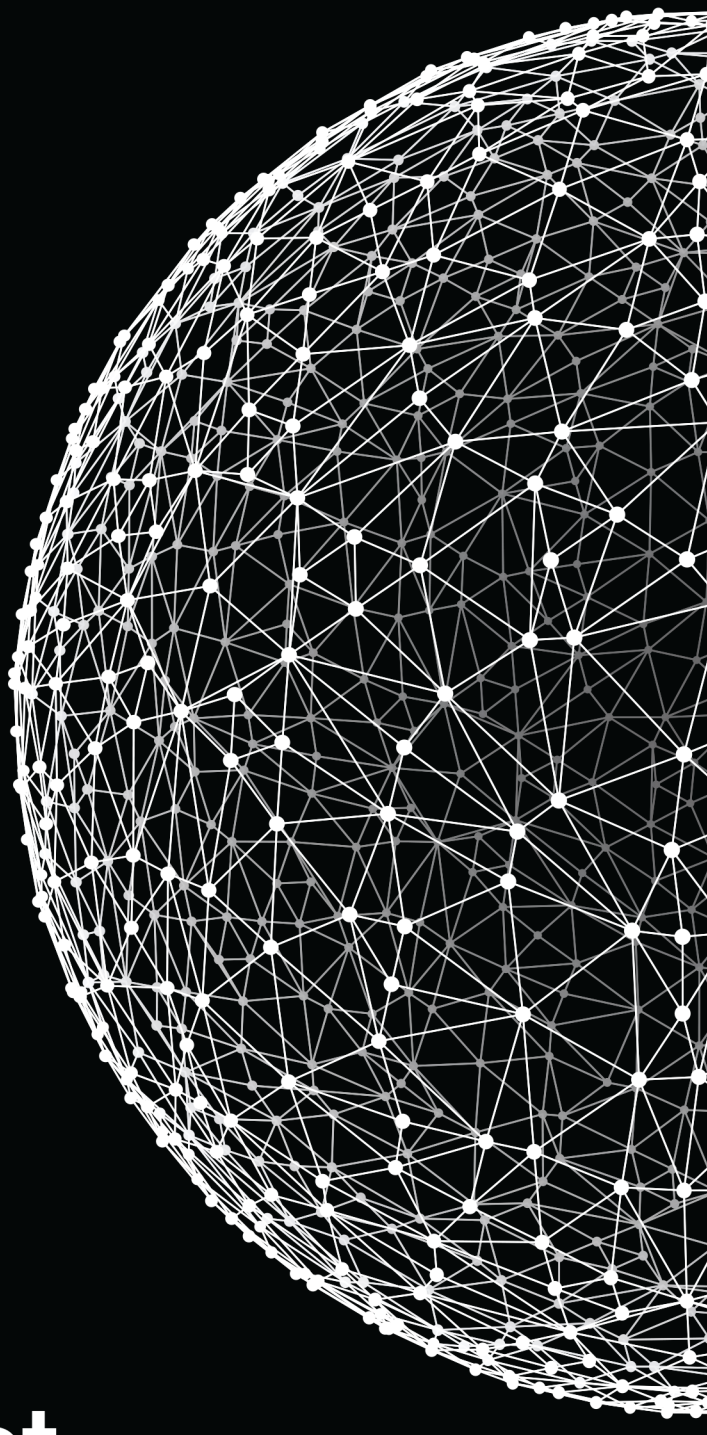
Sprint 11

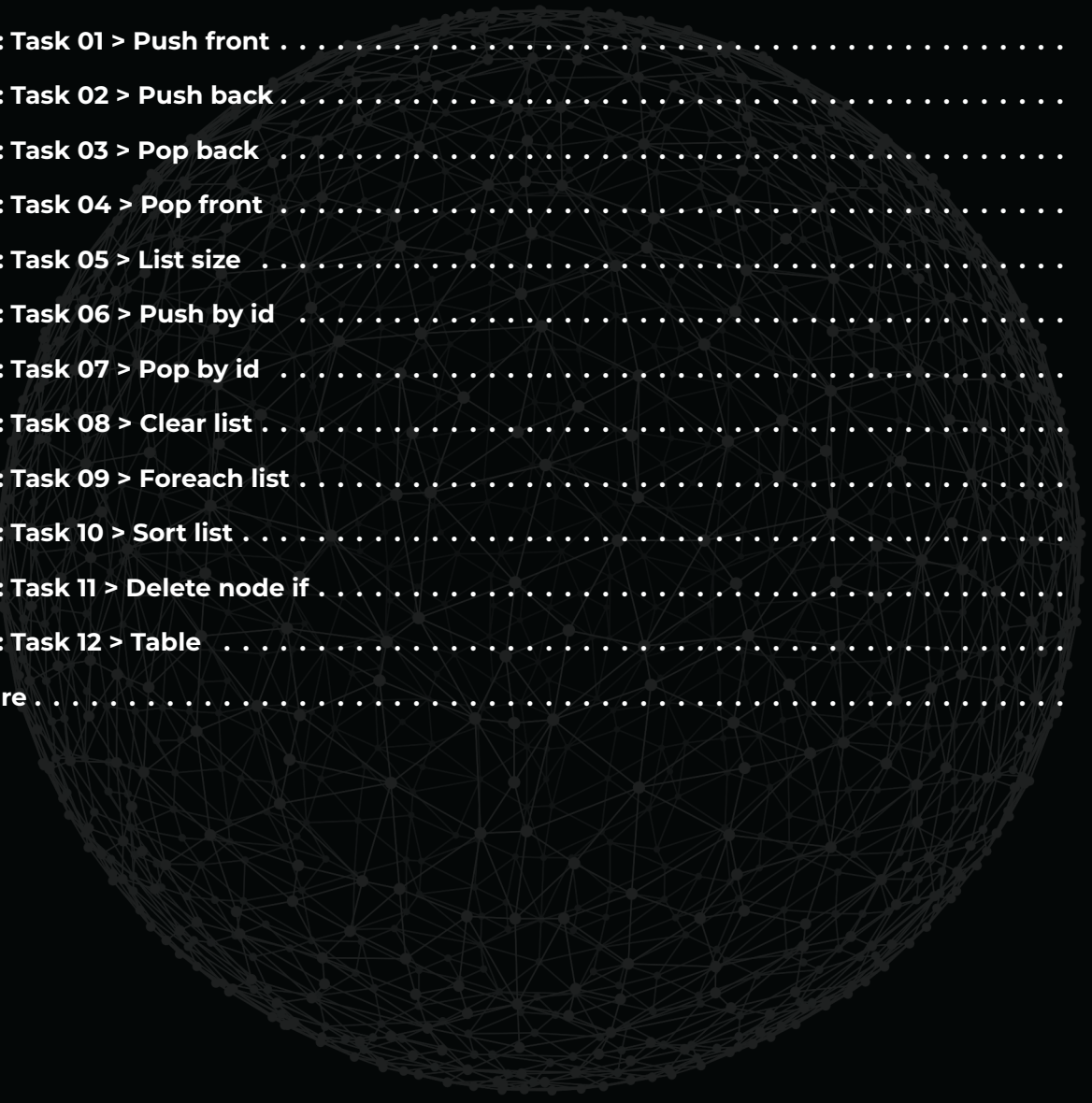
Marathon C

September 1, 2020



ucode connect

Contents



ucode connect

Engage

DESCRIPTION

Howdy!

