DIARY MANAGEMENT

Algorithmics and data structure
L2 C Project 2023/2024

Camille Dommergue
Morgane Cordier
Oussama Rachid

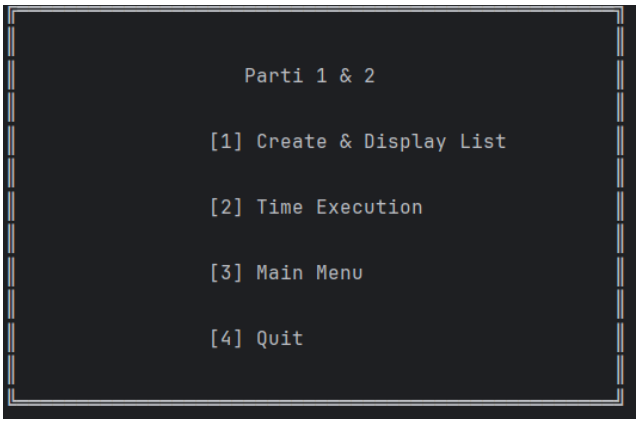
INTRODUCTION

This year project's goal is to create a diary, with three main objectives:
- Illustrate the organization of integers in multiple levels in linked lists.
- Conduct a performance evaluation by testing and comparing the execution time of classical and level-based search algorithms, aiming to identify the most efficient approach for diary management.
- Create and use a list of people's names to manage contacts and appointments, allowing users to search for patient names and view associated appointments. If a patient is not found, the program proposes easy insertion into the contact list with a new appointment.

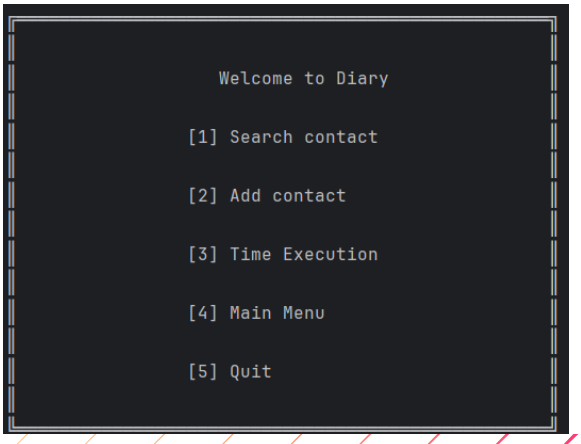
The goal is to create an intuitive and efficient diary management system by applying data structures and search algorithms.

The main menu

Part 1&2



Part 3



Main functions

To explain the main functions, we will divide it into 6 categories:

Data Structure

The program uses two main data structures: t_d_list and t_d_list_level. These structures are the same and represent a level-based list. The choice of these data structures is justified by their suitability for managing and organizing hierarchical data.

Search Algorithm

P_ClassicSearch (P1&2) & SimpleSearch (P3) conducts a classical linear search on level 0 of the list, while P_LevelSearch (P1&2) & LevelSearch (P3) is designed for level-based searches, which is more efficient. The performance of these algorithms is assessed by testing their execution time in a loop with several iterations.

Time Management

The program uses a custom timing mechanism with startTimer, stopTimer, and displayTime functions to measure the execution time of the search algorithms. This is crucial for evaluating the efficiency of the implemented algorithms.

File use

The code also employs a file-related function, ContactFileToList, which is responsible for implementing the contact list from a provided file.

Contact and Appointment Management:

Functions like CreateEntry, InsertCroissantList, Search, and DisplayAppointment are used for creating, inserting, searching, and displaying contact and appointment information. Functions provides options for creating new contacts, viewing appointments, and creating appointments based on user input.

Loop Structure

The program is organized into a main loop (Main_loop) that encapsulates the entire execution. Within this loop, there are nested loops for different functionalities, allowing the user to navigate through the program's diverse options.

Part 1 & 2

```
///// PARTIE 1
typedef struct s_d_cell
{
    int value;
    int nb_next;
    struct s_d_cell** tab_next;
}t_d_cell;

t_d_cell P_CreateCell(int value,int nb_next);


typedef struct s_d_list
{
    int nb_level;
    struct s_d_cell** tab_heads;
}t_d_list;

t_d_list P_CreateList(int nb_level);
bool P_InsertList(t_d_cell* cell, t_d_list* list);
void P_DisplayLevel(int level,t_d_list list);
void P_DisplayLevelsList(t_d_list list);
void nDisplayAlignedLevelsList(t_d_list* list);
void P_DisplayAlignedLevelsList(t_d_list list);
void P_InsertCroissantList(t_d_cell* cell, t_d_list* list);

///// PARTIE 2

t_d_list P_CreateLevelList(int nb);
bool P_ClassicSearch(int value, t_d_list list);
bool P_LevelSearch(int value, t_d_list list);
```

=> t_d_cell is a type representing a cell in a linked list. It will hold an integer value and an array of pointers to the next cells.

