

Ecole nationale des sciences appliquées
Année universitaire : 2021-2022
Module Complexité et structure de données

Rapport des Travaux pratiques



Nom et prénom: Abdeljalil FENNIRI

N° Apogée: 19000047

Niveau : 1^{er} Année cycle ingénieur – Génie Informatique

[Sommaire:](#)**Tableaux et temps d'exécution**

1. Exercice1 : Recherche du maximum.....	3
2. Exercice2 : Tri à Bulles.....	4
3. Exercice3 : Recherche dichotomique.....	5

Les listes chaînées (Liste des Personnes)

1. Insérer en tête de liste.....	7
2. Insérer en fin de liste.....	8
3. Retrait en tête de liste.....	9
4. Retrait en fin de liste.....	10
5. Retrait à partir d'un nom.....	11
6. Rechercher une personne à partir d'un nom.....	12

Les piles et les files

1. Les piles.....	15
2. Les files.....	17

Les arbres

1. Création de l'arbre généalogique.....	20
2. Création de l'arbre de l'expression arithmétique.....	20
3. Parcours Préfixé.....	20
4. Parcours Infixé.....	21
5. Parcours postfixé.....	21
6. Trouver Nœud	22
7. Taille	23
8. Hauteur.....	23
9. Parcours en largeur.....	24

TP n°1 : Tableaux et temps d'exécution

Exercice1 : Recherche du maximum :

Un programme en C qui implémente un algorithme de recherche du maximum dans un ensemble de n entiers. Voici ma version de code :

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  main()
5  {
6      printf("\nExercice1:RECHERCHE DU MAXIMUM\n");
7      //declaration des variables et demande à l'utilisateur de saisir un nbr
8      int n,i=0,*tab,max;
9      printf("\n Veuillez saisir un nombre: ");
10     scanf("%d",&n);
11     srand(time(NULL));
12     //allocation de mémoire pour la table et remplissage
13     tab=(int*)malloc(n*sizeof(int));
14     while (i<n)
15     {
16         *(tab+i)=rand();
17         i++;
18     }
19     //recherche de maximum de table
20     max=*tab;
21     for (i=1;i<n;i++)
22     {
23         if (*(tab+i)>max)
24         {
25             max=*(tab+i);
26         }
27     }
28     printf(" \n \n Le maximum est :    %d ",max,"\n\n");
29 }
30

```

En test le programme avec 2 cas n=5 et n=900 000 000

- Pour n=5:

```

Exercice1:RECHERCHE DU MAXIMUM

Veuillez saisir un nombre: 5

Le maximum est :    26202
Process returned 0 (0x0)   execution time : 3.794 s
Press any key to continue.

```

- Pour n=900 000 000 :

```

Exercice1:RECHERCHE DU MAXIMUM

Veuillez saisir un nombre: 900000000

Le maximum est :    32767
Process returned 0 (0x0)   execution time : 40.776 s
Press any key to continue.

```

Exercice2 : Tri à Bulles

Le tri à bulles s'effectue en $n-1$ étapes (pour un tableau de n éléments). A la i ème étape on balaye le tableau en partant de la fin jusqu'à la case i et à chaque fois que l'élément courant est plus petit que son prédécesseur, on échange leur position. Cette méthode a pour effet de faire remonter au fur et à mesure les bulles les plus légères vers la surface.

On implémente le tri à bulles en c :

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  main()
4  {
5      printf("\n\nExercice2: TRI A BULLES\n");
6      int n,i,dim,aide,*p,en_desordre=1;
7      printf("Veuillez saisir la dimension du tableau : ");
8      scanf("%d",&n);
9      p=(int*)malloc(n*sizeof(int));
10     //remplissage aleatoire du tableau
11     srand(time(NULL));
12     while (i<n){
13         *(p+i)=rand();
14         i++;
15     }
16     //affichage du tableau initial
17     printf("\n Tableau non triller: ");
18     for(i=0;i<n;i++){
19         printf("%d \t",*(p+i));
20     }
21     dim=n;
22     while(en_desordre){
23         en_desordre=0;
24         for(i=0;i<dim-1;i++){
25             if(*(p+i)>*(p+i+1)){
26                 //Permutation
27                 aide=*(p+i+1);
28                 *(p+i+1)=*(p+i);
29                 *(p+i)=aide;
30                 en_desordre=1;
31             }
32         }
33         dim--;
34     }
35     printf("\n voila le tableau triller : ");
36     for(i=0;i<n;i++){
37         printf("%d \t",*(p+i));
38     }
39 }
40
41
42

