Université de Yaoundé 1
Ecole Nationale Supérieure Polytechnique

University of Yaounde 1
National Advanced School of Engineering

EXPOSE D'INFORMATIQUE 4
SOUS-GROUPE A2 GROUPE N°2

LISTE DES PARTICIPANTS :

- DJAHAPPI NZEMBIA ELISEE (Chef de groupe)
- BILA ELIE DESIRE
- DIFFOUO TIYO NESLY
- DJANEA DOOH WILLIAM
- DINGUE WANDJI SERENA
- DZUGAING FOTSI

Sous la coordination de : Mr TAGOUDJEU
Mme WAMBA

ENONCE DU TRAVAIL DEMANDE

Il nous a été proposé de résoudre les exercices 3 et 5 dont les énoncés sont les suivants :

Exercice 3 : Recherche du maximum d'une suite finie

Ecrire un programme en L.E.A qui détermine le maximum d'une suite finie de nombres réels lus dans un fichier sur le disque.

N.B : *Pour la recherche du maximum proprement dite, proposer une* ***approche non récursive*** *et une* ***approche récursive***

Exercice 5 : Recherche des 2 plus petits éléments d'une suite finie

Ecrire un programme en L.E.A qui détermine les 2 plus petits éléments (au sens de la relation $\leqslant$) d'une suite finie de nombres réels lus dans un fichier sur le disque.

N.B : *Pour la recherche du maximum proprement dite, proposer une* ***approche non récursive*** *et une* ***approche récursive***

N.B : Dans les programmes de résolution qui seront présentés ci-dessous :

1. Nous n'avons pas modifié la disposition des éléments de la suite qui nous est donnée. Et donc :
2. Nous avons travailler avec les indices de sorte que :
3. Il suffit de déterminer par les intructions de style `valeur = Suite.val[indice]` les valeurs réellement cherchées.

Ceci a été motivé par le fait que nous ne voulions pas d'une quelconque façon **toucher** aux données de l'utilisateur gardant ainsi une certaine **confidentialité**.

LECTURE D'UNE SUITE CONTENUE DANS UN FICHIER

Bien que le professeur en cours magistrale a dit, pour le cadre des TDs, de considérer comme acquise la procédure permettant de lire une suite de réels contenue dans un fichier, nous la donnons ici à titre de rappel.

```
procedure Lecture_Suite(chemin : chaîne, var S : Suite)

/*Objectifs: Récuperer les termes d'une suite finie
             contenue dans un fichier sur le disque
             et les stocker dans une structure de type Suite. */

/*Entrée: chemin: chemin d'accès absolu au fichier
                  sur le disque
          S: Structure de type suite, qui contiendra les termes
             de la suite et, telle que :
             S.taille = nbre de termes de la suite et
             S.val[i] = i-ème terme de la suite. */

var Fic: Fichier_Texte;
    k : entier;

Debut
    Attacher(Fic, chemin);
    OuvrirLecture(Fic)
    k <-- 0;
    Tantque (non EOF(Fic)) faire
        k <-- k+1;
        Lire(Fic, S.val[k]);
    Ftantque
    S.taille = k
    Fermer(Fic)
Fin
```

Nous présentons maintenant une implémentation en python de ce programme.

```python
def Lecture_Suite(chemin : str, S : Suite) -> Suite:

    Fic = open(chemin, "r")

    lines = Fic.readlines()

    for line in lines :
        """
        Les lignes sont terminées par le caractère '\n'
        Donc il faut le remplacer par un caractére vide
        afin de pouvoir faire un casting pour avoir la
```

```
        valeur en float.
        """
        if "\n" in line :
            line = line.replace("\n", '')

        line = float(line)

        S.val.append(line)

    S.taille = len(lines)

    Fic.close()
    return S
```

RESOLUTION DE L'EXERCICE 3

Commençons par l'approche itérative.

• Approche itérative

Pour trouver le maximum recherché, nous allons considérer la taille de la suite. Nous renverons -1 si la suite est vide. De manière itérative, nous comparerons le maximum à la valeur courante pour en déduire le maximum global de la suite.

```
Fonction ResearchMax_ite(S : Suite) : entier
  /* Objectifs : Déterminer le maximum des termes d'une suite finie
                  de façon itérative */
  /* Entrée    : S : Suite extraire du fichier */
  /* Sortie    : indMax : indice de l'élément maximum de la suite S */

  Var indMax : entier;

  Debut :

    Si S.taille == 0 alors
      Si S.val[1] alors
        Retourner S.val[1];
      Sinon
        Retourner(-1);
      FSi
    Sinon
      indMax = 1;
      Pour k = 2(1)S.taille faire
        Si S.val[indMax] < S.val[k] alors
          indMax <-- k;
        FSi
      FPour
      Retourner(indMax) ;
Fin
```