=> t_d_list is a type and organize cells into levels and hold a head on each level.

=> P_DisplayLevel is a function that displays the cells of a linked list at a specified level (user's choice).

=> P_DisplayLevelsList is a function using the same prosses that P_DisplayLevel, but shows all cells (organized by levels) in the linked list.

=> P_InsertCroissantList is used to add a cell to the linked list while maintaining the sorting, in a croissant order.

=> P_CreateLevelList will initialize a new level-based list, with a level size given by the user

=> P_ClassicSearch will search the value by traversing the level 0 only, will return 1 if the value is found and 0 if it's not found.

=> P_LevelSearch will locate a given value by using the levels, starting firstly with the highest level will return 1 if the value is found and 0 if it's not found.

Part 3

```c
// contact
typedef struct s_d_contact
{
    char* name;//name_firstname

}t_d_contact;

// appointment
typedef struct s_d_appointment
{
    char* date; //jj/mm/yyyy
    char* hour; // hh:mm
    char* time; //hh:mm
    char* goal;

}t_d_appointment;

// cell for simply linked list
typedef struct s_cell
{
    struct s_d_appointment* appointment;
    struct s_cell* next;

}t_cell;
```

=> **t_d_contact** type represents a contact in the diary management system. It holds a name and a first name separated by an underscore.

=> **t_d_appointment** type represents an appointment, with its date at format jj/mm/yyyy, its hour format hh:mm, its time format hh:mm, and a brief description of the appointment's purpose (goal).

=>**t_cell** type represents a cell of a simply linked list. It contains a pointer to an appointment and a pointer to the next cell in the list.

```c
// simply linked list
typedef struct s_std_list
{
    t_cell* head;

}t_std_list;

// entry
typedef struct s_d_entry
{
    struct s_d_contact contact;
    t_std_list* list;
    int nbcap;
    struct s_d_entry** search;
    int lv;

}t_d_entry;

// level list
typedef struct s_d_list_level
{
    struct s_d_entry** tab_heads;
    int nb_level;

}t_d_list_level;
```

=> t_std_list type represents a simply linked list. It contains a pointer to the head of the list.

=> t_entry type is an entry of the diary management system. This is a central structure that brings together the contact details and the linked list of appointments with the level of the entry.

=>t_d_list_level type represents a level-based list of entries. This structure organizes entries into levels. Each level is essentially a linked list of entries, and the array of pointers holds the head of each level's linked list.

Part 3 (bis)

```c
char* ScanNAME();
char* scanDate();
char* scanHours();
char* scanTime();
char* scanGoal();
char* Getname(char* contact);
char** GetNAME(char* contact);
char** GetDATE(char* date);
char** GetHT(char* ht);
```

=> Scan functions are used to read the input of the user: name, date, time, and goal. We use Gets functions to split input and return it, separated by underscore for **GetNAME,** slash for **GetDATE** and **":"** for **GetHT**. (**Getname** allow to get only the firstname)

```c
//insert list
void InsertList(t_d_entry* entry, t_d_list_level * list) {...}//MARCHE
void InsertCroissantList(t_d_entry* entry, t_d_list_level * list) {...}//MARCHE

//DisplayLEVEL
void DisplayLevelsList(t_d_list_level list) {...}// MARCHE
void DisplayLevel(int level,t_d_list_level list) {...} // MARCHE

//Search function
t_d_entry* LevelSearch(char* value, t_d_list_level list) {...}

int DisplayAppointment(t_d_entry entry) {...}

void CreateInsertAppointment(t_d_entry* entry) {...}

int SimpleSearch(char* value, t_d_list_level list) {...}
```

=> insertCroissantList will insert an entry in the list while keeping alphabetical order.

=>DisplayAppointment will display the appointment of an entry given in argument

=> LevelSearch will allow you to search the first entry with the same 3 first characters, starting by the highest level and returning this entry.

