Assignment 1 : Linked List

Generated by Doxygen 1.8.6

Thu Sep 20 2018 09:37:06

Contents

1 File Index

1.1 File List

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

cachelist.c **This is a single linked list of nodes containing a value and a label**	**1**

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

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

list	the list on to which you want to add
value	the new value for that node
label	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().

```
{
  cl_node* newNode = make_node(value, label); /*! a new list node */
  cl_node* endOfList = list;                  /*! the end of the new list */

  /* find the end of the list */
  if(list == NULL)
  {
    list = newNode;
  }
  else
  {
    while(endOfList->next != NULL)
    {
      endOfList = endOfList->next;
    }
    endOfList->next = newNode;
  }

  return list; /* the head of the list */
}
```

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

list	the list on to which you want to add
value	the new value for that node
label	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().

```
{
  cl_node* newNode = make_node(value, label); /*! the new list node */

  /* connect the old list */
  newNode->next = list;

  return newNode; /* the head of the new list */
}
```

2.1.2.3 void cl_destroy (cl_node ∗ *list*)

Frees all the memory used for the given list

Parameters

list	the list you want to set free

Definition at line 366 of file cachelist.c.

```
{
  cl_node* nextNode; /* keep destroying until this is NULL */

  while(list != NULL)
  {
    nextNode = list->next;
    free(list);
    list = nextNode;
  }
}
```

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

list	the list to dump

Definition at line 69 of file cachelist.c.

```
{
  printf("==================\n");
  while (list)
  {
    printf("%4i: %s\n", list->value, list->label);
    list = list->next;
  }
}
```

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

list	the linked list
search_value	the sought after value
cache	bool to determine if the found value will be cached or not
compares	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.

```
{
  cl_node* node = list;      /*! current node while walking the list */
  cl_node* head = list;      /*! head of the list */
  cl_node* foundNode = NULL; /*! contains the sought after value */
  cl_node* previous = list;  /*! node previous to the found node */
  int compares_ = 0;         /*! number of compares it took to find the node */

  /* look for a matching value */
  while(node != NULL)
  {
    compares_++;
    if(node->value == search_value)
    {
      foundNode = node;
      break;
    }
    else
    {
      previous = node;
      node = node->next;
    }
  }

  /* found a matching value */
  if(foundNode != NULL)
  {
    *compares = compares_;

    /* cache the node if enabled
     * ensure that the found node isn't the head already */
    if(cache && foundNode != list)
    {
      cl_node* tempNode = foundNode->next;
      foundNode->next = list;
      previous->next = tempNode;
      head = foundNode;
    }
  }

  return head; /*! the head of the list */
}
```

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

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

contains the sought after value

Definition at line 269 of file cachelist.c.

References make_node().

```
{
  cl_node* node = list; /*! contains the sought after value */

  /* look for the value */
  while(node != NULL)
  {
    if(node->value == search_value)
    {
      break;
    }
    else
    {
      node = node->next;
    }
  }

  /* the value was found */
  if(node != NULL)
  {
    cl_node* newNode = make_node(value, label);
    newNode->next = node->next;
    node->next = newNode;
  }

}
```

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

list	the list on to which you want to add
search_value	the value we're looking for
value	the new value for that node
label	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().

```
{
  cl_node* node = list;     /*! contains the sought after value */
  cl_node* previous = list; /*! the node prior to the found value */
  cl_node* head = list;     /*! the head of the list */
  cl_node* newNode = NULL;  /*! new node to insert into the list */

  /* look for the specified value */
  while(node != NULL)
  {
    if(node->value == search_value)
    {
      break;
    }
    else
    {
      previous = node;
      node = node->next;
    }
  }

  /* if nothing is found insert nothing */
  if(node == NULL)
  {
    return list;
  }

  /* only make a node if we're going to insert it */
  newNode = make_node(value, label);

  /* this only happens with 2+ nodes */
  if(previous != node)
  {
    previous->next = newNode;
  }

  /* reset the head if we're inserting at the beginning */
  else
  {
    head = newNode;
  }
  newNode->next = node;

  return head; /* the head of the list */
}
```

2.1.2.8 cl_node∗ cl_remove (cl_node ∗ *list,* int *search_value*)

Removes the first node with a matching value

Parameters

list	the list in which you are searching
search_value	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.

```
{
  cl_node* node = list;     /*! remove this node */
  cl_node* previous = list; /*! node prior to the one marked for deleting */

  /* look for the specified value */
  while(node != NULL)
  {
    /* found value, exterminate! */
    if(node->value == search_value)
    {
      /* if we're dealing with the head of the list */
      if(node == previous)
      {
        list = list->next;
        free(node);
      }
      /* all other members of the list */
      else
      {
        previous->next = node->next;
        free(node);
      }
        return list;
    }
    /* no value found, keep moving */
    else
    {
      previous = node;
      node = node->next;
    }
  }

  return list; /* the head of the list */
}
```

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

value	the new node's new value
label	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
```

```
    the new node’s new label

  \return
    the head of the list
******************************************************************************/
static cl_node *make_node(int value, const char *label)
{
  cl_node *node = (cl_node *)malloc(sizeof(cl_node)); /*! a new list node */

  if (!node)
  {
    printf("Can’t allocate new node.\n");
    exit(1);
  }

  node->value = value;
  node->next = NULL;

    /* Be sure not to overwrite memory */
  strncpy(node->label, label, LABEL_SIZE - 1);
  node->label[LABEL_SIZE - 1] = 0;

  return node; /* the new node */
}

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

  \param list
    the list to dump
******************************************************************************/
void cl_dump(const cl_node *list)
{
  printf("==================\n");
  while (list)
  {
    printf("%4i: %s\n", list->value, list->label);
    list = list->next;
  }
}