Nous présentons maintenant une implémentation en python de ce programme.
Commençons par définir la structure suite qui nous aidera d'ici jusqu'à la fin de ce travail.

```python
# definition de la structure suite

class Suite :
  """
  On crée la suite en lui passant en paramètre un tableau
  qui est le coeur même de la suite.
  """
  def __init__(self, array : list) :
      self.val = array
      self.taille = len(array)
```

Ensuite vient le programme python pour la résolution itérative.

```
# approche iterative

  def ResearchMax_ite(S : Suite) -> int :
      """
      On compare la valeur de la suite à l'indice courant
      à cette à l'indice préalablement sélectionné comme celui
      du max de sorte à justement le mettre à jour si possible.
      """
      if S.taille == 0 :
          return -1
      elif S.taille == 1 :
          return 1
      else :
          indMax = 0
          for k in range(1, S.taille) : # attention en python l'indiçage commence à 0 et
                                                    non à 1 comme en L.E.A
              if S.val[indMax] < S.val[k] :
                  indMax = k
          return indMax
```

• Approche récursive

Ici, nous allons déterminer de façon récursive le maximum des sous-suites de la suite donnée. Lesquelles contiennent sucessivement 2, 3, ... jusqu'à tous les éléments de la suite. Pour ce faire, nous allons conserver graduellement la valeur du maximum de la sous-suite déjà parcourue tout en le comparant aux valeurs restantes de la suite. Ce qui augmentera donc la taille de la sous-suite déjà parcourue. Résultat, on obtiendra au final le maximum global de la suite.

```
/* Fonction principale

fonction ResearchMax_rec(S: Suite) : entier
  Debut
    Si S.taille == 0 alors
      Si S.val[1] alors
        Retourner S.val[1];
      Sinon
        Retourner(-1);
      FSi
    Sinon
      Retourner(Research(S, 1, 2));
    FSi
  Fin

/* fonction auxiliare : celle qui est recursive

fonction Research(S : Suite, n1 : entier, n2 : entier) : entier
  /* Objectifs : déterminer le plus grand nombre entre ceux d'indices n1 et n2
                 puis de façon récursive avancer dans le tableau */
  /* Entrées : S: suite extraite du fichier
               n1: indice du plus grand des éléments déjà analysés
```

```
              n2: indice cherchant le plus grand des éléments non analysés
                  i.e de ceux qui restent dans le tableau */
/* Sorties : indMax : qui sera l'indice du maximum de la suite */

var indMax : entier

Debut
  Si n2 <= S.taille alors /* On est à la fn du tableau */

    /* on cherche maintenant qui est le plus grand entre les nombres aux
       indices n1 et n2 */

    Si S.val[n1] > S.val[n2] alors
      Retourner n1
    Sinon
      Retourner n2
    FSi
  Sinon /* On est encore en cours de route */

    /* On définit maintenant la condition d'arrêt de la recursion */

    Si n2 <= S.taille alors
      Si S.val[n2] < S.val[n1] alors

        /* l'élément devant n'est pas plus grand que le max
           de ce qui sont derrière donc on avance n2 en conservant
           le max de ce qui sont derrière car il est le max de
           tous à l'heure actuelle des choses */

        indMax <-- Research(S, n1, n2 + 1);

      Sinon

        /*l'élément de devant est plus grand que le max de ceux qui
          sont derrière, il devient donc le max et on avance dans
          le parcours du tableau */

        indMax <-- Research(S, n2, n2+1);
      FSi
    FSi
  FSi
  Retourner indMax
Fin
```

Nous présentons maintenant une implémentation en python de ce programme.

```python
# fonction principale

def ResearchMax_Rec(S : Suite) -> int :
    if S.taille == 0 :
        return -1
    elif S.taille == 1 :
        return 1
    else :
        return Research(S, 0, 1) # attention en python l'indiçage commence à 0 et non à
                                                        1 comme en L.E.A

# fonction auxiliaire

def Research(S: Suite, n1 : int , n2 : int) -> int :
    """
    comme dans l'approche itérative, n1 contient l'indice du max déjà
    trouvé et n2 est l'indice qui avance dans le tableau en cherchant
    s'il existe un autre max de sorte de mettre à jour n1.
    """

    if n2 == S.taille -1 : # attention en python le dernier indice du tableau est S.
                                                    taille - 1
        if S.val[n1] <= S.val[n1] :
            return n2
        else :
            return n1
    else :
        if n2 < S.taille -1 :
            if S.val[n2] <= S.val[n1] :
                indMax = Research(S, n1, n2 + 1)
            else :
                indMax = Research(S, n2, n2 + 1)

    return indMax
```

RESOLUTION DE L'EXERCICE 5

De même, commençons par l'approche itérative.

