To-do List

Generated by Doxygen 1.8.11

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 NOD Struct Reference

Linked list used to store the priority, category and task.

`#include <list.h>`

Data Fields

- int priority

  Stores the priority. Is a number from 0 to 10.
- char * category

  Contains the category. Stores up to 1000 characters, can be made out of multiple words separated by a blank.
- char * task

  Contains the task. Stores up to 1000 characters, can be made out of multiple words separated by a blank.
- struct NOD * next

  Pointer to the next element in the list.

3.1.1 Detailed Description

Linked list used to store the priority, category and task.

The documentation for this struct was generated from the following file:

- list.h

Chapter 4

File Documentation

4.1 display.c File Reference

Source code for the display header file.

```
#include "display.h"
#include <stdio.h>
```

Functions

- void menu ()

  Prints menu with choices for the user.

4.1.1 Detailed Description

Source code for the display header file.

4.1.2 Function Documentation

4.1.2.1 void menu ( )

Prints menu with choices for the user.

Returns

Doesn't return anything.

4.2 display.h File Reference

Header file for display functions.

Functions

- void menu ()

  Prints menu with choices for the user.

4.2.1 Detailed Description

Header file for display functions.

4.2.2 Function Documentation

4.2.2.1 void menu ( )

Prints menu with choices for the user.

Returns

Doesn't return anything.

