COMP1511 18s2

Week-09 Laboratory Exercises

Notice: "autotest" and "give" submission commands are now available!

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

In particular, re-read (and/or watch the lecture video for) the lecture slides on the topics:

- "Linked List" (slide-12 to slide-28)
- "Recursion, Linked List with Recursion" (slide-2 to slide-9)

All the questions in this lab are based on what we discussed in the lectures.

You can also watch **videos** available on the following page for further explanations:

- Linked Lists
  - (video) Linked Lists - Prerequisites
  - (video) Linked Lists - Creating and traversing a linked list (Coding)
  - (video) Linked List - Sorted Insert
  - (video) Linked Lists - Adding elements to a Linked List (Coding)
  - (video) Linked List - Delete
  - (video) Linked Lists - Deleting elements from a Linked List (Coding)

We will only assess lab Exercises 01, 02, 03 and 04 to derive marks for this lab. However, you will learn a lot by attempting and solving the challenge exercises.

Getting Started

Create a new directory for this lab called `lab09` by typing:

```
$ mkdir lab09
```

Change to this directory by typing:

```
$ cd lab09
```

Exercise-01: Return How Many Even Numbers are in a Linked List (individual)

Hint: Lecture slides on list traversal (slide-12 and slide-16) in "Linked List"

Download **count_even_list.c**, the starting code for this exercise or use this cp command:

```
$ cp -n /web/cs1511/18s2/activities/count_even_list/count_even_list.c .
```

Note **count_even_list.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

Your task is to add code to this function:

```
// return the number of even values in a linked list
int count_even(struct node *head) {

    // PUT YOUR CODE HERE (change the next line!)
    return 42;

}
```

count_even is given one argument, **head**, which is the pointer to the first node in a linked list.
Add code to **count_even** so that its returns the number of even values in the linked list.

For example if the linked list contains these 8 elements:

```
16, 7, 8, 12, 13, 19, 21, 12
```

count_even should return **4**, because these 4 elements are even:

```
16, 8, 12, 12
```

Testing

count_even_list.c also contains a **main** function which allows you to test your **count_even** function.
This main function:

- converts the command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**
- calls **count_even(head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **count_even** function will be called directly in marking. The main function is only to let you test your **count_even** function

Here is how you use main function allows you to test **count_even**:

```
$ cp -n /web/cs1511/18s2/activities/count_even_list/count_even_list.c .
$ dcc count_even_list.c -o count_even_list
$ ./count_even_list 16 7 8 12 13 19 21 12
4
$ ./count_even_list 2 4 6 2 4 6
6
$ ./count_even_list 3 5 7 11 13 15 17 19 23 29
0
$ ./count_even_list 2 4 8 16 32 64 128 256
8
$ ./count_even_list
0
```

Assumptions/Restrictions/Clarifications.

An even number is divisible by 2.
count_even should return a single integer.

count_even should not change the linked list it is given. Your function should not change the next or data fields of list nodes.

count_even should not use arrays.

count_even should not call malloc.

count_even should not call scanf (or getchar or fgets).

You can assume the linked list only contains positive integers.

count_even should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1511 autotest count_even_list
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk09_count_even_list count_even_list.c
```

Exercise-02: Delete First Element of a Linked List (individual)

Hint: Lecture slides on delete (slide-27 and slide-28) in "Linked List"

Download **list_delete_first.c**, the starting code for this exercise or use this cp command:

```
$ cp -n /web/cs1511/18s2/activities/list_delete_first/list_delete_first.c .
```

Note **list_delete_first.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

Your task is to add code to this function:

```
struct node *delete_first(struct node *head) {

    // PUT YOUR CODE HERE (change the next line!)
    return NULL;

}
```

delete_first is given one argument, **head**, which is the pointer to the first node in the linked list.
Add code to **delete_first** so that it deletes the first node from list.

delete_first should return a pointer to the new first node in the list.

If the list is now empty **delete_first** should return **NULL**.

delete_first should call **free** to free the memory of the node it deletes.

For example if the linked list contains these 8 elements:

```
16, 7, 8, 12, 13, 19, 21, 12
```

delete_first should return a pointer to a list with these elements:

```
7, 8, 12, 13, 19, 21, 12
```

Hint: this is a simple task requiring only a few lines of code.

Testing

list_delete_first.c also contains a **main** function which allows you to test your **delete_first** function. It converts command-line arguments to a linked list, calls **delete_first**, and then prints the result.
Do not change this main function. If you want to change it, you have misread the question.

Your **delete_first** function will be called directly in marking. The main function is only to let you test your **delete_first** function

Here is how you the main function allows you to test **delete_first**:

