Cours langage C

Cours du 22/09/2020

Dernièrement ils ont fait les variables globales et tableau unidimensionnel

Initialisation du tableau exemple :
int tab[5] = {1, 2, 3, 4, 5} ;

Exemple remplissage d'un tableau

```c
#include <stdio.h>

#define NMAX 20

int main(){
    float notes[NMAX], moy=0.0;
    int i;

    for(i = 0; i < NMAX; i++){
        printf("\nEntrez la note : ");
        scanf("%f", &notes[i]);
        moy += notes[i];
    }

    moy /= NMAX;
    printf("\nLa moyenne est : %f", moy);

    return 0;
}
```

Tableau multidimensionnels

déclaration :
type nom_tab[dim1][dim2]...[dimN]

ex :
int tab[5][2] = { {1,2}, {3,4}, {5,6}, {7,8}, {9,10}} ;

Les pointeurs

La mémoire d'un ordinateur est organisée en une suite de « cases » repérées par une adresse
Les variables d'un programme sont stockées en mémoire, et possèdent (ou non) une valeur.

Qu'est ce qu'un pointeur ?

En règle général un pointeur permet d'accéder a une variable

- Objet contenant une référence (adresse) sur un autre objet
- Il est déclaré avec le type de la variable sur laquelle il pointe

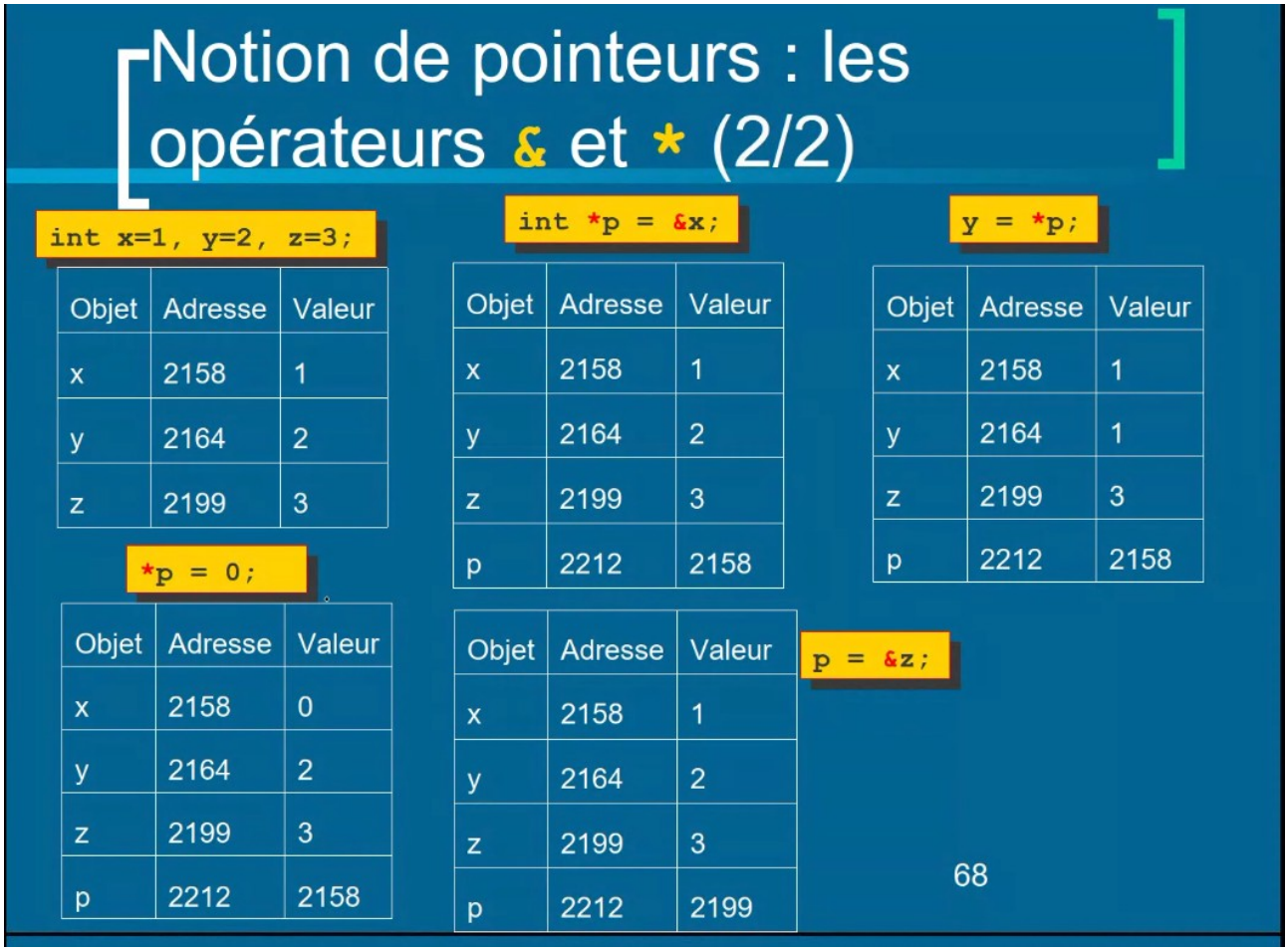
Ex :
type *nom_pointeur ;

