

# TP 1 Structure de données

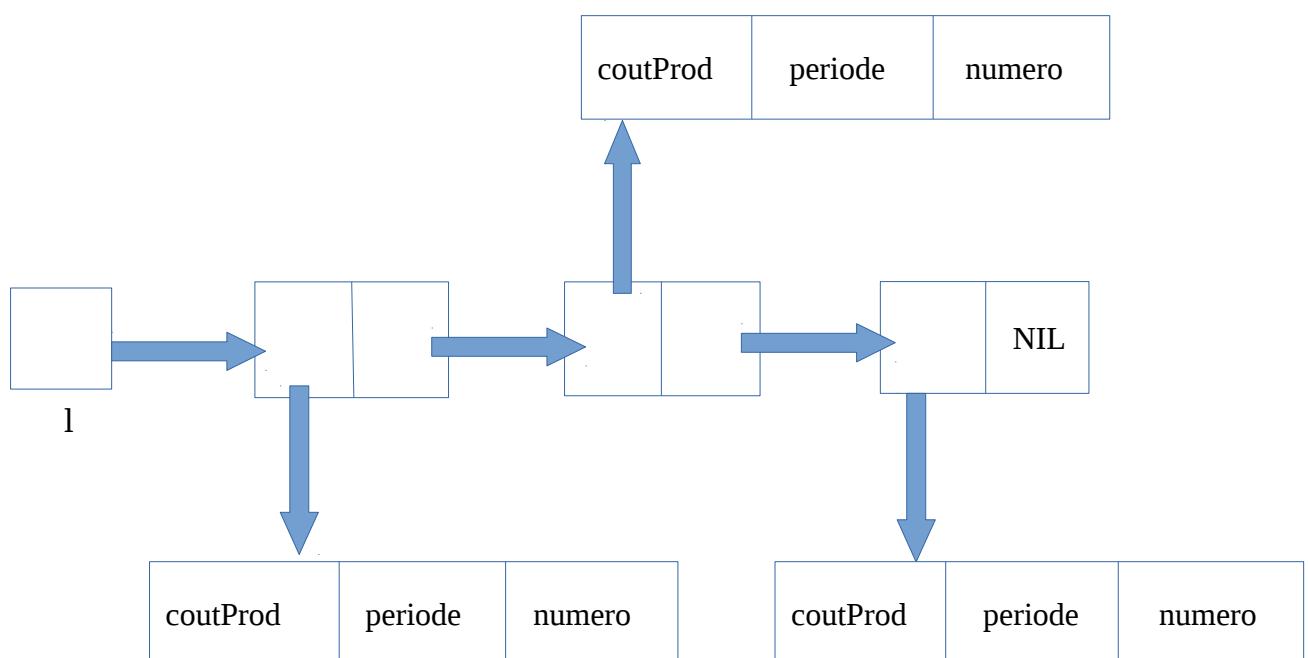
Monteilhet Gautier  
Champredonde Adrien

## 1) Présentation générale

Objectif :

Ce tp a pour objectif de récupérer d'un fichier une matrice contenant différents coûts de production par usine en fonction des périodes. Et ensuite de créer une liste chaînée des K valeurs (coût de production) les plus faibles. Enfin avec la possibilité de supprimer une usine de la liste chaînée et de pouvoir sauvegarder la liste dans un fichier.

Structure de données :



Organisation du code source :

liste.h      Structure de la liste et de l'usine avec en plus les prototypes des fonctions traitant la liste.  
liste.c      Fonctions : - NouvelleListe

- InsererEnTete
- InsererEnFin
- LibererListe
- AfficherListe
- LibererElt
- SupprElt
- SupprPremierElt

tp1.h Prototypes

tp1.c Fonctions : - TriTab  
- Chargement  
- AfficherMatrice  
- LibererMatrice  
- ConvertirMatrice  
- SupprUsine  
- Sauvegarde  
- Global

tp1main.c Programme principale

## 2) Présentation de chaque fonction

liste.c

InsererEnTete :

On alloue l'espace mémoire nécessaire pour créer un maillon m;  
Si aucune mémoire n'est trouvée pour ce nouveau maillon alors

    On affiche un message d'erreur;  
    On quitte le programme avec un code d'erreur;

fsi ;