You got some basic skills during the initial stage of C programming. We hope you have studied hard and understood the concepts and principles of programming so far. This is the last Sprint of the Marathon but not the last challenge.

Data structures provide the means to manage large amounts of data productively. Efficient data structures are the key to designing powerful algorithms. There are plenty of different data structures for various apps, e.g. array, linked list, stack, queue, binary tree, hash table, etc.

You will need to develop a set of functions to manage singly linked lists, which are one of the simplest and most common data structures.

BIG IDEA

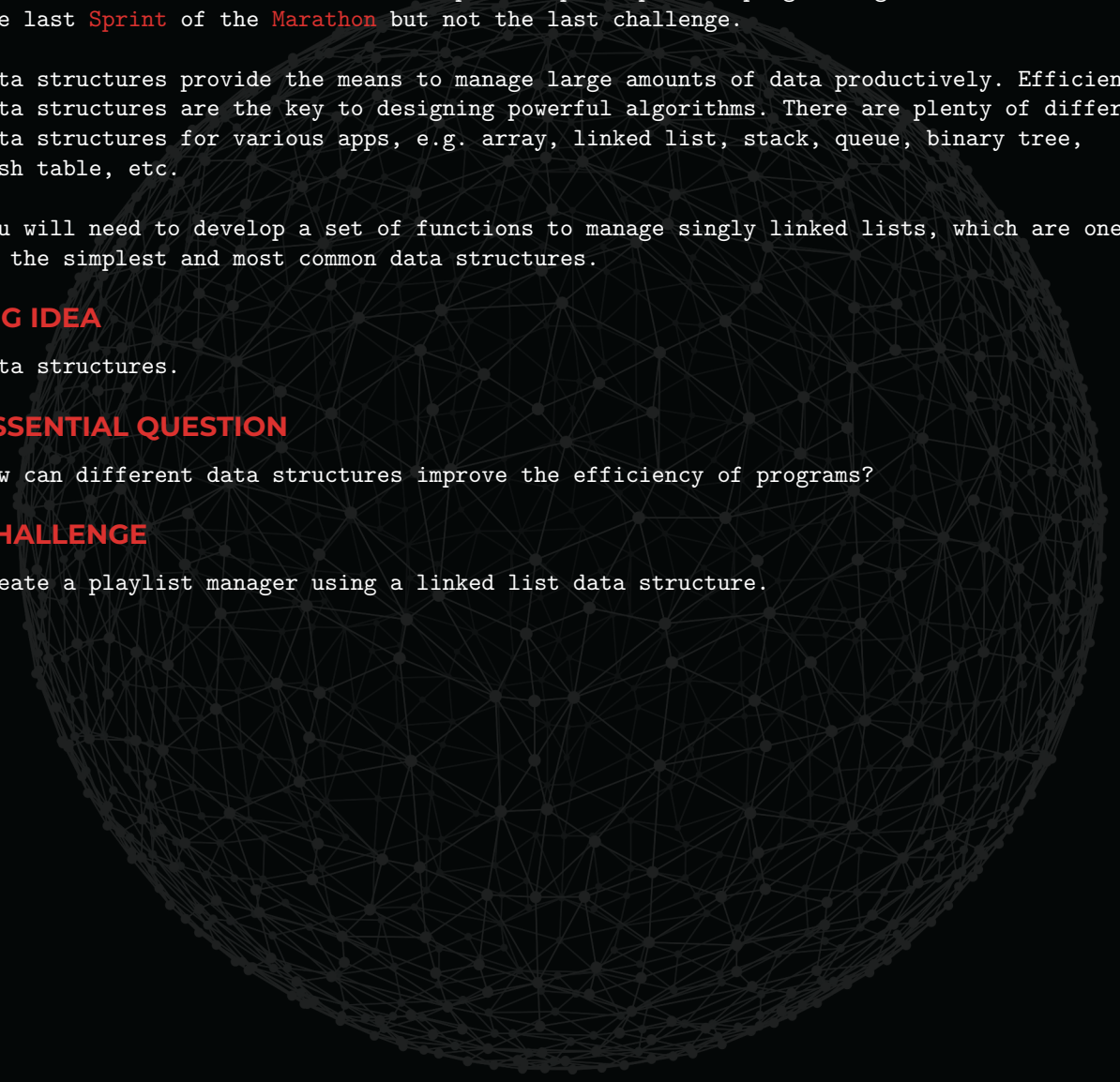
Data structures.

ESSENTIAL QUESTION

How can different data structures improve the efficiency of programs?

CHALLENGE

Create a playlist manager using a linked list data structure.



Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is concentration in programming and why is it so important during coding?
- What kind of music is best for your concentration?
- What kind of music players do you know?
- What is the principal benefit of a linked list over a conventional array?
- What is the principal benefit of an array over a linked list?
- What elements does a node consist of?
- What is the difference between singly linked list, doubly linked list, multiply linked list and circular linked list?
- Under what conditions is a list considered as empty?
- Is it possible to build other data structures (e.g. stack, queue, binary tree, etc.) with linked list nodes?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Plug in your headphones.
- Open a player and find the tracks listed below:
  - "Rock Is Dead" by Marilyn Manson
  - "Spybreak! (Short One)" by Propellerheads
  - "Bad Blood" by Ministry
  - "Clubbed to Death (Kurayamino Mix)" by Rob D
  - "Prime Audio Soup" by Meat Beat Manifesto
  - "Leave You Far Behind" by Lunatic Calm
  - "Mindfields" by The Prodigy
  - "Dragula (Hot Rod Herman Remix)" by Rob Zombie
  - "My Own Summer (Shove It)" by Deftones
  - "Ultrasonic Sound" by Hive
  - "Look to Your Orb for the Warning (Radio Edit)" by Monster Magnet
  - "Du hast" by Rammstein
  - "Wake Up" by Rage Against the Machine
- Programmers often listen to music during coding. Find your zen.
- Try to realize the whole programmer's way that you have done till now, step by step.

- Explore linked lists.
- Clone your git repository that is issued on the challenge page in the LMS.
- Arrange to brainstorm tasks with other students.
- Try to implement your thoughts in code.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story. Examine the given examples carefully. They may contain details that are not mentioned in the task.
- Analyze all information you have collected during the preparation stages.
- Perform only those tasks that are given in this document.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Your program must manage memory allocations correctly. A memory that is no longer needed must be freed, otherwise, the task is considered incomplete.
- Pay attention to what is allowed in a certain task. Use of forbidden stuff is considered a cheat and your tasks will be failed.
- Complete tasks according to the rules specified in the `Auditor`.
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.
- Use your brain and follow the white rabbit to prove that you are the Chosen one!

