0x12. C - Singly linked lists

CAlgorithmData structure

- By: Julien Barbier
- Weight: 1
- Project over - took place from Apr 21, 2022 6:00 AM to Apr 22, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 43.0/43 mandatory & 12.0/12 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Calculation: 100.0% + (100.0% * 100.0%) == **200.0%**

Concepts

For this project, we expect you to look at this concept:

- Data Structures

Resources

Read or watch:

- Linked Lists

- Google
- Youtube

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **without the help of Google**:

General

- When and why using linked lists vs arrays
- How to build and use linked lists

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 and 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`
- 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 list_s - singly linked list
 * @str: string - (malloc'ed string)
 * @len: length of the string
 * @next: points to the next node
 *
 * Description: singly linked list node structure
 */
typedef struct list_s
{
    char *str;
    unsigned int len;
    struct list_s *next;
} list_t;
```

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. Print list

mandatory

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

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

- Prototype: `size_t print_list(const list_t *h);`
- Return: the number of nodes
- Format: see example

- If `str` is `NULL`, print `[0] (nil)`
- You are allowed to use `printf`

```
julien@ubuntu:~/0x12. 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)
{
    list_t *head;
    list_t *new;
    list_t hello = {"World", 5, NULL};
    size_t n;

    head = &hello;
    new = malloc(sizeof(list_t));
    if (new == NULL)
    {
        printf("Error\n");
        return (1);
    }
    new->str = strdup("Hello");
    new->len = 5;
    new->next = head;
    head = new;
    n = print_list(head);
    printf("-> %lu elements\n", n);
```

```
    printf("\n");
    free(new->str);
    new->str = NULL;
    n = print_list(head);
    printf("-> %lu elements\n", n);

    free(new);
    return (0);
}
julien@ubuntu:~/0x12. Singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=g
nu89 0-main.c 0-print_list.c -o a
julien@ubuntu:~/0x12. Singly linked lists$ ./a
[5] Hello
[5] World
-> 2 elements

[0] (nil)
[5] World
-> 2 elements
julien@ubuntu:~/0x12. Singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `0-print_list.c`

Done! Help Check your code 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 `list_t` list.

- Prototype: `size_t list_len(const list_t *h);`

```
julien@ubuntu:~/0x12. Singly linked lists$ cat 1-main.c
```

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "lists.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    list_t *head;
    list_t *new;
    list_t hello = {"World", 5, NULL};
    size_t n;

    head = &hello;
    new = malloc(sizeof(list_t));
    if (new == NULL)
    {
        printf("Error\n");
        return (1);
    }
    new->str = strdup("Hello");
    new->len = 5;
    new->next = head;
    head = new;
    n = list_len(head);
    printf("-> %lu elements\n", n);
    free(new->str);
    free(new);
    return (0);
```

```
}
julien@ubuntu:~/0x12. Singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=g
nu89 1-main.c 1-list_len.c -o b
julien@ubuntu:~/0x12. Singly linked lists$ ./b
-> 2 elements
julien@ubuntu:~/0x12. Singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `1-list_len.c`

Done! Help Check your code 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 `list_t` list.

- Prototype: `list_t *add_node(list_t **head, const char *str);`
- Return: the address of the new element, or `NULL` if it failed
- `str` needs to be duplicated
- You are allowed to use `strdup`

```
julien@ubuntu:~/0x12. 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)
{
```

```
    list_t *head;


    head = NULL;
    add_node(&head, "Alexandro");
    add_node(&head, "Asaia");
    add_node(&head, "Augustin");
    add_node(&head, "Bennett");
    add_node(&head, "Bilal");
    add_node(&head, "Chandler");
    add_node(&head, "Damian");
    add_node(&head, "Daniel");
    add_node(&head, "Dora");
    add_node(&head, "Electra");
    add_node(&head, "Gloria");
    add_node(&head, "Joe");
    add_node(&head, "John");
    add_node(&head, "John");
    add_node(&head, "Josquin");
    add_node(&head, "Kris");
    add_node(&head, "Marine");
    add_node(&head, "Mason");
    add_node(&head, "Praylin");
    add_node(&head, "Rick");
    add_node(&head, "Rick");
    add_node(&head, "Rona");
    add_node(&head, "Siphan");
    add_node(&head, "Sravanthi");
    add_node(&head, "Steven");
    add_node(&head, "Tasneem");
    add_node(&head, "William");
    add_node(&head, "Zee");
    print_list(head);
    return (0);
```

```
}
julien@ubuntu:~/0x12. Singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=g
nu89 2-main.c 2-add_node.c 0-print_list.c -o c
julien@ubuntu:~/0x12. Singly linked lists$ ./c
[3] Zee
[7] William
[7] Tasneem
[6] Steven
[9] Sravanthi
[6] Siphan
[4] Rona
[4] Rick
[4] Rick
[7] Praylin
[5] Mason
[6] Marine
[4] Kris
[7] Josquin
[4] John
[4] John
[3] Joe
[6] Gloria
[7] Electra
[4] Dora
[6] Daniel
[6] Damian
[8] Chandler
[5] Bilal
[7] Bennett
[8] Augustin
[5] Asaia
[9] Alexandro
julien@ubuntu:~/0x12. Singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `2-add_node.c`

Done! Help Check your code 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 `list_t` list.

- Prototype: `list_t *add_node_end(list_t **head, const char *str);`
- Return: the address of the new element, or `NULL` if it failed
- `str` needs to be duplicated
- You are allowed to use `strdup`