```
$ cp -n /web/cs1511/18s2/activities/list_delete_first/list_delete_first.c .
$ dcc list_delete_first.c -o list_delete_first
$ ./list_delete_first 16 7 8 12 13 19 21 12
[7, 8, 12, 13, 19, 21, 12]
$ ./list_delete_first 2 4 6 2 4 6
[4, 6, 2, 4, 6]
$ ./list_delete_first 42
[]
$ ./list_delete_first
[]
```

Assumptions/Restrictions/Clarifications.

delete_first should call **free** to free the memory for the node it deletes
delete_first should not change the data fields of list nodes.

delete_first should not use arrays.

delete_first should not call malloc.

delete_first should not call scanf (or getchar or fgets).

delete_first should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest list_delete_first
```

When you are finished on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk09_list_delete_first list_delete_first.c
```

You must run **give** from your own account before **Sunday 23 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-03: Delete First Element Containing A Value from a Linked List (pair)

Hint: Lecture slides on "Finding an Item" and "Delete" in "Linked List"

Download **list_delete_contains.c**, the starting code for this exercise or use this cp command:

```
$ cp -n /web/cs1511/18s2/activities/list_delete_contains/list_delete_contains.c .
```

Note **list_delete_contains.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

Your task is to add code to this function:

```
// Delete the first node in the list containing the value `value`.
// The deleted node is freed.
// If no node contains `value`, the list is not changed.
// The head of the list is returned.
struct node *delete_contains(int value, struct node *head) {

    // PUT YOUR CODE HERE (change the next line!)
    return NULL;

}
```

delete_contains is given two argument, **value** and **head**. **value** is an int. **head** is the pointer to the first node in a linked list. Add code to **delete_contains** so that it deletes the first node in the linked list that whose *data* field equals **value**.

If **value** does not occur in the linked list, the list should not be changed.

If **value** occurs more than once in the linked list, only the first occurrence should be deleted.

delete_contains should return a pointer to the new list.

If the list is now empty **delete_contains** should return **NULL**.

delete_contains should call **free** to free the memory of the node it deletes.

For example if **value** is **12** and the linked list contains these 8 elements:

```
16, 7, 8, 12, 13, 19, 21, 12
```

delete_contains should return a pointer to a list with these elements:

```
16, 7, 8, 13, 19, 21, 12
```

Testing

list_delete_contains.c also contains a **main** function which allows you to test your **delete_contains** function.
This main function:

- converts the first command-line argument to **value**
- converts the remaining command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**
- calls **delete_contains(value, head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **delete_contains** function will be called directly in marking. The main function is only to let you test your **delete_contains** function

```
$ cp -n /web/cs1511/18s2/activities/list_delete_contains/list_delete_contains.c .
$ dcc list_delete_contains.c -o list_delete_contains
$ ./list_delete_contains 12 16 7 8 12 13 19 21 12
[16, 7, 8, 13, 19, 21, 12]
$ ./list_delete_contains 42 16 7 8 12 13 19 21 12
[16, 7, 8, 12, 13, 19, 21, 12]
$ ./list_delete_contains 2 4 6 2 4 6
[4, 6, 4, 6]
$ ./list_delete_contains 42 42
[]
$ ./list_delete_contains 42
[]
```

Assumptions/Restrictions/Clarifications.

delete_contains should call **free** to free the memory for the node it deletes
delete_first should not change the data fields of list nodes.

delete_contains should not use arrays.

delete_contains should not call malloc.

delete_contains should not call scanf (or getchar or fgets).

delete_contains should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest list_delete_contains
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk09_list_delete_contains list_delete_contains.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 23 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-04: Remove Consecutive Repeated Values From List (pair)

Hint: Lecture slides on "Linked List: Previous pattern" (slide-13) and "Delete a Node from a List" (slide-28) in "Linked List"

Download **list_delete_repeated.c**, the starting code for this exercise or use this cp command:

```
$ cp -n /web/cs1511/18s2/activities/list_delete_repeated/list_delete_repeated.c .
```

Note **list_delete_repeated.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

Your task is to add code to this function:

```
struct node *delete_repeated(struct node *head) {

    // PUT YOUR CODE HERE (change the next line!)
    return NULL;