=> CreateInsertAppointment will allow to create and insert appointments in an entry given as argument

=> SimpleSearch searches for the entry with the same name, only searching through the level 0. it returns 1 if it found, it or 0 if not.

```
#ifndef UNTITLED_DISPLAY_H
#define UNTITLED_DISPLAY_H

#include ...

void menuMain();
void menuP12();
void menuBASE();
void ContactFileToList(t_d_list_level* list);
t_d_entry* Search(t_d_list_level* list);
void Test_time(t_d_list_level* list);
#endif
```

(=> Menus are shown on the 2nd page.)
=> menuMain allows you to choose between running parts 1 & 2 or part 3.

=> menuP12 runs parts 1 and 2
=> menuBASE runs part 3

=> ContactFileToList function will read the file containing the list of contact, create an entry per contact and insert it into the list given in argument
=> Search function will search and return the entry of the first contact's name with the 3 first letter input by the user.
=> Test_time is used to obtain the complexity of part 1&2

Results

Part 1&2 :

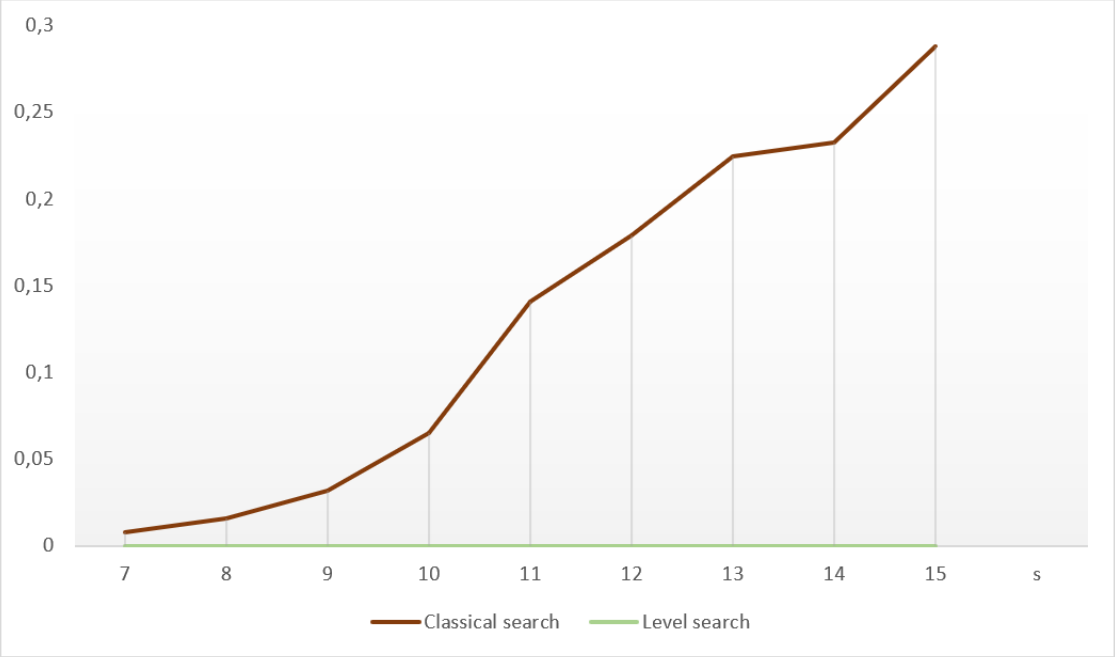
When we test the Part 1&2 we can chose the number of level and the program provides a simple display list and an aligned one

```
Welcome to C-Project L2

   [1] Part 1 & 2

   [2] Part 3

   [3] Quit

>> 1

      Parti 1 & 2

   [1] Create & Display List

   [2] Time Execution

   [3] Main Menu

   [4] Quit

>>1
Enter a number of lv :
>>4
```

```
Level list of grade 9
Classic Search [12] 000,012
Level Search[0] 000,000

Level list of grade 10
Classic Search [26] 000,026
Level Search[1] 000,001

Level list of grade 11
Classic Search [50] 000,050
Level Search[0] 000,000

Level list of grade 12
Classic Search [89] 000,089
Level Search[1] 000,001

Level list of grade 13
Classic Search [137] 000,137
Level Search[1] 000,001

Level list of grade 14
Classic Search [143] 000,143
Level Search[0] 000,000

Level list of grade 15
Classic Search [141] 000,141
Level Search[1] 000,001
Confer Graph in Rapport
```

```
Enter a number of lv :
>>4
Display List:
[list head_0 @-]-->[1|@-]-->[2|@-]-->[3|@-]-->[4|@-]-->[5|@-]-->[6|@-]-->[7|@-]-->[8|@-]-->[9|@-]-->[10|@-]-->[11|@-]-->
[12|@-]-->[13|@-]-->[14|@-]-->[15|@-]--> NULL
[list head_1 @-]-->[2|@-]-->[4|@-]-->[6|@-]-->[8|@-]-->[10|@-]-->[12|@-]-->[14|@-]--> NULL
[list head_2 @-]-->[4|@-]-->[8|@-]-->[12|@-]--> NULL
[list head_3 @-]-->[8|@-]--> NULL

Display List Aligned:
[list head_0 @-]-->[1|@-]-->[2|@-]-->[3|@-]-->[4|@-]-->[5|@-]-->[6|@-]-->[7|@-]-->[8|@-]-->[9|@-]-->[10|@-]-->[11|@-]-->
[12|@-]-->[13|@-]-->[14|@-]-->[15|@-]--> NULL
[list head_1 @-]----------->[2|@-]----------->[4|@-]----------->[6|@-]----------->[8|@-]----------->[10|@-]----------->
[12|@-]----------->[14|@-]----------> NULL
[list head_2 @-]----------------------------->[4|@-]----------------------------->[8|@-]----------------------------->
[12|@-]-----------------------------> NULL
[list head_3 @-]------------------------------------------------------------------->[8|@-]-------------------------------
-------------------------------------> NULL
```

