*//=========================================================================*

*//the function create an element , insert data into it and return its //pointer*

*//element->next recive NULL*

*//return NULL if memory allocation failed*

**Node** **createElement**(**int** data) {

**Node** element = **malloc**( **sizeof**(\*element) );

**if** (element == NULL) **return** NULL;

element->*data* = data;

element->*next* = NULL;

**return** element;

}

*//-------------------------------------------------------------------------*

*//copy element and returns its pointer*

**Node** **copyElement**(**Node** element) {

**Node** element\_copy = **malloc**( **sizeof**(\*element\_copy) );

**if** (element\_copy == NULL) **return** NULL;

element\_copy->*data* = element->*data*;

element\_copy->*next* = element->*next*;

**return** element\_copy;

}

*//-------------------------------------------------------------------------*

*//release all the memory allocated for list*

**void** **listDestroy**(**Node** list) {

**Node** temp = NULL;

**while** (list != NULL) {

temp = list->*next*;

**free**(list);

list = temp;

}

}

*//-------------------------------------------------------------------------*

*//add element to the list in original order*

*//return false if memory allocation failed and true otherwise*

bool **addToListDirect**(**Node**\* list\_ptr,**const** **Node** **const** element) {

**Node** element\_copy = **copyElement**(element);

**if** (element\_copy == NULL) **return** false;

**if** (\*list\_ptr == NULL) {

\*list\_ptr = element\_copy;

**return** true;

}

**Node** list\_running = \*list\_ptr;

**while** (list\_running->*next* != NULL) {

list\_running = list\_running->*next*;

}

list\_running->*next* = element\_copy;

**return** true;

}

*//-------------------------------------------------------------------------*

*//add element to the list in a reversed order*

*//return false if memory allocation failed and true otherwise*

bool **addToListRevers**(**Node**\* list\_ptr,**const** **Node** **const** element) {

**Node** element\_copy = **copyElement**(element);

**if** (element\_copy == NULL) **return** false;

element\_copy->*next* = \*list\_ptr;

\*list\_ptr = element\_copy;

**return** true;

}

*//-------------------------------------------------------------------------*

*//printing all the elements of a given list*

**void** **printList**(**const** **Node** **const** list) {

**Node** list\_running = list;

**while** (list\_running != NULL) {

**printf**("%d ",list\_running->*data*);

list\_running = list\_running->*next*;

}

}

*//-------------------------------------------------------------------------*

*//build a a list from element in even places and a reversed list from element*

*//placed in odd places*

*//prints the list and then the reversed list*

bool **printCheckmarkOrdered**(**const** **Node** **const** list) {

**if** (list == NULL) **return** true;

**Node** list\_running = list;

**int** counter = 0;

**Node** list\_direct = NULL;

**Node** list\_revers = NULL;

**Node** element = NULL;

bool result;

**while** (list\_running != NULL) {

element = **createElement**(list\_running->*data*);

**if** (element == NULL) **return** false;

**if** (counter % 2 == 0) {

result = **addToListDirect**(&list\_direct,element);

}**else**{

result = **addToListRevers**(&list\_revers,element);

}

**if** (result == false) {

**free**(element);

**listDestroy**(list\_direct);

**listDestroy**(list\_revers);

**return** false;

}

counter = counter + 1;

list\_running = list\_running->*next*;

**free**(element);

}

**printList**(list\_direct);

**printList**(list\_revers);

**listDestroy**(list\_direct);

**listDestroy**(list\_revers);

**return** true;

}

*//=========================================================================*