La période de la structure usine prend la valeur de la période de l'usine en entrée ;  
Le coût de production de la structure usine prend la valeur du coût de production de l'usine  
en entrée ;

Le numéro de la structure usine prend la valeur du numéro de l'usine en entrée ;

Le pointeur suivant dans le maillon m pointe sur la tête de liste ;

On retourne le maillon m qui est le premier élément de la liste;

Fin ;

```

9  /* -----
10 /* InsererEnTete      Insére un maillon en tête de liste */
11 /*
12 /* En entrée: l est la liste chaînée où la fonction ajoute un maillon */
13 /*           u représente l'usine qui sera ajouté dans le maillon */
14 /*
15 /* En sortie: la liste chainée l avec un maillon supplémentaire */
16 /*
17
18 liste_t InsererEnTete(liste_t l, usine_t u)
19 {
20     maillon_t * m;
21
22     m = (maillon_t *)malloc(sizeof(maillon_t));
23
24     if(m == NULL)
25     {
26         printf("Erreur : Problème d'allocation\n");
27         exit(2);
28     }
29
30     m->u.période = u.période;
31     m->u.coutProd = u.coutProd;
32     m->u.numero = u.numero;
33
34     m->suiv = l;
35
36     return m;
37 }
```

### InsererEnFin :

On alloue l'espace mémoire nécessaire pour créer un maillon m;

Si aucune mémoire n'est trouvée pour ce nouveau maillon alors

    On affiche un message d'erreur;

    On quitte le programme avec un code d'erreur;

fsi ;

On met le pointeur suivant du nouveau maillon égale à NIL ;

La période de la structure usine prend la valeur de la période de l'usine en entrée ;

Le coût de production de la structure usine prend la valeur du coût de production de l'usine en entrée ;

Le numéro de la structure usine prend la valeur du numéro de l'usine en entrée ;

Si la liste l n'est pas vide alors

    le pointeur a pointe sur la tête de liste ;

    Tant que a ne pointe pas sur le dernier maillon de la liste faire

        a pointe sur le maillon suivant ;

ftq ;

    On fait pointer le pointeur suiv du dernier maillon de la liste vers notre nouveau maillon m ;

Sinon

    le pointeur l pointe sur le nouveau maillon pointé par m ;

fsi ;

    On retourne la liste l;

Fin ;

```

39  /* -----
40  * InsererEnFin      Insére un maillon en fin de liste
41  */
42  /* En entrée: l est la liste chainée où la fonction ajoute un maillon
43  *           u représente l'usine qui sera ajouté dans le maillon
44  */
45  /* En sortie: la liste chainée l avec un maillon supplémentaire
46  */
47
48  liste_t InsererEnFin(liste_t l, usine_t u)
49  {
50      maillon_t * m,* a;
51
52      m = (maillon_t *)malloc(sizeof(maillon_t));
53      if(m == NULL)
54      {
55          printf("Erreur : Probleme d'allocation\n");
56          exit(2);
57      }
58      m->suiv = NouvelleListe();
59      m->u.periode = u.periode;
60      m->u.coutProd = u.coutProd;
61      m->u.numero = u.numero;
62      if(l != NULL)
63      {
64          a=l;
65          while(a->suiv != NULL)
66          {
67              a = a->suiv;
68          }
69
70          a->suiv = m;
71      }
72      else
73      {
74          l = m;
75      }
76      return l;
77  }
78

```

NouvelleListe :

On retourne NIL ;

Fin ;

```

78  /* -----
79  * NouvelleListe      Renvoie NULL pour faire une nouvelle liste
80  */
81  /*
82  * En entrée:
83  */
84  /* En sortie: retourne NULL
85  */
86
87  liste_t NouvelleListe(void)
88  {
89      return NULL;
90  }
91

```

### LibererListe :

Tant que la liste l n'a pas été parcourue en entière faire  
    On libère l'espace mémoire occupé par maillon pointé par l;  
    Le pointeur l pointe sur le maillon suivant;  
ftq ;

Fin ;

```
92  /* ----- */
93  /* LibererListe      Libère la mémoire occupé par la liste chaînée */
94  /*
95  /* En entrée: l est la liste chaînée à libérer
96  /*
97  /* En sortie:
98  /*
99
100 void LibererListe(liste_t l)
101 {
102     maillon_t * m;
103
104     while(l!=NULL)
105     {
106         m = l;
107         LibererElt(m);
108         l = l->suiv;
109     }
110 }
```