• Approche itérative

Pour trouver les 2 plus petits éléments recherchés, nous allons considérer la taille de la suite. Nous considérerons que la suite à au moins 2 valeurs. De manière itérative nous comparerons la valeur courante aux 2 plus petits éléments déjà determinés, pour en déduire les minimums globaux de la suite.

N.B : Nous utiliserons une procédure car nous avons à renvoyer **2** valeurs que sont les minimums de la suite.

```
procédure Research2mostSmall_ite(S : Suite, var inf1 : entier, var inf2 : entier)

  /*Objectifs: déterminer les 2 plus petits éléments de la suite
               par un simple parcours.*/
  /*Entrée: S : suite extraite du fichier
            inf1: qui contiendra plus petit élément au fur et à mesure du
                  parcours de la suite
            inf2 : qui contiendra le 2-ème plus petit, i.e : inf1 < inf2 */

  /*A la fin du programme, inf1 et inf2 seront les deux plus petits éléments
    de la suite avec Suite.val[inf1] < Suite.val[inf2] */

  Debut
    inf1 <-- 1, inf2 <-- 1;
    Si S.taille == 2 alors

    /* On compare juste les 2 valeurs de la suite */

      Si S.val[1] > S.val[2] alors
        inf1 <-- 2, inf2 <-- 1;
      Sinon
        inf1 <-- 1, inf2 <-- 2;
      FSi
    Sinon
      Pour k = 1(1)S.taille faire
        Si S.val[inf2] > S.val[k] alors
          Si S.val[inf1] > S.val[k] alors

          /* k est donc l'indice du plus petit, on arrange alors */

            inf2 <-- inf1;
            inf1 <-- k;

          Sinon

          /* k est l'indice du 2-ème plus petit, on arrange alors */
```

```
          inf2 <-- k;
        FSi
      FSi /*en effet, k n'aura pas à changer l'ordre
            vu qu'il est plus grand que inf1 et inf2 */
    FPour
  FSi
```

Nous présentons maintenant une implémentation en python de ce programme.

N.B : *Il y a une petite nuance avec le programme en python. En effet : inf1 et inf2 tous deux au début*tous deux au début *initialisées* tous deux au début *du tableau mais plutôt inf1 au debut et inf2 à la suite de inf1.*

Conséquence *, on va d'abord comparer les valeurs à ces indices là afin de savoir qui est réellement le plus petit.*

```python
    # approche iterative

def Research2mostSmall_ite(S : Suite) -> tuple :
  """
  On va juste comparer la valeur de la suite à l'indice
  courant pour déduire éventuellement si elle est plus
  petites qu'au moins un des deux plus petits déjà trouvés.
  Et on dicutera à propos de cette comparaison.
  """
    inf1, inf2 = 0, 1
    if S.taille == 2 :
        if S.val[0] <= S.val[1] :
            inf1 = 0
            inf2 = 1
        else :
            inf1 = 1
            inf2 = 0
    else :

        if S.val[inf1] > S.val[inf2] :
            inf1, inf2 = inf2, inf1

        for k in range(2, S.taille) :
            if S.val[inf2] > S.val[k] :
                inf2 = k
                if S.val[inf1] > S.val[inf2] :
                    inf1, inf2 = inf2, inf1

    return (inf1, inf2)
```

• Approche récursive

Pour l'approche recursive, nous avons procéder par détermination successive du minimum de la suite puis de la suite privée du minimum déjà trouvé. Ainsi de suite ... puis nous prenons les 2 premiers minimums trouvés qui représentent en réalité les plus petits éléments de la suite.

