ECOLE SUPERIEURE DE L'ÉDUCATION ET DE LA FORMATION - AGADIR

CREATION D'UNE APPLICATION DES ARBRES BINAIRES SUR DES SALARIÉS

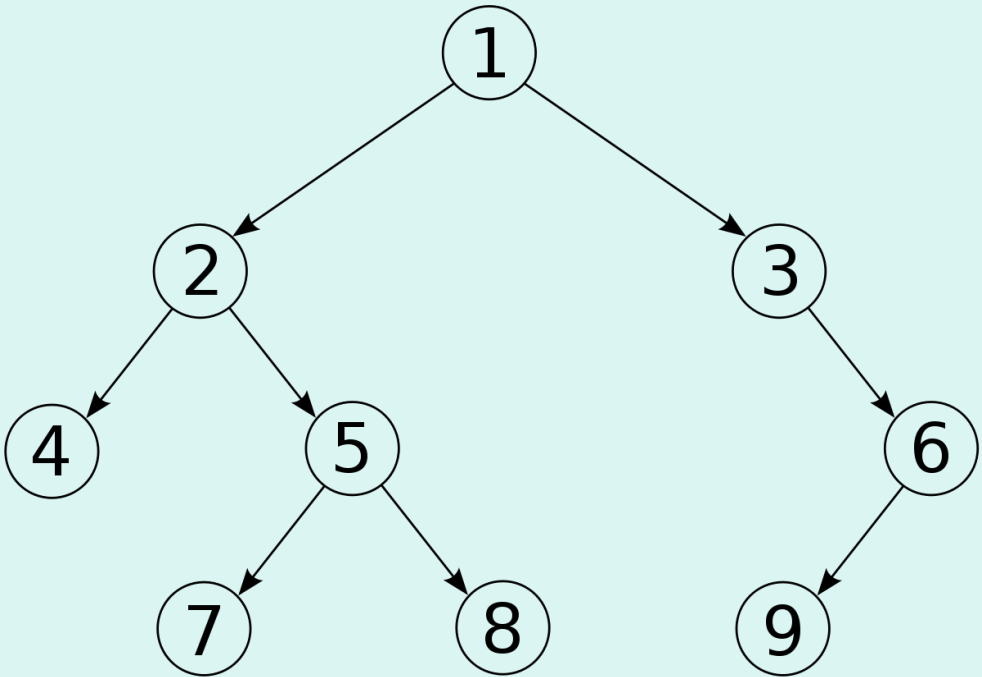
STRUCTURE DES DONNÉES

RÉALISÉ PAR:

- ASMA ELFAHIM
- LAILA HAMZA
- AMAL ADERDOUR

Encadré par:

- MR Yousef Es-saady



SOMMAIRE

INTRODUCTION

Dans le cadre de notre deuxième année en **E**cole **S**upérieure de l'**E**ducation et de la **F**ormation , nous avons eu pour tâche la réalisation d'un programme en langage C.
Pour poursuivre la découverte des différentes structures de données, nous allons maintenant nous attarder sur les arbres.
Un arbre binaire est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé nœud, le nœud initial étant appelé racine.
Dans un arbre binaire, chaque élément possède au plus deux éléments fils au niveau inférieur, habituellement appelés gauche et droit. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé père.
Pour ce faire, nous avons crée un exemple sur la gestion des salariés d'une entreprise, dont chaque employé possède un chef pour implémenter l'arbre binaire.
La première chose à faire est de créer les structures suivantes:

- **tree (arbre)**
- **employe**

Ensuite, nous avos crée un ensemble de fonctions pour faciliter la gestion de cette entreprise.
Après des nombreuses analyses, nous avons abouti à un code, qui sera bien détaillé et expliqué dans les chapitres suivants.

Chapitre 1 : Créations des structures

1. Structure d'un employé :

Afin de pouvoir gérer les salariés d'une entreprise, nous avons créé un type Employe en utilisant une structure, pour l'identifier dans l'arbre binaire de recherche. Cette dernière rassemble des informations sur ct employé, notamment :

- **son nom,**
- **son prénom**
- **son salaire**
- **année d'entrée à l'entreprise**

```
/* Structure d'un element */
    typedef struct Employe
    {
        char nom[30];
        char prenom[30];
        int salaire;
        int annee_Entree;

    }Employe;
```

2. Structure d'un arbre :

Tout comme les listes chaînées, les arbres sont basés sur une structure du langage C. La différence sera qu'elle contiendra des pointeurs pour lier les éléments :

