

## Assignment 1 : Linked List

Generated by Doxygen 1.8.6

Thu Sep 20 2018 09:37:06

## Contents

<b>1</b>	<b>File Index</b>	<b>1</b>
1.1	File List . . . . .	1
<b>2</b>	<b>File Documentation</b>	<b>1</b>
2.1	cachelist.c File Reference . . . . .	1
2.1.1	Detailed Description . . . . .	2
2.1.2	Function Documentation . . . . .	2
2.2	cachelist.c . . . . .	9
	<b>Index</b>	<b>15</b>

## 1 File Index

### 1.1 File List

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

<b>cachelist.c</b>	<b>This is a single linked list of nodes containing a value and a label</b>	<b>1</b>
--------------------	---	----------

## 2 File Documentation

### 2.1 cachelist.c File Reference

this is a single linked list of nodes containing a value and a label.

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

#### Functions

- static `cl_node *` [make\\_node](#) (int value, const char \*label)
- void [cl\\_dump](#) (const `cl_node *`list)
- `cl_node *` [cl\\_add\\_end](#) (`cl_node *`list, int value, const char \*label)
- `cl_node *` [cl\\_add\\_front](#) (`cl_node *`list, int value, const char \*label)
- `cl_node *` [cl\\_remove](#) (`cl_node *`list, int search\_value)
- `cl_node *` [cl\\_insert\\_before](#) (`cl_node *`list, int search\_value, int value, const char \*label)
- void [cl\\_insert\\_after](#) (`cl_node *`list, int search\_value, int value, const char \*label)
- `cl_node *` [cl\\_find](#) (`cl_node *`list, int search\_value, int cache, int \*compares)
- void [cl\\_destroy](#) (`cl_node *`list)

### 2.1.1 Detailed Description

this is a single linked list of nodes containing a value and a label.

#### Author

Cody Morgan

email: [cody.morgan@digipen.edu](mailto:cody.morgan@digipen.edu)

DigiPen login: [cody.morgan](#)

Course: CS180

#### Section A

#### Assignment #1

#### Date

19 SEP 2018 operations include:

- `cl_add_end` : add item to the end
- `cl_add_front` : add item to the front
- `cl_remove` : remove a give value from the list
- `cl_insert_before` : insert a new value before this value
- `cl_insert_after` : insert a new value after this value
- `cl_find` : find this value
- `cl_destroy` : destroy this value (first)
- `cl_dump` : print this list

Definition in file [cachelist.c](#).

### 2.1.2 Function Documentation

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

Adds a new node to the end of the list

#### Parameters

---

<i>list</i>	the list on to which you want to add
<i>value</i>	the new value for that node
<i>label</i>	the new label for that node

**Returns**

head of the list

a new list node

the end of the new list

Definition at line 94 of file [cachelist.c](#).

References [make\\_node\(\)](#).

```
00095 {
00096     cl_node* newNode = make_node(value, label); /*! a new list node */
00097     cl_node* endOfList = list;                /*! the end of the new list */
00098
00099     /* find the end of the list */
00100     if(list == NULL)
00101     {
00102         list = newNode;
00103     }
00104     else
00105     {
00106         while(endOfList->next != NULL)
00107         {
00108             endOfList = endOfList->next;
00109         }
00110         endOfList->next = newNode;
00111     }
00112
00113     return list; /* the head of the list */
00114 }
```

**2.1.2.2 cl\_node\* cl\_add\_front ( cl\_node \* list, int value, const char \* label )**

Adds a new node to the front of the list

**Parameters**

<i>list</i>	the list on to which you want to add
<i>value</i>	the new value for that node
<i>label</i>	the new label for that node

**Returns**

head of the list

the new list node

Definition at line 131 of file [cachelist.c](#).