Act: Task 00

NAME

Create node

DIRECTORY

`t00/`

SUBMIT

`list.h, mx_create_node.c`

ALLOWED FUNCTIONS

`malloc`

LEGEND

People in Nebuchadnezzar want to relax and listen to music after the constant battles in Matrix. It is a pity, but they do not have a player for this. So you can help them by writing some useful functions. This function will help to `create a playlist` with one song in it.

DESCRIPTION

Create a function that creates a new node of a linked list `t_list`. The function assigns a parameter `data` to the list variable `data` and assigns `next` to `NULL`.

`list.h` contains the structure `s_list` and required includes and prototypes to compile your function successfully. See in the SYNOPSIS.

NOTE: you must submit such `list.h` in every task of Sprint 11.
NOTE 2: all functions that create nodes or delete them must not allocate or free memory for list variable `data`.

SYNOPSIS

```
typedef struct s_list {
    void *data;
    struct s_list *next;
}              t_list;
```

```
t_list *mx_create_node(void *data);
```

SEE ALSO

Linked list

Act: Task 01

NAME

Push front

DIRECTORY

`t01/`

SUBMIT

`list.h, mx_push_front.c, mx_create_node.c`

ALLOWED FUNCTIONS

`malloc`

LEGEND

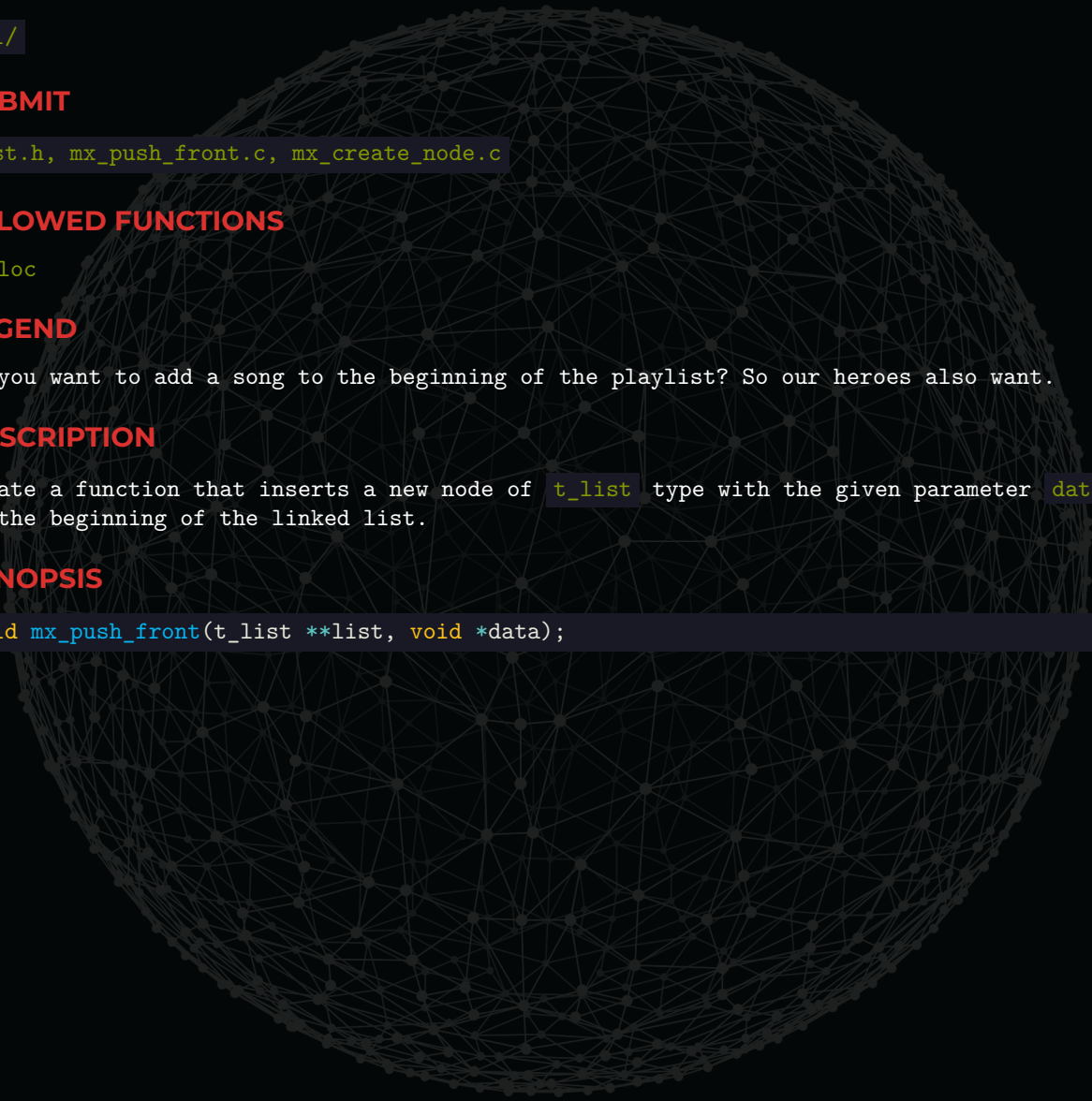
Do you want to add a song to the beginning of the playlist? So our heroes also want.

DESCRIPTION

Create a function that inserts a new node of `t_list` type with the given parameter `data` at the beginning of the linked list.

