0x13. C - More singly linked lists

C | Algorithm | Data structure

Weight: 1

Project over - took place from Aug 28, 2023 6:00 AM to Aug 30, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 90.0/90 mandatory & 35.0/35 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Calculation: 100.0% + (100.0% * 100.0%) == **200.0%**

Resources

Read or watch:

- Google (/rltoken/2-7-eVuWcPutbXf6YZZgiA)
- Youtube (/rltoken/wVWwl86ufLMsXeAigpxllg)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/jL0iK5DlEbQK5elwCNDa-g), **without the help of Google**:

General

- How to use linked lists
- Start to look for the right source of information without too much help

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc`, using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/betty-doc.pl)
- You are not allowed to use global variables
- No more than 5 functions per file
- The only C standard library functions allowed are `malloc`, `free` and `exit`. Any use of functions like `printf`, `puts`, `calloc`, `realloc` etc... is forbidden
- You are allowed to use _putchar (https://github.com/alx-tools/_putchar.c/blob/master/_putchar.c)
- You don't have to push `_putchar.c`, we will use our file. If you do it won't be taken into account
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions and the prototype of the function `_putchar` should be included in your header file called `lists.h`
- Don't forget to push your header file
- All your header files should be include guarded

More Info

Please use this data structure for this project:

```
/**
 * struct listint_s - singly linked list
 * @n: integer
 * @next: points to the next node
 *
 * Description: singly linked list node structure
 *
 */
typedef struct listint_s
{
    int n;
    struct listint_s *next;
} listint_t;
```

Tasks

0. Print list

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that prints all the elements of a `listint_t` list.

- Prototype: `size_t print_listint(const listint_t *h);`
- Return: the number of nodes
- Format: see example
- You are allowed to use `printf`

(/)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 0-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *new;
    listint_t hello = {8, NULL};
    size_t n;

    head = &hello;
    new = malloc(sizeof(listint_t));
    if (new == NULL)
    {
        printf("Error\n");
        return (1);
    }
    new->n = 9;
    new->next = head;
    head = new;
    n = print_listint(head);
    printf("-> %lu elements\n", n);
    free(new);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 0-main.c 0-print_listint.c -o a
julien@ubuntu:~/0x13. More singly linked lists$ ./a
9
8
-> 2 elements
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `0-print_listint.c`

Done! | Check your code | Get a sandbox | QA Review

1. List length (/)

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns the number of elements in a linked `listint_t` list.

- Prototype: `size_t listint_len(const listint_t *h);`

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 1-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *new;
    listint_t hello = {8, NULL};
    size_t n;

    head = &hello;
    new = malloc(sizeof(listint_t));
    if (new == NULL)
    {
        printf("Error\n");
        return (1);
    }
    new->n = 9;
    new->next = head;
    head = new;
    n = listint_len(head);
    printf("-> %lu elements\n", n);
    free(new);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 1-main.c 1-listint_len.c -o b
julien@ubuntu:~/0x13. More singly linked lists$ ./b
-> 2 elements
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`

- Directory: `0x13-more_singly_linked_lists`
- File: `1-listint_len.c`

Done! | Check your code | Get a sandbox | QA Review

2. Add node

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that adds a new node at the beginning of a `listint_t` list.

- Prototype: `listint_t *add_nodeint(listint_t **head, const int n);`
- Return: the address of the new element, or `NULL` if it failed

(/)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 2-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint(&head, 0);
    add_nodeint(&head, 1);
    add_nodeint(&head, 2);
    add_nodeint(&head, 3);
    add_nodeint(&head, 4);
    add_nodeint(&head, 98);
    add_nodeint(&head, 402);
    add_nodeint(&head, 1024);
    print_listint(head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 2-main.c 2-add_nodeint.c 0-print_listint.c -o c
julien@ubuntu:~/0x13. More singly linked lists$ ./c
1024
402
98
4
3
2
1
0
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `2-add_nodeint.c`

Done! | Check your code | Get a sandbox | QA Review

3. Add node at the end

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that adds a new node at the end of a `listint_t` list.