N.B : Une fois de plus, nous utiliserons une procédure car nous avons à renvoyer **2** valeurs que sont les minimums de la suite.

```
/*Procédure principale */
/* création d'une structure auxiliaire */
Type auxiliaire :
  Trier(N) :
    T1 : [-1, -1, ..., -1]; /* de taille N; qui contiendra les indices
                            des plus petits éléments dans l'ordre croissant
                            de ces éléments là.*/
    T2 : [1, 2, 3, ..., N]; /* de taille N; qui contiendra les indices
                             des valeurs du tableau et se remplira de -1
                             au fur et à mesure que les éléments de
                             la suite ont déjà été choisies de sorte
                             de ne plus compter dans les comparaisons
                             ces éléments là. */
    ecrase1 = 1  /* 1 est la valeur par défaut */

procédure Research2mostSmall_rec(S : Suite, var inf1 : entier, var inf2 : entier)

  /*Objectifs: déterminer les deux plus petits éléments de la suite
               par extraction successive du plus petit élément
               et ce, au travers de la recursion. */
  /*Entrée: S: Suite extraite du fichier */
  /* Sortie: indices inf1 et inf2 des deux plus petis éléments de la suite
             avec Suite.val[inf1] < Suite.val[inf2] */

  var T : Trier(S.taille)

  Debut

    /*On va déterminer les plus petits éléments qui seront
    dans T1 comme souligné plus haut */

    ResearchSmall(S, T)

    /* Et on prends les deux plus petits */

    inf1 <-- T1[1], inf2 <-- T1[2];

    Fin


/*Procédure auxiliaire */

procédure ResearchSmall(S : Suite, var T : Trier)

  /*Objectifs: déterminer le plus petit élément de la suite S
             et à partir des modificationsde de T */
```

```
/*Entrée : S: suite à travailler, T Type auxiliaire pour gerer les
            indices de S */

/*Sortie: cette procédure modifie la structure Trier T */

var indMin : entier; /*qui contiendra le minimum courant
                     de la suite S */
    indDebut : entier; /* variable auxiliare */
    temp : entier; /* variable auxiliaire */

Debut

  /*ecrase1 a donc cette 1-ère valeur où il y a -1 dans T1
  on viendra à cette endroit de T1 pour mettre le prochain
  plus petit élément */

  /*observons maintenant la condition suffisante d'arrêt
  de la recursion. C'est dés qu'on a déjà les 2 plus petits */

  Si T.ecrase1 == 3 alors

    break /*Fin du programme */

  Fsi
  /*On est encore en cours de processus, il faut donc
  déterminer le prochain plus petit élément de la suite S
  ou plutôt son indice à proprement parlé */

  indMin <-- 1;
  Pour k=1(1)S.taille faire
    Si T.T2[k] != -1 faire
      min <-- T.T2[k];
      indDebut <--- k;
    FSi
  FPour

  Pour k=indDebut(1)S.taille faire

  /*On ne compare qu'avec ceux qui ne sont pas déjà
    choisis comme plus petits élements */

    temp <-- T.T2[k] ;
    Si (temp != -1) et (S.val[min] > S.val[temp]) alors
      min <-- temp;
    FSi
  FPour
```

```
    T.T2[min] <-- -1; /*pour qu'on ne puisse plus
                          utilisé cette indice pour
                          comparer car il est déjà
                          un plus petit.*/
    T.T1[T.ecrase1] <-- min; /*et on le met à sa place
                              dans T1 */
    T.ecrase1 += 1 ; /* pour avancer dans T1 */

    /*Ensuite on reprends le procédé */
    ResearchSmall(S, T1, T2)

Fin
```

Nous présentons maintenant une implémentation en python de ce programme.

N.B : Notons ici que nous utiliserons des fonctions et non des procédures en ce qui concerne l'implémentation en python. En l'occurence :

1. la fonction principale renverra un tuple contenant les indices des 2 plus petits éléments dans l'ordre. Et :
2. la fonction auxiliaire quant à elle, renverra le tableau T1.

```python
class Trier :
    def __init__(self, taille) :
        # T1 est le tableau des indices des plus petits
        # elements de la suite. Les -1 vont prendre
        # progressivement ces indices là.
        self.T1 = [-1 for i in range(taille)]

        # T2 contient les indices de la suite dont ceux differents de -1
        # sont les indices de ceux qui n'ont pas encore été choisis.
        self.T2 = [ i for i in range(taille)]

        # ecrase1 est le plus petit indice de T1 contenant -1
        # c'est là dans T1 qu'on mettra l'indice du prochain plus petit.
        self.ecrase1 = 0

def Research2mostSmall( S : Suite)-> int :
    # on crée une instance de la classe Trier
    T = Trier(S.taille)

    # on determine les deux plus petits éléments de la suite
    # T1[0] et T1[1]
    T1 = Research(S, T)

    return( T1[0], T1[1] )

def Research( S : Suite, T : Trier) :
    """
    C'est en réalité la fonction qui est récursive
```

```
    """

    # la recursion s'arrête lorqu'on a déja les deux
    # plus petits éléments de la suite.
    if T.ecrase1 == 2 :
        return T.T1

    #######
    for i in range(S.taille) :
        if T.T2[i] != -1 :
            min = i
            debutParcours = i
            break

    for i in range(debutParcours +1, S.taille) :
        a = T.T2[i]

        if a != -1 and S.val[a] < S.val[min] :
            min = a

    ######
    """
    # la partie entre ## peut aussi se faire par :

    Temp = [x for x in  T.T2 if x!= -1]

    min = Temp[0]

    for i in range(1, len(Temp)) :
        a = T.T2[i]

        if S.val[a] < S.val[min] :
            min = a
    """

    # on met à jour les tableaux T1 et T2 ainsi que
    # ecrase1.
    T.T1[T.ecrase1] = min

    T.T2[min] = -1

    T.ecrase1 += 1

    return Research(S, T)
```

NOUS VOUS REMERCIONS POUR VOTRE AIMABLE ATTENTION !