### LibererElt :

On libère l'espace mémoire pointé par m :

Fin ;

```
112  /* ----- */
113  /* LibererElt      Libère la mémoire occupé d'un maillon de la liste chaînée */
114  /*
115  /* En entrée: m est le maillon à libérer
116  /*
117  /* En sortie:
118  /*
119
120 void LibererElt(maillon_t * m)
121 {
122     free(m);
123 }
```

SupprPremierElt :

Si la liste n'est pas vide alors

Le pointeur m pointe sur le premier maillon de la liste;

Le pointeur l pointe sur le maillon suivant;

On libère l'espace mémoire occupé pointé par m;

Sinon

Le pointeur l prend la valeur NIL;

fsi ;

On retourne la liste l;

Fin ;

```
125  /* ----- */
126  /* SupprPremierElt      Supprime le premier maillon de la liste */
127  /*
128  /* En entrée: l est la liste chainée
129  /*
130  /* En sortie: la liste chainée avec le premier maillon supprimé
131  /*
132
133 liste_t SupprPremierElt(liste_t l)
134 {
135     maillon_t * m;
136
137     if (l != NULL)
138     {
139         m=l;
140         l=l->suiv;
141         LibererElt(m);
142     }
143     else
144     {
145         l = NULL;
146     }
147     return l;
148 }
```

SupprElt :

Le pointeur suivant du maillon pointé par prec pointe sur le maillon suivant le maillon à supprimer;

On libère l'espace mémoire occupé par le maillon pointé par suppr;

On retourne le pointeur prec;

Fin ;

```
150  /* ----- */
151  /* SupprElt      Supprime un élément de la liste chainée
152  /*
153  /* En entrée: prec est le pointeur sur le maillon précédent à celui qu'on supprimera
154  /*           suppr est le pointeur sur le maillon à supprimer
155  /*
156  /* En sortie: renvoie le pointeur prec
157  /*
158
159 liste_t SupprElt(maillon_t * prec, maillon_t *suppr)
160 {
161     prec->suiv = suppr->suiv;
162     LibererElt(suppr);
163
164     return prec;
165 }
```

AfficherListe :

Si l est vide alors

On affiche que la liste est vide;

Sinon

Tant que la liste n'a pas été parcourue en entière faire

    On affiche l'usine du maillon pointé par l;

    l pointe sur le maillon suivant ;

        ftq ;

    Fsi ;

Fin ;

```

165  /* ----- */
166  /* AfficherListe          Afficher la liste chaînée      */
167  /* ----- */
168  /* En entrée: l est la liste chaînée à afficher           */
169  /* ----- */
170  /* En sortie:          */
171  /* ----- */
172  /* ----- */
173  /* ----- */
174
175 void AfficherListe(liste_t l)
176 {
177     if(l == NULL)
178     {
179         printf("Liste vide !\n");
180     }
181     else
182     {
183         printf("-----\n");
184         printf("| Usine | periode | cout de production |\n");
185         printf("|-----|-----|\n");
186
187         while(l!=NULL)
188         {
189             printf("|\t %d\t|\t %d\t|\t %d\t|\n",l->u.numero, l->u.periode, l->u.coutProd);
190             if(l->suiv==NULL) printf("-----|\n");
191             else printf("-----|\n");
192             l=l->suiv;
193         }
194     }
195 }
196 }
```

## ConvertirMatrice :

On crée une nouvelle liste l;  
 On crée un tableau tab trié en fonction du tableau mat;  
 On parcourt le tableau avec un pas de un, faire  
     on insère en fin de la liste l, la valeur du tableau où nous sommes situés;  
 ftq ;  
 On retourne la liste l;

```

10  /* ----- */
11  /* ConvertirMatrice      Convertie notre matrice en liste chaînée triée (ordre croissant) avec k valeurs */
12  /*
13  /* En entrée: mat matrice d'entier contenant les coûts de production des usines
14  /*           m nombre d'usine
15  /*           n nombre de période
16  /*           k nombre de valeur à conserver
17  /*
18  /* En sortie: la liste chaînée l triée en ordre croissant
19  /* -----
20
21 liste_t ConvertirMatrice(int ** mat, int m, int n, int k)
22 {
23     int i;
24     usine_t tab[m*n];
25     liste_t l;
26
27     l = NouvelleListe();
28     TriTab(tab,mat,m,n);
29     for(i=0;i<k;i++)
30     {
31         l = InsererEnFin(l,tab[i]);
32     }
33     return l;
34 }
```

