SDD : TP 1

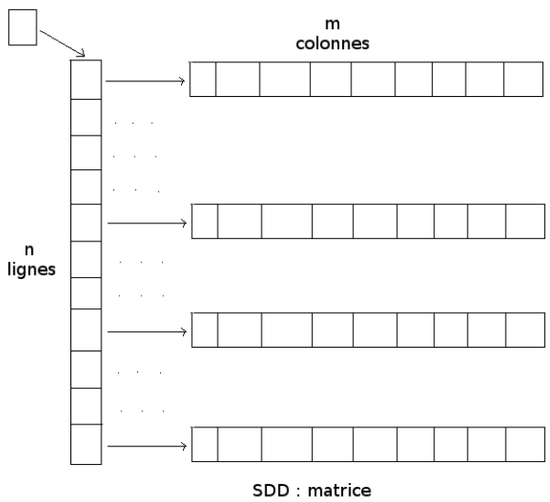
Mathieu Boutin - Jérémy Morceaux

March 12, 2018

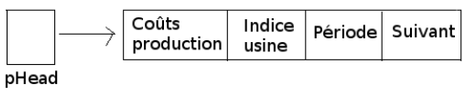
1 Présentation générale

- Ce TP à pour but de créer une matrice représentant des coûts de production de différentes usines à différentes périodes depuis un fichier. Il faut ensuite récupérer les K usines ayant le plus faible coût de production. Finalement, il faut repérer toutes les occurences d'une usines u dans cette liste et les supprimer.

- Schéma de base :



SDD : matrice



SDD : liste chainée

Structure des fichiers de matrices : prenons par exemple une matrice n*m

1ère ligne fichier : <nombre de lignes> <nombre de colonne>
2ème ligne du fichier: $m_{0,1}$ $m_{0,2}$ ... $m_{0,m-1}$
...
n+1ème ligne du fichier : $m_{n-1,1}$ $m_{0,2}$ ... $m_{n-1,m-1}$

- Les fichiers sources se trouvent dans le dossier **src**.Les fonctions relatives au matrices sont dans le fichier **ZZ_matrix.c** et les fonctions relatives aux listes chainées sont dans le fichier **ZZ_linked_list.c**. Les entêtes sont dans le dossier **include**.

2 Détail de chaque fonction

2.1 findElt

Principe : FindElt

On initialise un pointeur courant qui va parcourir la liste.
Tant qu'il n'est pas arrivé à la fin et que l'élément est inférieur à v
    On parcout la liste chainée.
On retourne l'adresse du pointeur courant
FIN

Lexique :

- Paramètre(s) de la fonction

- pHead est la tête fictive de la liste chainée
- v est la valeur que l'on cherche dans cette liste.

- Variable(s) locale(s)
  - curr est le pointeur courant qui parcourt la liste.

Programme commenté :

```
/* ------------------------------------------------------------------------ */
/* findElt:  - Research an element in a sorted linked list.                 */
/*                                                                          */
/* Input:    - pHead is a ficticious head pointer of the linked list.       */
/*           - v is the value we search in the list.                        */
/*                                                                          */
/* ------------------------------------------------------------------------ */
production_t * findElt(float v, production_t * pHead)
{
    production_t  *curr = pHead;

    while (curr != NULL && curr->value < v) /* while we aren't arrived at the end of the list and the value is lower that v, we go through the list */
    {
            curr = curr->next; /* we move forward in the list */
    }

    return(curr); /* we return the adress of the element researched if we found it, otherwise return NULL */
}
```

Source code : findElt()

2.2 linkBlock

Principe : linkBlock

On fait pointer notre nouveau bloc vers suivant
On fait pointer le bloc précédent vers notre nouveau bloc.

FIN

Lexique :

- Paramètre(s) de la fonction
  - prev est le pointeur qui pointe vers l'élément précédent.
  - element est le pointeur vers notre nouveau bloc.
  - next est le pointeur vers le bloc suivant.