References [make\\_node\(\)](#).

```
00132 {
00133     cl_node* newNode = make_node(value, label); /*! the new list node */
00134
00135     /* connect the old list */
00136     newNode->next = list;
00137
00138     return newNode; /* the head of the new list */
00139 }
```

#### 2.1.2.3 void cl\_destroy ( cl\_node \* *list* )

Frees all the memory used for the given list

**Parameters**

<i>list</i>	the list you want to set free
-------------	-------------------------------

Definition at line 366 of file [cachelist.c](#).

```

00367 {
00368     cl_node* nextNode; /* keep destroying until this is NULL */
00369
00370     while(list != NULL)
00371     {
00372         nextNode = list->next;
00373         free(list);
00374         list = nextNode;
00375     }
00376 }
```

**2.1.2.4 void cl\_dump ( const cl\_node \* list )**

Dumps all of the list info to the screen for your viewing pleasure

**Parameters**

<i>list</i>	the list to dump
-------------	------------------

Definition at line 69 of file [cachelist.c](#).

```

00070 {
00071     printf("=====\\n");
00072     while (list)
00073     {
00074         printf("%4i: %s\\n", list->value, list->label);
00075         list = list->next;
00076     }
00077 }
```

**2.1.2.5 cl\_node\* cl\_find ( cl\_node \* list, int search\_value, int cache, int \* compares )**

Find a given value, count the number of compares it took to find, and finally cache that value if enabled

**Parameters**

<i>list</i>	the linked list
<i>search_value</i>	the sought after value
<i>cache</i>	bool to determine if the found value will be cached or not
<i>compares</i>	how many compares it took to find that value

**Returns**

the head of the list

current node while walking the list

head of the list

contains the sought after value

node previous to the found node

number of compares it took to find the node

the head of the list

Definition at line 317 of file [cachelist.c](#).

```

00318 {
00319     cl_node* node = list;          /*! current node while walking the list */
00320     cl_node* head = list;         /*! head of the list */
00321     cl_node* foundNode = NULL; /*! contains the sought after value */
00322     cl_node* previous = list; /*! node previous to the found node */
00323     int compares_ = 0;            /*! number of compares it took to find the node */
00324
00325     /* look for a matching value */
00326     while(node != NULL)
00327     {
00328         compares_++;
00329         if(node->value == search_value)
00330         {
00331             foundNode = node;
00332             break;
00333         }
00334         else
00335         {
00336             previous = node;
00337             node = node->next;
00338         }
00339     }
00340
00341     /* found a matching value */
00342     if(foundNode != NULL)
00343     {
00344         *compares = compares_;
00345
00346         /* cache the node if enabled
00347          * ensure that the found node isn't the head already */
00348         if(cache && foundNode != list)
00349         {
00350             cl_node* tempNode = foundNode->next;
00351             foundNode->next = list;
00352             previous->next = tempNode;
00353             head = foundNode;
00354         }
00355     }
00356
00357     return head; /*! the head of the list */
00358 }

```

### 2.1.2.6 void cl\_insert\_after ( cl\_node \* list, int search\_value, int value, const char \* label )

Insert a node after the sought after value

#### Parameters

<i>list</i>	the list on to which you want to add
<i>search_value</i>	the value we're looking for
<i>value</i>	the new value for that node
<i>label</i>	the new label for that node

contains the sought after value

Definition at line 269 of file [cachelist.c](#).

References [make\\_node\(\)](#).

```

00271 {
00272     cl_node* node = list; /*! contains the sought after value */
00273
00274     /* look for the value */
00275     while(node != NULL)
00276     {
00277         if(node->value == search_value)
00278         {
00279             break;
00280         }
00281         else
00282         {
00283             node = node->next;
00284         }
00285     }
00286

```

```

00287  /* the value was found */
00288  if(node != NULL)
00289  {
00290      cl_node* newNode = make_node(value, label);
00291      newNode->next = node->next;
00292      node->next = newNode;
00293  }
00294
00295  }

```

### 2.1.2.7 cl\_node\* cl\_insert\_before ( cl\_node \* list, int search\_value, int value, const char \* label )

Insert a new node before a given value

#### Parameters

<i>list</i>	the list on to which you want to add
<i>search_value</i>	the value we're looking for
<i>value</i>	the new value for that node
<i>label</i>	the new label for that node

#### Returns

head of the list

contains the sought after value

the node prior to the found value

the head of the list

new node to insert into the list

Definition at line 207 of file [cachelist.c](#).