```

Après l'exécution :

```
Exercice2: TRI A BULLES
Veuillez saisir la dimension du tableau : 5

Tableau non trier: 27161      6651      15202      25018      13818
voila le tableau trier :  6651      13818      15202      25018      27161
Process returned 0 (0x0)   execution time : 4.946 s
Press any key to continue.
```

Exercice3 : Recherche dichotomique

Un algorithme de recherche pour trouver la position d'un élément dans un tableau trié.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  void afficher(int Tableau[],int N){
4      int i;
5      for (i=0;i<N;i++){
6          printf("%d\t",Tableau[i]);
7      }
8  }
9  int chercher_dich(int Tableau[],int N,int O){
10     int d=1,f=N;
11     int i;
12     int trouve=0;
13     do{
14         i=(d+f)/2;
15         if(Tableau[i]==O){
16             trouve=1;
17         }else if (Tableau[i]<O){
18             d=i+1;
19         }else{
20             f=i-1;
21         }
22     }
23     while(trouve!=1);
24     return i+1;
25 }
26 int main(){
27     int i,j,N,O;
28     int *Tableau=NULL;
29     int aide,emplacement;
30     printf("Veuillez saisir la taille du tableau:\t");
31     scanf("%d",&N);
32     Tableau= malloc(N*sizeof(int));
33     if(Tableau==NULL)
34     { printf("Echec d'allocation du tableau");
35       exit(0); }
36     for(i=0;i<N;i++)
37     { printf("\n A[%d]=\t",i);
38       scanf("%d",&Tableau[i]);}
39     for (i=0;i<N-1;i++){
40         for(j=0;j<N-(i+1);j++){
41             if(Tableau[j]>Tableau[j+1]){
42                 aide=Tableau[j];
43                 Tableau[j]=Tableau[j+1];
44                 Tableau[j+1]=aide;}}
45     printf("\n Votre tableau est:\t");
46     afficher(Tableau,N);
47     printf("\n entrer la cle : \t");
48     scanf("%d",&O);
49     emplacement = chercher_dich(Tableau,N,O);
50     printf("l'emplacement du nombre est dans la case: %d",emplacement);
51     free(Tableau);
52     return 0;
53 }
54
```

Après l'exécution :

```
Veillez saisir le taille du tableau: 10

A[0]= 1
A[1]= 7
A[2]= 8
A[3]= 9
A[4]= 13
A[5]= 15
A[6]= 17
A[7]= 22
A[8]= 30
A[9]= 31

Votre tableau est: 1 7 8 9 13 15 17 22 30 31
entrer le cle : 30
l'emplacement du nombre est dans la case: 9
Process returned 0 (0x0) execution time : 36.215 s
```

TP2 : Les listes chaînées (Liste des Personnes)

Une liste chaînée est une structure de donnée linéaire qui représente une collection ordonnée et de taille arbitraire d'éléments de même type. On a défini une liste par 5 éléments : le premier élément, le dernier élément, le nombre d'élément, l'élément courant et leur type.

Dans ce TP, on est intéressé de construire une liste de personne qui contient de nom prénom de personne. Faisons ça par l'implémentation 5 scripts

- liste.h : qui contient la déclaration de la structure liste et les prototypes des fonctions de gestion d'une liste
- liste.c : qui contient le corps des fonctions dont prototype est défini dans liste.h
- personne.h : qui contient la structure et les prototypes des fonctions de gestion d'une personne
- personne.c : qui contient le corps des fonctions dont prototype est défini dans personne.h
- main.c : le programme principal

1. Insérer en tête de liste :