Programme commenté :

```
/* ------------------------------------------------------------------------------------------*/
/* linkBlock:                  Link two block between them                                   */
/*                                                                                           */
/* Input:          - prev is the struct that comes before element                            */
/*                 - element is the struct we want to connect                                */
/*                 - next is the struct that comes after element                             */
/*                                                                                           */
/* ------------------------------------------------------------------------------------------*/
void linkBlock(production_t ** prev,production_t * element,production_t * next)
{
    element->next = next; /* we simply modify the element's next and prev's next */
    *prev = element;

}
```

Source code : linkBlock()

2.3 insertKSorted

Principe : insertKSorted

On initialise un pointeur courant et un pointeur qui pointera vers l'élément précédent;
Si la liste chainée est non vide
    On initialise un compteur à zéro
    Tant qu'on est pas à la fin de la liste
        On cherche le bloc où l'on doit insérer notre nouveau bloc. A chaque bloc visité, on incrémente le compteur
    Si le compteur est inférieur ou égale à K
        Si l'adresse du bloc où l'on doit ajouter le nouveau bloc est nulle
            On ajoute notre bloc à la liste chainée [c'est le dernier élément]
        Sinon
            On ajoute notre bloc à la liste chainée et on libère le reste
    Sinon [La valeur du bloc est trop haute]
        On libère le bloc
Sinon [La liste est vide]
    On ajoute le bloc à la liste chainée.

FIN

Lexique :

- Paramètre(s) de la fonction
  - pHead est la tête fictive de la liste chainée
  - address est l'adresse du bloc où l'on doit ajouter le nouveau bloc
  - element est un pointeur le nouveau bloc
  - K est le nombre de plus petit coût de production que l'on veut garder.
- Variable(s) locale(s)
  - prev est le pointeur qui pointe vers l'élément précédent.
  - curr est le pointeur courant qui parcourt la liste.
  - j est un compteur qui permet de ne pas dépasser la valeur K.
  - tmp est un pointeur temporaire pour libérer la fin de la liste chainée.

Programme commenté :

```c
/* ------------------------------------------------------------------------*/
/* insertKSorted: - Insert a block in a linked list so that it stays sorted and */
/*                  not exceed a length of K                               */
/*                                                                         */
/* Input:         - pHead is a ficticious head pointer of the linked list. */
/*                - address is the address of a block. We want to place our */
/*                  block before this one.                                 */
/*                - element is the block we want to place.                 */
/*                - K is the maximal length of our linked list             */
/*                  (note that she can obviously be of length < K).        */
/*                                                                         */
/* ------------------------------------------------------------------------*/
void insertKSorted(production_t ** pHead, production_t *address, production_t *element, int K)
{
    production_t  ** prev  =   pHead;
    production_t  *  curr  = * pHead;

    /* the linked list is not empty */
    if(curr != NULL)
    {
        int j = 0;

        /* We go through the linked list, searching for the address */
        while (curr != NULL && curr != address)
        {
            prev = &(curr->next);
            curr = curr->next;

            j++; /* We increment a counter, if it overpasses K, it means we can free the block because it has a to high value */
        }

        /* if j <= K-1 , we need to free the rest of the linked list so that its length stay <= K */
        if(j <= K-1 )
        {

            /* address = null means we are at the end of the linked list, so there is no block after this one */
            if(address == NULL)
            {
                /* We add the new block */
                linkBlock(prev,element,curr);

            }
            else
            {
                /* We add the new block */
                linkBlock(prev,element,curr);

                /* We need to set curr to the element because the block is inserted before it */
                curr = element;

                /* we find the last element of the linked list */
                while (curr != NULL && j <= K-2)
                {
                    prev = &(curr->next);
                    curr = curr->next;

                    j++;
                }
                /* we free the rest of the linked list */
                if(curr != NULL)
                {
                    production_t *tmp;

                    tmp = curr->next; /* set next's last element to null */
                    curr->next = NULL;

                    freeLinkedList(tmp); /* free the rest of the linked list */
                    tmp = NULL;
                }

            }
        }
        else /* j > k-1, this value is to high, we can free it */
        {
            free(element); /* we free the element because it's outranged */

        }
    }
    else /* the linked list is empty, just need to add the bloc to it */
    {
        linkBlock(pHead,element,curr);
    }

}
```

Source code : insertKSorted()

2.4 removeFactory

Lexique :

- Paramètre(s) de la fonction
  - pHead est la tête fictive de la liste chainée

Principe : removeFactory

On initialise un pointeur courant, un pointeur précédent et un pointeur temporaire.
Tant qu'on est pas arrivé à la fin de la liste.
Si l'entreprise courante est celle recherchée alors :
On supprime cet élément de la liste.
Sinon
On avance dans la liste.