References [make\\_node\(\)](#).

```

00209 {
00210     cl_node* node = list;      /*! contains the sought after value */
00211     cl_node* previous = list; /*! the node prior to the found value */
00212     cl_node* head = list;     /*! the head of the list */
00213     cl_node* newNode = NULL;  /*! new node to insert into the list */
00214
00215     /* look for the specified value */
00216     while(node != NULL)
00217     {
00218         if(node->value == search_value)
00219         {
00220             break;
00221         }
00222         else
00223         {
00224             previous = node;
00225             node = node->next;
00226         }
00227     }
00228
00229     /* if nothing is found insert nothing */
00230     if(node == NULL)
00231     {
00232         return list;
00233     }
00234
00235     /* only make a node if we're going to insert it */
00236     newNode = make_node(value, label);
00237
00238     /* this only happens with 2+ nodes */
00239     if(previous != node)
00240     {
00241         previous->next = newNode;
00242     }

```



```

00243
00244  /* reset the head if we're inserting at the beginning */
00245  else
00246  {
00247      head = newNode;
00248  }
00249  newNode->next = node;
00250
00251  return head; /* the head of the list */
00252 }

```

### 2.1.2.8 `cl_node* cl_remove ( cl_node * list, int search_value )`

Removes the first node with a matching value

#### Parameters

<i>list</i>	the list in which you are searching
<i>search_value</i>	the sought after value

#### Returns

head of the list

remove this node

node prior to the one marked for deleting

Definition at line 153 of file [cachelist.c](#).

```

00154 {
00155     cl_node* node = list;      /*! remove this node */
00156     cl_node* previous = list; /*! node prior to the one marked for deleting */
00157
00158     /* look for the specified value */
00159     while(node != NULL)
00160     {
00161         /* found value, exterminate! */
00162         if(node->value == search_value)
00163         {
00164             /* if we're dealing with the head of the list */
00165             if(node == previous)
00166             {
00167                 list = list->next;
00168                 free(node);
00169             }
00170             /* all other members of the list */
00171             else
00172             {
00173                 previous->next = node->next;
00174                 free(node);
00175             }
00176             return list;
00177         }
00178         /* no value found, keep moving */
00179         else
00180         {
00181             previous = node;
00182             node = node->next;
00183         }
00184     }
00185
00186     return list; /* the head of the list */
00187 }

```

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

Makes a new node for a single linked list

## Parameters

<i>value</i>	the new node's new value
<i>label</i>	the new node's new label

## Returns

the head of the list

a new list node

Definition at line 43 of file [cachelist.c](#).

Referenced by [cl\\_add\\_end\(\)](#), [cl\\_add\\_front\(\)](#), [cl\\_insert\\_after\(\)](#), and [cl\\_insert\\_before\(\)](#).

```

00044 {
00045     cl_node *node = (cl_node *)malloc(sizeof(cl_node)); /*! a new list node */
00046
00047     if (!node)
00048     {
00049         printf("Can't allocate new node.\n");
00050         exit(1);
00051     }
00052
00053     node->value = value;
00054     node->next = NULL;
00055
00056     /* Be sure not to overwrite memory */
00057     strncpy(node->label, label, LABEL_SIZE - 1);
00058     node->label[LABEL_SIZE - 1] = 0;
00059
00060     return node; /* the new node */
00061 }
```