```
void insererEnTeteDeListe (Liste* li, Objet* objet){
    Element* nouveau = creerElement();
    nouveau->reference = objet;
    nouveau->suivant = li->premier;
    li->premier = nouveau;
    if(li->dernier == nullptr) li->dernier = nouveau;
    li->nbElt++;
}
```

Déclaration de la fonction insérer en tête de liste

```
GERTION D'UNE LISTE DE PERSONNE
0 - Fin
1 - Insertion    en tete de liste
2 - Insertion    en fin de liste
3 - Retrait      en tete de liste
4 - Retrait      en fin de liste
5 - Retrait      d'un element a partir de son nom
6 - Parcours     de la liste
7 - Recherche    d'un element a partie de son nom
8 - Destruction  de la liste

Votre choix ? 1

Nom de la personne ^_< cr^_<er ? Abdeljalil
Pr^_<nom de la personne ^_< cr^_<er ?Fenniri
l'element Abdeljalil Fenniri inserer en tete de liste
```


2. Insérer en fin de liste :

Pour insérer en fin de liste, il faut d'abord savoir comment insérer après un élément, ça nous permet d'insérer après le dernier élément de liste.

```
static void insererApres (Liste* li , Element* precedent , Objet* objet){
    if(precedent == nullptr){
        insererEnTeteDeListe(li,objet);
    }else{
        Element* nouveau = creerElement();
        nouveau->reference = objet;
        nouveau->suivant = precedent->suivant;
        precedent->suivant=nouveau;
        if(precedent == li->dernier) li->dernier = nouveau;
        li->nbElt++;
    }
}

void insererEnFinDeListe(Liste* li , Objet* objet){
    insererApres (li,li->dernier,objet);
}
```

Déclaration de la fonction insérer en fin de liste et insérer en Après

```
GERTION D'UNE LISTE DE PERSONNE

0 - Fin
1 - Insertion   en tete de liste
2 - Insertion   en fin de liste
3 - Retrait     en tete de liste
4 - Retrait     en fin de liste
5 - Retrait     d'un element a partir de son nom
6 - Parcours    de la liste
7 - Recherche   d'un element a partie de son nom
8 - Destruction de la liste

Votre choix ? 2

Nom de la personne ^1$ cr^1$er ?Nom1
Pr^1$nom de la personne ^1$ cr^1$er ?Prenom1
l'element Nom1 Prenom1 inserer en fin de liste
```

En fait un parcours de liste pour savoir si le code fonctionne correctement

```
Votre choix ? 6

Abdeljalil Fenniri

Nom1 Prenom1
```


3. Retrait en tête de liste :

```
Nom2 Prenom2
Abdeljalil Fenniri
Nom1 Prenom1
Nom3 Prenom2
```

J'ai rempli la liste par des éléments pour retrait en tête et en fin de liste

```
Objet* extraireEnTeteDeListe (Liste* li){
    Element* extrait = li->premier;
    if(!listeVide(li)){
        li->premier=li->premier->suivant;
        if(li->premier==NULL) li->dernier=NULL;
        li->nbElt--;
    }
    return extrait != NULL ? extrait->reference : NULL ;
}
```

Déclaration de la fonction Extrait en tête de liste

```
GERTION D'UNE LISTE DE PERSONNE
0 - Fin
1 - Insertion    en tete de liste
2 - Insertion    en fin de liste
3 - Retrait      en tete de liste
4 - Retrait      en fin de liste
5 - Retrait      d'un element a partir de son nom
6 - Parcours     de la liste
7 - Recherche    d'un element a partie de son nom
8 - Destruction  de la liste

Votre choix ? 3
Element Nom2 Prenom2 extrait en tete de liste
```

L'extraction l'élément Nom2 Prenom2 de la tête de liste

On affiche la nouvelle liste