- Prototype: `listint_t *add_nodeint_end(listint_t **head, const int n);`
- Return: the address of the new element, or `NULL` if it failed

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 3-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 3-main.c 3-add_nodeint_end.c 0-print_listint.c -o d
julien@ubuntu:~/0x13. More singly linked lists$ ./d
0
1
2
3
4
98
402
1024
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `3-add_nodeint_end.c`

Done! | Check your code | Get a sandbox | QA Review

4. Free list

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that frees a `listint_t` list.

- Prototype: `void free_listint(listint_t *head);`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 4-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    free_listint(head);
    head = NULL;
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 4-main.c 3-add_nodeint_end.c 0-print_listint.c 4-free_listint.c -o e
julien@ubuntu:~/0x13. More singly linked lists$ valgrind ./e
==3643== Memcheck, a memory error detector
==3643== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3643== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3643== Command: ./e
==3643==
0
1
2
3
4
98
402
1024
==3643==
==3643== HEAP SUMMARY:
==3643==     in use at exit: 0 bytes in 0 blocks
==3643==   total heap usage: 9 allocs, 9 frees, 1,152 bytes allocated
==3643==
==3643== All heap blocks were freed -- no leaks are possible
==3643==
```

```
==3643== For counts of detected and suppressed errors, rerun with: -v
==3643== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `4-free_listint.c`

Done! | Check your code | Get a sandbox | QA Review

5. Free

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that frees a `listint_t` list.

- Prototype: `void free_listint2(listint_t **head);`
- The function sets the `head` to `NULL`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 5-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    free_listint2(&head);
    printf("%p\n", (void *)head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 5-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c -o f
julien@ubuntu:~/0x13. More singly linked lists$ valgrind ./f
==3843== Memcheck, a memory error detector
==3843== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3843== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3843== Command: ./f
==3843==
0
1
2
3
4
98
402
1024
(nil)
==3843==
==3843== HEAP SUMMARY:
==3843==     in use at exit: 0 bytes in 0 blocks
==3843==   total heap usage: 9 allocs, 9 frees, 1,152 bytes allocated
==3843==
==3843== All heap blocks were freed -- no leaks are possible
```

```
==3843==
==3843== For counts of detected and suppressed errors, rerun with: -v
==3843== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `5-free_listint2.c`

Done! | Check your code | Get a sandbox | QA Review

6. Pop

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that deletes the head node of a `listint_t` linked list, and returns the head node's data (n).

- Prototype: `int pop_listint(listint_t **head);`
- if the linked list is empty return `0`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 6-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    int n;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    n = pop_listint(&head);
    printf("- %d\n", n);
    print_listint(head);
    n = pop_listint(&head);
    printf("- %d\n", n);
    print_listint(head);
    free_listint2(&head);
    printf("%p\n", (void *)head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 6-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c 6-pop_listint.c -o g
julien@ubuntu:~/0x13. More singly linked lists$ valgrind ./g
==4369== Memcheck, a memory error detector
==4369== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4369== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4369== Command: ./g
==4369==
0
1
2
3
4
98
402
1024
```

```
- 0
(1)
2
3
4
98
402
1024
- 1
2
3
4
98
402
1024
(nil)
==4369==
==4369== HEAP SUMMARY:
==4369==     in use at exit: 0 bytes in 0 blocks
==4369==   total heap usage: 9 allocs, 9 frees, 1,152 bytes allocated
==4369==
==4369== All heap blocks were freed -- no leaks are possible
==4369==
==4369== For counts of detected and suppressed errors, rerun with: -v
==4369== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `6-pop_listint.c`

Done! | Check your code | Get a sandbox | QA Review

7. Get node at index

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns the nth node of a `listint_t` linked list.

