

Assignment #1: Adrian Kacmarcik

Generated by Doxygen 1.8.13

Contents

1	File Index	1
1.1	File List	1
2	File Documentation	1
2.1	cachelist.c File Reference	1
2.1.1	Detailed Description	2
2.1.2	Function Documentation	3
	Index	11

1 File Index

1.1 File List

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

cachelist.c	Description: Linked lists operstions with cacheing	1
--------------------	---	----------

2 File Documentation

2.1 cachelist.c File Reference

Description: Linked lists operstions with cacheing.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cachelist.h"
```

Functions

- static `cl_node * make_node` (int value, const char *label)
Make a new node based on inputs.
- `cl_node * cl_add_end` (cl_node *list, int value, const char *label)
Add a node to the end of |list|.
- `cl_node * cl_add_front` (cl_node *list, int value, const char *label)
Add a new node to the front of |list|.
- `cl_node * cl_remove` (cl_node *list, int search_value)
Search |list| for a match and delete it.
- `cl_node * cl_insert_before` (cl_node *list, int search_value, int value, const char *label)
Add a new node before the matching value.
- void `cl_insert_after` (cl_node *list, int search_value, int value, const char *label)
Add new node after matching value.
- `cl_node * cl_find` (cl_node *list, int search_value, bool cache, int *compares)
Find a node in |list|, and if |cache| then move to the beginning if found.
- void `cl_destroy` (cl_node *list)
Deletes and frees all of |list|.
- void `cl_dump` (const cl_node *list)
print out the list

2.1.1 Detailed Description

Description: Linked lists operations with caching.

Author

Adrian T.P. Kacmarcik

DP email a.kacmarcik@digipen.edu

course CS180

section A

assignment 1

Date

2020/01/23

cl_add_end: Add a node to the end of the list contained with the inputted list pointer

cl_add_front: Add a node to the front of the list that is contained in the pointer passed in (this will change the head pointer to point to the new beginning of the list)

cl_remove: Search the list contained with the head pointer for the inputted search value and removes the first one that it finds from the list

cl_insert_before: Inserts a new node with the inputted value and label before the matching value. If no matching value is found, then nothing is done.

cl_insert_after: Inserts a new node with the inputted value and label after the matching value. If no matching value is found, then nothing is done.

cl_find: Searches through the list looking for the first matching value, counting the number of comparisons it took to find the matching node. If cacheing is true then the found node will be moved to the beginning of the list to simulate cacheing.

cl_destroy: Frees the memory in the linked list that the head pointer is pointing to.

cl_dump: Prints out the list to the console.

2.1.2 Function Documentation**2.1.2.1 cl_add_end()**

```
cl_node* cl_add_end (
    cl_node * list,
    int value,
    const char * label )
```

Add a node to the end of |list|.

Parameters

<i>list</i>	Pointer to the start of the linked list.
<i>value</i>	Value to store in the new node.
<i>label</i>	Label for the new node.

Returns

The pointer to the head of the list.

Definition at line 87 of file cachelist.c.

References `make_node()`.

```

88 {
89     // copy of the start of the list
90     cl_node *listLocation = list;
91     cl_node *newNode = make_node(value, label); // node to add to |list|
92
93     // check if the list exists
94     if (list)
95     {
96         // find the end of the list
97         while (listLocation->next) {
98             listLocation = listLocation->next;
99         }
100         listLocation->next = newNode; // add |newNode| to end of |list|
101
102         return list; // return pointer to start of list
103     }
104     else
105     {
106         list = newNode; // set the head of the list to be the new node
107         return newNode; // return the head of the new list
108     }
109 }

```

2.1.2.2 cl_add_front()

```

cl_node* cl_add_front (
    cl_node * list,
    int value,
    const char * label )

```

Add a new node to the front of |list|.

Parameters

<i>list</i>	Head pointer.
<i>value</i>	Value for new node.
<i>label</i>	String for the new node.

Returns

The pointer to the head of |list|.

Definition at line 118 of file cachelist.c.

References `make_node()`.