Time (in seconds) consumed by the program to search for a value in a classical list and in a levelled list



When operating the second part we can see the difference between the classical search and levelled search mostly while using a big amount of data.

Part 3:

Screen Creation of contact & appointment

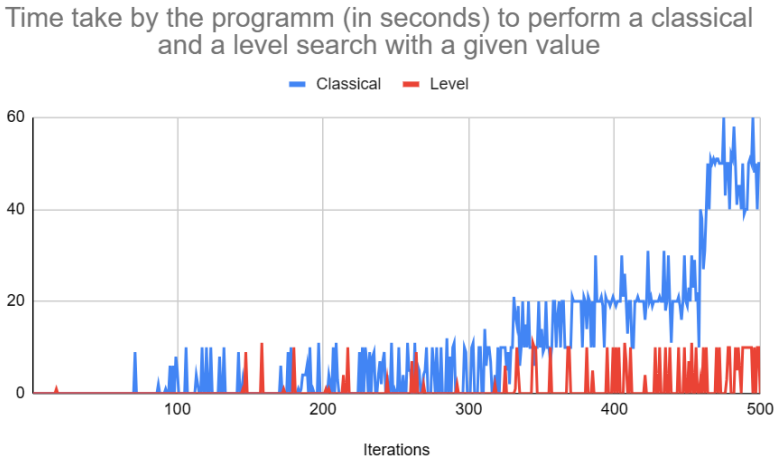
Display:

