destroy_element_function_type f);

```
NodeList* nodelist_ini(){
   NodeList *pn;
   if ((pn = (NodeList*) malloc(sizeof(NodeList))) ==NULL)
      return NULL;
   pn->info = pn->next = NULL;
   return pn;
}

void nodelist_free(NodeList *pn, destroy_element_function_type f){
   if (pn != NULL){
      if (pn->info != NULL)
         f(pn->info);
      free(pn);
   }
}
```

a) list_ini: check parameters, reserve memory, initialize parameters

b)list_destroy: while it is not empty extract (first), and destroy the extracted elements, then free

c) list_insertFirst:
create a nodelist, and insert in "info" a copy of the element passed
if the list is empty the node will have to point in next to itself and the list will have to point to this nodelist
if not: The next of the new node points to the previous next of the last node in the list
and
The next of the last node points to the this new nodelist we have created

d) list_insertLast:
create a nodelist, and insert in "info" a copy of the element passed
if the list is empty the node will have to point in next to itself and the list will have to point to this nodelist
if not: The next of the new node points to the previous next of the last node in the list
and
The next of the previous last node points to the this new nodelist we have created
and
the last element of the list is now the new nodelist

e) list_insertInOrder:

```
create a nodelist, and insert in "info" a copy of the element passed (pn)
if the list is empty the node will have to point in next to itself and the list will have to point to this nodelist
if not:
aux= get the first element of the list ( the next element of last)
if there is a need to insert at the begining [we compare elements with the function in the struct (pElem and
the info in aux) and is >0]
we destroy the node we have just created and return a call  to insertFirst
else
while aux is not the last element and the info of the next element of aux and pElem comparation function is
>0
        we get the next element of aux
        if aux is the last element of the list
        the last element of the list is set to the nodelist we have created (pn)
next of pn is next of aux
next of aux is pn
```

f) list_extractFirst:

```
  if empty ->  return null
  pElem = info of next element of last
  info of next element of last = null

  // list of 1 element
  if (next of the last element equals last) {
      free the node (last element)
      last element = null //empty the list
  }
  // list of 2 elements or more
  else {
       paux = next of last element        // paux point to the first node
      next of last element = next of paux     // the next of the last element points to the second node
      free paux
  }
  return pElem;
```

g) list_extractLast:

```
  if empty ->  return null
  pElem = info of last element
  info of last element = null

  // list of 1 element
  if (next of the last element equals last) {
      nodelist_free  // free the node (last element)
      last element = null //empty the list
      return pElem
  }

  // list of 2 elements or more
  for (paux=pl->last; paux->next!=pl->last; paux=paux->next)  // Set pn to the the penultimate node
  next of peaux = next of last  // The next of pn points to the first node
  nodelist_free //The last node is free
  last = paux          //Now the last node is paux
  return pElem
```

h)list_isEmpty: if last is null

i) list_get:
```
get the nodelist element in last field of the list
do : count and get the next element
while: it is not the last element of the list and you have not reached index
check if index is equal to your count (you have left the while because you found something and not because you have finished checking the list) in that case return the info field of the last element you got else return null.
```
j) list_size:
```
get the nodelist element in last field of the list
do : count and get the next element
while: it is not the last element of the list
```

k) list_print:
```
if not the list is not empty
print the size and count characters,
get the nodelist element in last field of the list
do : get the next element, print using the function in the struct of list the info field of that element, coutn the printing
while: it is not the last element of the list
```

p3_e2 (is explained step by step):
```
create clean function: int mainCleanUp (int ret_value, List *pl1, List *pl2, int *pe)
main (check errors when needed and return failure when needed):
//declare variables
// check that there are 2 input elements , if not failure
// read the input element
//initialize 2 lists
//initialize an integer
// for all numbers from input to 1
  //Set in the interger the number
 //check odd or even:
    //   In the first list, if the number is even it will insert it at the beginning and if it is odd at the end.
    //  In the second list, inserts the number in order (always keep the list sorted).
// Prints both lists.
//Free resources used
```

Create integer.h:
```
void int_destroy(void* e);
void * int_copy(const void* e);
int    int_print(FILE * f, const void* e);
int    int_cmp(const void* e1, const void* e2);
void * int_ini ();
void * int_setInfo (void *e, int v);
```

Create integer.c. The following functions will be used when you want to store integers in the list. Beware! These functions receive an integer pointer, not a structure with an integer pointer (as in exercise P2_E1)
in p3_testqueuel.c: change include for #include "queuel.h"
in queuel.h: leave everything the same.
in queuel.c: change struct so that it now contains the circular list, always check the input parameters of functions:

```
struct _Queue {
   List *plist;
   destroy_element_function_type destroy_element_function;
   copy_element_function_type copy_element_function;
   print_element_function_type print_element_function;
};
```

a) queue_ini: initialize all values of the struct. Remember to check errors after reserving memory. After calling for the function to initialize the list, remember to check its result, in case of error free the queue
b) destroy: destroy the list then free
c) empty: check the list
d) isFull: is never full
e) insert: insert last in the list
f)extract: extract first from the list
g) size: check the size of the list
h) print: print the list.