## TriTab :

Pour li allant de 0 à m-1 faire  
 Pour co allant de 0 à n-1 faire  
     La période prend la valeur de co;  
     Le coût de production prend la valeur de la matrice avec les indices li et co ;;  
     Le numero prend la valeur de li;  
     On incrément cpt de 1;  
 ftq ;  
 ftq ;  
 On initialise estTrie à -1;  
 On initialise cour à 0;  
 Tant que estTrie vaut -1 faire  
     Pour i allant de 0 à m\*n -2 faire  
         Si le coût de production à l'indice i est strictement supérieur au coût de
             production à l'indice i+1 alors  
                 On échange la période à l'indice i avec la période à l'indice i+1 ;
                 On échange le coût de production à l'indice i avec la période à
                     l'indice i+1 ;
                 On échange le numéro à l'indice i avec le numéro à l'indice i+1 ;
                 On initialise cour à -1;  
     Fsi ;  
 ftq ;  
 si cour est égal à -1 alors  
     on met la valeur 0 à cour;  
 sinon  
     on met la valeur 1 à estTrie;  
 fsi ;  
 ftq ;

Fin ;

```
36  /* ----- */
37  /* TriTab      Tri un tableau d'usine en ordre croissant de leur coût de production */
38  /*
39  /* En entrée: tab est le tableau à rendre trié
40  /*           mat matrice d'entier contenant les coûts de production des usines
41  /*           m nombre d'usine
42  /*           n nombre de période
43  /*
44  /* En sortie: tab le tableau des usines triées
45  /*
46
47 void TriTab(usine_t * tab, int ** mat, int m, int n)
48 {
49     usine_t tmp;
50     int estTrie,cour, li,co,i, cpt=0;
51
52     for(li=0; li<m; li++)
53     {
54         for(co=0; co<n; co++)
55         {
56             tab[cpt].periode = co;
57             tab[cpt].coutProd = mat[li][co];
58             tab[cpt].numero = li;
59             cpt++;
60         }
61     }
62
63     estTrie=-1;
64     cour = 0;
65     while(estTrie== -1)
66     {
67         for(i=0; i < m*n - 1; i++)
68         {
69             if(tab[i].coutProd > tab[i+1].coutProd)
70             {
71                 tmp.periode=tab[i].periode;
72                 tmp.coutProd=tab[i].coutProd;
73                 tmp.numero=tab[i].numero;
74                 tab[i].periode=tab[i+1].periode;
75                 tab[i].coutProd=tab[i+1].coutProd;
76                 tab[i].numero=tab[i+1].numero;
77                 tab[i+1].periode = tmp.periode;
78                 tab[i+1].coutProd = tmp.coutProd;
79                 tab[i+1].numero = tmp.numero;
80                 cour = -1;
81             }
82         }
83         if(cour== -1)
84             cour = 0;
85         else estTrie = 1;
86     }
87 }
```

Chargement :

On ouvre le fichier f en mode lecture;

Si l'ouverture n'a pas marché alors

    On affiche un message d'erreur;

    On quitte le programme avec un code d'erreur;

fsi ;

Lecture des entiers m et n de la première ligne du fichier f;

On alloue de l'espace mémoire à la matrice mat;

Si l'allocation n'a pas marché alors

    On affiche un message d'erreur;

    On quitte le programme avec un code d'erreur;

fsi ;

Pour i allant de 0 à m-1 faire

    On alloue de l'espace mémoire pour la tableau d'entier mat;

    Si l'allocation n'a pas marché alors

On affiche un message d'erreur;  
 On quitte le programme avec un code d'erreur;  
 fsi ;  
 Pour j allant de 0 à n-1 faire  
     On ajoute dans le tableau le coût de production du fichier;  
     ftq ;  
 ftq ;  
 On ferme le fichier f;  
  