SYNOPSIS

```
void mx_push_front(t_list **list, void *data);
```



Act: Task 02

NAME

Push back

DIRECTORY

`t02/`

SUBMIT

`list.h, mx_push_back.c, mx_create_node.c`

ALLOWED FUNCTIONS

`malloc`

LEGEND

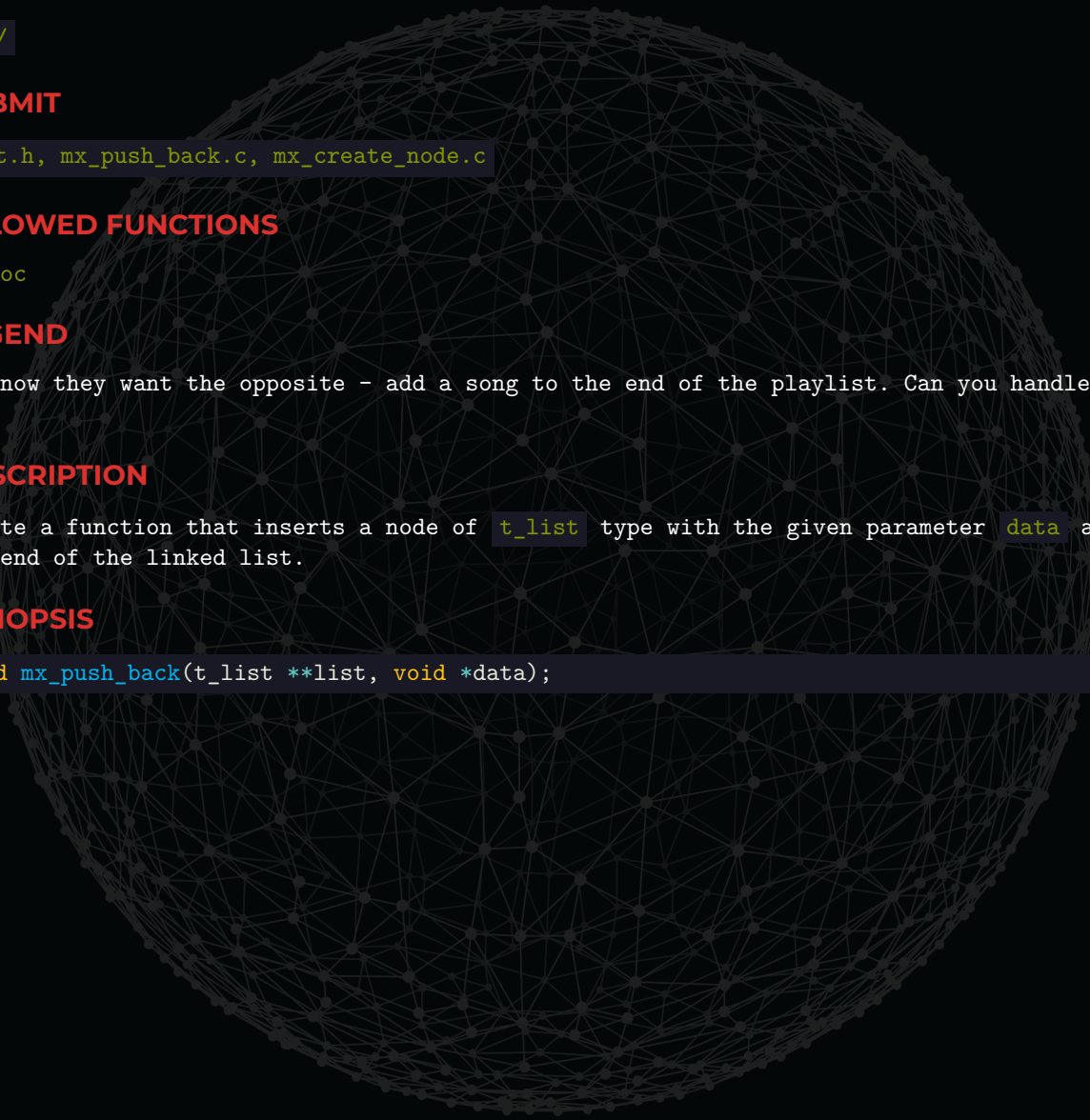
Oh, now they want the opposite - add a song to the end of the playlist. Can you handle it?

DESCRIPTION

Create a function that inserts a node of `t_list` type with the given parameter `data` at the end of the linked list.

SYNOPSIS

```
void mx_push_back(t_list **list, void *data);
```



Act: Task 03

NAME

Pop back

DIRECTORY

`t03/`

SUBMIT

`list.h, mx_pop_back.c`

ALLOWED FUNCTIONS

`free`

LEGEND

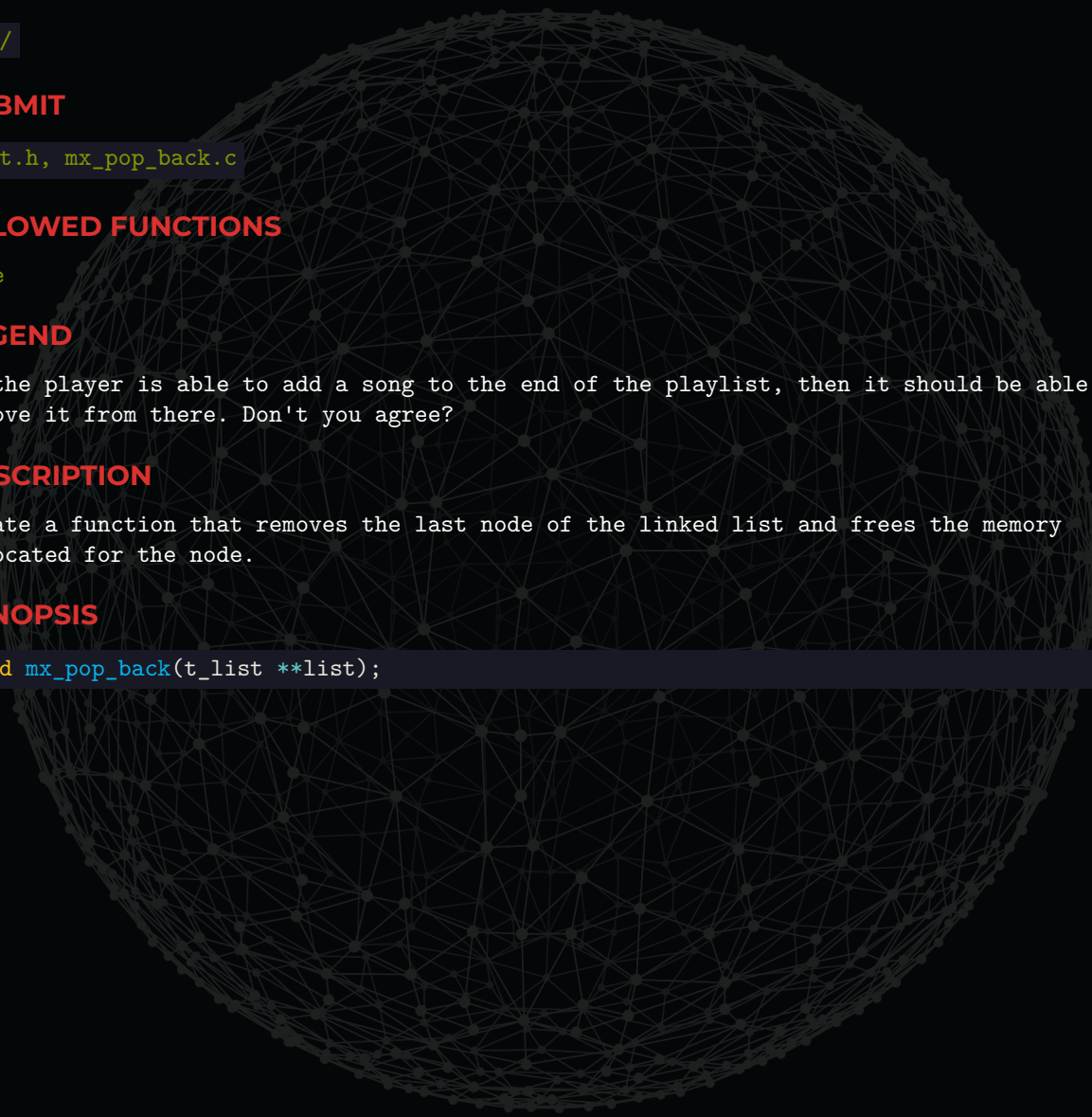
If the player is able to add a song to the end of the playlist, then it should be able to remove it from there. Don't you agree?

DESCRIPTION

Create a function that removes the last node of the linked list and frees the memory allocated for the node.