- un pointeur pour accéder à la branche de gauche,
- l'autre pour accéder à la branche de droite,
- un autre pour accéder à la racine.
- Une valeur pour ordonner les éléments

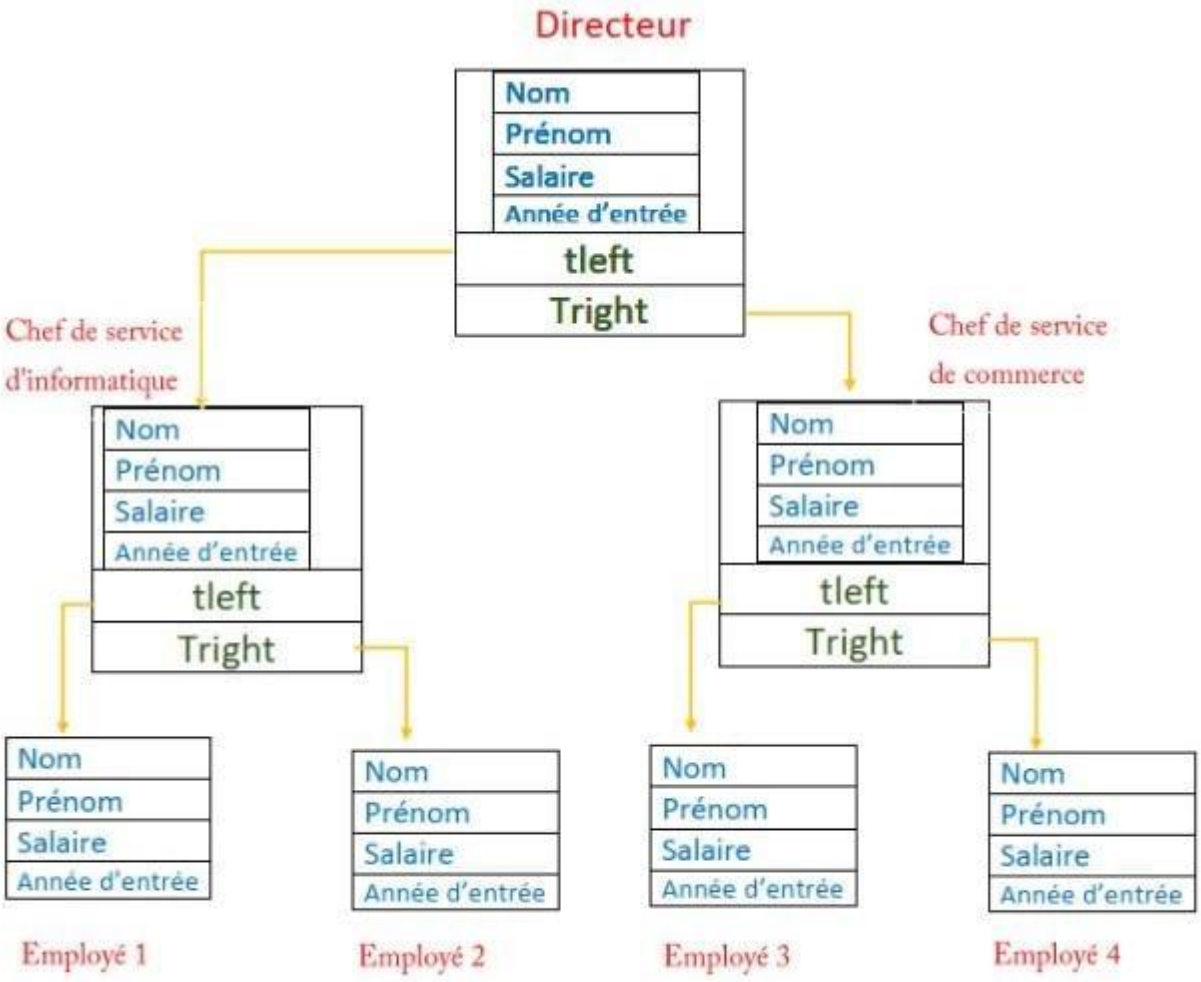
Nous avons maintenant suffisamment d'éléments pour constituer la structure d'un nœud.

```
/* Structure d'un Arbre */
typedef struct Tree
{
    Employe info;
    struct Tree *tleft;
    struct Tree *tright;
    struct Tree *parent;
}Tree;
```

Les prototypes :

