Name: Ai Yoshida
Neptun code: FPYYT9
Class: The Basic of Programming
Professor: Harvoth Gabor

Task manager
documentation for developer

Declaration in the beginning

Header Files

This program has two head files.

#include <stdio.h>

To use basic functions, such as printf function, scanf function, and file handling

#include <stdlib.h>

To use malloc function

#include <string.h>

To use string operation.

Definition of Macros

#define _CRT_SECURE_NO_WARNINGS

To avoid errors when the program uses the fopen function.

Data structure

This program uses structure and linked lists to manage data of the tasks.

“struct tasks” is the title of nodes. Since this is declared with typedef, all nodes inside this code will be written as just “tasks”. This node consists of these variables below.

char task_name[20];
char task_description[200];
char categories[20]

to connect tasks and categories.

int s_year;
int s_month;
int s_day;

The int variables to store the data of the start date.

int e_year;
int e_month;
int e_day;

The int variables to store the data of the end date.

int priority;

to store priority 1 to 10.

char dep[20]

to set what this task depends on.

struct tasks *next

To connect the next node.

Main function()

In the very beginning of the main function, database.txt is open with a+ mode. This makes sure the file exists. Then this file is immediately closed by the fclose function.

This is the first menu which will be shown to the users. It provides options by printf function, and gets the number from the user by using scanf function. To get numbers from the user, the int choice (int type of valuable) is used.

There are 1 to 4 options which the user can take, (1, create a new task, 2, see unfinished task list, 3, see the category list and task list, 4, See dependent tasks) and it is controlled by if loops. From 1 to 4, each number connects to each function below, and leads the user to the next program. If the user puts invalid value, the printf function shows “There is no such option!”.

In the case if the user inputs 1

The first if statement starts. Since the purpose of this choice is to create a function, the file opens with a+ mode. Then, head1 is allocated space for one task, to store the data for user input of a new task. In the case if code failed to allocate memory to head1, checking whether head1 is equal to NULL, if so, printf function shows “allocating memory has failed”, and then code will be finished by exit function.

If allocating memory succeeds, change head1 value by assigning function CreateNewTask(). By this function, head1 gets data of a new task from user input (detail of CreateNewTask() is mentioned below. Next, by using the fprintf function, write all the elements into the database.txt file. Each element has written into the file with extra mark “#”. This is for later use of the sscanf function to divide elements into each variable.

After all operations are done, head1’s memory will be released by free function, and database.txt file will be closed by fclose function.

In the case if the user inputs 2

The second if statement starts. The file opened by fopen function with “read” mode, since the purpose of this option is to see a list of unfinished tasks. First, head2 of tasks are set as NULL. This head2 is used to point to the linked list which this statement will read from file. Also, char str1 [500] is set to get string from file.

The next while statement will keep getting one sentence from the database file until its end. fgets function will read one line (which means one sentence till “\n”) from database.txt, and put that line into char str1 for 500 characters. In the first step, the line which is received from file save into str1, but in the next step, it will be divided into elements of tasks by the sscanf function. To divide elements into tasks, tasks “current” has prepared.Like other nodes, tasks “current” allocated the memory by malloc function, then check whether allocation has succeeded or not by checking the value is equal to NULL or not by if statement. In the case tasks “current” is equal to NULL, printf function will print out “allocating memory has failed”, and finished by exit function. sscanf function will separate the element into tasks “current” by “#”marks. Each element which divided by “#” goes to the current element, such as current-> task_name...until current->dep. After this procedure, “current->next” will be replaced with “head2”. Then “head2” will be replaced by current.

The procedure above will be repeated until fgets reach to an end of the file. Then, the fclose function closes the database.txt file. Since now head2 has a pointer to access to the nodes of the tasks, so put this in the SeeUrUnfinishedTasks() function. (The details of this function are mentioned below.) Then this function will show the user a list of unfinished tasks chronologically.

In the case if the user inputs 3

the third if statement starts. First of all, the fopen function opens the file database.txt with "read" mode. Set tasks "head3" and char str2 [500] to get information from the opened file. Basically, the procedure of how to obtain data from a file is exactly the same as the previous "in the case of the user input 2". With that method, let head3 has the pointer of read data. Then put it into the SeeUrCategoriesAndTasks() function(The details of this function are mentioned below). In this way, this statement will show the user a list of categories and tasks.

In the case if the user inputs 4

The fourth if statement starts. In the same method, this statement will obtain the elements from the database.txt file, and send the pointer to the SeeUrdep() function. The difference is only the names of variables, such as setting tasks node as "head4", and char str4 [500], and the name of the tasks to put element is "current4".

Functions

tasks *CreateNewTask()

This function receives nothing from the main function, but returns a task pointer.

First of all, this function sets s as a tasks variable, and allocates s heap memory the size of the struct tasks, by using malloc function. As same as other malloc allocation, check whether this value is valid or not by using if statement with NULL. If s is invalid, the exit function will end all.