Un pointeur qui ne pointe sur rien envoie la constante NULL.

Pour obtenir l'adresse d'un objet on utilise « & » (s'applique aux objets mémoire et élements de tableaux (pas aux expressions, constantes, …)

L'opérateur unaire « * » représente l'opérateur de déréférence.
Il donne accès à l'objet pointé par un pointeur

```c
int x = 1, y = 2, z = 3;

int *p; // p est un pointeur sur un entier

p = &x; // p pointe maintenant sur x
y = *p; // y vaut désormais 1
*p = 0; // x vaut désormais 0
p = &z; // p pointe désormais sur z
```



Exemple de programme

```c
#include <stdio.h>

int main(){
    int u1, u2, v = 3;
    int *p; // déclaration d'un pointeur
    u1 = 2 * (v + 5);
    p = &v; // pointeur pointe sur l'addresse de v
    *p = 5; // la valeur de v change à 5
    u2 = 2 * (*p + 5);
    printf("\n u1 = %d, u2 = %d, v = %d", u1, u2, v);

    return 0;
}
```

Quelques source d’erreurs :

```c
p = v; // fait pointer p à l'adresse 3
p = 5; // fait pointer p à l'adresse 5
*p = &v // met l'adresse de v dans la case pointée par p
```

Allocation dynamique de pointeurs

- On peut initialiser un pointeur « p » en lui affectant l’adresse d’une autre variables

1 – il faut d’abord réserver à *p un espace mémoire de taille adéquate.
L’adresse de cet espace mémoire sera la valeur de p.
On parle d’allocation dynamique

2- fonction « malloc » de la librarie stdlib.h

ex : nom_pointeur = (type *)malloc(sizeof(type))

```c
#include <stdlib.h> // include lib

int main(){
    int *p; // pointeur p

    p = (int *)malloc(sizeof(int)); // allocation dynamique

    free(p); // libère la mémoire
}
```

Problèmes d'allocation mémoire

- Le système ne peut plus allouer de mémoire
- Des zones réservées n'ont pas été liberées
- La taille des zones réservées est trop grande
- Manipulation de données dans une zone de mémoire non réservée
- Dans tous les cas il faut vérifier que la zone mémoire est bien valide (test sur la nullité du pointeur sur cette zone)

Exemple :

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int *p;

    do{

        // allocation zone de 2**17 bites
        p = (int *)malloc(131072*sizeof(int));
        printf("%p\n", p);