FIN

- factory est l'indice de l'usine que l'on supprimé de la liste chainée.

- Variable(s) locale(s)
  - prev est le pointeur qui pointe vers l'élément précédant le pointeur curr.
  - curr est le pointeur courant qui parcourt la liste.
  - tmp est un pointeur qui permet de libérer un bloc proprement.

Programme commenté :

```
/* ------------------------------------------------------------------------------------*/
/* removeFactory:  - Remove all the factory having "factory" as index in the linked list.  */
/*                                                                                     */
/* Input:          - pHead is a ficticious head pointer of the sorted linked list.         */
/*                 - factory is an intenger which represent the index of a factory.        */
/* ------------------------------------------------------------------------------------*/
void removeFactory(production_t **pHead,int factory)
{
    /* We initiliaze a previous and a current pointer for the research */
    production_t  ** prev  =  pHead;
    production_t  *  curr  = *pHead;
    production_t  *  tmp;

    while (curr != NULL) /* while we  aren't arrived at the end of the list we go through the list */
    {
        tmp = NULL;

        if(curr->factory == factory) /* if we found the factory researched  */
        {
            tmp = curr;
            *prev = curr->next;
        }
        else /*  if the current element is not the factory researched */
        {
            prev = &(curr->next); /* the previous pointer points to the adress of the next element , and NOT to the next element !! */
        }

        curr = curr->next; /* we move forward in the list, the current pointer points to the next element */

        if(tmp != NULL)
        {
            free(tmp); /* we free it to avoid a problem of memory not restored */
        }
    }
}
```

Source code : removeFactory()

2.5 writeLinkedListToFile

Principe : writeLinkedListToFile

On initialise un pointeur courant
Tant qu'on est pas arrivé à la fin de la liste
On affiche un message avec des données de l'élément courant
Le pointeur courant avance dans la liste

FIN

Lexique :

- Paramètre(s) de la fonction
  - file est le fichier dans lequel on veut afficher la liste chainée. (stdout, stderr, ou un fichier texte).
  - pHead est la tête fictive de la liste chainée.

- Variable(s) locale(s)
  - curr est le pointeur courant qui parcourt la liste.

Programme commenté :

```c
/* ------------------------------------------------------------------------------------------------*/
/* writeLinkedListToFile :  - print a linked list in a file (it can be stdout, stderr, or a created file). */
/*                                                                                                 */
/* Input:                   - pHead is a ficticious head pointer of the linked list.               */
/*                                                                                                 */
/* ------------------------------------------------------------------------------------------------*/
void writeLinkedListToFile(FILE* file,production_t *pHead)
{
    production_t  * curr = pHead; /* we initialize a current pointer which will go through the list  */

    while (curr != NULL) /* while we aren't arrived at the end of the list, we print a message with datas picked up in the list */
    {
        fprintf(file,"L'usine %d a une production de %f à la période %d \n",curr->factory,curr->value,curr->period);
        curr = curr->next;
    }

}
```

Source code : writeLinkedListToFile()

2.6 insertProductionBlock

Principe : insertProductionBlock

On récupère l'adresse à laquelle on doit insérer le nouvel élément afin de garder la liste triée.
On crée un nouveau bloc.
Si on est capable d'allouer le bloc alors
    On affecte les nouvelles valeurs à ce bloc.
    On l'insert dans la liste chaînée triée.
Sinon
    On affiche un message d'erreur
FIN

Lexique :

- Paramètre(s) de la fonction
  - pHead est la tête fictive de la liste chainée.
  - value est le coût de production de l'usine.
  - factory est l'indice de l'usine.
  - period est la période de production de l'usine.
  - K est le nombre de plus petit coût de production que l'on veut garder.
- Variable(s) locale(s)
  - insertAdress pointe vers le bloc où on doit insérer le nouveau bloc.
  - newElement pointe vers le nouveau bloc.