SYNOPSIS

```
void mx_pop_back(t_list **list);
```



Act: Task 04

NAME

Pop front

DIRECTORY

`t04/`

SUBMIT

`list.h, mx_pop_front.c`

ALLOWED FUNCTIONS

`free`

LEGEND

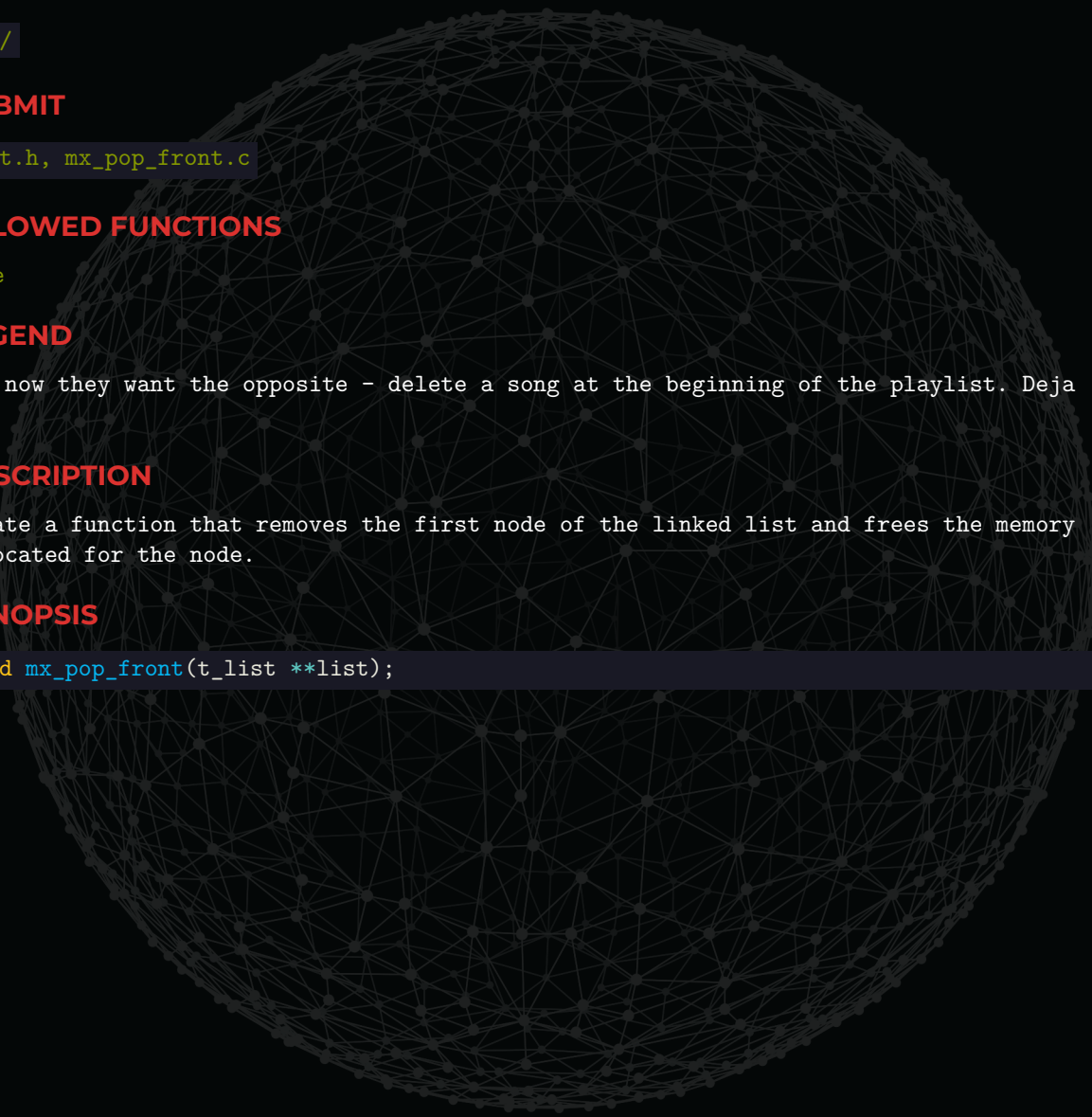
Oh, now they want the opposite - delete a song at the beginning of the playlist. Deja vu?

DESCRIPTION

Create a function that removes the first node of the linked list and frees the memory allocated for the node.

SYNOPSIS

```
void mx_pop_front(t_list **list);
```



Act: Task 05

NAME

List size

DIRECTORY

`t05/`

SUBMIT

`list.h, mx_list_size.c`

ALLOWED FUNCTIONS

None

LEGEND

"1, 2, 3,...,31,... Oh, I lost count of songs in the playlist." - Apoc says.
It would be nice to have a player which counts the number of songs in a playlist.

DESCRIPTION

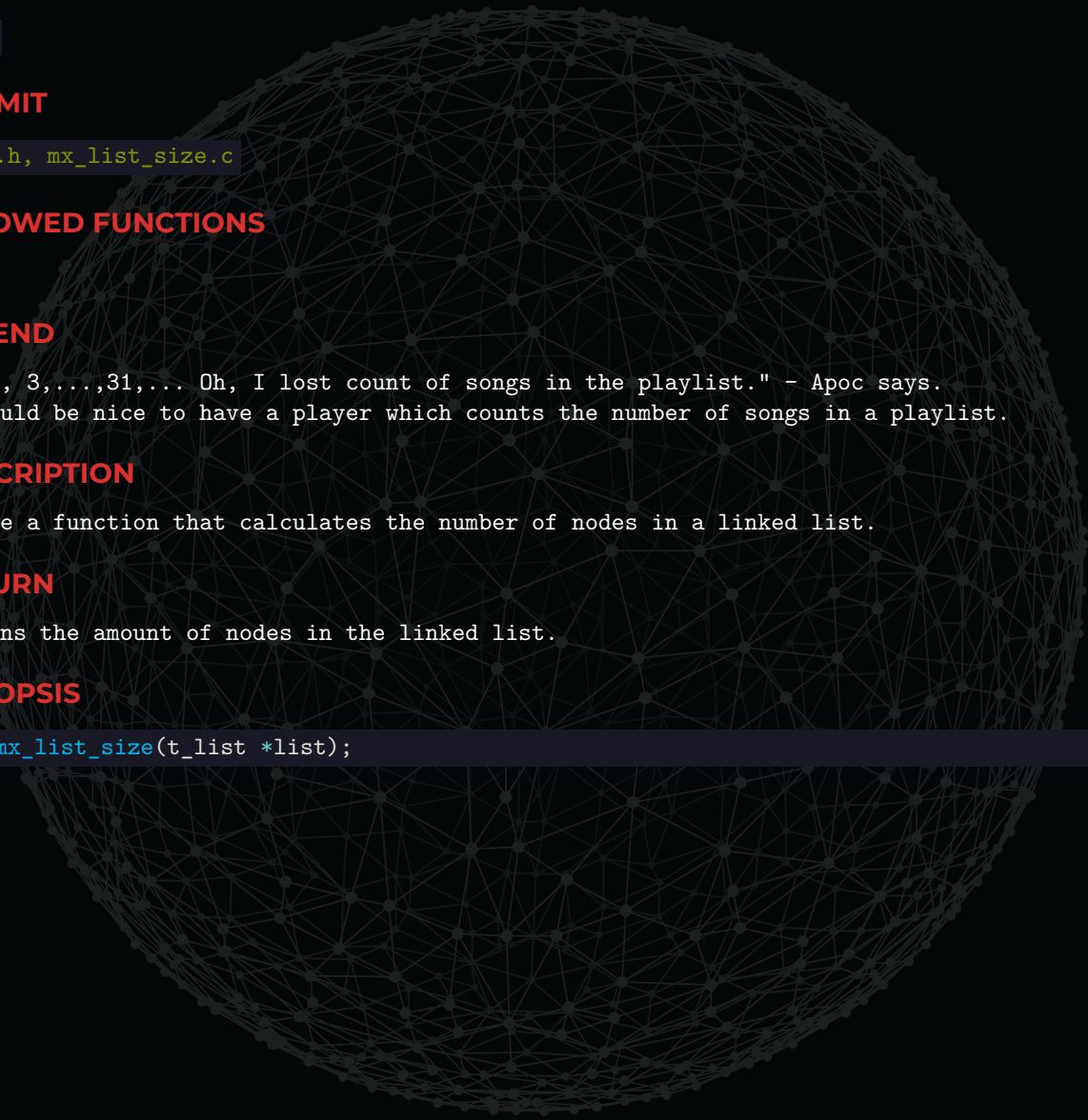
Create a function that calculates the number of nodes in a linked list.