}
```

delete_repeated is given one argument, **head**, a pointer to the first node in a linked list.
Add code to **delete_repeated** so that it deletes any **consecutive** repeated values from the linked list.

If there are no **consecutive** repeated values, the list should not be changed.

delete_repeated should return a pointer to the new list.

delete_repeated should call **free** to free the memory of the node it deletes.

For example if the linked list contains these 9 elements:

```
16, 16, 7, 8, 12, 19, 19, 19, 7
```

delete_repeated should return a pointer to a list with 6 elements:

```
16, 7, 8, 12, 19, 7
```

Testing

list_delete_repeated.c also contains a **main** function which allows you to test your **delete_repeated** function.
This main function:

- converts the command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**
- calls **delete_repeated(head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **delete_repeated** function will be called directly in marking. The main function is only to let you test your **delete_repeated** function

```
$ cp -n /web/cs1511/18s2/activities/list_delete_repeated/list_delete_repeated.c .
$ dcc list_delete_repeated.c -o list_delete_repeated
$ ./list_delete_repeated 16 16 7 8 12 19 19 19 7
[16, 7, 8, 12, 19, 7]
$ ./list_delete_repeated 42 42 42 42 42 42 42
[42]
$ ./list_delete_repeated 42 42 42 77 42 42 42 42
[42, 77, 42]
$  ./list_delete_repeated 1 0 1 0 1 0 1 0 1
[1, 0, 1, 0, 1, 0, 1, 0, 1]
$  ./list_delete_repeated 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
[1, 2, 3, 4, 5]
$ ./list_delete_repeated
[]
```

Assumptions/Restrictions/Clarifications.

If there are two repeated values, **delete_repeated** may delete either list node.
Similarly if there are **n** repeated values **delete_repeated** may delete any **n - 1** of the list nodes.

delete_repeated should call **free** to free the memory for the node it deletes

delete_repeated should not change the data fields of list nodes.

delete_repeated should not use arrays.

delete_repeated should not call malloc.

delete_repeated should not call scanf (or getchar or fgets).

delete_repeated should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1511 autotest list_delete_repeated
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give  cs1511  wk09_list_delete_repeated  list_delete_repeated.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 23 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-05: Challenge Exercise: Reverse a Linked List (individual)

Download **list_reverse.c**, the starting code for this exercise or use this cp command:

```
$ cp -n /web/cs1511/18s2/activities/list_reverse/list_reverse.c .
```

Note **list_reverse.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

Your task is to add code to this function:

```
struct node *reverse(struct node *head) {

    // PUT YOUR CODE HERE (change the next line!)
    return NULL;

}
```

list_reverse is given one argument, **head**, which is the pointer to the first node in the linked list.
Add code to **reverse** which rearranges the list to be in reverse order.

reverse should return a pointer to the new list.

reverse must rearrange the list by changing the **next** fields of nodes.

reverse must not change the **data** fields of nodes.

For example if the linked list contains these 8 elements:

```
16, 7, 8, 12, 13, 19, 21, 12
```

reverse should return a pointer to a list with these elements:

```
12, 21, 19, 13, 12, 8, 7, 16
```

Testing

list_reverse.c also contains a **main** function which allows you to test your **list_reverse** function.
This main function:

- converts the command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**
- calls **reverse(head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **list_reverse** function will be called directly in marking. The main function is only to let you test your **list_reverse** function

```
$ cp -n /web/cs1511/18s2/activities/list_reverse/list_reverse.c .
$ dcc list_reverse.c -o list_reverse
$ ./list_reverse 16 7 8 12 13 19 21 12
[12, 21, 19, 13, 12, 8, 7, 16]
$ ./list_reverse 2 4 6 2 4 6
[6, 4, 2, 6, 4, 2]
$ ./list_reverse 42
[42]
$ ./list_reverse
[]
```

Assumptions/Restrictions/Clarifications.

list_reverse should not change the data fields of list nodes.
list_reverse should not use arrays.

list_reverse should not call malloc.

list_reverse should not call scanf (or getchar or fgets).

list_reverse should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest list_reverse
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk09_list_reverse list_reverse.c
```

You must run give before **Sunday 23 September 23:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Submission

When you are finished each exercises make sure you submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Sunday 23 September 23:59:59** to submit your work.

COMP1511 18s2: Programming Fundamentals is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1511@cse.unsw.edu.au

CRICOS Provider 00098G