Programme commenté :

```c
/* ------------------------------------------------------------------------------------------------*/
/* insertProductionBlock: - Creates and insert a production block in the linked list.              */
/*                                                                                                 */
/* Input:                 - pHead is a ficticious head pointer of the linked list.                 */
/*                        - value is the production's cost of the factory.                         */
/*                        - factory is an intenger which represent the index of the factory.       */
/*                        - period is an integer which represent the period of the production.     */
/*                        - K is the number of the smallest production we want to pick out.        */
/* ------------------------------------------------------------------------------------------------ */
void insertProductionBlock(production_t **pHead, float value, int factory, int period, int K)
{
    production_t * insertAdress;
    production_t * newElement;

    insertAdress = findElt(value,*pHead); /* we research the adress in the list where we could insert the new element */
    newElement = (production_t *)malloc(sizeof(production_t));  /* creation of a new block */

    /* If we were able to allocate the block */
    if(newElement != NULL)
    {
        /* we assign datas to the new block */
        newElement->value = value;
        newElement->factory = factory;
        newElement->period = period;
        newElement->next = NULL;

        insertKSorted(pHead,insertAdress,newElement,K); /* we insert the new block in the sorted linked list by keeping the list sorted */
    }
    else
    {
        printf("[ERROR]: Problem during allocation.\n");
    }
}
```

Source code : insertProductionBlock()

2.7 freeLinkedList

Lexique :

- Paramètre(s) de la fonction
  - pHead est la tête fictive de la liste chainée.
- Variable(s) locale(s)

Principe : freeLinkedList

Tant qu'on est pas arrivé au bout de la liste faire
On libère proprement le bloc courant.
On passe à l'élément suivant.

FIN

- curr est le pointeur courant qui parcourt la liste.
- tmp est un pointeur qui permet de libérer un bloc proprement.

Programme commenté :

```
/* ------------------------------------------------------------------------------*/
/* freeLinkedList:  - Free the linked list                                       */
/*                                                                               */
/* Input:           -pHead is a ficticious head pointer of the sorted linked list*/
/*                                                                               */
/* ------------------------------------------------------------------------------*/
void freeLinkedList(production_t *pHead)
{
    /* We initiliaze a temp and a current pointer for the release */
    production_t * curr = pHead;
    production_t * temp = NULL;

    while(curr != NULL) /* while we  aren't arrived at the end of the list we go through the list */
    {
        temp = curr;

        curr = curr->next;/* we move forward in the list */
        free(temp); /* we free the temporary pointer which pointed to the former current element */
    }
}
```

Source code : freeLinkedList()

2.8 loadMatrixFromFile

Principe : loadMatrixFromFile

On crée un flux vers notre fichier contenant notre matrice et une matrice pour stocker ses valeurs, sa ligne, et sa colonne.

Si le fichier s'est ouvert correctement alors
On lit les dimensions de la matrice situées sur la première ligne du fichier
On alloue dynamiquement une matrice.
Si on a un problème d'allocation à un certain moment
on libère proprement la matrice et on rapporte une erreur à la fonction appelante.
Sinon
On lit l'élément $m_{i,j}$ depuis le fichier et on l'insère dans la matrice
On ferme le fichier
Sinon
On rapporte une erreur à la fonction appelante.

FIN

Lexique :

- Paramètre(s) de la fonction
  - fileName est le nom du fichier qui contient la matrice.
  - errorCode est un pointeur sur un entier qui indique si la fonction s'est bien passée.
- Variable(s) locale(s)
  - matrix est une structure contenant les valeurs de la matrice et sa taille.
  - issue = 1, signifie qu'on doit s'arrêter d'allouer la matrice.
  - file est le flux que l'on ouvre avec le nom du fichier contenant la matrice
  - i et j permettent de parcourir la matrice.