4.3 list.c File Reference

Source code for the list header file.

```
#include "list.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Functions

- void add_tasks (NODE *head)

  Adds tasks from input.txt file to the linked list.
- void print_all (NODE *head)

  Prints all the nodes of the list.
- void print_by_categ (NODE *head)

  Prints every element that has the same category as the one entered by the user.
- void merge_sort (NODE **headRef)

  Merge sort function.
- NODE * sorted_merge (NODE *a, NODE *b)

  Function that merges two sublists together.
- void front_back_split (NODE *source, NODE **frontRef, NODE **backRef)

  Function that splits the list in two sublists. Splits the nodes into front and back halves and returns the two lists using pointers. If the length is odd, the extra node goes in the front list.
- void delete_task (NODE *head)

  Function that deletes a node, based on input provided by user.
- void modify_categ (NODE *head)

  Function that modifies the category based on user input.
- void delete_categ (NODE *head)

  Function that deletes a category. It replaces the category with a blank to symbolise that it's empty.

4.3.1 Detailed Description

Source code for the list header file.

4.3.2 Function Documentation

4.3.2.1 void add_tasks (NODE * *head*)

Adds tasks from input.txt file to the linked list.

Parameters

head	Takes as parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

File is opened.

Prints message in case it can't be opened.

Allocates memory for new node.

Saves the priority (with 5 maximum number of digits).

Allocates memory for the category and reads it.

Deletes newline character left by fgets.

Allocates memory for task and reads it.

If task is empty, it frees the memory allocated for the node and quits.

Deletes newline character left by fgets.

New node points to the first non-empty element.

New node becomes the first non-empty element.

Closes file.

4.3.2.2 void delete_categ (NODE * *head*)

Function that deletes a category. It replaces the category with a blank to symbolise that it's empty.

Parameters

head	Takes as a parameter a pointer to the first element in the list.

Returns

Doesn't return anything.

Asks user for the category to be deleted.

Goes through each node and if its category matches the one provided by the user, it is deleted (replaced by a blank space).

4.3.2.3 void delete_task (NODE * *head*)

Function that deletes a node, based on input provided by user.

Parameters

head	Takes as a parameter a pointer to the first element in the list.

Returns

Doesn't return anything.

Asks user to enter a task.

Prints message in case list is empty.

Goes through every element and checks if its category matches the provided one. If it does, it is deleted.

4.3.2.4 void front_back_split (NODE * *source,* NODE ** *frontRef,* NODE ** *backRef*)

Function that splits the list in two sublists. Splits the nodes into front and back halves and returns the two lists using pointers. If the length is odd, the extra node goes in the front list.

Parameters

source	Pointer to the first element of the list that has to be split.
frontRef	Pointer to a pointer of the first sublist.
backRef	Pointer to a pointer of the second sublist.

Returns

Doesn't return anything.

If the list is empty or has just one element, it doesn't split it.

'fast' advances two nodes and 'slow' one node.

'slow' is before the midpoint in the list, so it gets split in two.

4.3.2.5 void merge_sort (NODE ** *headRef*)

Merge sort function.

Parameters

headRef	Takes as a parameter a pointer to the pointer of the first element.

Returns

Doesn't return anything.

If list is empty or has 1 element, it quits.

It else splits it into 2 sublists.

Recursively sorts the sublists.

Merges them together.

4.3.2.6 void modify_categ (NODE * *head*)

Function that modifies the category based on user input.

Parameters

head	Takes as a parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

Asks user to enter the category to be modified.

Asks user to enter new name for the category.

It goes through every element and checks if its category matches the one provided. If it does, it changes it to the new one.

4.3.2.7 void print_all (NODE * *head*)

Prints all the nodes of the list.

Parameters

head	Takes a pointer to the first element of the list as a parameter.

Returns

Doesn't return anything.

Opens file.

Prints message in case it can't be opened and quits.

Prints message in case the list is empty.

Prints everything stored in each node.

Closes file.

4.3.2.8 void print_by_categ (NODE * *head*)

Prints every element that has the same category as the one entered by the user.

Parameters

head	Takes as a parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

Opens (or creates) output file (output.txt). If one already exists and it has data in it, the results will be printed after.

Asks user to enter a category.

Prints a message in case it can't be opened and quits.

Prints message in case list is empty.

Goes through every element and checks if its category matches the one provided by the user. If it does, it gets printed.

File is closed.

4.3.2.9 NODE* sorted_merge (NODE * *a*, NODE * *b*)

Function that merges two sublists together.

Parameters

a	Pointer to the first sublist.
b	Pointer to the second sublist.

Returns

Returns a pointer to the now merged list.

If one of them is empty, the other is returned.

Based on priority, it picks a or b and recurs.

4.4 list.h File Reference

Header file for functions that use the linked list.

Data Structures

- struct NOD

  Linked list used to store the priority, category and task.

Macros

- #define MAX_CHAR 1000

  The maximum number of characters of the variables in which the tasks and categories are stored.

Typedefs

- typedef struct NOD NODE

  Used for declarations.

Functions

- void add_tasks (NODE *head)

  Adds tasks from input.txt file to the linked list.
- void print_all (NODE *head)

  Prints all the nodes of the list.
- void print_by_categ (NODE *head)

  Prints every element that has the same category as the one entered by the user.
- NODE * sorted_merge (NODE *a, NODE *b)

  Function that merges two sublists together.
- void front_back_split (NODE *source, NODE **frontRef, NODE **backRef)

  Function that splits the list in two sublists. Splits the nodes into front and back halves and returns the two lists using pointers. If the length is odd, the extra node goes in the front list.
- void merge_sort (NODE **headRef)

  Merge sort function.
- void delete_task (NODE *head)

  Function that deletes a node, based on input provided by user.
- void modify_categ (NODE *head)

  Function that modifies the category based on user input.
- void delete_categ (NODE *head)

  Function that deletes a category. It replaces the category with a blank to symbolise that it's empty.

4.4.1 Detailed Description

Header file for functions that use the linked list.

4.4.2 Function Documentation

4.4.2.1 void add_tasks (NODE * *head*)

Adds tasks from input.txt file to the linked list.

Parameters

head	Takes as parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

File is opened.

Prints message in case it can't be opened.

Allocates memory for new node.

Saves the priority (with 5 maximum number of digits).

Allocates memory for the category and reads it.

Deletes newline character left by fgets.

Allocates memory for task and reads it.

If task is empty, it frees the memory allocated for the node and quits.

Deletes newline character left by fgets.

New node points to the first non-empty element.

New node becomes the first non-empty element.

Closes file.

4.4.2.2 void delete_categ (NODE * *head*)

Function that deletes a category. It replaces the category with a blank to symbolise that it's empty.

Parameters

head	Takes as a parameter a pointer to the first element in the list.

Returns

Doesn't return anything.

Asks user for the category to be deleted.

Goes through each node and if its category matches the one provided by the user, it is deleted (replaced by a blank space).

4.4.2.3 void delete_task (NODE * *head*)

Function that deletes a node, based on input provided by user.

Parameters

head	Takes as a parameter a pointer to the first element in the list.

Returns

Doesn't return anything.

Asks user to enter a task.

Prints message in case list is empty.

Goes through every element and checks if its category matches the provided one. If it does, it is deleted.

4.4.2.4 void front_back_split (NODE * *source,* NODE ** *frontRef,* NODE ** *backRef*)

Function that splits the list in two sublists. Splits the nodes into front and back halves and returns the two lists using pointers. If the length is odd, the extra node goes in the front list.

Parameters

source	Pointer to the first element of the list that has to be split.
frontRef	Pointer to a pointer of the first sublist.
backRef	Pointer to a pointer of the second sublist.

Returns

Doesn't return anything.

If the list is empty or has just one element, it doesn't split it.

'fast' advances two nodes and 'slow' one node.

'slow' is before the midpoint in the list, so it gets split in two.

4.4.2.5 void merge_sort (NODE ** *headRef*)

Merge sort function.

Parameters

headRef	Takes as a parameter a pointer to the pointer of the first element.

Returns

Doesn't return anything.

If list is empty or has 1 element, it quits.

It else splits it into 2 sublists.

Recursively sorts the sublists.

Merges them together.

4.4.2.6 void modify_categ (NODE * *head*)

Function that modifies the category based on user input.

Parameters

head	Takes as a parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

Asks user to enter the category to be modified.

Asks user to enter new name for the category.

It goes through every element and checks if its category matches the one provided. If it does, it changes it to the new one.

4.4.2.7 void print_all (NODE * *head*)

Prints all the nodes of the list.

Parameters

head	Takes a pointer to the first element of the list as a parameter.

Returns

Doesn't return anything.

Opens file.

Prints message in case it can't be opened and quits.

Prints message in case the list is empty.

Prints everything stored in each node.

Closes file.

4.4.2.8 void print_by_categ (NODE * *head*)

Prints every element that has the same category as the one entered by the user.

Parameters

head	Takes as a parameter a pointer to the first element of the list.

Returns

Doesn't return anything.

Opens (or creates) output file (output.txt). If one already exists and it has data in it, the results will be printed after.

Asks user to enter a category.

Prints a message in case it can't be opened and quits.

Prints message in case list is empty.

Goes through every element and checks if its category matches the one provided by the user. If it does, it gets printed.

File is closed.

4.4.2.9 NODE* sorted_merge (NODE * *a*, NODE * *b*)

Function that merges two sublists together.

Parameters

a	Pointer to the first sublist.
b	Pointer to the second sublist.

Returns

Returns a pointer to the now merged list.

If one of them is empty, the other is returned.

Based on priority, it picks a or b and recurs.

4.5 main.c File Reference

Main function.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list.h"
#include "display.h"
```

Functions

- int main ()

4.5.1 Detailed Description

Main function.

Returns

An integer 0 upon exit succes.

4.5.2 Function Documentation

4.5.2.1 int main ( )

Start node is created

Allocating memory for it

Prints menu.

User makes a choice.