RETURN

Returns the amount of nodes in the linked list.

SYNOPSIS

```c
int mx_list_size(t_list *list);
```



Act: Task 06

NAME

Push by id

DIRECTORY

`t06/`

SUBMIT

`list.h, mx_push_index.c, mx_create_node.c, mx_push_front.c, mx_push_back.c`

ALLOWED FUNCTIONS

`malloc`

LEGEND

It's time for the ultimate insertion feature! With it, the player can insert a song at any position in the playlist.

DESCRIPTION

Create a function that inserts a new node of `t_list` type with the given parameter `data` at the the position of the linked list indicated by the index.

1. The list numeration starts from `0`.
2. If `index` is below zero, the node becomes the first in the list.
3. If `index` is greater than the list size, the node becomes the last in the list.

SYNOPSIS

```c
void mx_push_index(t_list **list, void *data, int index);
```

Act: Task 07

NAME

Pop by id

DIRECTORY

`t07/`

SUBMIT

`list.h, mx_pop_index.c, mx_pop_front.c, mx_pop_back.c`

ALLOWED FUNCTIONS

`free`

LEGEND

It's time for the ultimate deletion feature! With it, the player can delete a song from any position in the playlist.

DESCRIPTION

Create a function that removes the node located at the given position `index` of the linked list and frees the memory allocated for the node.

1. The list numeration starts from `0`.
2. If `index` is below zero, delete the first node.
3. If `index` is larger than the list size, delete the last node.

SYNOPSIS

```c
void mx_pop_index(t_list **list, int index);
```

Act: Task 08

NAME

Clear list

DIRECTORY

`t08/`

SUBMIT

`list.h, mx_clear_list.c`

ALLOWED FUNCTIONS

`free`

LEGEND

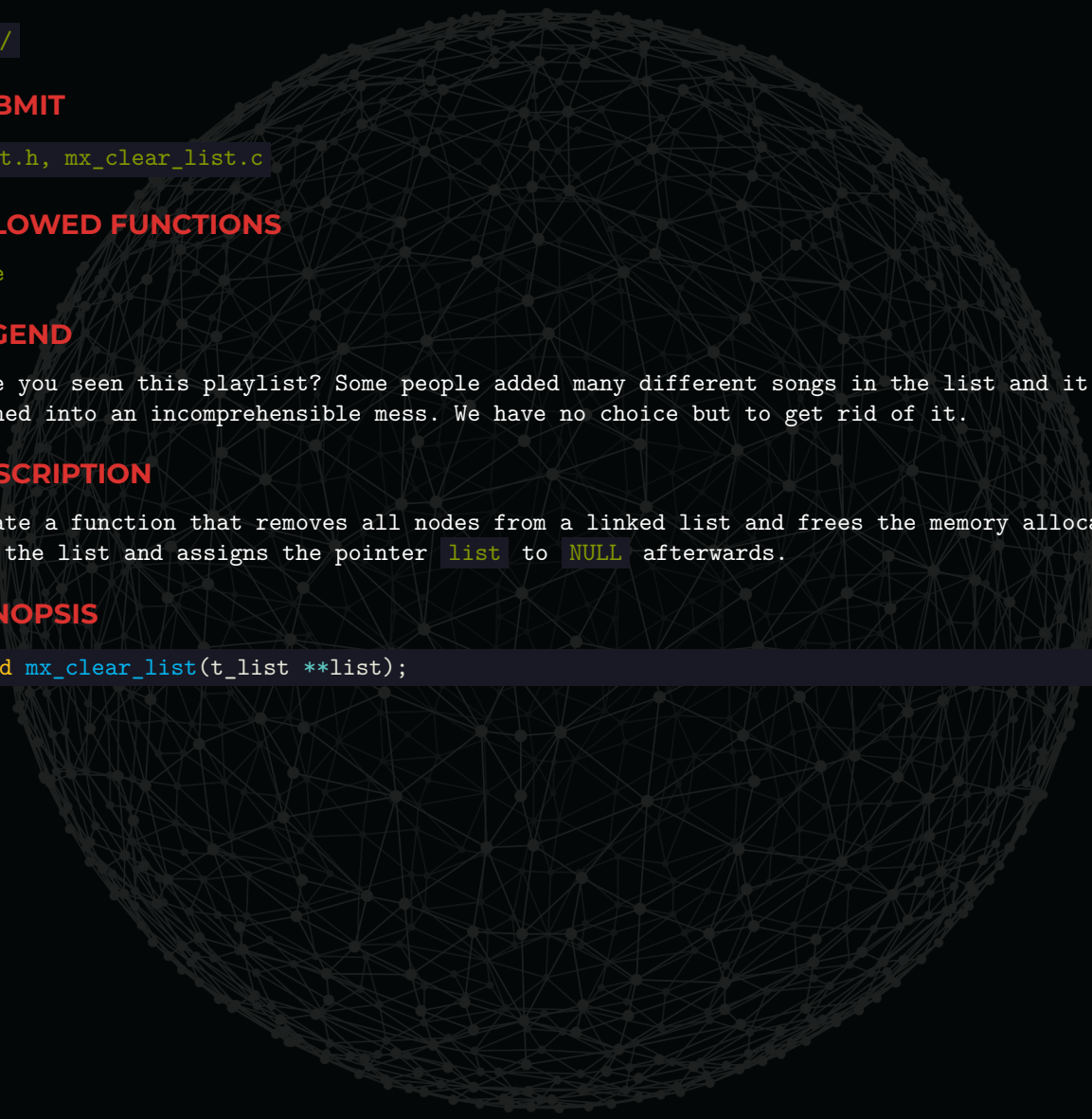
Have you seen this playlist? Some people added many different songs in the list and it turned into an incomprehensible mess. We have no choice but to get rid of it.