Programme commenté :

```c
/* ------------------------------------------------------------------------------------------ */
/* loadMatrixFromFile                  Create a matrix from a file                            */
/*                                                                                            */
/* Inputs:  - fileName is the name of the file where it is written some data of production    */
/*            of a company                                                                    */
/*          - errorCode is a pointer of an error message :                                    */
/*                 -1 = problem while openin file                                             */
/*                  1 = no problem                                                            */
/*                  0 = problem during allocation                                             */
/*                                                                                            */
/* Output:  - Return a matrix created from a file                                             */
/* ------------------------------------------------------------------------------------------ */
matrix_t loadMatrixFromFile(char* fileName,int *errorCode)
{

    /* We create a matrix_t to store its value, column and line */
    matrix_t    matrix;
    int         issue;
    FILE*       file = fopen(fileName,"r"); /* creation of a flow */

    /* Initialization */
    matrix.value = NULL;
    *errorCode = -1;

    if(file!= NULL) /* We make sure, we actually opened the file */
    {
        int i = 0,
            j = 0;

        *errorCode = 0;

        /* We can get dimension of the matrice n*m through the first line*/
        fscanf(file,"%d %d",&matrix.line,&matrix.column);

        matrix.value = malloc(matrix.line * sizeof(float*)); /* First, we create lines of the matrix */

        issue = 0;

        /* No error during first allocation */
        if (matrix.value != NULL)
        {

            for(i=0; i< matrix.line; i++)
            {
                if(issue == 0) /* So far so good */
                {

                    matrix.value[i]= malloc(matrix.column * sizeof(float)); /* Now, we create columns of the matrix */

                }

                if(issue == 0 && matrix.value[i]==NULL) /* error during allocation, aborting */
                {

                    issue = 1;
                    freeMatrix(matrix); /* we had an issue during allocation, we free the matrix */

                }
            }

        }
        else /* error during allocation */
        {
            freeMatrix(matrix);
        }

        /* Then, we get the values of the matrix */
        *errorCode = 1;


        for(i = 0;i < matrix.line;i++)  /* we go through lines of the matrix */
        {

            for(j=0;j < matrix.column;j++)      /* we go through columns of the matrix */
            {

                fscanf(file,"%f",&matrix.value[i][j]); /* get the data from the file */

            }

        }

        fclose(file); /* Don't forget to close the flow */
    }
    else
    {
        *errorCode = -1; /* We had a problem while opening the file*/
    }


    return(matrix);
}
```

Source code : loadMatrixFromFile()

2.9 freeMatrix

Principe : freeMatrix

Si la matrice est non nulle alors :
On parcourt les lignes de la matrice :
Si la valeur du bloc de la ligne i est non nulle alors:
On libère ce bloc
On libère la matrice;

FIN

Lexique :

- Paramètre(s) de la fonction
  - matrix est la structure de la matrice que l'on veut afficher
- Variable(s) locale(s)

– i est l'entier qui permet de supprimer chaque ligne de la matrice.

Programme commenté :

```c
/* ------------------------------------------------------------------------------------------ */
/* freeMatrix          Free the matrix even if it is badly created.                           */
/*                                                                                            */
/* Input:   - matrix is a struct that stores values of the matrix and its size                */
/*                                                                                            */
/* ------------------------------------------------------------------------------------------ */
void freeMatrix(matrix_t matrix)
{
    int i;

    if (matrix.value != NULL)
    {
        for(i=0; i< matrix.line; i++) /* For each line */
        {
            if(matrix.value[i] != NULL) /* We check that there is something to free */
            {

                free(matrix.value[i]); /* We free the line */
                matrix.value[i] = NULL;

            }
        }

        free(matrix.value); /* We free the main pointer */
        matrix.value = NULL;

    }

}
```

Source code : freeMatrix()

2.10 printMatrix

Principe : printMatrix

On parcourt les lignes de la matrice
    On parcourt les colonnes de la matrice
        On affiche l'élément de la i-ème ligne et de la j-ème colonne

FIN

Lexique :

- Paramètre(s) de la fonction
  - matrix est la structure de la matrice que l'on veut afficher.
- Variable(s) locale(s)
  - i et j permettent d'afficher l'élément de la i-ème ligne et de la j-ème colonne.

Programme commenté :

```c
/* ------------------------------------------------------------------------------------------ */
/* printMatrix             Display the matrix in the terminal                                 */
/*                                                                                            */
/* Input:   - matrix is a struct that stores values of the matrix and its size                */
/*                                                                                            */
/* ------------------------------------------------------------------------------------------ */
void printMatrix(matrix_t matrix)
{
    int i,
        j;

    for(i = 0;i < matrix.line;i++) /* we go through lines of the matrix */
    {
        for(j=0;j < matrix.column;j++) /* we go through columns of the matrix */
        {

            printf("%f \t",matrix.value[i][j]); /* we display the value of the element in the line i and the column j */

        }

        printf("\n");
    }
}
```

Source code : printMatrix()

