libADT

An abstract data type library written in C.

1 Lists

Lists are an abstract data type which are used to store a collection of (often) related objects.
They can be ordered or unordered and are an example of a more general ADT called a 'container. Functions for the list ADT include:

- Constructor - to instantiate a new list.
- Destructor - to remove an instance of a list from memory.
- Add - to add an item to the end of the list.
- Read - to read the value at a given index of the list into a variable.
- Remove - to remove an item in a list at a given index.
- Size - to return the number of items in the list.
- isEmpty - to return true if the list is empty, otherwise false.
- Search - to see if a given value exists within the list.

ArrayList

Constructor

To instantiate a new ArrayList fist initialise a pointer to a List struct.
Then instantiate it with the constructor method *listConst()*.
This will initialise an integer ArrayList with a capacity of 50.

```
struct List* newList;
newList = listConst();
```

Destructor

To free a list from memory call the *listDest()* method and pass the list as an argument.

```
struct List* newList;
newList = listConst();

listDest(newList);
```

listAdd(struct List* list, int entity)

To add an item to the ArrayList use the *listAdd()* method and pass the list and item as arguments.

```
struct List* newList;
newList = listConst();

listAdd(newList, 3);
//ArrayList now reads [3] + 0's for empty initialised indexes.
```

listRead(struct List* list, int index, int* var)

To add an item to the ArrayList use the *listRead()* method and pass the list and item as arguments.

```
int var;
int newList;
listConst(newList);

listRead(newList, 3, &var);

printf("%d", var); //prints 3
```

listAdd(struct List* list, int entity)

To remove an item from the ArrayList use the *listRem()* method and pass the list and index to remove item.

```
struct List* newList;
newList = listConst();

listAdd(newList, 3); newList = [3]
listRem(newList, 1); //Removes 3 from list.
```

Objects will cascade down ArrayLists when removed too.

```
struct List* newList;
newList = listConst();

listAdd(newList, 5);
listAdd(newList, 1);
listAdd(newList, 3);

//List reads [5,1,3] + '0's for empty initialised indexes.

listRem(newList, 1); //Removes '1' from index 1 of list.

//List now reads [5,3] + '0's for empty initialised index.
```

listSize(struct List* list)

To get the number of items currently stored in the ArrayList use the *listSize()* method.

```
struct List* newList;
newList = listConst();

for(int i=0; i<5; i++)
{
    listAdd(newList, i);
}

printf("%d", listSize(newList)); //prints 5
```

listIsEmpty(struct List* list)

The *listIsEmpty()* method returns 1 if the ArrayList is empty otherwise 0.

```
struct List* newList;
newList = listConst();

print("%d", listIsEmpty(newList)); //prints 1

listAdd(newList, 5); //newList = [5]

print("%d", listIsEmpty(newList)); //prints 0
```