Exercises — Dlist advanced

version #

Assistants C/Unix 2022 <assistants@tickets.assistants.epita.fr>

Copyright

Contents

*https://intra.assistants.epita.fr

1 Dlist advanced

Files to submit:

- dlist_advanced/Makefile
- dlist_advanced/*

Provided files:

- dlist_advanced/dlist.h

Makefile: Your makefile should define at least the following targets:

- library: Produce the libdlist.a library
- clean: Delete everything produced by make

Authorized functions: You are only allowed to use the following functions:

- malloc(3)
- calloc(3)
- free(3)
- putchar(3)
- puts(3)
- printf(3)

Authorized headers: You are only allowed to use the functions defined in the following headers:

- stddef.h
- errno.h
- err.h
- assert.h

2 Goal

Be careful!

This exercise is the next step after having completed the *Dlist* exercise. Make sure you have already completed it, because you will need some of the functions you wrote in the first part.

In this step, you will manipulate doubly linked lists in a more advanced way.

2.1 Threshold 4

2.1.1 dlist_clear

```
void dlist_clear(struct dlist *list);
```

Empty `list` by freeing its nodes. However, `list` must not be freed.

2.1.2 dlist_shift

```
void dlist_shift(struct dlist *list, int offset);
```

Shift the list (by looping) of $|$`offset`$|$ elements (where $|x|$ is the absolute value of x). This shift is a left shift if `offset` is negative and is a right shift if `offset` is positive.

Example:

```
list1 = [1,2,3,4,5]
/* Shift 1 */
list1 = [5,1,2,3,4]

list2 = [2,4,6,8,10]
/* Shift -7 */
list2 = [6,8,10,2,4]
```

2.1.3 dlist_add_sort

```
int dlist_add_sort(struct dlist *list, int element);
```

Add `element`, keeping `list` sorted. This will only be tested with an already sorted list. If any argument is invalid, return `-1`. Otherwise, return `1`.

2.1.4 dlist_remove_eq

```
int dlist_remove_eq(struct dlist *list, int element);
```

Remove the first element equal to `element`. Return `1` if an element has been removed, `0` otherwise.

2.1.5 dlist_copy

```
struct dlist *dlist_copy(const struct dlist *list);
```

Return a **deep** copy of `list`: each element of the list must be copied. If anything goes wrong, return `NULL`.

2.2 Threshold 5

2.2.1 dlist_sort

```
void dlist_sort(struct dlist *list);
```

Sort `list` with a bubble sort.

2.2.2 dlist_merge

```
void dlist_merge(struct dlist *list1, struct dlist *list2);
```

Merge two sorted lists. The result of the merge must be a sorted list. `list2` must be emptied.

Before:

```
list1 = [1,4,7,8]
list2 = [2,5,6,7]
```

After:

```
List1 = [1,2,4,5,6,7,7,8]
List2 = []
```

2.3 Threshold 6

2.3.1 dlist_levenshtein

```
unsigned int dlist_levenshtein(struct dlist *list1, struct dlist *list2);
```

Returns the levenshtein distance between `list1` and `list2`.

It is my job to make sure you do yours.