Printf function instructs the user what to input (name of the task, description of the task, category, the start date, the end date, the priority, dependent task) . For the first two scanf functions, which ask the user to input the name of the task and the description of the task, use %[^\n]s to put the data into the variables. To clear the output console, fflush(stdin) is used. All scanf functions will put the data into tasks "s" elements. Only s->next element will be put NULL.

After succeeding in collecting necessary data from the user, this function will return "s" pointer to the first if statement in the main function to save all data into the database.txt file. When the procedure all above finishes, the free function will release the allocated memory of tasks "s".

void SeeUrUnfinishedTasks(tasks *head2)

This function receives tasks *head2 from the main function. This pointer contains the information of the node of the linked list of the data from the file. If there was no linked list inside the sent pointer(checking by if statement whether content equals to NULL or not), printf function will show "You have no unfinished task!".

Parameters int year, month, day are used to collect today's date from the user by scanf function. Printf function instructs how to put numbers.

To pick up nodes from the list which have a later date compared to today's date, we use the while statement. Task "*tmp2" is to manipulate head information.

By the while function, the program goes through all elements in head2 by tmp2. To find which task to show, first condition by if sentence is below.

(tmp2->s_year < year) && (tmp2->e_year > year)

If the dates of tmp2 meet this condition, all elements of it are shown by printf function. This condition checks the date of the year of the elements.
Second,

```
((tmp2->s_year == year) && (tmp2->e_year == year) && (tmp2->s_month < month) &&
                                        (tmp2->e_month > month))
```

In the case when the start year and end year are the same, and cannot be classified by previous condition, this sentence checks the element's month.

Third,

```
((tmp2->s_year == year) && (tmp2->e_year == year) &&(tmp2->s_month == month) &&
    (tmp2->e_month == month) && (tmp2->s_day < day) && (tmp2->e_day > day))
```

In the case when the start month and end month are the same, and cannot be classified by previous condition, this sentence checks the element's day.

In the end of whlie loop. by setting

```
tmp2 = tmp2->next;
```

proceed this loop to next step.

tasks *sortdate(tasks *unfinished)

This function will sort the given linked list into chronological order. This function receives the pointer which is a head of the linked list. Task "*i" and "*j" are for shifting nodes to change node order in the linked list. Also, the parameters are used for temporary storing to shift content of nodes too below.

```
char tmp_task_name[20];
char tmp_task_description[200];
char tmp_category[20];
int tmp_s_year;//start date
int tmp_s_month;
int tmp_s_day;
int tmp_e_year;//end date
int tmp_e_month;
int tmp_e_day;
int tmp_priority;
char tmp_dep[20];
```

The algorithm consists of two for loops and one if statement.

```
for(i = unfinished; i->next != NULL; i= i->next)
for(j=i->next; j != NULL; j=j->next)
if((i->s_year > j->s_year) && (i->s_month > j->s_month) && (i->s_day > j->s_day))
```

By this way, comparing two nodes which are next to each other, and if the second one has a later date than the first one, all elements are switched.

Then it returns the pointer which connects to the switched linked list to the SeeUrUnfinishedTasks().

void SeeUrCategoriesAndTasks(tasks *head3)

This function receives a head of a linked list which contains the nodes from the file.

char show[30];

tasks *tmp3 = head3;

are set, first of all. Next, by scanf function, get information of which category the user wants to see. printf function instructs how to enter by showing “Please enter the name of the category you would like to see: " .The given name of the category is stored into char show [30]. Next step is to find nodes which have the same category name with char show [30].

By using strcmp function, the name of the category in nodes and the given name from the user will be compared. if they are identical, printf function will show all data about the nodes which meet the condition. This loop continues till tmp3 shows NULL.”tmp3 = tmp3->next;” by putting this, tmp3 will move to the next node, and keep looping.

void SeeUrdep(tasks *head4)

This function will show a list of dependent tasks to the user.

This function receives a pointer which connects with nodes from the file from the main function. The if statement below confirms whether the linked list is empty or not.

```
if(head4 == NULL) {printf("Task list is empty.");}
```

```
char depend[30];
tasks *tmp4 = head4;
tasks *tmp44 = head4;
```

are declared.

printf function will instruct the user to put data ”Please enter the name of the dependent task you would like to see: ”. Then get the string by scanf function, and put it to char depend [30].

Firstly, this function will show the independent task detail, which has several dependent tasks. After the printf function shows “An independent task”, the while statement will find out the nodes which have identical category names with char depend [30] by strcmp function. printf function will show the detail of the nodes which meet the criteria.

Second, dependent tasks are shown. printf function describe "Tasks which depends on the task above” to the user, and show the nodes which have identical value with depend in “tmp4->dep”, and list the detail of the nodes.