CS 180A Assignment 1b (Cachelist) - Ryman Barnett

Generated by Doxygen 1.8.17

1 File Index

1.1 File List

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

cachelist.c
Source file for cachelist functions to create a linked list of nodes that will be used to store, manipulate, and retrieve data using either caching or non-caching methods 1

2 File Documentation

2.1 cachelist.c File Reference

Source file for cachelist functions to create a linked list of nodes that will be used to store, manipulate, and retrieve data using either caching or non-caching methods.

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

Functions

- static cl_node * make_node (int value, const char *label)
  Creates a new node.
- cl_node * cl_add_end (cl_node *list, int value, const char *label)
  adds a new node to the end of the list
- cl_node * cl_add_front (cl_node *list, int value, const char *label)
  adds a new node to the front of the list
- cl_node * cl_remove (cl_node *list, int search_value)
  removes node with a given value from the list

- cl_node * cl_insert_before (cl_node *list, int search_value, int value, const char *label)
  inserts a new node into the list before the node with a given value
- void cl_insert_after (cl_node *list, int search_value, int value, const char *label)
  inserts a new node into the list after the node with a given value
- cl_node * cl_find (cl_node *list, int search_value, bool cache, int *compares)
  finds first occurrence of a node with a given value in the list and counts how many compares were required to find it
- void cl_destroy (cl_node *list)
  frees the memory of the linked list
- void cl_dump (const cl_node *list)
  prints the linked list

2.1.1 Detailed Description

Source file for cachelist functions to create a linked list of nodes that will be used to store, manipulate, and retrieve data using either caching or non-caching methods.

Author

Ryman Barnett

email: ryman.b@digipen.edu

DigiPen login: ryman.b

Course: CS180

Section: A

Assignment #1b

Date

2022-09-20

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 )
```

adds a new node to the end of the list

Parameters

list	head of the linked list
value	value to store in the node
label	label to store in the node

Returns

head of the linked list

Definition at line 70 of file cachelist.c.

```
71 {
72   // store the head of the list for walking
73   cl_node *temp = list;
74
75   // if the list is empty, make a new node at front
76   if (!list)
77     return make_node(value, label);
78
79   // walk to the end of the list
80   while (temp->next != NULL)
81     temp = temp->next;
82
83   // add the new node to the end of the list
84   temp->next = make_node(value, label);
85
86   return list; // return the head of the list
87 }
```

References make_node().

2.1.2.2 cl_add_front()

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

adds a new node to the front of the list

Parameters

list	head of the linked list
value	value to store in the node
label	label to store in the node

Returns

head of the linked list

Definition at line 105 of file cachelist.c.

```
106 {
107   // make a new node with the given value and label
108   cl_node *newNode = make_node(value, label);
109   cl_node *temp = list; // store the head of the list for walking
110
```

```
111   // put newNode at the front of the list
112   list = newNode;
113   list->next = temp; // point the new node to the old head of the list
114
115   return list; // return the head of the list
116 }
```

References make_node().

Referenced by cl_find(), and cl_insert_before().

2.1.2.3 cl_destroy() void cl_destroy (cl_node * *list*)

frees the memory of the linked list

Parameters

list	head of the linked list to free

Definition at line 324 of file cachelist.c.

```
325 {
326   cl_node *temp = list; // store the head of the list for walking
327
328   // walk the list until we hit end
329   while (temp != NULL)
330   {
331     // store the next node
332     cl_node *nextNode = temp->next;
333     free(temp); // free the current node
334     temp = nextNode; // move next node to head
335   }
336 }
```

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

prints the linked list

Parameters

list	head of the linked list to print

Definition at line 345 of file cachelist.c.

```
346 {
347   printf("==================\n");
348
349   /* Print each value in the list */
350   while (list)
351   {
352     printf("%4i: %s\n", list->value, list->label);
353     list = list->next;
354   }
```

```
355 }
```

2.1.2.5 cl_find()

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

finds first occurrence of a node with a given value in the list and counts how many compares were required to find it

Parameters

list	head of the linked list
search_value	value to search for in the list
cache	whether or not to use cacheing
compares	to store the number of compares that occured

Returns

head of the linked list

Definition at line 278 of file cachelist.c.