DESCRIPTION

Create a function that removes all nodes from a linked list and frees the memory allocated for the list and assigns the pointer `list` to `NULL` afterwards.

SYNOPSIS

```
void mx_clear_list(t_list **list);
```



Act: Task 09

NAME

Foreach list

DIRECTORY

`t09/`

SUBMIT

`list.h, mx_foreach_list.c`

ALLOWED FUNCTIONS

None

LEGEND

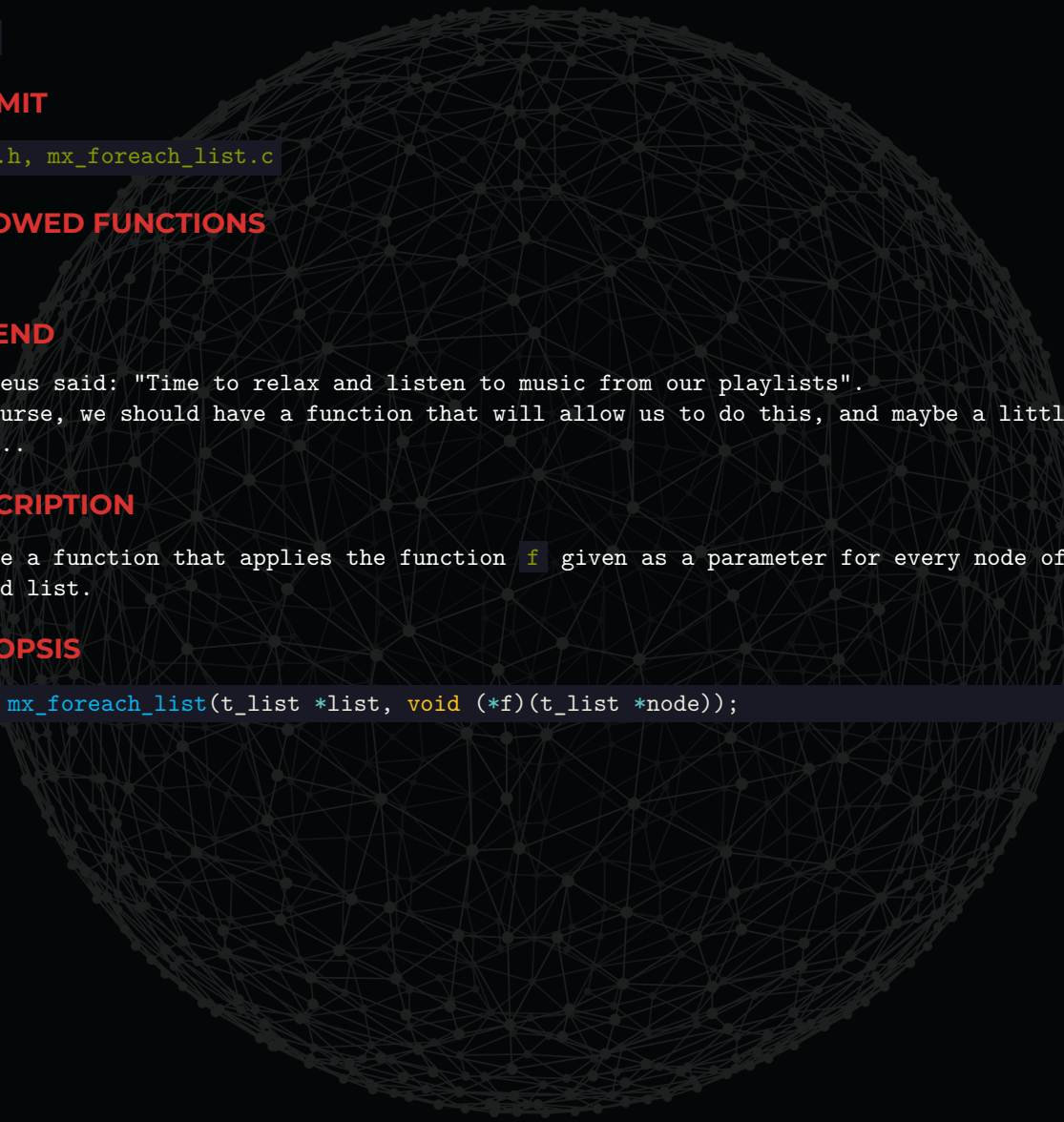
Morpheus said: "Time to relax and listen to music from our playlists".
Of course, we should have a function that will allow us to do this, and maybe a little more...

DESCRIPTION

Create a function that applies the function `f` given as a parameter for every node of the linked list.

SYNOPSIS

```
void mx_foreach_list(t_list *list, void (*f)(t_list *node));
```



Act: Task 10

NAME

Sort list

DIRECTORY

`t10/`

SUBMIT

`list.h, mx_sort_list.c`

ALLOWED FUNCTIONS

None

LEGEND

Oh, the songs in our playlists are not in order. Can we write a feature that allows us to do this?