- Prototype: `listint_t *get_nodeint_at_index(listint_t *head, unsigned int index);`
- where `index` is the index of the node, starting at `0`
- if the node does not exist, return `NULL`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 7-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *node;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    node = get_nodeint_at_index(head, 5);
    printf("%d\n", node->n);
    print_listint(head);
    free_listint2(&head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 7-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c 7-get_nodeint.c -o h
julien@ubuntu:~/0x13. More singly linked lists$ ./h
0
1
2
3
4
98
402
1024
98
0
1
2
3
4
98
402
```

```
1024
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `7-get_nodeint.c`

Done! | Check your code | Get a sandbox | QA Review

8. Sum list

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that returns the sum of all the data (n) of a `listint_t` linked list.

- Prototype: `int sum_listint(listint_t *head);`
- if the list is empty, return `0`

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 8-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    int sum;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    sum = sum_listint(head);
    printf("sum = %d\n", sum);
    free_listint2(&head);
    return (0);
}
julien@ubuntu:~/c0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 8-main.c 3-add_nodeint_end.c 5-free_listint2.c 8-sum_listint.c -o i
julien@ubuntu:~/0x13. More singly linked lists$ ./i
sum = 1534
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `8-sum_listint.c`

Done! | Check your code | Get a sandbox | QA Review

9. Insert

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that inserts a new node at a given position.

(/)

- Prototype: `listint_t *insert_nodeint_at_index(listint_t **head, unsigned int idx, int n);`
- where `idx` is the index of the list where the new node should be added. Index starts at `0`
- Returns: the address of the new node, or `NULL` if it failed
- if it is not possible to add the new node at index `idx`, do not add the new node and return `NULL`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 9-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    printf("-----------------\n");
    insert_nodeint_at_index(&head, 5, 4096);
    print_listint(head);
    free_listint2(&head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 9-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c 9-insert_nodeint.c -o j
julien@ubuntu:~/0x13. More singly linked lists$ ./j
0
1
2
3
4
98
402
1024
-----------------
0
1
2
3
4
4096
98
402
```

```
1024
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `9-insert_nodeint.c`

Done! | Check your code | Get a sandbox | QA Review

10. Delete at index

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that deletes the node at index `index` of a `listint_t` linked list.

- Prototype: `int delete_nodeint_at_index(listint_t **head, unsigned int index);`
- where `index` is the index of the node that should be deleted. Index starts at `0`
- Returns: `1` if it succeeded, `-1` if it failed

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 10-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 5);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
```

```
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    printf("-----------------\n");
    delete_nodeint_at_index(&head, 0);
    print_listint(head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 10-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c 10-delete_nodeint.c -o
k
julien@ubuntu:~/0x13. More singly linked lists$ valgrind ./k
==5571== Memcheck, a memory error detector
==5571== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5571== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5571== Command: ./k
==5571==
0
1
2
3
4
98
402
1024
-----------------
0
1
2
3
4
402
1024
-----------------
1
2
3
4
402
1024
-----------------
2
3
```

```
4
(4)402
1024
-----------------
3
4
402
1024
-----------------
4
402
1024
-----------------
402
1024
-----------------
1024
-----------------
-----------------
-----------------
-----------------
-----------------
-----------------
-----------------
-----------------
-----------------
-----------------
==5571==
==5571== HEAP SUMMARY:
==5571==     in use at exit: 0 bytes in 0 blocks
==5571==   total heap usage: 9 allocs, 9 frees, 1,152 bytes allocated
==5571==
==5571== All heap blocks were freed -- no leaks are possible
==5571==
==5571== For counts of detected and suppressed errors, rerun with: -v
==5571== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `10-delete_nodeint.c`

Done! | Check your code | Get a sandbox | QA Review

11. Reverse list

#advanced

Write a function that reverses a `listint_t` linked list.