    }while(p != NULL);

    return 0;
}
```

Saturation de la mémoire qui peut afficher ce message :
***** malloc[1307] : error : can't allocate region**
0x0

Les pointeurs et le passage en argument de fonction

Le C passe les arguments des fonctions par valeur
- On transmet en fait une copie de la valeur
- La fonction appelée ne modifie que localement la variable transmise par la fonction appelante

En faisant passer des pointeurs comme argument, on peut accéder au contenu des cases pointées dans la mémoire.

Exemple de passage en argument

```c
#include <stdio.h>

void fct(int *x){
    *x = *x + 2;
```

```
}

int main(){
    int a = 3;
    fct(&a);
    printf("a = %d", a);
}
```

Tableaux et pointeurs

Déclarer un tableau de taille données c'est réserver l'emplacement mémoire correspondant :

```
int tab1[5] = {1,2,3,4,5};
int *tab2;

tab2 = (int *)malloc(5 * sizeof(int));
```

Accès aux éléments du tableau :

```
*tab = 1; // on met 1 dans la case pointées
tab++; // on pointe sur la case suivant
*tab++ = 2; // on met 2 dans la case pointée et on pointe sur la case suivante
tab[0] = 3; // on met 3 dans la 1 ere case pointée par tab2
tab[1] = 4; // on met 4 dans la 2 eme case pointée par tab2
tab += 2; // on pointe deux cases plus loin
*tab = 5; // On met 5 dans la case pointée
```

Une mémoire est linéaire et chaque case correspond a 1 octet

Un double pointeur, est un pointeur qui pointe sur un pointeur et se déclare :
« int **ptr », il permet également de parcourir un tableau deux dimensions.

Exemple : (code qu'il a écrit a la main)

```
#include <stdio.h>
#include <stdlib.h>

#define N 17

int main(){
    int **tab2; // pointeur pour tableau
    int i;
    tab2 = (int**)malloc(N * sizeof(int*)); // 1)

    for(i = 0; i < N; i++){
        tab2[i] = (int*)malloc(N * sizeof(int)); // on remplie le tableau
    }
}
```

les chaînes de caractères

- Les chaînes de caractères sont des tableaux de caractères
- Chaque case du tableau contient un caractère
- Le dernier élément est toujours ‘\0’ (null byte)
- Pour stocker une chaîne de n éléments, il faut n + 1 case (à cause du ‘\0’)
- Lorsqu’on déclare une chaîne de caractères en « char * » il ne faut pas oublier d’allouer l’espace mémoire nécessaire (malloc)

Structures

- Structure de données complexe permettant de regrouper des données de types différents
- C’est un nouveau type différencié par un nom
- Déclaration de variable de ce type
- Peut s’initialiser comme un tableau

```c
struct point{
    int x;
    int y;
    char couleur[10];
};

struct point a = {2, 3, "rouge"};
```

- Accès aux membres :
    - Structure statique : opérateur ‘.’
    - Structure dynamique : opérateur ‘→’
- Les membres s’utilisent comme des variables
- On peut copier et affecter des structures
- On ne peut pas les comparer
- On peut construire des structures de structures

```c
struct point a, *b;
...
a.x
b->x
```

Fonctions de structures

- Une fonction peut prendre des structures comme arguments
- Une fonction peut renvoyer une structure

```
struct point addpoint(struct point a, struct point b){
    a.x += b.x;
    a.y += b.y;
    a.couleur = b.couleur;
    return a;
}
```

a.couleur = b.couleur => ce n'est pas une copie !

Tableaux de structures

- Il est possible de construire des tableaux de structure :
    - Déclaration : « struct point tab[3] »
    - On ne peut accéder qu'élément par élément au tableau et champs par champs à la structure : tab[2].x

Intérêt

- Le programmeur pourra utiliser une structure quand les objets à manipuler ne peuvent se résumer à :
    - Une seule valeur
    - Un ensemble de valeurs de même type
- Par exemple, un type de donnée pour représenter les informations relatives à un individu (nom, prénom, taille …)

Fonctions standard

- Cet ensemble regroupe les fonctions :
    - D'affichage
    - De saisie de caractères
    - D'accès à des fichiers
    - De manipulation de chaînes de caractères
    - De fonctions mathématiques ..
-Elles possèdent toute un « man »

Fonctions math.h

Fonctions trigonométriques			Fonctions élémentaires		
`sin(x)`	`cos(x)`	`tan(x)`	`exp(x)`	`pow(x,y)`	`sqrt(x)`
`asin(x)`	`acos(x)`	`atan(x)`	`log(x)`	`ceil(x)`	`floor(x)`
`sinh(x)`	`cosh(x)`	`tanh(x)`	`log10(x)`	`fabs(x)`	`fmod(x,y)`

Nécessite l'option **`-lm`** à la compilation