 On retourne la matrice mat ;  
 Fin ;

```

88  /* -----
89  /* Chargement      Charge un fichier pour renvoyer une matrice des coûts de production des usines
90  /*
91  /* En entrée: nomFichier représente le nom de fichier que l'on doit charger
92  /*           m pointeur sur le nombre d'usine
93  /*           n pointeur sur le nombre de période
94  /*
95  /*
96  /* En sortie: mat matrice des coûts de production des usines
97  /*           m nombre d'usine
98  /*           n nombre
99  /*
100 /* -----
101 int ** Chargement(char * nomFichier, int * m, int * n)
102 {
103     FILE * f;
104     int i,j, ** mat;
105
106     f=fopen(nomFichier,"r");
107     if(f == NULL)
108     {
109         printf("Erreur : Problème de lecture\n");
110         exit(1);
111     }
112     fscanf(f,"%d%c %d%c",m,n);
113     mat = (int **)malloc(* m * sizeof(int));
114     if(mat == NULL)
115     {
116         printf("Erreur : Problème d'allocation\n");
117         exit(2);
118     }
119
120     for(i=0;i<*m;i++)
121     {
122         mat[i] = (int *)malloc(* n * sizeof(int));
123         if(mat[i] == NULL)
124         {
125             printf("Erreur : Problème d'allocation\n");
126             exit(2);
127         }
128         for(j=0;j<* n;j++)
129         {
130             fscanf(f,"%d%c",mat[i]+j);
131         }
132     }
133     fclose(f);
134
135     return mat;
136
137 }
138

```

### AfficherMatrice :

Pour i allant de 0 à m-1 faire

Pour j allant de 0 à n-1 faire

    On affiche la valeur de la matrice avec les indices i et j;

    ftq ;

    ftq ;

Fin ;

```
140  /* ----- */  
141  /* AfficherMatrice      Affiche toute la matrice de dimension m*n */  
142  /* */  
143  /* En entrée: mat matrice d'entier contenant les coûts de production des usines */  
144  /*          m nombre d'usine */  
145  /*          n nombre de période */  
146  /* */  
147  /* En sortie: */  
148  /* */  
149  
150 void AfficherMatrice(int ** mat, int m, int n)  
151 {  
152     int i,j;  
153  
154     for(i=0;i<m;i++)  
155     {  
156         for(j=0;j<n;j++)  
157             printf("%d ",mat[i][j]);  
158         printf("\n");  
159     }  
160     printf("\n");  
161 }
```

### LibererMatrice :

Pour i allant de 0 à m-1 faire

    Libération de l'espace mémoire occupé par mat[i];

    ftq ;

    Libération de l'espace mémoire occupé par la matrice mat;

```
165  /* ----- */  
166  /* LibererMatrice      Libère l'espace occupé par la matrice */  
167  /* */  
168  /* En entrée: mat matrice d'entier contenant les coûts de production des usines */  
169  /*          m nombre d'usine */  
170  /* */  
171  /* En sortie: */  
172  /* */  
173 void LibererMatrice(int ** mat, int m)  
174 {  
175     int i;  
176  
177     for(i=0;i<m;i++)  
178     {  
179         free(mat[i]);  
180     }  
181     free(mat);  
182 }
```

SupprUsine :

Tant que la liste n'est pas vide et que le maillon de tête a un numéro d'usine est égale à u faire :

    On supprime le premier élément de la liste l;

    ftq ;

    Si l n'est pas vide alors

        Le pointeur a pointe sur la tête de la liste ;

        Tant que le pointeur ne pointe pas sur l'avant dernier maillon de la liste faire

            Si le numéro d'usine du maillon suivant de a vaut u alors

                On supprime le maillon suivant du maillon pointé par a;

            Sinon

                a pointe sur le maillon suivant ;

            fsi ;

    ftq ;

    fsi ;

    On retourne la liste l;

Fin ;

```
183 /* ----- */  
184 /* SupprUsine      Supprime de la liste chainée tous les éléments d'une usine */  
185 /* ----- */  
186 /*  
187 /* En entrée: l est la liste chainée  
188 /*           u numéro d'usine à supprimer de la liste chainée */  
189 /*-----*/  
190 /* En sortie: l la nouvelle liste chainée après suppression */  
191 /*-----*/  
192  
193 liste_t SupprUsine(liste_t l, int u)  
194 {  
195     maillon_t * a;  
196  
197     while(l!=NULL && l->u.numero == u)  
198     {  
199         l = SupprPremierElt(l);  
200     }  
201     if(l != NULL)  
202     {  
203         a=l;  
204         while(a->suiv!=NULL)  
205         {  
206             if(a->suiv->u.numero == u)  
207             {  
208                 a = SupprElt(a, a->suiv);  
209             }  
210             else  
211             {  
212                 a = a->suiv;  
213             }  
214         }  
215     }  
216     return l;  
217 }  
218 }  
219
```