[it's only a screen of the first part of the head_0 display]

```
[list head_0 @-]-->[abdelhafed|@-]-->[abdelhafid|@-]-->[abdelhafide|@-]-->[abdelhak|@-]-->[abdelhake|@-]-->[abdelhakim
|@-]-->[abdelhakime|@-]-->[abdelhali|@-]-->[abdelhalim|@-]-->[abdelhalime|@-]-->[achyl|@-]-->[acil|@-]-->[acyl|@-]-->[ad
a|@-]-->[adael|@-]-->[adal|@-]-->[adalbert|@-]-->[adam|@-]-->[afdal|@-]-->[affan|@-]-->[affif|@-]-->[afid|@-]-->[afif|@-
]-->[afnan|@-]-->[afonso|@-]-->[afrahim|@-]-->[afrim|@-]-->[aftab|@-]-->[afzal|@-]-->[agapito|@-]-->[agash|@-]-->[agathe
|@-]-->[agathon|@-]-->[akhenaton|@-]-->[akhil|@-]-->[akhim|@-]-->[akhmad|@-]-->[akhmed|@-]-->[aki|@-]-->[akif|@-]-->[bab
acar|@-]-->[babak|@-]-->[babakar|@-]-->[babou|@-]-->[babylas|@-]-->[bacar|@-]-->[bacari|@-]-->[bacary|@-]-->[bacem|@-]--
>[bachar|@-]-->[bela|@-]-->[belaid|@-]-->[belal|@-]-->[belarmino|@-]-->[belel|@-]-->[belgacem|@-]-->[belhadj|@-]-->[belh
assen|@-]-->[believe|@-]-->[biaggio|@-]-->[biagio|@-]-->[biagui|@-]-->[bicente|@-]-->[bien-aime|@-]-->[bienaime|@-]-->[b
ienvenido|@-]-->[bienvenu|@-]-->[bienvenue|@-]-->[bijan|@-]-->[bouabdallah|@-]-->[bouabdellah|@-]-->[boualam|@-]-->[boua
lem|@-]-->[bouaza|@-]-->[bouazza|@-]-->[bouba|@-]-->[boubacar|@-]-->[boubacary|@-]-->[boubakar|@-]-->[camy|@-]-->[camyl|
@-]-->[camylle|@-]-->[can|@-]-->[candassamy|@-]-->[candide|@-]-->[candido|@-]-->[caner|@-]-->[cantin|@-]-->[cantor|@-]--
>[cemil|@-]-->[cendro|@-]-->[cengiz|@-]-->[cengizhan|@-]-->[cenk|@-]-->[cension|@-]-->[cenzo|@-]-->[cephas|@-]-->[cerron
e|@-]-->[cesaire|@-]-->[chalom|@-]-->[chalvi|@-]-->[cham|@-]-->[chamil|@-]-->[champion|@-]-->[chams|@-]-->[chams-dine|@-
]-->[chams-eddine|@-]-->[chamsdine|@-]-->[chamseddine|@-]-->[clint|@-]-->[clinton|@-]-->[clive|@-]-->[clivens|@-]-->[clo
dius|@-]-->[clodomir|@-]-->[clotaire|@-]-->[clothaire|@-]-->[clotilde|@-]-->[cloud|@-]-->[cordier|@-]-->[dalla|@-]-->[da
llil|@-]-->[dalton|@-]-->[dalvin|@-]-->[daly|@-]-->[dalyan|@-]-->[dalyl|@-]-->[dama|@-]-->[daman|@-]-->[damas|@-]-->[dec
ius|@-]-->[declan|@-]-->[dede|@-]-->[deeclan|@-]-->[deelan|@-]-->[deen|@-]-->[deivy|@-]-->[dejan|@-]-->[deklan|@-]-->[de
l|@-]-->[dick|@-]-->[dicklan|@-]-->[dickson|@-]-->[diclan|@-]-->[diclane|@-]-->[didi|@-]-->[didie|@-]-->[didier|@-]-->[d
iego|@-]-->[duy|@-]-->[dwain|@-]-->[dwayn|@-]-->[dwayne|@-]-->[dwen|@-]-->[dwight|@-]-->[dyan|@-]-->[dycklan|@-]-->[dycl
an|@-]-->[dyclane|@-]-->[dziou|@-]-->[eddie|@-]-->[eddine|@-]-->[eddy|@-]-->[eddyne|@-]-->[eddyson|@-]-->[edeen|@-]-->[e
dek|@-]-->[edel|@-]-->[edelbert|@-]-->[edem|@-]-->[egan|@-]-->[ege|@-]-->[egehan|@-]-->[egemen|@-]-->[egide|@-]-->[egidi
o|@-]-->[egiste|@-]-->[egon|@-]-->[egor|@-]-->[egzon|@-]-->[eldar|@-]-->[elder|@-]-->[eldi|@-]-->[eldin|@-]-->[eldine|@-
]-->[eldrick|@-]-->[eleazar|@-]-->[eleodore|@-]-->[eleonor|@-]-->[eleonore|@-]-->[elvys|@-]-->[elwan|@-]-->[elwann|@-]--
```

Complexity:

```
Classic Search 0[0] 000,000
Level Search 0[0] 000,000

Classic Search 1[0] 000,000
Level Search 1[0] 000,000

        [···]

Classic Search 498[15] 000,015
Level Search 498[0] 000,000

Classic Search 499[10] 000,010
Level Search 499[0] 000,000
```



Conclusion

As a team, this diary management project provided us with some important lessons. On the technical side, we have put in application the courses seen in class into using lists and optimizing search algorithms, usage of lists and trees. Collaboration was important. We realized how important is the communication and documentation are for efficient teamwork. We managed time wisely to try providing the most user-friendly design to smooth the user experience. The most important thing is that we now have a clear example of the usage of trees and lists.