(/)

- Prototype: `listint_t *reverse_listint(listint_t **head);`
- Returns: a pointer to the first node of the reversed list
- You are not allowed to use more than `1` loop.
- You are not allowed to use `malloc`, `free` or arrays
- You can only declare a maximum of two variables in your function

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 100-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;

    head = NULL;
    add_nodeint_end(&head, 0);
    add_nodeint_end(&head, 1);
    add_nodeint_end(&head, 2);
    add_nodeint_end(&head, 3);
    add_nodeint_end(&head, 4);
    add_nodeint_end(&head, 98);
    add_nodeint_end(&head, 402);
    add_nodeint_end(&head, 1024);
    print_listint(head);
    reverse_listint(&head);
    print_listint(head);
    free_listint2(&head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 100-main.c 3-add_nodeint_end.c 0-print_listint.c 5-free_listint2.c 100-reverse_listint.c
-o l
julien@ubuntu:~/0x13. More singly linked lists$ valgrind ./l
==3117== Memcheck, a memory error detector
==3117== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3117== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3117== Command: ./l
==3117==
0
1
2
3
4
98
402
1024
1024
402
98
4
3
```

```
2
(1)
0
==3117==
==3117== HEAP SUMMARY:
==3117==     in use at exit: 0 bytes in 0 blocks
==3117==   total heap usage: 9 allocs, 9 frees, 1,152 bytes allocated
==3117==
==3117== All heap blocks were freed -- no leaks are possible
==3117==
==3117== For counts of detected and suppressed errors, rerun with: -v
==3117== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `100-reverse_listint.c`

Done! | Check your code | Get a sandbox | QA Review

12. Print (safe version)

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that prints a `listint_t` linked list.

- Prototype: `size_t print_listint_safe(const listint_t *head);`
- Returns: the number of nodes in the list
- This function can print lists with a loop
- You should go through the list only once
- If the function fails, exit the program with status `98`
- Output format: see example

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 101-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *head2;
    listint_t *node;

    head2 = NULL;
    add_nodeint(&head2, 0);
    add_nodeint(&head2, 1);
    add_nodeint(&head2, 2);
    add_nodeint(&head2, 3);
    add_nodeint(&head2, 4);
    add_nodeint(&head2, 98);
    add_nodeint(&head2, 402);
    add_nodeint(&head2, 1024);
    print_listint_safe(head2);
    head = NULL;
    node = add_nodeint(&head, 0);
    add_nodeint(&head, 1);
    add_nodeint(&head, 2);
    add_nodeint(&head, 3);
    add_nodeint(&head, 4);
    node->next = add_nodeint(&head, 98);
    add_nodeint(&head, 402);
    add_nodeint(&head, 1024);
    print_listint_safe(head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 101-main.c 2-add_nodeint.c 101-print_listint_safe.c -o m
julien@ubuntu:~/0x13. More singly linked lists$ ./m
[0x1b500f0] 1024
[0x1b500d0] 402
[0x1b500b0] 98
[0x1b50090] 4
[0x1b50070] 3
[0x1b50050] 2
[0x1b50030] 1
[0x1b50010] 0
[0x1b50600] 1024
[0x1b505e0] 402
```

```
[0x1b505c0] 98
[0x1b505a0] 4
[0x1b50580] 3
[0x1b50560] 2
[0x1b50540] 1
[0x1b50110] 0
-> [0x1b505c0] 98
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `101-print_listint_safe.c`

Done! | Check your code | Get a sandbox | QA Review

13. Free (safe version)

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that frees a `listint_t` list.

- Prototype: `size_t free_listint_safe(listint_t **h);`
- This function can free lists with a loop
- You should go though the list only once
- Returns: the size of the list that was free'd
- The function sets the `head` to `NULL`