```

119 {
120     cl_node *newNode = make_node(value, label); // node to add to |list|
121
122     newNode->next = list; // point next to be the old start
123     list = newNode;      // update the head pointer
124
125     return newNode; // return new head pointer
126 }

```

2.1.2.3 cl_destroy()

```
void cl_destroy (
    cl_node * list )
```

Deletes and frees all of |list|.

Parameters

<i>list</i>	Pointer to the head of the list to free.
-------------	--

Definition at line 294 of file cachelist.c.

```
295 {
296     cl_node *listLocation = list; // temp storage of location in |list|
297
298     // loop through |list| freeing all of the nodes
299     while (list)
300     {
301         listLocation = list; // store the current location in |listLocation|
302         list = list->next;    // move |list| to be the next location
303         free(listLocation);  // free |listLocation|
304     }
305 }
```

2.1.2.4 cl_dump()

```
void cl_dump (
    const cl_node * list )
```

print out the list

Parameters

<i>list</i>	Pointer to the head of the list to walkk through.
-------------	---

Definition at line 311 of file cachelist.c.

```
312 {
313     printf("=====\n"); // print seperator
314
315     // loop through |list|
316     while (list)
317     {
318         printf("%4i: %s\n", list->value, list->label); // print formatted
319         list = list->next;                             // next node in |list|
320     }
321 }
```

2.1.2.5 cl_find()

```
cl_node* cl_find (
    cl_node * list,
    int search_value,
    bool cache,
    int * compares )
```

Find a node in |list|, and if |cache| then move to the beginning if found.

Parameters

<i>list</i>	Pointer to the head of the list
<i>search_value</i>	Value to search for in list
<i>cache</i>	Whether we should be cacheing
<i>compares</i>	Pointer to store the number of comparasons

Returns

The pointer to the head of the list.

Definition at line 254 of file cachelist.c.

```
255 {
256     cl_node *listLocation = list;    // current location in |list|
257     cl_node *prevListLocation = list; // previous location in |list|
258     *compares = 0;                    // set the number of compares to |0|
259
260     // loop through |list|
261     while (listLocation)
262     {
263         (*compares)++; // increment the number of compares
264
265         // check if we have found a matching value
266         if (listLocation->value == search_value)
267         {
268             // check if we are cacheing
269             if (cache)
270             {
271                 // check if we have moved from the start
272                 if (prevListLocation != listLocation)
273                 {
274                     prevListLocation->next = listLocation->next; // skip over curr loc
275                     listLocation->next = list;                    // add beggining
276                     list = listLocation;                          // point head to curr
277                 }
278             }
279             // return pointer to the start of the list
280             return list;
281         }
282         // if we havent found anything then go to next node
283         prevListLocation = listLocation;
284         listLocation = listLocation->next;
285     }
286     // if there was nothing to match in the list then return the head
287     return list;
288 }
```

2.1.2.6 cl_insert_after()

```
void cl_insert_after (
    cl_node * list,
    int search_value,
    int value,
    const char * label )
```

Add new node after matching value.

Parameters

<i>list</i>	Pointer to the start of the list.
<i>search_value</i>	Value to match
<i>value</i>	Value of the new node
<i>label</i>	String for the new node

Definition at line 223 of file cachelist.c.

References `make_node()`.

```
225 {
226     cl_node *listLocation = list; // current location in |list|
227
228     // loop through |list|
229     while (listLocation)
230     {
231         // check if we founnd a match
232         if (listLocation->value == search_value)
233         {
234             cl_node *newNode = make_node(value, label); // node to add to |list|
235
236             // insert node in |list|
237             newNode->next = listLocation->next;
238             listLocation->next = newNode;
239
240             return; // escape with first match found
241         }
242         listLocation = listLocation->next; // next node
243     }
244 }
```

2.1.2.7 cl_insert_before()

```
cl_node* cl_insert_before (
    cl_node * list,
    int search_value,
    int value,
    const char * label )
```

Add a new node before the matching value.

Parameters

<i>list</i>	Pointer to the start of the list.
<i>search_value</i>	Value to search list for.
<i>value</i>	Value for new node.
<i>label</i>	String for the new node.

Returns

The pointer to the head of the list.

Definition at line 178 of file cachelist.c.

References `make_node()`.