## Sauvegarde :

On ouvre le fichier en mode écriture;  
Si l'ouverture n'a pas marché alors  
    On affiche un message d'erreur;  
    On quitte le programme avec un code d'erreur;  
fsi ;  
Tant que l ne vaut pas NIL faire  
    On écrit dans le fichier les informations du premier élément de la liste l;  
    l pointe sur le maillon suivant ;  
ftq ;  
On ferme le fichier f;

```
219
220  /* -----
221  /* Sauvegarde      Sauvegarde la liste chainée dans un fichier
222  /*
223  /* En entrée: nomFichier est le nom du fichier où la sauvegarde sera effectuée
224  /*           l liste chainée à sauvegarder
225  /*
226  /* En sortie:
227  /* -----
228
229 void Sauvegarde(char * nomFichier, liste_t l)
230 {
231     FILE * f;
232
233     f=fopen(nomFichier,"w");
234     if(f==NULL)
235     {
236         printf("Probleme d'ouverture de fichier\n");
237         exit(1);
238     }
239
240     fprintf(f, "Usine\tPeriode\t\tCout de production\n");
241
242     while(l != NULL)
243     {
244
245         fprintf(f, " %d\t\t %d\t\t %d\n", l->u.numero, l->u.periode, l->u.coutProd);
246         l = l->suiv;
247     }
248
249     fclose(f);
250 }
```

## Global :

On crée la matrice mat en fonction du fichier donné;  
Si k est supérieur à m\*n alors  
    k prend la valeur m\*n;  
fsi ;  
On affiche la matrice mat;  
On convertie la matrice mat en liste chaînée avec k usines;  
On affiche la liste l;  
On supprime l'usine 1 dans la liste l;  
On affiche la liste l;  
On sauvegarde la liste l dans un fichier;  
On libère l'espace mémoire occupé par la matrice mat  
On libère l'espace mémoire occupé par la liste l;

```

251  /*
252  *----- Fonction qui fait appel à toutes les autres fonctions nécessaire pour notre tp */
253  /*
254  */
255  /* En entrée: argv est le tableau de caractères contenant les arguments */
256  /* k est le pointeur d'entier sur le nombre de valeurs */
257  /* à conserver dans la conversion de la matrice */
258  /*
259  * En sortie:
260  */
261
262 void Global(char argv[], int *k)
263 {
264     int **mat, m, n;
265     liste_t l;
266
267     mat = Chargement(argv,&m,&n);
268
269     if(*k > m*n)
270         *k = m*n;
271     printf("\nK : %d\n\n",* k);
272     printf("Matrice du fichier : \n");
273     AfficherMatrice(mat, m, n);
274
275     l=ConvertirMatrice(mat, m, n, * k);
276     printf("Liste chainée : \n");
277
278     AfficherListe(l);
279
280     l = SupprUsine(l, 1);
281
282     printf("\nAprès suppression des usines :\n\n");
283
284     AfficherListe(l);
285
286     Sauvegarde("liste.txt",l);
287
288     LibererMatrice(mat, m);
289     LibererListe(l);
290 }

```

### tp1main.c :

main :

Si le nombre d'argument est strictement inférieur à 2 ou strictement supérieur à 3 alors

On affiche un message d'erreur;

On quitte le programme avec un code d'erreur;

fsi ;

Si le nombre d'argument est égal à 2 alors

l'entier k prend la valeur 0;

Sinon

k prend la valeur de l'argument donné en paramètre

fsi ;

On appelle le programme principal;

On retourne 0;

Fin ;

```
9  /* ----- */  
10 /* main           Fonction main */  
11 /* */  
12 /* En entrée: argc nombre d'arguments */  
13 /*          argv tableau de chaîne de caractères des arguments */  
14 /* */  
15 /* En sortie: le code de retour */  
16 /* ----- */  
17  
18  
19 int main(int argc, char ** argv)  
20 {  
21     int k;  
22  
23     if(argc<2 || argc>3)  
24     {  
25         printf("Erreur : il faut 1 argument minimum et 2 arguments maximum\n");  
26         exit(1);  
27     }  
28     if(argc==2)  
29     {  
30         k=0;  
31     }  
32     else k = atoi(argv[2]);  
33     Global(argv[1], &k);  
34     return 0;  
35 }  
36 |
```

