warsztat5

Generated by Doxygen 1.9.3

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 ListInfo Struct Reference

Data Fields

- struct ListRec * **firstRec**
- int(* **compare**)(void *, void *)
- void(* **copy**)(void *, void *, size_t)
- size_t **elem_size**

3.2 ListRec Struct Reference

Data Fields

- void * **storage**
- struct ListRec * **next**

Chapter 4

File Documentation

4.1 list.c File Reference

Functions

- void **listInitialize** (List *list, int(*compare)(void *, void *), void(*copy)(void *, void *, size_t), size_t elem_↩ size)
- int **listEmpty** (List list)
- void **listInsert** (List *list, void *elem)
- void * **listFind** (List *list, void *elem)
- void **listRemove** (List *list, void *elem)
- void **listDestroy** (List *list)
- void **listTraverse** (List list, void(*visit)(void *))
- List reverseList (List list)

  Funkcja do odwracania listy w miejscu (bez tworzenia nowej listy).
- List **copyList** (List list)
- List **listReverse** (List list)
- int **listCount** (List list)
- int **listCompare** (List l1, List l2)
- void concatenateLists (List *list1, List *list2)

  Funkcja do łączenia (konkatenowania) dwóch list.
- List * splitListInHalf (List *originalList, List *secondHalf)

  Funkcja do dzielenia listy na pół (z jednej listy tworzy dwie o rozmiarze połowy wejściowej listy)
- void listClear (List *list)

  Funkcja do usuwania wszytkich elementów z listy (czyszczenie listy), ale nie niszcząca jej.
- void mergeSortedLists (List *firstList, List *secondList)

  Funkcja do łączenia dwóch posortowanych list w jedną (również posortowaną).
- void listSort (List *list)

  Funkcja do sortowania listy jednokierunkowej.

4.1.1 Function Documentation

4.1.1.1 concatenateLists()

```
void concatenateLists (
            List * list1,
            List * list2 )
```

Funkcja do łączenia (konkatenowania) dwóch list.

Parameters

in	*list1*	
in	*list2*	

4.1.1.2 listClear()

```
void listClear (
            List * list )
```

Funkcja do usuwania wszytkich elementów z listy (czyszczenie listy), ale nie niszcząca jej.

Parameters

in	*list*	

4.1.1.3 listSort()

```
void listSort (
            List * list )
```

Funkcja do sortowania listy jednokierunkowej.

Wykorzystano algorytm MergeSort. Złożoność obliczeniowa zaimplementowanego rozwiązania wynosi O(nlog(n)).

Parameters

in	*list*	

4.1.1.4 mergeSortedLists()

```
void mergeSortedLists (
            List * firstList,
            List * secondList )
```

Funkcja do łączenia dwóch posortowanych list w jedną (również posortowaną).

Parameters

in	*firstList*	
in	*secondList*	

4.1.1.5 reverseList()

```
List reverseList (
            List list )
```

Funkcja do odwracania listy w miejscu (bez tworzenia nowej listy).

Parameters

in	*list*	

Returns

List

4.1.1.6 splitListInHalf()

```
List * splitListInHalf (
            List * originalList,
            List * secondHalf )
```

Funkcja do dzielenia listy na pół (z jednej listy tworzy dwie o rozmiarze połowy wejściowej listy)

Parameters

in	*originalList*	
in	*secondHalf*	

Returns

List*

4.2 list.h

```
1 #ifndef LIST_H
2 #define LIST_H
3
4 #include <stdio.h>
5
6
7 struct ListRec{
8
9     void * storage;
10     struct ListRec * next;
11
12 };
13
14 typedef struct ListRec * listType;
15
16
17 struct ListInfo{
```

```
18
19     struct ListRec * firstRec;
20     int (*compare)(void*, void*);
21     void (*copy)(void*, void*, size_t);
22     size_t elem_size;
23
24 };
25
26 typedef struct ListInfo * List;
27
28
29 void listInitialize(List * list, int (*compare)(void*, void*), void (*copy)(void*, void*, size_t), size_t
       elem_size);
30 int listEmpty(List list);
31 void listDestroy(List * list);
32 void * listFind(List * list, void * elem);
33 void listRemove(List * list, void * elem);
34 void listInsert(List * list, void * elem);
35 void listTraverse(List list, void (*visit)(void*));
36 List copyList(List list);
37 List listReverse(List list);
38 List reverseList(List list);
39 int listCount(List list);
40 int listCompare(List l1, List l2);
41 void listClear(List* list);
42 List* splitListInHalf(List* originalList, List* secondHalf);
43 void concatenateLists(List* list1, List* list2);
44 void mergeSortedLists(List* firstList, List* secondList);
45 void listSort(List* list);
46 #endif
```

4.3 tests.c File Reference

Functions

- void **displayElement** (void *element)
- void **copy** (void *a, void *b, size_t elem_size)
- int **compare** (void *a, void *b)
- int testIfPalindrome (List *list)

  Funkcja do sprawdzania, czy dana lista jest palindromem, ale nie zużywająca dodatkowej pamięci.
- int main (void)

  Główna funkcja testująca zaimplementowane funkcjonalności.

4.3.1 Function Documentation

4.3.1.1 main()

```
int main (
            void  )
```

Główna funkcja testująca zaimplementowane funkcjonalności.

Returns

int

4.3.1.2 testIfPalindrome()

```
int testIfPalindrome (
            List * list )
```

Funkcja do sprawdzania, czy dana lista jest palindromem, ale nie zużywająca dodatkowej pamięci.

Funkcja dzieli listę na pół, po czym sprawdza, czy odwrotność drugiej połowy jest taka sama jak pierwsza połowa (wtedy lista jest palindromem).

Parameters

in	*list*	

Returns

int

Index