```

180 {
181     cl_node *listLocation = list;    // current location in |list|
182     cl_node *prevListLocation = list; // previous location in |list|
183
184     // loop through |list|
185     while (listLocation)
186     {
187         // check if ew have found a match
188         if (listLocation->value == search_value)
189         {
190             cl_node *newNode = make_node(value, label); // create new node
191             newNode->next = listLocation;               // point next to curr
192
193             //check if we are at the start of the list
194             if (prevListLocation == listLocation)
195             {
196                 // new node is at the beggining change the head pointer
197                 list = newNode;
198                 return newNode; // return new head pointer
199             }
200             else
201             {
202                 // set the previous node to be pointing at the new one
203                 prevListLocation->next = newNode;
204
205                 return list; // return the head pointer
206             }
207         }
208         // move to the next node
209         prevListLocation = listLocation;
210         listLocation = listLocation->next;
211     }
212     // if none are found return the head pointer
213     return list;
214 }

```

2.1.2.8 cl_remove()

```

cl_node* cl_remove (
    cl_node * list,
    int search_value )

```

Search `|list|` for a match and delete it.

Parameters

<i>list</i>	Head pointer of the list.
<i>search_value</i>	Value to search <code> list </code> for.

Returns

The head pointer.

Definition at line 134 of file cachelist.c.

```

135 {
136     cl_node *listLocation = list;    // current location in |list|
137     cl_node *prevListLocation = list; // previous location in |list|
138
139     // loop through |list|
140     while (listLocation)
141     {
142         // check for match
143         if (listLocation->value == search_value)
144         {
145             // check for begginging of |list|
146             if (prevListLocation == listLocation)
147             {
148                 // move head pointer to next node
149                 cl_node *temp = listLocation->next;
150                 free(listLocation); // free the head pointer
151                 list = temp;        // change the head pointer to new one
152                 return temp;        // return new head pointer
153             }
154             else
155             {
156                 // move the pointers to skip over |listLocation|
157                 prevListLocation->next = listLocation->next;
158                 free(listLocation); // free |listLocaiton|
159                 return list;        // return head pointer
160             }
161         }
162         // move to next node
163         prevListLocation = listLocation;
164         listLocation = listLocation->next;
165     }
166     // if nothing found return head pointer
167     return list;
168 }

```

2.1.2.9 make_node()

```

static cl_node* make_node (
    int value,
    const char * label ) [static]

```

Make a new node based on inputs.

Parameters

<i>value</i>	Number to be stored in the newNodes's value .
<i>label</i>	String to be stored in label .

Returns

The pointer to the new node.

Definition at line 57 of file cachelist.c.

Referenced by `cl_add_end()`, `cl_add_front()`, `cl_insert_after()`, and `cl_insert_before()`.

```
58 {
59     // malloc the space for a new node
60     cl_node *node = (cl_node *)malloc(sizeof(cl_node));
61
62     // check if malloc succeeded
63     if (!node)
64     {
65         printf("Can't allocate new node.\n"); // print that it failed
66         exit(1);                             // exit the code early
67     }
68
69     // set the values of |node|
70     node->value = value;
71     node->next = NULL;
72
73     /* Be sure not to overwrite memory */
74     strncpy(node->label, label, LABEL_SIZE - 1); // copy the label in |node|
75     node->label[LABEL_SIZE - 1] = 0;             // set end |NULL|
76
77     return node; // return the pointer to the new node
78 }
```

Index

- cachelist.c, [1](#)
 - cl_add_end, [3](#)
 - cl_add_front, [4](#)
 - cl_destroy, [4](#)
 - cl_dump, [5](#)
 - cl_find, [5](#)
 - cl_insert_after, [6](#)
 - cl_insert_before, [7](#)
 - cl_remove, [8](#)
 - make_node, [9](#)
- cl_add_end
 - cachelist.c, [3](#)
- cl_add_front
 - cachelist.c, [4](#)
- cl_destroy
 - cachelist.c, [4](#)
- cl_dump
 - cachelist.c, [5](#)
- cl_find
 - cachelist.c, [5](#)
- cl_insert_after
 - cachelist.c, [6](#)
- cl_insert_before
 - cachelist.c, [7](#)
- cl_remove
 - cachelist.c, [8](#)
- make_node
 - cachelist.c, [9](#)