```
Abdeljalil Fenniri
Nom1 Prenom1
Nom3 Prenom2
```

4. Retrait en fin de liste :

Du même liste on extrait le dernier élément de liste par la fonction `extraireEnFinListe()` mais avant on doit déclarer la fonction `extraire` après pour qu'on puisse extraire l'élément qui vient après le dernier élément dans le cas où la liste n'est pas vide, si on la liste contient un seul élément alors on va extraire en tête de liste

```
static Objet* extraireApres(Liste* li, Element* precedent){
    if(precedent == NULL){
        return extraireEnTeteDeListe(li);
    }else{
        Element* extrait = precedent->suitant;
        if(extrait != NULL){
            precedent->suitant = extrait->suitant;
            if(extrait == li->dernier) li->dernier = precedent;
            li->nbElt--;
        }
        return extrait != NULL ? extrait->reference : NULL;
    }
}

Objet* extraireEnFinDeListe(Liste* li){
    Objet* extrait;
    if(listeVide(li)){
        extrait = NULL;
    }else if(li->premier == li->dernier){
        extrait = extraireEnTeteDeListe(li);
    }else{
        Element* ptc = li->premier;
        while (ptc->suitant != li->dernier) ptc = ptc->suitant;
        extrait = extraireApres(li, ptc);
    }
    return extrait;
}
```

Déclaration de la fonction Extraire en Fin de liste et Extraire Après

```
GERTION D'UNE LISTE DE PERSONNE

0 - Fin
1 - Insertion    en tete de liste
2 - Insertion    en fin de liste
3 - Retrait      en tete de liste
4 - Retrait      en fin de liste
5 - Retrait      d'un element a partir de son nom
6 - Parcours     de la liste
7 - Recherche    d'un element a partie de son nom
8 - Destruction  de la liste

Votre choix ? 4

Element Nom3 Prenom2 extrait en Fin de liste
```

5. Retrait à partir d'un nom :

Il nous reste 2 élément de la liste « Abdeljalil Fenniri » et « Nom1 Prenom1 », maintenant on va extraire l'élément « Nom1 Prenom1 » d'après son nom avec la fonction `extraireUnobjet()`

```

booléen extraireUnObjet(Liste* li, Objet* objet){
    Element* precedent = NULL;
    Element* ptc       = NULL;
    booléen trouve = faux;

    ouvrirListe(li);
    while (!finListe(li) && !trouve){
        precedent=ptc;
        ptc =elementCourant(li);
        trouve =(ptc->reference==objet)? vrai : faux;
    }
    if(!trouve) return faux;
    Objet* extrait = extraireApres(li,precedent);
    return vrai;
}

```

Déclaration de la fonction Extraire en Fin de liste et Extraire Après

```

GERTION D'UNE LISTE DE PERSONNE

0 - Fin
1 - Insertion   en tete de liste
2 - Insertion   en fin de liste
3 - Retrait     en tete de liste
4 - Retrait     en fin de liste
5 - Retrait     d'un element a partir de son nom
6 - Parcours    de la liste
7 - Recherche   d'un element a partie de son nom
8 - Destruction de la liste

Votre choix ? 5

Nom de la personne ^\4 extraire ?Nom1
Element Nom1 Prenom1 extrait de la liste

```

Et on fait un parcours de liste et il nous reste que l'élément « Abdeljalil Fenniri »

```

Votre choix ? 6
Abdeljalil Fenniri

```

6. Rechercher une personne à partir d'un nom :

On va chercher un élément de liste à partir la fonction chercherObjet()

```
Objet* chercherUnObjet(Liste* li, Objet* objetCherche){
    boolean trouve=false;
    Objet* objet;
    ouvrirListe(li);
    while(!finListe(li) && !trouve){
        objet = objetCourant(li);
        trouve = li->comparer(objetCherche, objet)==0;
    }
    return trouve ? objet : NULL;
}
```

Déclaration de la fonction Extrait en Fin de liste et Extrait Après