Le choix de ses fonctions sera expliqué dans le chapitre suivant.

```
/* Prototypes des fonctions */
Tree *new_tree(Employe X);
void clean_tree(Tree *tr);
Tree *join_tree(Tree *left, Tree *right,Employe node);
void print_tree_prefix(Tree *tr);
Tree* ajouter(Tree *tr,Employe X);
int count_tree_nodes(Tree *tr);
int nbfeuille(Tree *tr);
int Rechercher(Tree *tr,Employe value);
int Calcul_2020(Tree *tr);
int is_empty_Tree(Tree* tr);
```



Chapitre 2 : Les fonctions :

1. Création d'un arbre :

Tout d'abord, nous avons commencé par créer un élément de type **Tree** *(Créer un arbre)* et lui réserver la mémoire.

Ensuite, nous vérifions que l'allocation est bien passée, sinon, un message d'erreur s'affichera.

L'étape suivante est d'initialiser les champs :

- **Les trois pointeurs à NULL**
- **La valeur**

Ces éléments seront donc positionnés au bout d'une branche et n'ont donc pas d'enfants.

```
Tree* new_tree(Employe X)
{
    Tree *tr = malloc(sizeof(*tr));
    if(tr == NULL)
    {
        fprintf(stderr, "Erreur allocation memoire.\n");
        exit(EXIT_FAILURE);
    }

    tr->info = X;
    tr->tleft = NULL;
    tr->tright = NULL;
    tr->parent = NULL;

    return tr;
}
```

2. Suppression d'un arbre :

Comme nous avons alloué de la mémoire avec ***malloc***, il faut donc la libérer. Une fonction est alors nécessaire dans notre exemple, pour ce faire :

- **Débrancher l'arbre gauche**
- **Débrancher l'arbre droit**
- **Débrancher le parent**
- **Libérer la mémoire en utilisant free()**

```
void clean_tree(Tree *tr)
{
    if(tr == NULL)
        return;

    tr->tleft=NULL;
    tr->tright=NULL;
    tr->parent=NULL;
    free(&tr->info);
}
```

3. Jointure d'un arbre gauche, un arbre droit et le parent :

Cette fonction prend en paramètres d'entrée un arbre gauche, un arbre droit et les informations de l'employé, elle joint deux arbres pour n'en former qu'un.

Tout d'abord, nous avons créé un nouvel arbre en utilisant la fonction **new_tree()**. Cet arbre n'aura ni des sous arbres gauches, ni des sous arbres droits, seulement la racine. **node** est le noeud qui lie les deux arbres

Ensuite, nous avons affecté à :

- **L'arbre gauche de tr : left**
- **L'arbre droit de tr : right**
- **Si left existe : son parent est tr**
- **Pareil pour right**

Elle retourne le nouvel arbre formé.

```
Tree *join_tree(Tree *left, Tree *right,Employe node)
{
    Tree *tr = new_tree(node);

    tr->tleft = left;
    tr->tright = right;

    if(left != NULL)
        left->parent = tr;
    if(right != NULL)
        right->parent = tr;
    return tr;
}
```

4. Afficher l'arbre binaire :

On teste d'abord: si l'arbre est vide, on affiche: il n'y a rien à afficher, l'arbre est vide.

Si la racine de l'arbre n'est pas vide, alors nous affichons son nom et son prénom.

Si le nœud gauche n'est pas vide, nous appelons fonction lui même pour que le nœud gauche soit l'arbre.

Si le nœud droit n'est pas vide, nous appelons fonction lui même pour que le nœud droit soit l'arbre

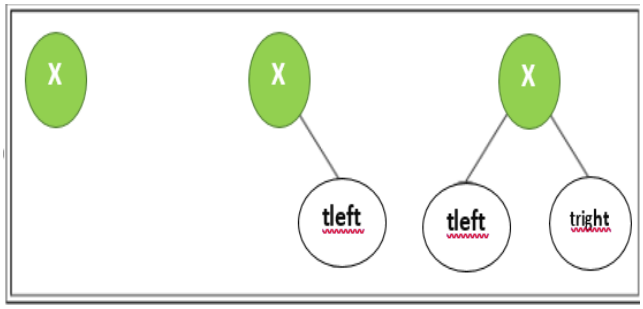
```
void print_tree_prefix(Tree *tr)
{
    if(tr== NULL)
        {
            printf("\tRien a afficher, l'arbre est vide.\n");
            return;
        }
    if(tr->parent != NULL)
        printf("(%s %s) -> (%s %s)\n",tr->parent->info.nom,tr->parent->info.prenom,tr->info.nom,tr->info.prenom);
    else
        printf("(%s %s)\n", tr->info.nom, tr->info.prenom);
    if(tr->tleft != NULL)
        print_tree_prefix(tr->tleft);

    if(tr->tright != NULL)
        print_tree_prefix(tr->tright);
}
```

5. Ajouter un nœud :

Cette fonction nous permet d'ajouter un élément x dans une nouvelle racine de l'arbre.