```
279 {
280   cl_node *temp = list; // store the head of the list for walking
281   *compares = 0;        // initialize the number of compares to 0
282
283   // walk the list until we hit end or find the node
284   while (temp != NULL)
285   {
286     *compares += 1; // another compare has been done
287
288     // if the node is the one we are looking for
289     if (temp->value == search_value)
290     {
291       // if we are caching, move the node to the front of the list
292       if (cache)
293       {
294         // copy label of node to be moved
295         char *label = (char*)malloc(LABEL_SIZE);
296         strncpy(label, temp->label, LABEL_SIZE - 1);
297         label[LABEL_SIZE - 1] = '\0'; // make sure the string is null terminated
298
299         // remove the node from the list
300         list = cl_remove(list, search_value);
301
302         // add the node to the front of the list
303         list = cl_add_front(list, search_value, label);
304
305         free(label); // free the memory of the label
306       }
307
308       break; // stop walking the list
309     }
310
311     temp = temp->next; // move to the next node
312   }
313
314   return list; // return the head of the list
315 }
```

References cl_add_front(), and cl_remove().

2.1.2.6 cl_insert_after() void cl_insert_after (

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

inserts a new node into the list after the node with a given value

Parameters

list	head of the linked list
search_value	value of node to insert after for in the list
value	value to store in the node
label	label to store in the node

Definition at line 235 of file cachelist.c.

```
237 {
238   cl_node *temp = list; // store the head of the list for walking
239
240   // walk the list until we hit end or find the node to insert after
241   while (temp != NULL)
242   {
243     // if the node is the one to insert after
244     if (temp->value == search_value)
245     {
246       cl_node *newNode = make_node(value, label); // make a new node
247       // insert the new node after the node and connect the list
248       newNode->next = temp->next;
249       temp->next = newNode;
250
251       break; // stop walking the list
252     }
253
254     temp = temp->next; // move to the next node
255   }
256 }
```

References make_node().

2.1.2.7 cl_insert_before() cl_node* cl_insert_before (

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

inserts a new node into the list before the node with a given value

Parameters

list	head of the linked list
search_value	value of node to insert before for in the list
value	value to store in the node
label	label to store in the node

Returns

head of the linked list

Definition at line 186 of file cachelist.c.

```
188 {
189   cl_node *temp = list; // store the head of the list for walking
190
191   // if the list is empty, nothing to insert before
192   if (!list)
193     return list;
194
195   // if the first node is the one to insert before
196   if (list->value == search_value)
197     list = cl_add_front(list, value, label); // add the new node to the front
198
199   // walk the list until we find the node to insert before
200   while (temp->next != NULL)
201   {
202     // if the next node is the one to insert before
203     if (temp->next->value == search_value)
204     {
205       cl_node *newNode = make_node(value, label); // make a new node
206       // insert the new node before the node and connect the list
207       newNode->next = temp->next;
208       temp->next = newNode;
209
210       break; // stop walking the list
211     }
212
213     temp = temp->next; // move to the next node
214   }
215
216   return list; // return the head of the list
217 }
```

References cl_add_front(), and make_node().

2.1.2.8 cl_remove() `cl_node* cl_remove (`

```
cl_node * list,
int search_value )
```

removes node with a given value from the list

Parameters

list	head of the linked list
search_value	value to search for in the list

Returns

head of the linked list

Definition at line 131 of file cachelist.c.

```
132 {
133   cl_node *temp = list; // store the head of the list for walking
134
135   // if the list is empty, nothing to remove
136   if (!list)
137     return list;
138
```

```
139   // if the first node is the one to remove
140   if (list->value == search_value)
141   {
142     list = list->next; // point the head of the list to the next node
143     free(temp);        // free the memory of the removed node
144
145     return list;       // return the new head of the list
146   }
147
148   // walk the list until we find the node to remove
149   while (temp->next != NULL)
150   {
151     // if the next node is the one to remove
152     if (temp->next->value == search_value)
153     {
154       cl_node *temp2 = temp->next; // store the node to remove
155       temp->next = temp->next->next; // point current node to the next node
156       free(temp2); // free the memory of the removed node
157
158       break; // stop walking the list
159     }
160
161     temp = temp->next; // move to the next node
162   }
163
164   return list; // return the head of the list
165 }
```

Referenced by cl_find().

2.1.2.9 make_node() `static cl_node* make_node (`

```
int value,
const char * label )  [static]
```

Creates a new node.

Parameters

value	value to store in the node
label	label to store in the node

Returns

pointer to the new node

Definition at line 34 of file cachelist.c.

```
35 {
36   /* Allocate the memory */
37   cl_node *node = (cl_node *)malloc(sizeof(cl_node));
38
39   if (!node)
40   {
41     printf("Can't allocate new node.\n");
42     exit(1);
43   }
44
45   /* Set the initial values */
46   node->value = value;
47   node->next = NULL;
48   /* Be sure not to overwrite memory */
49   strncpy(node->label, label, LABEL_SIZE - 1);
50   node->label[LABEL_SIZE - 1] = 0;
51   return node;
52 }
```

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

Index