DESCRIPTION

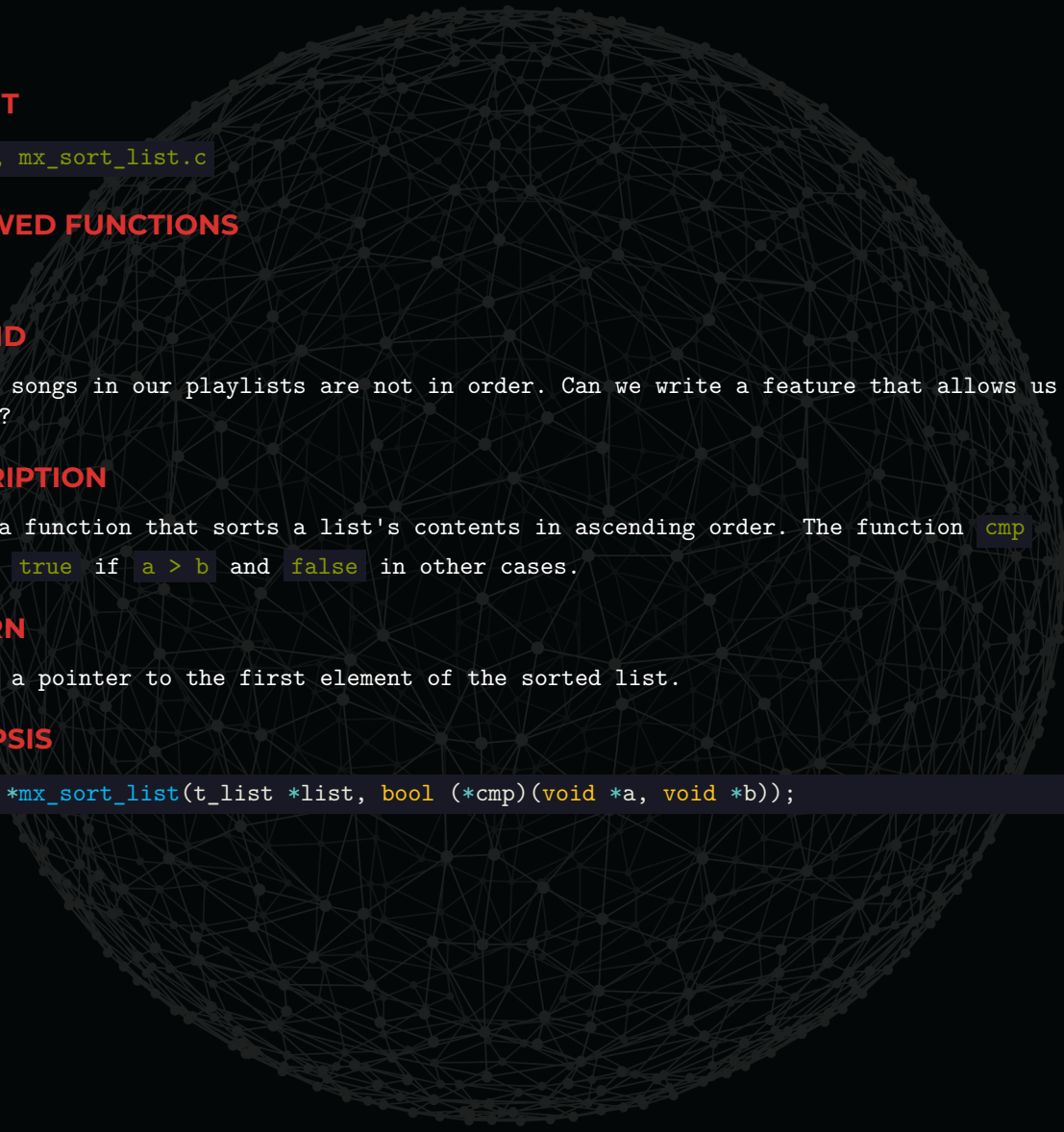
Create a function that sorts a list's contents in ascending order. The function `cmp` returns `true` if `a > b` and `false` in other cases.

RETURN

Returns a pointer to the first element of the sorted list.

SYNOPSIS

```
t_list *mx_sort_list(t_list *list, bool (*cmp)(void *a, void *b));
```



Act: Task 11

NAME

Delete node if

DIRECTORY

`t11/`

SUBMIT

`list.h, mx_del_node_if.c`

ALLOWED FUNCTIONS

`free`

LEGEND

Some songs in our playlists may be outdated or no longer meet any criteria.
Guess what you need to do.

DESCRIPTION

Create a function that removes a list's nodes, in which the node's `data` is equal to `del_data`. The function `cmp` returns `true` if `a == b` and `false` in other cases.

SYNOPSIS

```
void
mx_del_node_if(t_list **list, void *del_data, bool (*cmp)(void *a, void *b));
```

Act: Task 12

NAME

Table

DIRECTORY

`t12/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

`open, close, read, write, malloc, free, exit`

BINARY

`playlist`

LEGEND

Having now gone all this way - collecting knowledge and materials, you can now create a music player just like our ancestors created this ship.

DESCRIPTION

Create a program that manages playlists. The playlist data is stored in a file in CSV-like format, see in the CONSOLE OUTPUT for more references.
Every song entity starts from a new line.
Every song entity is described by `artist` and `name` which are specified in a playlist file in a respective order. The program supports 4 options: `add`, `remove`, `sort`, `print`. These options will be passed as command-line arguments as stated below:

- `./playlist [file] add [artist] [name]` - to add the song in the playlist file. Create a new playlist file if the file with the given name does not exist
- `./playlist [file] remove [index]` - to remove a song from the playlist file by index
- `./playlist [file] sort [artist | name]` - to sort songs in the playlist file in alphabetical order by artist or song
- `./playlist [file] print` - to print all songs from the playlist prefixed by their `index` to the standard output

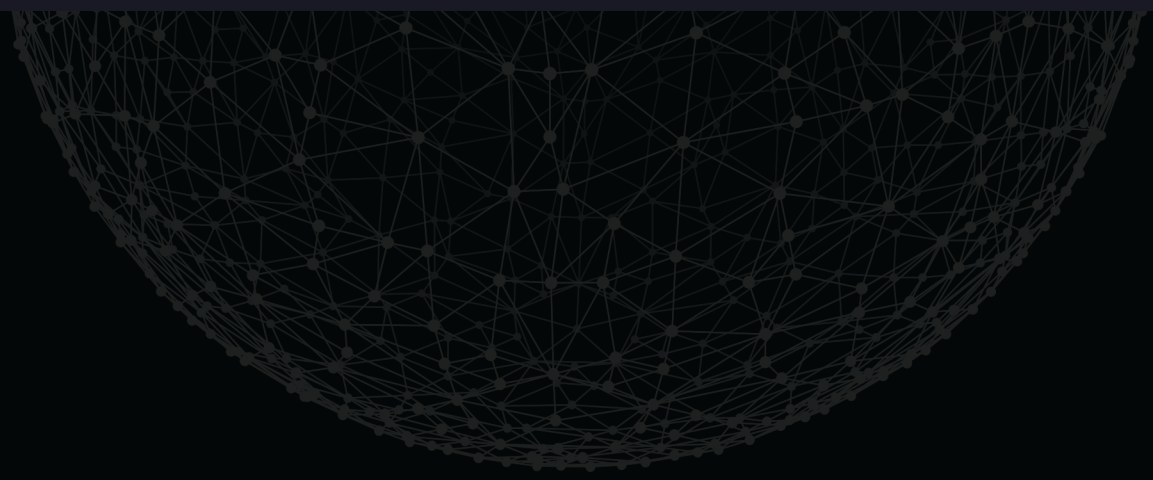
Error handling. The program prints to the standard error stream `stderr` the following:

- in case of invalid command-line arguments - `ERROR`
- in case of usage - `usage: ./playlist [file] [command] [args]`

You must use lists for this task, otherwise, the task will not be graded.

CONSOLE OUTPUT

```
>./playlist | cat -e
usage: ./playlist [file] [command] [args]
>cat -e plist
Marilyn Manson,rock is dead$
Red Hot Chili Peppers,Californication$
iron Maiden,Seventh Son of a Seventh Son$
linkin park,numb$
>./playlist plist sort artist
>./playlist plist print | cat -e
0. iron Maiden - Seventh Son of a Seventh Son$
1. linkin park - numb$
2. Marilyn Manson - rock is dead$
3. Red Hot Chili Peppers - Californication$
>./playlist plist add "The Prodigy" "Mindfields"
>./playlist plist sort name
>./playlist plist remove 0
>./playlist plist print | cat -e
0. The Prodigy - Mindfields$
1. linkin park - numb$
2. Marilyn Manson - rock is dead$
3. iron Maiden - Seventh Son of a Seventh Son$
>./playlist plist remove a | cat -e
ERROR
>./playlist plist print asdf | cat -e
ERROR
>./playlist plist wow | cat -e
ERROR
>
```



Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.