/*!****************************************************************************
  Adds a new node to the end of the list

  \param list
    the list on to which you want to add

  \param value
    the new value for that node

  \param label
    the new label for that node

  \return
    head of the list
******************************************************************************/
cl_node *cl_add_end(cl_node *list, int value, const char *label)
{
  cl_node* newNode = make_node(value, label); /*! a new list node */
  cl_node* endOfList = list;                  /*! the end of the new list */

  /* find the end of the list */
  if(list == NULL)
  {
    list = newNode;
  }
  else
  {
    while(endOfList->next != NULL)
    {
      endOfList = endOfList->next;
    }
    endOfList->next = newNode;
  }

  return list; /* the head of the list */
}

/*!****************************************************************************
  Adds a new node to the front of the list

  \param list
    the list on to which you want to add

  \param value
    the new value for that node

  \param label
    the new label for that node

  \return
    head of the list
*****************************************************************************/
cl_node *cl_add_front(cl_node *list, int value, const char *label)
{
  cl_node* newNode = make_node(value, label); /*! the new list node */

  /* connect the old list */
  newNode->next = list;

  return newNode; /* the head of the new list */
}

/*!****************************************************************************
  Removes the first node with a matching value

  \param list
    the list in which you are searching

  \param search_value
    the sought after value

  \return
    head of the list
*****************************************************************************/
cl_node *cl_remove(cl_node *list, int search_value)
{
  cl_node* node = list;     /*! remove this node */
  cl_node* previous = list; /*! node prior to the one marked for deleting */

  /* look for the specified value */
  while(node != NULL)
  {
    /* found value, exterminate! */
    if(node->value == search_value)
    {
      /* if we're dealing with the head of the list */
      if(node == previous)
      {
        list = list->next;
        free(node);
      }
      /* all other members of the list */
      else
      {
        previous->next = node->next;
        free(node);
      }
        return list;
    }
    /* no value found, keep moving */
    else
    {
      previous = node;
      node = node->next;
    }
  }

  return list; /* the head of the list */
}

/*!****************************************************************************
  Insert a new node before a given value

  \param list
    the list on to which you want to add

  \param search_value
    the value we're looking for

  \param value
    the new value for that node

  \param label
    the new label for that node

  \return
    head of the list
*****************************************************************************/
cl_node *cl_insert_before(cl_node *list, int search_value, int value,
                          const char *label)
{
  cl_node* node = list;     /*! contains the sought after value */
  cl_node* previous = list; /*! the node prior to the found value */
  cl_node* head = list;     /*! the head of the list */
  cl_node* newNode = NULL;  /*! new node to insert into the list */

  /* look for the specified value */
  while(node != NULL)
  {
    if(node->value == search_value)
    {
      break;
    }
    else
    {
      previous = node;
      node = node->next;
    }
  }

  /* if nothing is found insert nothing */
  if(node == NULL)
  {
    return list;
  }

  /* only make a node if we're going to insert it */
  newNode = make_node(value, label);

  /* this only happens with 2+ nodes */
  if(previous != node)
  {
    previous->next = newNode;
  }

  /* reset the head if we're inserting at the beginning */
  else
  {
    head = newNode;
  }
  newNode->next = node;

  return head; /* the head of the list */
}

/*!****************************************************************************
  Insert a node after the sought after value

  \param list
    the list on to which you want to add

  \param search_value
    the value we're looking for

  \param value
    the new value for that node

  \param label
    the new label for that node
*****************************************************************************/
void cl_insert_after(cl_node *list, int search_value, int value,
                     const char *label)
{
  cl_node* node = list; /*! contains the sought after value */

  /* look for the value */
  while(node != NULL)
  {
    if(node->value == search_value)
    {
      break;
    }
```

```c
    else
    {
      node = node->next;
    }
  }

  /* the value was found */
  if(node != NULL)
  {
    cl_node* newNode = make_node(value, label);
    newNode->next = node->next;
    node->next = newNode;
  }

}

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

  \param list
    the linked list

  \param search_value
    the sought after value

  \param cache
    bool to determine if the found value will be cached or not

  \param compares
    how many compares it took to find that value

  \return
    the head of the list
*******************************************************************************/
cl_node *cl_find(cl_node *list, int search_value, int cache, int *compares)
{
  cl_node* node = list;      /*! current node while walking the list */
  cl_node* head = list;      /*! head of the list */
  cl_node* foundNode = NULL; /*! contains the sought after value */
  cl_node* previous = list;  /*! node previous to the found node */
  int compares_ = 0;         /*! number of compares it took to find the node */

  /* look for a matching value */
  while(node != NULL)
  {
    compares_++;
    if(node->value == search_value)
    {
      foundNode = node;
      break;
    }
    else
    {
      previous = node;
      node = node->next;
    }
  }

  /* found a matching value */
  if(foundNode != NULL)
  {
    *compares = compares_;

    /* cache the node if enabled
     * ensure that the found node isn't the head already */
    if(cache && foundNode != list)
    {
      cl_node* tempNode = foundNode->next;
      foundNode->next = list;
      previous->next = tempNode;
      head = foundNode;
    }
  }

  return head; /*! the head of the list */
}

/*!*****************************************************************************
  Frees all the memory used for the given list

  \param list
    the list you want to set free
******************************************************************************/
void cl_destroy(cl_node *list)
{
  cl_node* nextNode; /* keep destroying until this is NULL */

  while(list != NULL)
  {
    nextNode = list->next;
    free(list);
    list = nextNode;
  }
}
```

Index