### 3) Compte rendu d'exécution

Makefile :

```
1 |CC = gcc
2 |CFLAGS = -W -Wall -g
3 |LDFLAGS = -lm
4 |
5 |OBJ = tp1main.o tp1.o liste.o
6 |
7 |prog : $(OBJ)
8 |    $(CC) $(OBJ) $(LDFLAGS) -o prog
9 |
10 |liste.o : tp1.h liste.h liste.c
11 |    $(CC) -c liste.c $(CFLAGS)
12 |
13 |tp1.o : liste.h tp1.h tp1.c
14 |    $(CC) -c tp1.c $(CFLAGS)
15 |
16 |clean:
17 |    rm -rf *.o
18 |
19 |mrproper: clean
20 |    rm -rf prog
```

Les cas particuliers :  
- fichier qui n'existe pas  
- k inférieur ou égal à 0  
- k supérieur au nombre possible  
- absence de k  
- matrice vide  
- suppression d'une usine pas présente

Le fichier n'existe pas :

```
Terminal Fichier Édition Affichage Rechercher Terminal Aide
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ make
gcc -c tp1.c -W -Wall -g
gcc tp1main.o tp1.o liste.o -lm -o prog
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices2.txt 10
Erreur : Probleme de lecture
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$
```

k ne correspondant pas aux attentes :

matrice.txt :

1	3	5			
2	14	10	5	45	48
3	48	31	14	47	25
4	23	58	94	12	2

Sans k on prend aucune valeur.

```
Terminal Fichier Édition Affichage Rechercher Terminal Aide
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices1.txt

K : 0

Matrice du fichier :

Liste chaînée :
Liste vide !
```

Avec k supérieur au nombre max de valeurs.

```
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices.txt 17

K : 15

Matrice du fichier :
14 10 5 45 48
48 31 14 47 25
23 58 94 12 2

Liste chaînée :
+-----+
| Usine | periode | cout de production |
+-----+
| 2     |    4     |          2           |
+-----+
| 0     |    2     |          5           |
+-----+
| 0     |    1     |         10          |
+-----+
| 2     |    3     |         12          |
+-----+
| 0     |    0     |         14          |
+-----+
| 1     |    2     |         14          |
+-----+
| 2     |    0     |         23          |
+-----+
| 1     |    4     |         25          |
+-----+
| 1     |    1     |         31          |
+-----+
| 0     |    3     |         45          |
+-----+
| 1     |    3     |         47          |
+-----+
| 0     |    4     |         48          |
+-----+
| 1     |    0     |         48          |
+-----+
| 2     |    1     |         58          |
+-----+
| 2     |    2     |         94          |
+-----+
```

Avec k inférieur ou égale à 0 :

```
Terminal Fichier Édition Affichage Rechercher Terminal Aide
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices.txt -2

K : -2

Matrice du fichier :
14 10 5 45 48
48 31 14 47 25
23 58 94 12 2

Liste chaînée :
Liste vide !
```

Matrice vide :

```
Terminal Fichier Édition Affichage Rechercher Terminal Aide
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices1.txt

K : 0

Matrice du fichier :

Liste chaînée :
Liste vide !
```

Suppression d'une usine pas présente :

Dans notre cas nous voulons supprimer l'usine numéro 1.

```
Terminal Fichier Édition Affichage Rechercher Terminal Aide
adrien@adrien-AsusROG:/media/adrien/DATA/ISIMA/ZZ1/SDD/tp/TP-BINOME$ ./prog matrices.txt 3

K : 3

Matrice du fichier :
14 10 5 45 48
48 31 14 47 25
23 58 94 12 2

Liste chaînée :
-----
| Usine | periode | cout de production |
|-----|
| 2    |    4    |      2   |
|-----|
| 0    |    2    |      5   |
|-----|
| 0    |    1    |     10  |
|-----|

Après suppression des usines :

-----
| Usine | periode | cout de production |
|-----|
| 2    |    4    |      2   |
|-----|
| 0    |    2    |      5   |
|-----|
| 0    |    1    |     10  |
|-----|
```