## 2.2 cachelist.c

```

00001 /*****
00002  *!
00003  *file    cachelist.c
00004  *author  Cody Morgan
00005  *par     email: cody.morgan@digipen.edu
00006  *par     DigiPen login: cody.morgan
00007  *par     Course: CS180
00008  *par     Section A
00009  *par     Assignment #1
00010  *date    19 SEP 2018
00011  *brief
00012  *this is a single linked list of nodes containing a value and a label.
00013
00014  *operations include:
00015  * - cl_add_end      : add item to the end
00016  * - cl_add_front    : add item to the front
00017  * - cl_remove       : remove a give value from the list
00018  * - cl_insert_before : insert a new value before this value
00019  * - cl_insert_after  : insert a new value after this value
00020  * - cl_find         : find this value
00021  * - cl_destroy      : destroy this value (first)
00022  * - cl_dump         : print this list
00023  */
00024 /*****
00025
00026 #include <stdio.h>      /* printf      */
00027 #include <stdlib.h>     /* malloc, exit */
00028 #include <string.h>     /* strncpy      */
00029 #include "cachelist.h" /* cachelist stuff */
00030
00031 /*!*****
00032  *Makes a new node for a single linked list
00033
00034  *param value
00035  *the new node's new value
00036
00037  *param label
```

```

00038     the new node's new label
00039
00040     \return
00041     the head of the list
00042     *****/
00043 static cl_node *make_node(int value, const char *label)
00044 {
00045     cl_node *node = (cl_node *)malloc(sizeof(cl_node)); /*! a new list node */
00046
00047     if (!node)
00048     {
00049         printf("Can't allocate new node.\n");
00050         exit(1);
00051     }
00052
00053     node->value = value;
00054     node->next = NULL;
00055
00056     /* Be sure not to overwrite memory */
00057     strncpy(node->label, label, LABEL_SIZE - 1);
00058     node->label[LABEL_SIZE - 1] = 0;
00059
00060     return node; /* the new node */
00061 }
00062
00063 /*!*****
00064 Dumps all of the list info to the screen for your viewing pleasure
00065
00066 \param list
00067     the list to dump
00068 *****/
00069 void cl_dump(const cl_node *list)
00070 {
00071     printf("===== \n");
00072     while (list)
00073     {
00074         printf("%4i: %s \n", list->value, list->label);
00075         list = list->next;
00076     }
00077 }
00078
00079 /*!*****
00080 Adds a new node to the end of the list
00081
00082 \param list
00083     the list on to which you want to add
00084
00085 \param value
00086     the new value for that node
00087
00088 \param label
00089     the new label for that node
00090
00091 \return
00092     head of the list
00093 *****/
00094 cl_node *cl_add_end(cl_node *list, int value, const char *label)
00095 {
00096     cl_node* newNode = make_node(value, label); /*! a new list node */
00097     cl_node* endOfList = list; /*! the end of the new list */
00098
00099     /* find the end of the list */
00100     if(list == NULL)
00101     {
00102         list = newNode;
00103     }
00104     else
00105     {
00106         while(endOfList->next != NULL)
00107         {
00108             endOfList = endOfList->next;
00109         }
00110         endOfList->next = newNode;
00111     }
00112
00113     return list; /* the head of the list */
00114 }
00115
00116 /*!*****
00117 Adds a new node to the front of the list
00118

```

```

00119  \param list
00120      the list on to which you want to add
00121
00122  \param value
00123      the new value for that node
00124
00125  \param label
00126      the new label for that node
00127
00128  \return
00129      head of the list
00130  *****/
00131  cl_node *cl_add_front(cl_node *list, int value, const char *label)
00132  {
00133      cl_node* newNode = make_node(value, label); /*! the new list node */
00134
00135      /* connect the old list */
00136      newNode->next = list;
00137
00138      return newNode; /* the head of the new list */
00139  }
00140
00141  /*!*****
00142      Removes the first node with a matching value
00143
00144      \param list
00145          the list in which you are searching
00146
00147      \param search_value
00148          the sought after value
00149
00150      \return
00151          head of the list
00152  *****/
00153  cl_node *cl_remove(cl_node *list, int search_value)
00154  {
00155      cl_node* node = list; /*! remove this node */
00156      cl_node* previous = list; /*! node prior to the one marked for deleting */
00157
00158      /* look for the specified value */
00159      while(node != NULL)
00160      {
00161          /* found value, exterminate! */
00162          if(node->value == search_value)
00163          {
00164              /* if we're dealing with the head of the list */
00165              if(node == previous)
00166              {
00167                  list = list->next;
00168                  free(node);
00169              }
00170              /* all other members of the list */
00171              else
00172              {
00173                  previous->next = node->next;
00174                  free(node);
00175              }
00176              return list;
00177          }
00178          /* no value found, keep moving */
00179          else
00180          {
00181              previous = node;
00182              node = node->next;
00183          }
00184      }
00185
00186      return list; /* the head of the list */
00187  }
00188
00189  /*!*****
00190      Insert a new node before a given value
00191
00192      \param list
00193          the list on to which you want to add
00194
00195      \param search_value
00196          the value we're looking for
00197
00198      \param value
00199          the new value for that node