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 102-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *head2;
    listint_t *node;

    head2 = NULL;
    add_nodeint(&head2, 0);
    add_nodeint(&head2, 1);
    add_nodeint(&head2, 2);
    add_nodeint(&head2, 3);
    add_nodeint(&head2, 4);
    add_nodeint(&head2, 98);
    add_nodeint(&head2, 402);
    add_nodeint(&head2, 1024);
    print_listint_safe(head2);
    head = NULL;
    node = add_nodeint(&head, 0);
    add_nodeint(&head, 1);
    add_nodeint(&head, 2);
    add_nodeint(&head, 3);
    add_nodeint(&head, 4);
    node->next = add_nodeint(&head, 98);
    add_nodeint(&head, 402);
    add_nodeint(&head, 1024);
    print_listint_safe(head);
    free_listint_safe(&head2);
    free_listint_safe(&head);
    printf("%p, %p\n", (void *)head2, (void *)head);
    return (0);
}
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 102-main.c 2-add_nodeint.c 101-print_listint_safe.c 102-free_listint_safe.c -o n
julien@ubuntu:~/0x13. More singly linked lists$ ./n
[0x11260f0] 1024
[0x11260d0] 402
[0x11260b0] 98
[0x1126090] 4
[0x1126070] 3
[0x1126050] 2
[0x1126030] 1
```

```
[0x1126010] 0
[0x1126600] 1024
[0x11265e0] 402
[0x11265c0] 98
[0x11265a0] 4
[0x1126580] 3
[0x1126560] 2
[0x1126540] 1
[0x1126110] 0
-> [0x11265c0] 98
(nil), (nil)
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `102-free_listint_safe.c`

Done! | Check your code | Get a sandbox | QA Review

14. Find the loop

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that finds the loop in a linked list.

- Prototype: `listint_t *find_listint_loop(listint_t *head);`
- Returns: The address of the node where the loop starts, or `NULL` if there is no loop
- You are not allowed to use `malloc`, `free` or arrays
- You can only declare a maximum of two variables in your function

(7)

```
julien@ubuntu:~/0x13. More singly linked lists$ cat 103-main.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    listint_t *head;
    listint_t *head2;
    listint_t *node;

    head2 = NULL;
    add_nodeint(&head2, 0);
    add_nodeint(&head2, 1);
    add_nodeint(&head2, 2);
    add_nodeint(&head2, 3);
    add_nodeint(&head2, 4);
    add_nodeint(&head2, 98);
    add_nodeint(&head2, 402);
    add_nodeint(&head2, 1024);
    print_listint_safe(head2);
    node = find_listint_loop(head2);
    if (node != NULL)
    {
        printf("Loop starts at [%p] %d\n", (void *)node, node->n);
    }
    free_listint_safe(&head2);
    head = NULL;
    node = add_nodeint(&head, 0);
    add_nodeint(&head, 1);
    add_nodeint(&head, 2);
    add_nodeint(&head, 3);
    add_nodeint(&head, 4);
    add_nodeint(&head, 5);
    add_nodeint(&head, 6);
    node->next = add_nodeint(&head, 7);
    add_nodeint(&head, 98);
    add_nodeint(&head, 402);
    add_nodeint(&head, 1024);
    print_listint_safe(head);
    node = find_listint_loop(head);
    if (node != NULL)
    {
        printf("Loop starts at [%p] %d\n", (void *)node, node->n);
    }
    free_listint_safe(&head);
```

```
    return (0);
```

(/)

```
julien@ubuntu:~/0x13. More singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=gnu
89 103-main.c 2-add_nodeint.c 101-print_listint_safe.c 102-free_listint_safe.c 103-find_loo
p.c -o o
julien@ubuntu:~/0x13. More singly linked lists$ ./o
[0x13700f0] 1024
[0x13700d0] 402
[0x13700b0] 98
[0x1370090] 4
[0x1370070] 3
[0x1370050] 2
[0x1370030] 1
[0x1370010] 0
[0x1370560] 1024
[0x1370540] 402
[0x1370010] 98
[0x1370030] 7
[0x1370050] 6
[0x1370070] 5
[0x1370090] 4
[0x13700b0] 3
[0x13700d0] 2
[0x13700f0] 1
[0x1370110] 0
-> [0x1370030] 7
Loop starts at [0x1370030] 7
julien@ubuntu:~/0x13. More singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x13-more_singly_linked_lists`
- File: `103-find_loop.c`

Done! | Check your code | Get a sandbox | QA Review