Algo et Structures de données

Introduction

- Ce cours aborde les structures classiques rencontrées en informatique pour organiser des données.
- On suppose que vous connaissez déjà les tableaux et les enregistrements (struct en C).
- Pour aborder les différentes structures de données présentées ici, il vous faudra également bien maîtriser la notion de pointeurs et de gestion dynamique de la mémoire.

Introduction

Les structures de données présentées ici sont:

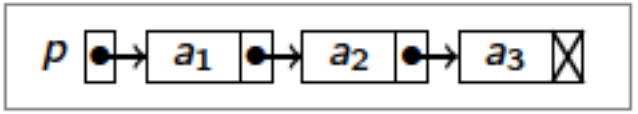
- les tableaux (arrays en anglais),
- les listes chaînées (linked lists en anglais),
- les piles (stacks en anglais),
- les files (queues en anglais),
- les arbres
  - Arbres binaires (binary trees en Anglais).
  - B-arbres (B-trees)

Les structures de données lineaire

Les listes chainées

Notion de liste

- Une liste est une suite d'éléments dont la notion de successeur est explicite c'est à dire à chaque élément, on lui associe des informations et l'adresse de son successeur.



- Elle est caractérisée par trois propriétés:
  - Chaque éléments contient un ou plusieurs champs constituant l'information et qui matérialise la fonction successeur,
  - Le dernier élément se distingue par une valeurs conventionnelle contenu dans le champs successeur,
  - On accède à la liste par l'adresse du premier élément (tête de la liste)

Les listes chainées

Déclaration :

Une liste chaînée est obtenue à partir du type cellule(structure) définie par:

```
typedef … Objet
typedef  struct  maillon {
            Objet  info;
             struct maillon *suiv;
      }  MAILLON, *PTR
```

Plusieurs types de liste:

- Listes simplement chainées,
- Listes doublement chainées,
- Listes circulaires,
- Listes ordonnées

Les listes chainées

Affichage d'une liste

```
Void parcours(PTR L){
      PTR pc =L;
      while (pc!= Null)
         {
            afficher(pc-> info);
            pc=pc->suiv;
         }
}
```

Les listes chainées

Définition récursive

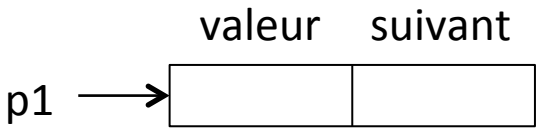
Soit X un ensemble ordonné, une liste chainée d'un ensemble X est définie comme:

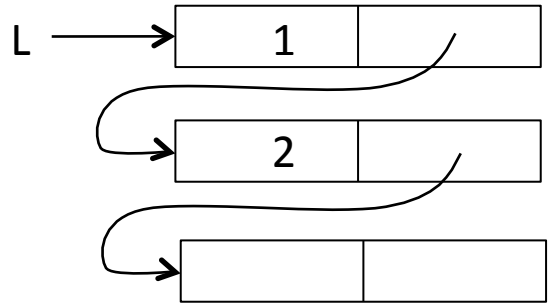
- un symbole conventionnel appelé liste vide,
- Ou un couple (i,L) ou i appartient à X et L une liste chainée d'éléments de X

```
Void  Parcourtrecurssif(PTR L) {
      if(L!=Null)
         {
            afficher(L -> info);
            Parcourtrecurssif(L ->suiv);
         }
}
```

Les listes chainées: Insertion d'un élément

Insertion en début






```
Void insertiondebut(objet X, PTR *L) {
        PTR p=(PTR)malloc(sizeof(MAILLON));
         p ->info = X;
         p ->suiv = *L;
         *L=p;
}
```

Les listes chainées: Insertion d'un élément

Insertion en Fin

```
Void insererenfin(Objet X, PTR L){
          PTR PC =L;
          while(pc->suiv != Null)
          pc=pc->suiv;

          PTR P(PTR)  malloc(sizeof(MAILLON));
          P->info=X;
          P->suiv=Null;
          Pc->suiv=p;
}
```