Nous allons donc avoir besoin de la fonction **join_tree()** que nous avons déjà crée et elle aura comme paramètre les deux sous-arbres gauche, Droit et la nouvelle racine x.




```
Tree* ajouter(Tree *tr,Employe X)
{
        if(tr ==NULL)
            return new_tree(X);
        tr=join_tree(tr->tleft,tr->tright,X);
    return (tr);
}
```

6. Compter le nombre des nœuds :

si l'arbre est vide on return 0.

Sinon Nous supposons que le nœud gauche est comme un arbre et nous calculons le nombre de nœuds dans cet arbre en appelant la même fonction, et la même chose s'applique au nœud droit, et on return leur somme plus 1.

```c
/**
* Compte le nombre de noeuds d'un arbre(Taille)
* @param tr L'arbre dont il faut compter les noeuds
* @return Le nombre de noeuds de l'arbre binaire
*/
int count_tree_nodes(Tree *tr)
{
    if(tr == NULL)
        return 0;

    return (count_tree_nodes(tr->tleft) + count_tree_nodes(tr->tright) + 1);
}
```

7. Compter le nombre des feuilles :

- si l'arbre est vide on retourne 0.
- Si le nœud gauche et le nœud droit sont vides on retourne 1.

Sinon Nous supposons que le nœud gauche est comme un arbre et nous calculons le nombre de feuille dans cet arbre en appelant la même fonction, et la même chose s'applique au nœud droit, et on return leur somme.

```c
/**
* Nombre de feuilles d'un arbre binaire
*/
int nbfeuille(Tree *tr)
{
    if(tr == NULL)
    return 0;

        if ((tr->tleft == NULL)&&(tr->tright == NULL))
        return 1;
        else
         return (nbfeuille(tr->tleft)+nbfeuille(tr->tright));
}
```

8. Rechercher un élément dans l'arbre binaire :

Nous avons dit que notre arbre est un arbre de recherche. C'est donc la fonction **Recherche()** que nous allons créer, elle devra nous indiquer si un élément avec une clé de valeur x est présente dans l'arbre ou non.

- **Tout d'abord, on teste si l'arbre est vide.**
- **On vérifie si on est en présence de l'élément recherché (l'année d'entrée), si oui on retourne 1.**
- **Si le salaire est strictement supérieur, alors elle est dans le sous-arbre gauche, sur lequel on effectue récursivement la recherche. De même si la clé recherchée est strictement inférieure à la clé de la racine, la recherche continue dans le sous-arbre droit.**

Un appel récursif de la fonction a été fait pour parcourir l'arbre élément par élément

```
int Rechercher(Tree *tr,Employe value)
{
        if(tr ==NULL)return 0;

         if ((tr->info.annee_Entree) == (value.annee_Entree)) return 1;
         if (value.salaire > tr->info.salaire)
                    Rechercher(tr->tleft,value);
               else Rechercher(tr->tright,value);
```

9. Calculer le nombre des employés selon leur année d'entrée :

La fonction **Calcul_2020()** permet de calculer le nombre des employés qui ont intégré l'entreprise avant ou durant l'année 2020.

Pour ce faire, on va suivre les étapes suivantes :

- **Vérifier si l'arbre est vide, le nombre d'employés est n=0**
- **Sinon, nous comparons les années d'entrées de tous les employés par la valeur 2020**
- **Appel récursif de la fonction pour parcourir l'ensemble de l'arbre (sous-branches droites et sous-branches gauches)**
- **Si la condition est satisfaite : incrémentation.**

```c
int Calcul_2020(Tree *tr)
{
    int n;
    if(tr ==NULL)
       n=0;
    else
    {
        if((tr->info.annee_Entree)<=2020)
        n=1+Calcul_2020(tr->tleft)+Calcul_2020(tr->tright);
        else
        n=Calcul_2020(tr->tleft)+Calcul_2020(tr->tright);
    }
    return n;
}
```