```
GERTION D'UNE LISTE DE PERSONNE

0 - Fin
1 - Insertion    en tete de liste
2 - Insertion    en fin de liste
3 - Retrait      en tete de liste
4 - Retrait      en fin de liste
5 - Retrait      d'un element a partir de son nom
6 - Parcours     de la liste
7 - Recherche    d'un element a partie de son nom
8 - Destruction  de la liste

Votre choix ? 7

Nom de la personne recherchée ?Abdeljalil
Abdeljalil Fenniri trouvée dans la liste
```

Et en va détruire la liste utilisant la fonction detruireliste()

```
void detruireListe(Liste* li){
    ouvrirListe(li);
    while(!finListe(li)){
        Element* ptc = elementCourant(li);
        free(ptc);
    }
    initListe(li, 0, NULL, NULL);
}
```


L'affichage et la recherche de tous les éléments qu'on a faits ci-dessus c'est grâce à des fonctions qu'on a implémenté dans le fichier `personne.c`

La création de l'élément `Personne` d'abord ce fait pour qu'on définit le type des éléments structurée dans la liste, on fait une allocation dynamique de mémoire pour un élément de type « `Personne` » qui est définie dans le fichier `personne.h` puis on crée l'élément `Personne`

```
Personne* creerPersonne(char* nom, char* prenom) {
    Personne* p = new Personne();
    strcpy(p->nom, nom);
    strcpy(p->prenom, prenom);
    return p;
}
```

L'affichage des éléments ce fait par la fonction `ecrirePersonne()` :

```
char* ecrirePersonne(Objet* objet) {
    Personne* p = (Personne*) objet;
    char* output = (char*) new Personne();

    snprintf(output, sizeof(Personne), "%s %s\n", p->nom, p->prenom);
    return output;
}
```

Dans la liste des personnes on a donnée à la fonction `li->comparer = comparerPersonne` pour qu'on puisse l'utiliser dans la recherche d'un personne du liste alors cette fonction est implémente dans le fichier `personne.c` comme ceci :

```
int comparerPersonne(Objet* objet1, Objet* objet2) {
    Personne* p1 = (Personne*) objet1;
    Personne* p2 = (Personne*) objet2;
    return strcmp(p1->nom, p2->nom);
}
```

Il y a des fonctions qu'on a les utiliser aussi comme `listerListe()` dans la partir du parcours de liste qui utilise lui-même la fonction `Objetcourant()`

```
static Element* elementCourant(Liste* li) {
    Element* ptc = li->courant;
    if(li->courant != NULL) {
        li->courant = li->courant->suivant;
    }
    return ptc;
}

Objet* objetCourant(Liste* li) {
    Element* ptc = elementCourant(li);
    return ptc==NULL ? NULL : ptc->reference;
}

void listerListe(Liste* li) {
    ouvrirListe(li);
    while(!finListe(li)) {
        Objet* objet = objetCourant(li);
        printf("%s\n", li->afficher(objet));
    }
}
```

Liste des entiers

TP3: Les piles et les files

Les piles et les files sont deux variantes des listes chaînées qui permettent de contrôler la manière dont sont ajoutés les nouveaux éléments. Cette fois, on ne va plus insérer de nouveaux éléments au milieu de la liste, mais seulement au début ou à la fin.

Les piles permettent de stocker des données au fur et à mesure, les unes au-dessus des autres pour pouvoir les récupérer plus tard. On appelle cette méthode de stockage LIFO (Last In First Out). On veut alors construire une pile, empiler des éléments, et dépiler la pile en implémentant ce programme :

```
int main() {
    printf("GESTION D'UNE PILE DE PERSONNES \n");

    Pile* pile1 = creerPile();

    empiler(pile1, creerPersonne("Alaoui", "Karim"));
    empiler(pile1, creerPersonne("Alaoui", "Ali"));
    empiler(pile1, creerPersonne("Alaoui", "Sarah"));

    printf("Valeurs dans la pile : du sommet vers la base\n");
    listerPile(pile1, écrirePersonne);
    printf("\nValeur dépilée: ");
    Personne* p = (Personne*) depiler(pile1);
    if(p != NULL) écrirePersonne(p);
    return 0;
}
```

D'abord on définit les fonctions (empiler, dépiler..) en les implémentant dans un fichier `pile.c` et en définissant leur protocole dans un fichier `pile.h`