Les listes chainées: Insertion d'un élément

Insertion en milieu

```
Void Insertionenmilieu(Objet X, PTR P) {
     PTR pc=(PTR)malloc(sizeof(MAILLON));
     pc->info=X;
     pc->suiv=P->suiv;
     P->suiv = pc;
}
```

Les listes chainées

Recherche d'un élément:

```
PTR chercher (Objet X, PTR L){
      PTR pc=L;
      while (pc!= Null && pc->info!=X)
      pc=pc->suiv;
      Return pc;

}
```

Les listes chainées

Supression d'un élément:

```
Void suprimer (PTR * L, Objet X) {
PTR p,pc;
pc=NULL
p=*L;
while (p!=NULL) {
    if (p->info!=X) { //on avance dans la liste
    prec=p;
    p=p->suiv;
    }
   else { //on a trouvé l'élément à supprimer
        if(pc!=NULL) pc->suiv=p->suiv;
    else

        free(p);


    }
   }}
```

Les listes chainées

Listes ordonnées:

Une liste ordonnée est une liste tel que les maillons ont un champ clé dont les valeurs appartiennent à un ensemble totalement ordonnée et qui vérifie:

si p->suiv != Null alors

P->clé $\leq$ p->suiv->clé

Les listes chainées

Listes ordonnées:

Insertion:

```
Void inserer(Objet X, PTR *L){
        PTR pr,pc,p;
        pc=*L;
                while (pc!= Null && pc->info<X)
                   {
                     pr=pc;
                     pc=pc->suiv;
                   }
       p=(PTR)malloc(sizeof(MAILLON));
       p->info=X;
       p->suiv=pc;
       if(pc==*L)
           *L=P;
       else
           pr->suiv=P;
```

Les listes chainées

Listes ordonnées:

Recherche:

```
PTR chercher (Objet X, PTR L){
      PTR pc=L;
      while (pc!= Null && pc->info<X)
      pc=pc->suiv;
      Return pc;

}
```

Exercice d'application

Exercice 1:

Ecrire un algorithme qui permet de déterminer la valeur maximale d'une liste chaînée d'entiers positifs.

Exercice 2:

On considère deux listes chaînées L1 et L2 dont les éléments sont des entiers. Ecrire un algorithme qui rattache la liste L2 à la suite de la liste L1. Tous les cas particuliers doivent être pris en compte.

Les listes chainées: Les Piles:

Les Piles:

- Une pile est une structure qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé le sommet de la pile.
- Quant on ajoute un élément, celui-ci devient le sommet de la pile, c'est-à-dire le seul élément accessible.
- Quant on retire un élément de la pile, on retire toujours le sommet, et le dernier élément ajouté avant lui devient alors le sommet de la pile.
- le dernier élément ajouté dans la pile est le premier élément à en être retiré.
- Cette structure est également appelée une liste LIFO (Last In, First Out).

Les listes chainées Les Piles:

Les Piles:

- Deux principales fonctions de manipulation des Piles:
  - Empiler(valeur,Pile)
  - Depiler(Pile) ->valeur

- Pour empiler, on doit tester si la Pile n'est pas pleine
- Pour dépiler, on doit tester si la Pile n'est pas vide

Les Piles:

Réalisation d'une pile à l'aide d'un tableau

```
typedef  … Objet
#define MAXPLIE  100
 typedef stuct pile {
          Objet T[MAXPILE];
            int sp;
} Pile;

Pile p;
```

```
Void initialiser()
           {
              p.sp=-1;
          }
```

Les Piles:

Réalisation d'une pile à l'aide d'un tableau :

Empiler:

```
void  empiler (Objet x){
          p.T[sp+1]=x;
          p.sp=p.sp+1;
   }
```

Dépiler:

```
Objet  depiler (void){
      Return p.T[p.sp--];
}
```

```
int  depiler (void){
          int x=p.T[p.sp];
          p.sp=p.sp-1;
          Return x;
}
```

