**4 - 1** Implemente uma lista estática, conforme a interface *lista.h*:

1. O TAD TLista, com vetor e a quantidade de elementos;
2. Operação de criar;
3. Operação inserir (no final)
4. Operação de remover;
5. Operação de buscar;

|  |
| --- |
| const int TAM = 100;    struct List{  int vector[100];  int quantity;  };    TList \*create() {  TList \*list = (TList\*) malloc(sizeof(TList));  list -> quantity = 0;    return list;  }    int size(TList \*list) {  return list -> quantity;  }    void insert(TList \*list, int value) {  int i;  int quantityElements = list -> quantity;  if(list -> quantity == TAM) return;    list -> vector[quantityElements] = value;  list -> quantity++;  }  int extract(TList \*list, int value) {  int positionValueFound = search(list, value), index;  int quantity = size(list);    if(positionValueFound == -1) {  return -1;  }  for(index = positionValueFound +1; index < quantity; index++) {  list -> vector[index - 1] = list -> vector[index];  }    list -> quantity--;  return positionValueFound;  }  int search(TList \*list, int value) {  int quantityElements = list -> quantity, i;    for(i = 0; i < quantityElements; i++) {  if(list -> vector[i] == value) {  return i;  }  }  return -1;  } |

**4 - 2** Inclua na implementação da sua lista estática o método *inserir\_pos*, o qual recebe uma lista, um valor e a posição onde o valor deverá ser inserido dentro da lista.

|  |
| --- |
| void insertInPosition(TList\* list, int value, int position) {  int index, quantityElements = size(list);  if(list -> quantity == TAM || position >= list -> quantity +1) return;  for(index = quantityElements - 1; index >= 0; index--) {  list -> vector[index + 1] = list -> vector[index];  if(index == position || index + 1 == position) {  list -> vector[position] = value;  list -> quantity++;  return;  }  }  } |

**4 - 3** Inclua na implementação da sua lista estática o método *remover\_pos*, o qual recebe uma lista, e a posição de onde o valor deverá ser removido da lista.

|  |
| --- |
| void extractInPosition(TList \*list, int position) {  int index, quantityElements = size(list);  if(list -> quantity == TAM) return;  if(position < quantityElements) {  for(index = position + 1; index < quantityElements; index++) {  list -> vector[index - 1] = list -> vector[index];  }  list -> quantity--;  } else {  return;  }  } |

**4 - 4** Implemente uma lista simplesmente encadeada, conforme a interface *listaenc.h*:

1. O elemento TNo;
2. Operação de criar;
3. Operação inserir (no final)
4. Operação de remover;
5. Operação de buscar;

|  |
| --- |
| struct List {  int element;  TListSimple \*next;  };  TListSimple \*create(int value) {  TListSimple \*list = (TListSimple\*)malloc(sizeof(TListSimple));  list -> element = value;  list -> next = NULL;  return list;  }  int size(TListSimple \*list) {  if(list == NULL) return -1;  TListSimple \*listAux = list -> next;  int count = 0;  while(listAux != NULL) {  count++;  listAux = listAux -> next;  }  return count;  }  void insert(TListSimple \*list, int value) {  if(list == NULL) return;  TListSimple \*listAux = list;  while(listAux -> next != NULL) {  listAux = listAux -> next;  }  listAux -> next = create(value);  }  int extract(TListSimple \*list, int value) {  if(list == NULL) return;  TListSimple \*listAux = list;  while(listAux -> next != NULL) {  if(listAux -> next -> element == value) {  listAux -> next = listAux -> next -> next;  } else {  listAux = listAux -> next;  }  }  }  TListSimple \*search(TListSimple \*list, int value) {  if(list == NULL) return;  TListSimple \*listAux = list -> next;  while(listAux != NULL) {  if(listAux -> element == value) {  return listAux;  } else {  listAux = listAux -> next;  }  }  } |

**4 - 5** Inclua na implementação da sua lista simplesmente encadeada o método *inserir\_pos*, o qual recebe uma lista, um valor e a posição onde o valor deverá ser inserido dentro da lista.