2.11 main

Principe : main

Si le nombre d'argument est égale à 4
On crée une matrice à partir du nom de fichier donné en argument
Si la création s'est bien passée
On affiche la matrice
Si K est supérieur à 0
Si la matrice est non-nulle
On crée la liste chainée sur toutes les valeurs de la matrice avec insertProductionBlock()
On libère la matrice
On affiche la liste chainée
Si l'indice de l'indice a supprimé est valide
On supprime toutes les occurrences de cette usine dans la liste chainée
On réaffiche la liste chainée
On sauvegarde la liste chainée dans un fichier
On libère la liste chainée
Sinon
On prévient l'utilisateur [Matrice-nulle !]
Sinon
On prévient l'utilisateur [K = 0 !!]
Sinon
On prévient l'utilisateur [Problème d'ouverture de fichier !!]
Sinon
On prévient l'utilisateur [Pas assez/trop peu d'arguments !!]
FIN

Lexique :

- Paramètre(s) de la fonction
  - argc est le nombre d'arguments du programme.
  - argv est la liste des arguments du programme.
- Variable(s) locale(s)
  - matrixA est la structure de la matrice.
  - codeError enregistre l'état de sortie d'une fonction.
  - i et j permettent de parcourir la matrice.
  - K est le nombre de plus petit coût de production que l'on veut garder.
  - factoryIndex est l'indice de l'usine a supprimée dans la liste chainée.
  - file est le flux pour enregistrer la liste chainée.

Programme commenté :

```c
int main(int argc, char * argv[])
{
    if(argc == 4) /* if we have enough arguments */
    {
        /* Declare variables */
        matrix_t        matrixA;
        int             codeError,
                        i
                        j,
                        K,
                        factoryIndex;

        production_t  * pTete;
        FILE          * file;

        /* Load some of them thanks to arguments of the program */
        K = atoi(argv[2]);
        factoryIndex = atoi(argv[3]);

        printf("%sI/ Loading the matrix from the file %s.%s\n",VERT,argv[1],NORMAL);

        matrixA = loadMatrixFromFile(argv[1],&codeError);

        /* Start with the creation of the matrix, if we manage to load it */
        if(codeError == 1)
        {
            printMatrix(matrixA);
            pTete = NULL;

            printf("%sII/ Creating the linked list.%s\n",VERT,NORMAL);

            if(K > 0)
            {
                if(matrixA.line > 0 && matrixA.column > 0) /* we are dealing with a non-null matrix */
                {


                    if( K > matrixA.line * matrixA.column )
                    {
                        fprintf(stdout,"%s[WARNING] K > number of data in the matrix.%s\n",JAUNE,NORMAL);
                    }


                    for(i = 0;i < matrixA.line;i++)
                    {
                        for(j = 0;j < matrixA.column;j++)
                        {
                            if( K != 0) /* there is no use in allocating block when we don't need to sort any value */
                            {
                                insertProductionBlock(&pTete,matrixA.value[i][j],i,j,K);
                            }
                        }
                    }

                    /* Delete the matrix, we don't need it anymore */
                    freeMatrix(matrixA);

                    printf("%sIII/ Displaying the result.%s\n",VERT,NORMAL);
                    writeLinkedListToFile(stdout,pTete);

                    printf("%sIV/ Removing the factory with the indice %d%s\n",VERT,factoryIndex,NORMAL);

                    if(!(factoryIndex > matrixA.line ||  factoryIndex < 0))
                    {
                        /* Removing all the block having factoryIndex as value */
                        removeFactory(&pTete,factoryIndex);
                    }
                    else
                    {
                        fprintf(stdout,"%s[WARNING] You're attempting to remove a non-existing factory.%s\n",JAUNE,NORMAL);
                    }


                    /* Print the new linked list */
                    writeLinkedListToFile(stdout,pTete);


                    printf("%sV/ Saving the linked list in the file data.%s\n",VERT,NORMAL);
                    file = fopen("data","w");

                    if(file != NULL)
                    {
                        /* In order to re-use this linked list, we save it in a file */
                        writeLinkedListToFile(file,pTete);
                        fclose(file);
                    }
                    else
                    {
                        fprintf(stdout,"%s[WARNING] Unable to write the linked list to the file data.%s\n",JAUNE,NORMAL);
                    }

                    /* We can finally free the linked list properly */
                    freeLinkedList(pTete);
                }
                else
                {
                    fprintf(stdout,"%s[WARNING] Null Matrix. There is no data to work with.%s\n",JAUNE,NORMAL);
                }
            }
            else
            {
                fprintf(stdout,"%s[ERROR] K is negative. No data will be shown%s\n",ROUGE,NORMAL);
            }
        }
        else
        {
            fprintf(stderr,"%s[ERROR]%s Can't create the matrix from the file : %s \n%s[ERROR]%s Reason :
",ROUGE,NORMAL,argv[1],ROUGE,NORMAL);

            if(codeError == 0)
            {
                fprintf(stderr,"Unable to allocate enough space for the matrix.\n");
            }
            else
            {
                fprintf(stderr,"The file doesn't exist.\n");
            }
        }
    }
    else
    {
        fprintf(stderr,"%s[ERROR] Too few/much arguments.%s\n",ROUGE,NORMAL);
        printf("[INFO] Usage : %s matrixFile A B \nWhere : \nmatrixFile - It's the name of the file where the matrix is
stored. It needs to be in directory where you start the program \nA - It's the number of smallest value you want to keep
from the matrix\nB - It's the indice of the factory you want to remove from the linked list containing the x smallest
value \n",argv[0]);

    }
    return(0);
}
```