```
typedef Liste Pile;

Pile* creerPile();
booléen pileVide(Pile* p);
void empiler(Pile* p, Objet* objet);
Objet* depiler(Pile* p);
void listerPile(Pile* p, char* (*f)(Objet*));
void detruirePile(Pile* p);
```

Fichier : `pile.h`

```

Pile* creerPile() {
    return creerListe(0, écrirePersonne, comparerPersonne);
}

bool pileVide(Pile* p) {
    return listeVide(p);
}

void empiler(Pile* p, Objet* objet) {
    insererEnTeteDeListe(p, objet);
}

Objet* depiler(Pile* p) {
    if (pileVide(p)) {
        return NULL;
    } else {
        return extraireEnTeteDeListe(p);
    }
}

void listerPile(Pile* p, char* (*f)(Objet*)) {
    listerListe(p);
}

void detruirePile(Pile* p) {
    detruireListe(p);
}

```

Comme en avant déjà mentionner que les pile sont des variantes des listes chaînées alors en doit appeler tous les fonctions de liste.c et puisqu'on veut empiler des élément de type « personne » on doit aussi appelée les fonctions de création, affichage et comparaison défini dans le fichier personne.c dans le fichier pile.c

Alors après l'exécution on trouve :

```

GESTION D'UNR PILE DE PERSONNES
Valeurs dans la pile : du sommet vers la base
Alaoui Sarah

Alaoui Ali

Alaoui Karim

Valeur d'úpilúe: Alaoui Sarah
Process returned 0 (0x0)   execution time : 0.067 s
Press any key to continue.

```

LES FILES

Les Files permettent de stocker des données au fur et à mesure, les unes au-dessus des autres pour pouvoir les récupérer plus tard. On appelle cette méthode de stockage FIFO (First In First Out). On veut alors construire une pile, enfiler des éléments, et défiler la file en implémentant ce programme :

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4  #include"liste.h"
5  #include"file.h"
6  #include"personne.h"
7
8  int main() {
9      printf("GESTION D'UNR File DE PERSONNES \n");
10
11      File* File1 = creerFile();
12
13      emFiler(File1, creerPersonne("Alaoui", "Karim"));
14      emFiler(File1, creerPersonne("Alaoui", "Ali"));
15      emFiler(File1, creerPersonne("Alaoui", "Sarah"));
16
17      printf("Valeurs dans la File : du sommet vers la base\n");
18      listerFile(File1, écrirePersonne);
19      printf("\nValeur défilée: ");
20      Personne* p = (Personne*) deFiler (File1);
21      if(p!=NULL) écrirePersonne(p);
22      return 0;
23  }
24

```

D'abord on définit les fonctions (enfiler, défiler..) en les implémentant dans un fichier flie.c et en définissant leur protocole dans un fichier file.h

```

1  typedef Liste File;
2
3  File* creerFile ();
4  boolean FileVide (File* p);
5  void emFiler (File* p, Objet* objet);
6  Objet* deFiler (File* p);
7  void listerFile(File* p , char* (*f) (Objet*));
8  void detruireFile(File* p);
9

```

```

8  File* creerFile() {
9      return creerListe(0, écrirePersonne, comparerPersonne);
10 }
11 bool FileVide(File* p) {
12     return listeVide(p);
13 }
14 void emFiler(File* p, Objet* objet) {
15     insererEnFinDeListe(p, objet);
16 }
17 Objet* deFiler(File* p) {
18     if(FileVide(p)) {
19         return NULL;
20     } else {
21         return extraireEnTeteDeListe(p);
22     }
23 }
24 void listerFile(File* p, char* (*f)(Objet*)) {
25     listerListe(p);
26 }
27 void detruireFile(File* p) {
28     detruireListe(p);
29 }
30

```

Alors après l'exécution on trouve :

```

GESTION D'UNR File DE PERSONNES
Valeurs dans la File : du sommet vers la base
Alaoui Karim

Alaoui Ali

Alaoui Sarah

Valeur d'upilue: Alaoui Karim
Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.