|  |
| --- |
| void insertInPosition(TListSimple\* list, int value, int position) {  TListSimple\* listAux = list;  int counter = 0;  if(list == NULL) return;  while(counter < position) {  listAux = listAux -> next;  counter++;  }  TListSimple\* listNew = create(value);    listNew -> next = listAux -> next;  listAux -> next = listNew;  } |

**4 - 6** Inclua na implementação da sua lista simplesmente encadeada o método *remover\_pos*, o qual recebe uma lista e a posição de onde o valor deverá ser removido da lista.

|  |
| --- |
| void removeInPosition(TListSimple \*list, int position) {  TListSimple \*listAux = list;  int counter = 0;  if(list == NULL) return;  while(counter < position) {  listAux = listAux -> next;  counter++;  }  if(listAux -> next -> next == NULL) {  listAux -> next = NULL;  } else {  listAux -> next = listAux -> next -> next;  }  } |

**4 - 7** Implemente uma lista duplamente encadeada, conforme a interface *listadenc.h*:

1. O elemento TNo;
2. Operação de criar;
3. Operação inserir (no final)
4. Operação de remover;
5. Operação de buscar;

|  |
| --- |
| struct List {  int element;  TListDoubly \*next;  TListDoubly \*previous;  };  TListDoubly \*create(int value) {  TListDoubly \*list = (TListDoubly \*)malloc(sizeof(TListDoubly));  list -> element = value;  list -> next = NULL;  list -> previous = NULL;  }  void insert(TListDoubly \*list, int value) {  if(list == NULL) return;  TListDoubly \*listAux = list;  while(listAux -> next != NULL){  listAux = listAux -> next;  }  TListDoubly \*newElement = create(value);    listAux -> next = newElement;  newElement -> previous = listAux;  }  int extract(TListDoubly \*list, int value) {  if(list == NULL) return;  TListDoubly \*listAux = list -> next;  while(listAux != NULL) {  if(listAux -> element == value) {  if(listAux -> next == NULL) {  listAux -> previous -> next = NULL;  free(listAux);  } else {  listAux -> previous -> next = listAux -> next;  listAux -> next -> previous = listAux -> previous;  free(listAux);  }  return listAux -> element;  }  listAux = listAux -> next;  }  }  TListDoubly \*search(TListDoubly \*list, int value) {  if(list == NULL) return NULL;  TListDoubly \*listAux = list -> next;  while(listAux != NULL) {  if(listAux -> element == value) {  return listAux;  }  listAux = listAux -> next;  }  return NULL;  } |

**4 - 8** Inclua na implementação da sua lista duplamente encadeada o método *inserir\_pos*, o qual recebe uma lista, um valor e a posição onde o valor deverá ser inserido dentro da lista.

|  |
| --- |
| void insertInPosition(TListDoubly \*list, int value, int position) {  int counter = 0;  if(no == NULL) return;  TListDoubly \*listAux = list;  while(counter < position) {  listAux = listAux -> next;  counter++;  }  TListDoubly\* newElement = create(value);  if (listAux -> next != NULL) newElement -> next = listAux -> next;  if (listAux -> next != NULL) listAux -> next -> previous = newElement;  listAux -> next = newElement;  newElement -> previous = listAux;  } |

**4 - 9** Inclua na implementação da sua lista duplamente encadeada o método *remover\_pos*, o qual recebe uma lista e a posição de onde o valor deverá ser removido da lista.

|  |
| --- |
| TListDoubly\* removeInPosition(TListDoubly \*list, int position){  int counter = 0;  if(list == NULL) return NULL;  TListDoubly\* listAux = list;  while(counter <= position) {  if(listAux -> next == NULL) {  return NULL;  }  listAux = listAux -> next;  counter++;  }    if(listAux -> next == NULL) {  listAux -> previous -> next = NULL;  free(listAux);  } else {  listAux -> previous -> next = listAux -> next;  listAux -> next -> previous = listAux -> previous;  free(listAux);  }  return NULL;  } |