Source code : main()

3 Compte rendu d'exécution

Makefile :

```
CC = gcc
LD_FLAGS = -I./include -I./src
CX_FLAGS = -Wall -Wextra -ansi -pedantic -I./include -I./src

all: main

main: main.o matrix.o linkedList.o
	$(CC) ./obj/main.o ./obj/matrix.o ./obj/linkedList.o -o ./bin/main_program $(LD_FLAGS)

main.o: ./src/ZZ_TP1_main.c ./include/ZZ_matrix.h
	$(CC) -c ./src/ZZ_TP1_main.c -o ./obj/main.o $(CX_FLAGS)

matrix.o: ./src/ZZ_matrix.c ./include/ZZ_matrix.h
	$(CC) -c ./src/ZZ_matrix.c -o ./obj/matrix.o $(CX_FLAGS)

linkedList.o: ./src/ZZ_linked_list.c ./include/ZZ_linked_list.h
	$(CC) -c ./src/ZZ_linked_list.c -o ./obj/linkedList.o $(CX_FLAGS)
```

Makefile

Jeux de test complets:

```
Trace 0: Pas assez d'argument

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program

[ERROR] Too few/much arguments.
[INFO] Usage : ./bin/main_program matrixFile A B
Where :
matrixFile - It's the name of the file where the matrix is stored. It needs to be in directory where you start the program
A - It's the number of smallest value you want to keep from the matrix
B - It's the indice of the factory you want to remove from the linked list containing the x smallest value

Press any key to continue ...
```

Terminal

```
Trace 1 : Matrice vide

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice_vide.txt 1 5

I/ Loading the matrix from the file matrix/matrice_vide.txt.
II/ Creating the linked list.
[WARNING] Null Matrix. There is no data to work with.

Press any key to continue ...
```

```
Ouvrir  matr...  ~/Doc...  Enregistrer
0 0
brut  Largeur des tabulations : 8  Lig 1, Col 1  INS
```

Terminal | fichier matrice

```
Trace 2 : Matrice à 1 element

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice_un_element.txt 1 0

I/ Loading the matrix from the file matrix/matrice_un_element.txt.
0.100000
II/ Creating the linked list.
III/ Displaying the result.
L'usine 0 a une production de 0.100000 à la période 0
IV/ Removing the factory with the indice 0
V/ Saving the linked list in the file data.

Press any key to continue ...
```

```
Ouvrir  matr...  ~/Doc...  Enregistrer
1 1
0.1
brut  Largeur des tabulations : 8  Lig 1, Col 1  INS
```

Terminal | fichier matrice

```
Trace 3 : Matrice à 2 element de la même usine

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice_2.txt 2 0

I/ Loading the matrix from the file matrix/matrice_2.txt.
1.200000        3.000000
II/ Creating the linked list.
III/ Displaying the result.
L'usine 0 a une production de 1.200000 à la période 0
L'usine 0 a une production de 3.000000 à la période 1
IV/ Removing the factory with the indice 0
V/ Saving the linked list in the file data.

Press any key to continue ...
```

```
Ouvrir  matr...  ~/Doc...  Enregistrer
1 2
1.2 3
brut  Largeur des tabulations : 8  Lig 1, Col 1  INS
```