```

TP4 : Les arbres

Un arbre est une structure de données composée de nœuds reliés entre eux par des branches, selon une organisation hiérarchique, à partir d'un nœud racine

Dans ce TP, on est intéressé de construire 2 arbres - Arbre généalogique et un arbre de l'expression arithmétique qui contient de nom prénom de personne. Faisons ça par l'implémentation 3 fichiers

- arbre.h : qui contient la déclaration du type arbre et les prototypes des fonctions de gestion d'un arbre.
- arbre.cpp : qui contient le corps des fonctions dont le prototype est défini dans arbre.h.
- mainarbre.cpp : qui est le programme principal des arbres.

Dans le main, on va afficher le menu suivant :

```
GESTION D'ARBRES
ARBRES BINAIRES
0 - Fin du programme
1 - Creation de l'arbre genealogique
2 - Creation de l'arbre de l'expression arithmetique
3 - Parcours prefixe
4 - Parcours infixe
5 - Parcours postfixe
6 - Trouver Noeud
7 - Taille
8 - Hauteur
9 - Parcours en Largeur
Votre choix ?
```

On commence par la création des 2 arbres puis on va faire un parcours préfixe, infixe et postfixé de chaque arbre en utilisant les fonctions de différent type de parcours défini dans arbre.cpp

1.2. Création des 2 arbres :

```
Votre choix ? 1
arbre genealogique creer
```

```
Votre choix ? 2
arbre arithmetique creer
```

3. Parcours Préfixé

A l'aide de la fonction « préfixe » défini dans le fichier arbre.cpp, on va parcourir les 2 arbres un parcours préfixe

```
void prefixe(Noeud* racine, char* (*affichier) (Objet*))
{
    if(racine!=NULL) {
        printf("%s ", affichier(racine->reference));
        prefixe(racine->gauche, affichier);
        prefixe(racine->droite, affichier);
    }
}
```

Déclaration de la fonction préfixe

```
Votre choix ? 3

Parcours prefixe de l'arbre genealogique :
Samir Kamal Yassine Hind Yasmine Sarah Karim

Parcours prefixe de l'arbre arithmetique :
- * + a b - c d e
```

Le parcour prefixe de l'arbre genealogique nous a donné :


Samir Kamal Yassine Hind Yasmine Sarah Karim

Le parcours prefixe de l'arbre arithmetique nous a donné :

- * + a b - c d e

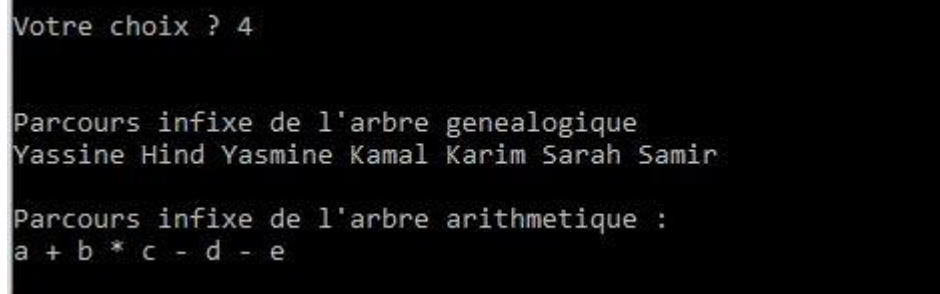
4. Parcours infixe:

A l'aide de la fonction « infixe » défini dans le fichier arbre.cpp, on va parcourir les 2 arbres un parcours infixe



```
void infixe (Noeud* racine, char* (*affichier) (Objet*))
{
    if (racine != NULL) {
        infixe (racine->gauche, affichier);
        printf ("%s ", affichier (racine->reference));
        infixe (racine->droite, affichier);
    }
}
```