10. Vérifier si l'arbre est vide :

La fonction *is_empty_Tree* permet de vérifier si l'arbre est vide ou pas. Elle doit prendre en paramètres d'entrée l'arbre et retourne 1 si l'arbre est vide, 0 sinon.

```c
int is_empty_Tree(Tree* tr)
{
    if(tr == NULL)
        return 1;

    return 0;
}
```

Programme principal

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mes_fonctions.h"
int main()
{
    printf("********************GESTION D'UNE
ENTREPRISE/SALARIES********************\n");
    Tree* arbre=malloc(sizeof(Tree));
    int choix;
    Employe* derct =malloc(sizeof(Employe));// directeur
    Employe* chef_I =malloc(sizeof(Employe));// chef de service
d'informatique
    Employe* chef_C =malloc(sizeof(Employe));//chef de service de commerce
    Employe* OfficerI =malloc(sizeof(Employe));// 1er employé dans service
d'informatique
    Employe* OfficerI2 =malloc(sizeof(Employe));// 2émè employé dans service
d'inforrmatique
    Employe* OfficerC =malloc(sizeof(Employe));// 1er employé dans service de
commerce
    Employe* OfficerC2 =malloc(sizeof(Employe));// 2émè employé dans service
de commerce
    Employe* newD =malloc(sizeof(Employe));

    printf("Salut! Veuillez commencer par saisir les informations du
directeur:");
    printf("\n\t - Nom et Prenom: ");
    scanf("%s %s",&derct->nom,&derct->prenom);
    printf("\t - Salaire : ");
    scanf("%d",&derct->salaire);
    printf("\t - Ann%ce d'entr%ce: ",130,130);
    scanf("%d",&derct->annee_Entree);

    arbre = new_tree(*derct);//cree un arbre

    //system("cls");
    printf("\nBienvenue '%s %s' tu peux maintenant g%crer votre
entreprise.\n" , derct->nom,derct->prenom, 130);
    printf("REMARQUE: Cette entreprise contiendra DEUX chefs de services. A
chaque service, il y aura deux employ%cs.\n",130);
     printf("\nSaisissez les informations du chef de service
'd'informatique':\n");
    printf("\t - Nom et Prenom: ");
    scanf("%s %s",&chef_I->nom,&chef_I->prenom);
    printf("\t - Salaire: ");
    scanf("%d",&chef_I->salaire);
    printf("\t - Ann%ce d'entr%ce: ",130,130);
    scanf("%d",&chef_I->annee_Entree);
    printf("Saisissez les informations du chef service de 'commerce':\n");
    printf("\t - Nom et Prenom: ");
    scanf("%s %s",&chef_C->nom,&chef_C->prenom);
    printf("\t - Salaire: ");
    scanf("%d",&chef_C->salaire);
    printf("\t - Ann%ce d'entr%ce: ",130,130);
    scanf("%d",&chef_C->annee_Entree);
    printf("\n**Service d'informatique:**\n __REMARQUE: Vous pouvez ajouter
deux employ%cs a chaque service.__\n",130);
    printf("\nSaisissez les informations des deux employ%cs :\n",130);
                printf("\t - Nom et Prenom: ");
```

```
                scanf("%s %s",&OfficerI->nom,&OfficerI->prenom);
                printf("\t - Salaire: ");
                scanf("%d",&OfficerI->salaire);
                printf("\t - Ann%ce d'entr%ce: ",130,130);
                scanf("%d",&OfficerI->annee_Entree);
                printf("--> Deuxieme employ%c:\n\t - Nom et Prenom: ", 130);
//  2eme employé
                scanf("%s %s",&OfficerI2->nom,&OfficerI2->prenom);
                printf("\t - Salaire: ");
                scanf("%d",&OfficerI2->salaire);
                printf("\t - Ann%ce d'entr%ce: ",130,130);
                scanf("%d",&OfficerI2->annee_Entree);
    printf("\n**Service de commerce:**\n");
    printf("\nSaisissez les informations des deux employ%cs :\n",130);
                printf("\t - Nom et Prenom: ");
                scanf("%s %s",&OfficerC->nom,&OfficerC->prenom);
                printf("\t - Salaire: ");
                scanf("%d",&OfficerC->salaire);
                printf("\t - Ann%ce d'entr%ce: ",130,130);
                scanf("%d",&OfficerC->annee_Entree);
                printf("--> Deuxieme employ%c:\n\t - Nom et Prenom: ",130);
//  2eme employé
                scanf("%s %s",&OfficerC2->nom,&OfficerC2->prenom);
                printf("\t - Salaire: ");
                scanf("%d",&OfficerC2->salaire);
                printf("\t - Ann%ce d'entr%ce: ",130,130);
                scanf("%d",&OfficerC2->annee_Entree);