Terminal | fichier matrice

```
Trace 4 : Matrice avec K > nombre de données dans la matrice

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice3.txt 5 0

I/ Loading the matrix from the file matrix/matrice3.txt.
0.400000        0.200000
0.300000        0.100000
II/ Creating the linked list.
[WARNING] K > number of data in the matrix.
III/ Displaying the result.
L'usine 1 a une production de 0.100000 à la période 1
L'usine 0 a une production de 0.200000 à la période 1
L'usine 1 a une production de 0.300000 à la période 0
L'usine 0 a une production de 0.400000 à la période 0
IV/ Removing the factory with the indice 0
L'usine 1 a une production de 0.100000 à la période 1
L'usine 1 a une production de 0.300000 à la période 0
V/ Saving the linked list in the file data.

Press any key to continue ...
```

```
2 2
0.4 0.2
0.3 0.1
```

Terminal | fichier matrice

```
Trace 5 : K negatif ou nulle

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice3.txt -2 0

I/ Loading the matrix from the file matrix/matrice3.txt.
0.400000        0.200000
0.300000        0.100000
II/ Creating the linked list.
[ERROR] K is negative. No data will be shown

Press any key to continue ...
```

```
2 2
0.4 0.2
0.3 0.1
```

Terminal | fichier matrice

```
Trace 6 : Retirer une usine non-existante

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice3.txt 2 3

I/ Loading the matrix from the file matrix/matrice3.txt.
0.400000        0.200000
0.300000        0.100000
II/ Creating the linked list.
III/ Displaying the result.
L'usine 1 a une production de 0.100000 à la période 1
L'usine 0 a une production de 0.200000 à la période 1
IV/ Removing the factory with the indice 3
[WARNING] You're attempting to remove a non-existing factory.
L'usine 1 a une production de 0.100000 à la période 1
L'usine 0 a une production de 0.200000 à la période 1
V/ Saving the linked list in the file data.

Press any key to continue ...
```

```
2 2
0.4 0.2
0.3 0.1
```

Terminal | fichier matrice

```
Trace 7 : Mauvais fichier de matrice

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice_blabla.txt 2 3

I/ Loading the matrix from the file matrix/matrice_blabla.txt.
[ERROR] Can't create the matrix from the file : matrix/matrice_blabla.txt
[ERROR] Reason : The file doesn't exist.

Press any key to continue ...
```

Terminal | fichier inexistant

```
Trace 8 : Cas général

[FORMAT] : ./bin/main_program matrix_file K indice_factory

./bin/main_program matrix/matrice1.txt 7 5

I/ Loading the matrix from the file matrix/matrice1.txt.
100.000000      1.600000        1.000000        9.000000        5.600000        3.200000        1.050000
5.000000        6.000000        7.000000        8.000000        5.300000        4.600000        1.900000
10.000000       11.000000       12.000000       13.000000       5.900000        9.800000        7.400000
14.000000       15.000000       16.000000       0.010000        2.300000        2.300000        0.100000
1.300000        1.600000        1.900000        9.100000        5.600000        6.500000        4.300000
7.800000        9.400000        1.500000        6.400000        0.200000        0.300000        0.800000
5.600000        7.400000        5.600000        4.800000        9.600000        4.600000        5.100000
II/ Creating the linked list.
III/ Displaying the result.
L'usine 3 a une production de 0.010000 à la période 3
L'usine 3 a une production de 0.100000 à la période 6
L'usine 5 a une production de 0.200000 à la période 4
L'usine 5 a une production de 0.300000 à la période 5
L'usine 5 a une production de 0.800000 à la période 6
L'usine 0 a une production de 1.000000 à la période 2
L'usine 0 a une production de 1.050000 à la période 6
IV/ Removing the factory with the indice 5
L'usine 3 a une production de 0.010000 à la période 3
L'usine 3 a une production de 0.100000 à la période 6
L'usine 0 a une production de 1.000000 à la période 2
L'usine 0 a une production de 1.050000 à la période 6
V/ Saving the linked list in the file data.

Press any key to finish.
```

```
7 7
100.0 1.6 1.0 9.0 5.6 3.2 1.05
5.0 6.0 7.0 8.0 5.3 4.6 1.9
10.0 11.0 12.0 13.0 5.9 9.8 7.4
14.0 15.0 16.0 0.01 2.3 2.3 0.1
1.3 1.6 1.9 9.1 5.6 6.5 4.3
7.8 9.4 1.5 6.4 0.2 0.3 0.8
5.6 7.4 5.6 4.8 9.6 4.6 5.1
```

Terminal | fichier matrice