```
gcc -o prg prg.c -lm
```

-o => nomme le fichier de sortie
-lm => lier avec la lib m

Librairies string.h

- **char *strcpy(char *s, char *ct)** : copie **ct** dans **s** et renvoie **s**
- **char *strcat(char *s, char *ct)** : concatène **ct** à la fin s et renvoie **s**
- **int strcmp(char *s, char *ct)** : compare **s** et **ct** et renvoie un nb < 0 si s < ct, 0 si s == ct et un nb > 0 sinon
- **size_t strlen(char *c)** : renvoie la longueur de la chaîne c (entier)

Cours du 29/09

Entrée-sortie, fichiers

Une entrée-sortie (E/S) :
- Transfert d'informations entre la mémoire de l'ordinateur et l'un de ses périphériques (écran, clavier, disque …)

Bloc : volume de données traité lors d'une opération de transfert

Fichier : suite de blocs

Modes d'accès aux données

- Accès séquentiel
  - Accès dans un fichier à un bloc en lisant préalablement les blocs le précédant

- Accès direct
    - Accès direct à n'importe quel bloc

Descripteur de fichier

- contient les informations nécessaires à l'utilisation d'un fichier
- les droits d'accès à ce fichier
- les conditions d'accès au fichier (lecture ou écriture)
- la position courante dans le fichier
- En C, le descripteur de fichier est une structure de type FILE

```
FILE *fd // descripteur de fichier
```

// mettre le screen machin

Ouverture de fichier

Permet d'ouvrir un fichier présent sur le DD

En C on ouvre un fichier avec la commande fopen :

```
FILE * fopen(char * filename, char *mode);
```

(r => lecture seule, w => fichier ecriture seule, rw => lecture / ecriture)

Exemple d'ouverture

```
FILE *fd;
fd = fopen("toto.txt", "r");

if (fd == NULL)
{
 printf("Erreur ouverture");
 exit(1);
}
```

Opérations d'écriture/lecture dans un fichier texte

```
int fprintf(FILE *flux, const char *format) // lis expression
int fscanf(FILE *flux, const char *format) // liste adresses
int fputs(const char* chaine, FILE *flux);
char *fgets(char *chaine, int nbCar, FILE *flux);
```

Exemple :

```
char nom;
int x, y, nbVals;
FILE *entree;

entree = fopen("monfichier", "r");

if(!entree){
   printf("Erreur d'ouverture du fichier");
   exit(-1);
}

while(1) {
  nbVals = fscanf(entree, "%c%d%d", &nom, &x, &y);
  if(feof(entree)) {
     if(nbVals == EOF) {
         printf("Fin de fichier normale");
         break;
      }
      else {
          printf("Fin de fichier anormale!!");
          break;
        }
    }
}
```

Lecture binaire en C

- Lecture du blocs d'octets consécutifs dans un fichier par la fonction « fread »
- Prototype

```
size_t fread(void *adr, size_t taille, size_t nblocs, FILE *fd);
```

« adr » : addresse en mémoire ou serotn stockées les données lues
« taille » : taille en octet d'un bloc de données
« nblocs »:nombre de blocs à écrire
« fd »:descripteur de fichier
Valeur de retour : nombre de blocs effectivement lus

Lecture binaire en C (2)