    arbre=
join_tree(join_tree(new_tree(*OfficerI),new_tree(*OfficerI2),*chef_I),join_tr
ee(new_tree(*OfficerC),new_tree(*OfficerC2),*chef_C),*derct);
    system("cls");
    printf("\n--------Effectuez un choix:--------\n");

    printf("\t1- Tapez 1 pour changer le directeur \n");
    printf("\t2- Tapez 2 pour Afficher l'ensemble des employes de
l'entreprise\n");
    printf("\t3- Le nombre des employes de l'entreprise \n");
    printf("\t4- Le nombre total des personnes dans l'entreprise\n");
    printf("\t5- Chercher un employe dans l'entreprise \n");
    printf("\t6- Nombre des employes qui ont integre l'entreprise avant ou
egal a l'an 2020 \n", 130 );
    printf("\t7- Supprimer tout les employes de l'entreprise\n");

    printf("\t!!ATTENTION!!: Si vous souhaitez quitter, TAPEZ 0\n");
    do{
    printf("\nSaisissez votre choix:\t");
    scanf("%d",&choix);
    switch(choix)
    {
        case 1:{
                printf("\nSaisissez les informations du nouveau
directeur:\n");
                printf("\t - Nom et Prenom: ");
                scanf("%s %s",newD->nom,&newD->prenom);
                printf("\t - Salaire: ");
                scanf("%d",&newD->salaire);
                printf("\t - Ann%ce d'entr%ce: ",130,130);
                scanf("%d",&newD->annee_Entree);
                arbre=ajouter(arbre,*newD);
              };break;
        case 2:{
                print_tree_prefix(arbre);
```

```
                };break;
        case 3:{
                 printf("\n- Le nombre des employ%cs est: %d\n",130,
nbfeuille(arbre));
                };break;
        case 4:{
                 printf("\n- Le nombre total des personnes dans l'entreprise:
%d",count_tree_nodes(arbre));
                };break;
        case 5:{
                  Employe* val=malloc(sizeof(Employe));
                  printf("\n Entrez les informations de l'employ%c
cherch%c:\n",130, 130);
                  printf("\t Nom et Prenom: ");
                  scanf("%s %s",&val->nom,&val->prenom);
                  printf("\t salaire: ");
                  scanf("%d",&val->salaire);
                  printf("\t Ann%ce d'entr%ce: ",130,130);
                  scanf("%d",&val->annee_Entree);
                  if(Rechercher(arbre,*val))
                      printf("\t- OUI, Employ%c trouv%c",130,130);
                   else   printf("\t- OUPS!! Cet employ%c n'existe pas",130);
                };break;
        case 6:{
                  printf("- Le nombre des employes qui ont integrE
l'entreprise avant ou egal a l'an de 2020 sont :%d\n",Calcul_2020(arbre));
                };break;
        case 7:{
                 clean_tree(arbre);
                  printf("\n Entreprise vide");
                  free(arbre);
                };break;
     }
     printf("\n");
     }while(choix !=0);



    free(arbre);
    return 1;
}
```

Conclusion :

Le programme a été organisé en trois fichiers : **« main.c »**, **« MesFonctions.h »** et **« MesFonctions.c »**. Ainsi qu'une petite vidéo descriptive du code. Vous trouverez, ci-joint, le code principal dans un fichier zip **« Arbres »**.

A l'aide de ce projet nous avons pu comprendre et expérimenter les différentes étapes de l'implémentation d'un arbre en commençant par l'analyse des différentes fonctions dont nous aurons besoin pour enrichir le code.

De plus la programmation nous a permis d'améliorer nos connaissances du langage C.

Vu les circonstances qu'on est en train de vivre ces derniers temps, nous avons fait de notre mieux pour pouvoir concevoir ce programme. Nous avons rencontré de nombreuses difficultés notamment pour la fonction de suppression de l'arbre complet et la recherche. La fonction de recherche a été réglée or celle de la suppression est toujours en cours d'étude.

Cependant, malgré les nombreuses fonctions que nous avons créées, nous ne pensons pas avoir épuisé la question, et nous espérons sincèrement susciter d'autres fonctions pour mieux maîtriser les arbres.