```

```

00200
00201 \param label
00202     the new label for that node
00203
00204 \return
00205     head of the list
00206 *****
00207 cl_node *cl_insert_before(cl_node *list, int search_value, int value,
00208                          const char *label)
00209 {
00210     cl_node* node = list;      /*! contains the sought after value */
00211     cl_node* previous = list; /*! the node prior to the found value */
00212     cl_node* head = list;     /*! the head of the list */
00213     cl_node* newNode = NULL;  /*! new node to insert into the list */
00214
00215     /* look for the specified value */
00216     while(node != NULL)
00217     {
00218         if(node->value == search_value)
00219         {
00220             break;
00221         }
00222         else
00223         {
00224             previous = node;
00225             node = node->next;
00226         }
00227     }
00228
00229     /* if nothing is found insert nothing */
00230     if(node == NULL)
00231     {
00232         return list;
00233     }
00234
00235     /* only make a node if we're going to insert it */
00236     newNode = make_node(value, label);
00237
00238     /* this only happens with 2+ nodes */
00239     if(previous != node)
00240     {
00241         previous->next = newNode;
00242     }
00243
00244     /* reset the head if we're inserting at the beginning */
00245     else
00246     {
00247         head = newNode;
00248     }
00249     newNode->next = node;
00250
00251     return head; /* the head of the list */
00252 }
00253
00254 /*!*****
00255 Insert a node after the sought after value
00256
00257 \param list
00258     the list on to which you want to add
00259
00260 \param search_value
00261     the value we're looking for
00262
00263 \param value
00264     the new value for that node
00265
00266 \param label
00267     the new label for that node
00268 *****
00269 void cl_insert_after(cl_node *list, int search_value, int value,
00270                   const char *label)
00271 {
00272     cl_node* node = list; /*! contains the sought after value */
00273
00274     /* look for the value */
00275     while(node != NULL)
00276     {
00277         if(node->value == search_value)
00278         {
00279             break;
00280         }

```

```

00281     else
00282     {
00283         node = node->next;
00284     }
00285 }
00286
00287 /* the value was found */
00288 if(node != NULL)
00289 {
00290     cl_node* newNode = make_node(value, label);
00291     newNode->next = node->next;
00292     node->next = newNode;
00293 }
00294
00295 }
00296
00297 /*!*****
00298 Find a given value,
00299 count the number of compares it took to find,
00300 and finally cache that value if enabled
00301
00302 \param list
00303     the linked list
00304
00305 \param search_value
00306     the sought after value
00307
00308 \param cache
00309     bool to determine if the found value will be cached or not
00310
00311 \param compares
00312     how many compares it took to find that value
00313
00314 \return
00315     the head of the list
00316 *****/
00317 cl_node *cl_find(cl_node *list, int search_value, int cache, int *compares)
00318 {
00319     cl_node* node = list;        /*! current node while walking the list */
00320     cl_node* head = list;        /*! head of the list */
00321     cl_node* foundNode = NULL; /*! contains the sought after value */
00322     cl_node* previous = list;    /*! node previous to the found node */
00323     int compares_ = 0;           /*! number of compares it took to find the node */
00324
00325     /* look for a matching value */
00326     while(node != NULL)
00327     {
00328         compares_++;
00329         if(node->value == search_value)
00330         {
00331             foundNode = node;
00332             break;
00333         }
00334         else
00335         {
00336             previous = node;
00337             node = node->next;
00338         }
00339     }
00340
00341     /* found a matching value */
00342     if(foundNode != NULL)
00343     {
00344         *compares = compares_;
00345
00346         /* cache the node if enabled
00347          * ensure that the found node isn't the head already */
00348         if(cache && foundNode != list)
00349         {
00350             cl_node* tempNode = foundNode->next;
00351             foundNode->next = list;
00352             previous->next = tempNode;
00353             head = foundNode;
00354         }
00355     }
00356
00357     return head; /*! the head of the list */
00358 }
00359
00360 /*!*****
00361 Frees all the memory used for the given list

```

```
00362
00363 \param list
00364     the list you want to set free
00365 *****/
00366 void cl_destroy(cl_node *list)
00367 {
00368     cl_node* nextNode; /* keep destroying until this is NULL */
00369
00370     while(list != NULL)
00371     {
00372         nextNode = list->next;
00373         free(list);
00374         list = nextNode;
00375     }
00376 }
```

## Index

- cachelist.c, [1](#)
  - cl\_add\_end, [2](#)
  - cl\_add\_front, [3](#)
  - cl\_destroy, [3](#)
  - cl\_dump, [5](#)
  - cl\_find, [5](#)
  - cl\_insert\_after, [6](#)
  - cl\_insert\_before, [7](#)
  - cl\_remove, [8](#)
  - make\_node, [8](#)
- cl\_add\_end
  - cachelist.c, [2](#)
- cl\_add\_front
  - cachelist.c, [3](#)
- cl\_destroy
  - cachelist.c, [3](#)
- 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, [8](#)