```
julien@ubuntu:~/0x12. 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)
{
    list_t *head;

    head = NULL;
    add_node_end(&head, "Anne");
    add_node_end(&head, "Colton");
    add_node_end(&head, "Corbin");
    add_node_end(&head, "Daniel");
```

```
    add_node_end(&head, "Danton");
    add_node_end(&head, "David");
    add_node_end(&head, "Gary");
    add_node_end(&head, "Holden");
    add_node_end(&head, "Ian");
    add_node_end(&head, "Ian");
    add_node_end(&head, "Jay");
    add_node_end(&head, "Jennie");
    add_node_end(&head, "Jimmy");
    add_node_end(&head, "Justin");
    add_node_end(&head, "Kalson");
    add_node_end(&head, "Kina");
    add_node_end(&head, "Matthew");
    add_node_end(&head, "Max");
    add_node_end(&head, "Michael");
    add_node_end(&head, "Ntuj");
    add_node_end(&head, "Philip");
    add_node_end(&head, "Richard");
    add_node_end(&head, "Samantha");
    add_node_end(&head, "Stuart");
    add_node_end(&head, "Swati");
    add_node_end(&head, "Timothy");
    add_node_end(&head, "Victor");
    add_node_end(&head, "Walton");
    print_list(head);
    return (0);
}
julien@ubuntu:~/0x12. Singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=g
nu89 3-main.c 3-add_node_end.c 0-print_list.c -o d
julien@ubuntu:~/0x12. Singly linked lists$ ./d
[4] Anne
[6] Colton
[6] Corbin
[6] Daniel
```

```
[6] Danton
[5] David
[4] Gary
[6] Holden
[3] Ian
[3] Ian
[3] Jay
[6] Jennie
[5] Jimmy
[6] Justin
[6] Kalson
[4] Kina
[7] Matthew
[3] Max
[7] Michael
[4] Ntuj
[6] Philip
[7] Richard
[8] Samantha
[6] Stuart
[5] Swati
[7] Timothy
[6] Victor
[6] Walton
julien@ubuntu:~/0x12. Singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `3-add_node_end.c`

Done! Help Check your code QA Review

4. Free list

mandatory

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

Write a function that frees a `list_t` list.

- Prototype: `void free_list(list_t *head);`

```
julien@ubuntu:~/0x12. 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)
{
    list_t *head;

    head = NULL;
    add_node_end(&head, "Bob");
    add_node_end(&head, "&");
    add_node_end(&head, "Kris");
    add_node_end(&head, "love");
    add_node_end(&head, "asm");
    print_list(head);
    free_list(head);
    head = NULL;
    return (0);
}
julien@ubuntu:~/0x12. Singly linked lists$ gcc -Wall -pedantic -Werror -Wextra -std=g
nu89 4-main.c 4-free_list.c 3-add_node_end.c 0-print_list.c -o e
julien@ubuntu:~/0x12. Singly linked lists$ valgrind ./e
```

```
==3598== Memcheck, a memory error detector
==3598== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3598== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3598== Command: ./e
==3598==
[6] Bob
[1] &
[3] Kris
[4] love
[3] asm
==3598==
==3598== HEAP SUMMARY:
==3598==     in use at exit: 0 bytes in 0 blocks
==3598==   total heap usage: 11 allocs, 11 frees, 1,166 bytes allocated
==3598==
==3598== All heap blocks were freed -- no leaks are possible
==3598==
==3598== For counts of detected and suppressed errors, rerun with: -v
==3598== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
julien@ubuntu:~/0x12. Singly linked lists$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `4-free_list.c`

Done! Help Check your code QA Review

5. The Hare and the Tortoise

#advanced

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

Write a function that prints `You're beat! and yet, you must allow,\nI bore my house upon my back!\n` before the `main` function is executed.

- You are allowed to use the `printf` function

```
julien@ubuntu:~/0x12. Singly linked lists$ cat 100-main.c
#include <stdio.h>


/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    printf("(A tortoise, having pretty good sense of a hare's nature, challenges one
to a race.)\n");
    return (0);
}
julien@ubuntu:~/$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 100-main.c 100-first
.c -o first
julien@ubuntu:~/$ ./first
You're beat! and yet, you must allow,
I bore my house upon my back!
(A tortoise, having pretty good sense of a hare's nature, challenges one to a race.)
julien@ubuntu:~/$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `100-first.c`

Done! Help Check your code QA Review

6. Real programmers can write assembly code in any language

#advanced

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

Write a 64-bit program in assembly that prints `Hello, Holberton`, followed by a new line.

- You are only allowed to use the `printf` function
- You are not allowed to use interrupts
- Your program will be compiled using `nasm` and `gcc`:

```
julien@ubuntu:~/$ nasm -f elf64 101-hello_holberton.asm && gcc -no-pie -std=gnu89 101
-hello_holberton.o -o hello

julien@ubuntu:~/$ ./hello

Hello, Holberton

julien@ubuntu:~/$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x12-singly_linked_lists`
- File: `101-hello_holberton.asm`

Done! Help Check your code Get a sandbox QA Review