Les Piles:

Réalisation d'une pile à l'aide d'un tableau :

Teste Pile vide:

```
int  Pilevide (void){
      Return p.sp==-1;
}
```

Teste Pile pleine:

```
int  Pilevide (void){
      Return p.sp= MAXPILE-1;
}
```

Les Piles:

Réalisation d'une pile à l'aide d'une liste chainée :

Déclaration :

```
Typedef struct maillon {
          Objet  info;
          Stuct maillon *suiv;
        }MAILLON,PILE;
```

```
PILE p;
```

Initialisation

```
Void initialiser (void) {
          p=NULL;
}
```

Les Piles:

Réalisation d'une pile à l'aide d'une liste chainée :

Empiler :

```
Void  empiler(Objet x){
       P=Nouveau_maillon(x,p);
   }
```

```
PILE Nouveau_maillon(Objet x, PILE S){
        PILE pc=(PILE)malloc(sizeof(MAILLON));
        pc->info=x;
        pc->suiv=S;
        return pc;
}
```

Les Piles:

Réalisation d'une pile à l'aide d'une liste chainée :

Depiler :

```
Objet depiler (void) {
          Objet x;
          PILE pc;
          x=p->info;
          pc=p;
          p=p->suiv;
          free(pc);
          return x;
    }
```

Les Files d'attente

Les Files:

Une file c'est une structure de données qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé la tête de la file.

C'est une structure dans laquelle, le premier arrivé est le premier sorti.

- Quant on ajoute un élément, celui-ci devient le dernier élément qui sera accessible.
- Quant on retire un élément de la file, on retire toujours la tête, celle-ci étant le premier élément qui a été placé dans la file.

Cette structure est également appelée une liste FIFO (First In, First Out).

Les Files d'attente

Réalisation d'une file à l'aide d'un tableau :

Pour implanter une file nous nous donnerons deux indices, appelés tete et queue, et nous utiliserons un tableau circulaire, obtenu à partir d' un tableau ordinaire en décidant que la première case du tableau fait suite à la dernière.

D' un point de vue technique, il suffira de modifier légèrement l' opération « passage au successeur ». Pour un tableau ordinaire, cette opération se traduit par l' incrémentation d' un indice :

Les Files d'attente

Réalisation d'une file à l'aide d'un tableau :

Déclaration:

```
typedef  … Objet
#define MAXFLIE  100
 typedef stuct file {
         Objet T[MAXFILE];
         int tete, queue;
} FILE;

FILE f;
```

```
#define INCR(i) (((i)+1)% MAXFILE)
```

initialisation:

```
Void initialiser()
          {
            f.tete=f.queue=-1;
          }
```

Les Files d'attente

Réalisation d'une file à l'aide d'un tableau :

enfiler:

```
Void enfiler (Objet x)
        {
            if (filvide())
            f.tete=0;
            f.T[INCR(f.queue)]=x;
          }
```

defiler:

```
Objet defiler ()
        {
            return f.T[f.tete++]
         }
```

Teste file vide:

```
Int filevide()
        {
            Return f.tete==-1;
         }
```

Les Files d'attente

Réalisation d'une file à l'aide d'une liste chainnée :

```
Typedef struct maillon {
          Objet  info;
          Stuct maillon *suiv;
        }MAILLON,*PTR;
```

```
typedef stuct file {
         PTR tete, queue;
} FILE;

FILE f;
```

Les Files d'attente

Réalisation d'une file à l'aide d'une liste chainnée :

Enfiler

```
Void enfiler (Objet x){
      PTR p=nouveau_maillon(x,NULL);
      if (f.tete!=NULL)
          f.queue->suiv=p;
      else
          f.queue=p;
          f.tete=p;
}
```

Defiler

```
Objet defiler(void){
        Objet x=f.tete->info;
        PTR p=f.tete;
        f.tete=f.tete->suiv;
        free(p);
        Return x;
}
```