Déclaration de la fonction prefixe



```
Votre choix ? 4

Parcours infixe de l'arbre genealogique
Yassine Hind Yasmine Kamal Karim Sarah Samir

Parcours infixe de l'arbre arithmetique :
a + b * c - d - e
```

5. Parcours postfixé

A l'aide de la fonction « postfixe » défini dans le fichier arbre.cpp, on va parcourir les 2 arbres un parcours postfixe

```
void postfixe (Noeud* racine, char* (*affichier) (Objet*))
{
    if (racine != NULL) {
        postfixe (racine->gauche, affichier);
        postfixe (racine->droite, affichier);
        printf ("%s ", affichier (racine->reference));
    }
}
```

Déclaration de la fonction postfixe

```
Votre choix ? 5

Parcours postfixe de l'arbre genealogique
Yasmine Hind Yassine Karim Sarah Kamal Samir

Parcours postfixe de l'arbre arithmetique :
a b + c d - * e -
```

6. Trouver nœuds

Maintenant on va chercher un nœud de l'arbre à l'aide de la fonction `trouvernoeud()` et puis l'afficher et afficher l'arbre qui contient ce nœud

```
Noeud* trouverNoeud(Noeud* racine, Objet* objet, int (*comparer) (Objet*, Objet*)) {
    Noeud* pNom;
    if(racine==NULL) {
        pNom=NULL;
    }
    else if(comparer(racine->reference, objet)==0) {
        pNom=racine;
    }
    else {
        pNom = trouverNoeud(racine->gauche, objet, comparer);
        if(pNom==NULL) pNom = trouverNoeud(racine->droite, objet, comparer);
    }
    return pNom;
}
```

Déclaration de la fonction `trouvernoeud()`

On cherche le nom « Hind » :

```
Votre choix ? 6

Entrer le noeud a chercher: Hind
L'objet Hind est trouve dans l'arbre generatrice
```

On cherche l'objet « a » :

```
Votre choix ? 6

Entrer le noeud a chercher: a
L'objet a est trouve dans l'arbre arithmetique
```

Et puis on cherche un objet qui n'existe pas dans l'arbre pour savoir si la fonction est bien implémentée

```
Votre choix ? 6

Entrer le noeud a chercher: Abdeljalil
L'objet ne se trouve pas dans l'arbre
```

7. Taille de l'arbre:

```
int taille(Noeud* racine)
{
    if(racine==NULL) {
        return 0;
    }
    else{
        return 1+ taille(racine->droite)+taille(racine->gauche);
    }
}
```

Déclaration de la fonction taille

```
Votre choix ? 7
La taille de l'arbre genealogique est: 7
La taille de l'arbre arithmetique est: 9
```

8. Hauteur:

```
int hauteur(Noeud* racine) {
    if (racine == NULL) {
        return 0;
    }
    else{
        return 1+max(hauteur(racine->gauche), hauteur(racine->droite));
    }
}
```

Déclaration de la fonction hauteur

```
Votre choix ? 8
La hauteur de l'arbre genealogique est: 5
La hauteur de l'arbre arithmetique est: 4
```

9. Parcours en largeur

```
void enLargeur(Noeud* racine, char* (*affichier)(Objet*))
{
    Liste* li = creerListe(0, affichierChar, NULL);
    insererEnFinDeListe(li, racine);
    while(!listeVide(li)) {
        Noeud* extrait = (Noeud*) extraireEnTeteDeListe(li);
        printf("%s ", affichierChar(extrait->reference));
        if(extrait->gauche != NULL) insererEnFinDeListe(li, extrait->gauche);
        if(extrait->droite != NULL) insererEnFinDeListe(li, extrait->droite);
    }
}
```

Déclaration de la fonction en largeur

```
Votre choix ? 9

Parcours en largeur de l'arbre genealogique est:
Samir Kamal Sarah Karim Yassine Hind Yasmine

Parcours en largeur de l'arbre arithmetique est:
- e * - d c + b a
```