```
nbBlocs = fread(&monPoint, sizeof(struct point), 1, entree);
```

Ecriture dans un fichier binaire

Ecriture de blocs d'octets consécutifs dans un fichier par la fonction « fwrite »

```
size_t fwrite(const void *adr, size_t taille, size_t nblocs, FILE *flux)
```

Exemple

```
struct point { char nom; int x, y; }
struct point point[NB_POINTS];
int nbBlocs;
FILE *sortie;

sortie = fopen("monFichier", "wb");
if (sortie == NULL) {
  printf("Impossible de créer le fichier");
  exit(-1);
}
nbBlocs = fwrite(&points[0], sizeof(struct point), NB_POINTS, sortie);
if (nbBlocs != NB_POINTS) {
  printf("Erreur d'écriture dans le fichier");
  exit(-1);
}
fclose(sortie);
```

Fermeture d'un fichier

La fonction pour fermet un fichier est : flcose() ;

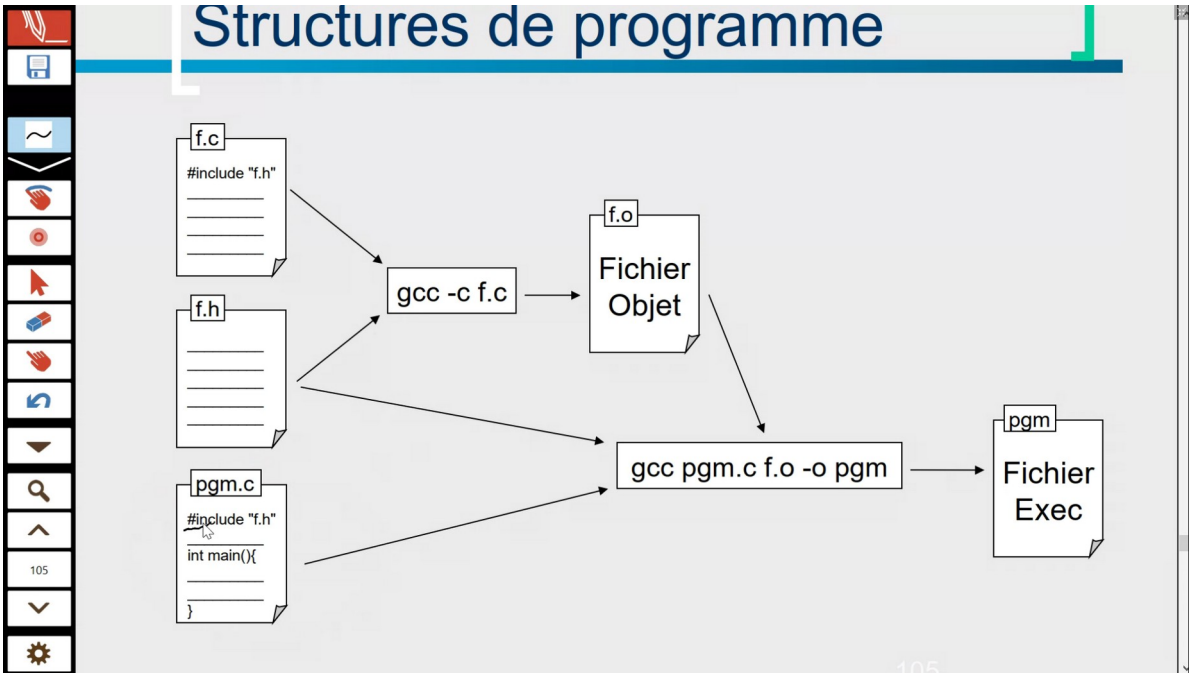
Exemple :

fclose(fd) ;

Structuration de programmes

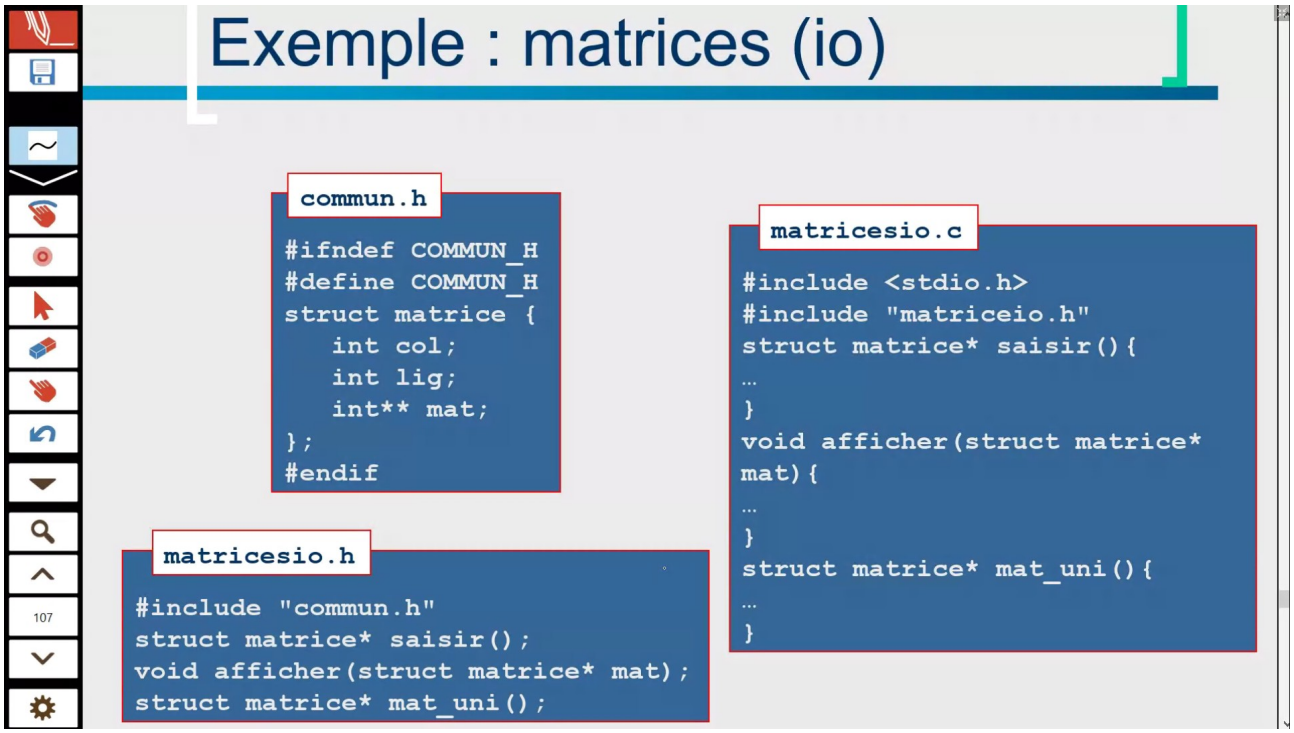
- Découpage d'un programme en plusieurs fichiers
    - Clarté du code
    - Réutilisation de fonctions communes
    - Compilation plus rapide

- Un fichier principal qui contient le main
- Des fichiers secondaires regroupant les fonctions et définitions communes



Dans le fichier d'entête on met :
- Prototype des fonctions
- Définition de nouveaux types (structures)
- Définition de variables communes (globales)
  - Attention aux effets de bords
  - A utiliser le moins possible

Exemple : matrices (io)



Cours du 30/09

Le typedef renomme une variable

Exemple

typedef int toto ;
toto a, b ;

typedef struct point chose ;
chose aa, bb ;

Tri par sélection

Dans un tableau non trié on va faire varier
i de 0 à n – 2 et j de i + 1 à n – 1

Exemple :

```c
#include <stdio.h>
#define N 8

void tri_s(int *tab, int n){
    int i, j, petit, i_petit;
    int tmp;
    for (i = 0; i < n - 1; i++)
    {
        i_petit = i;
        petit = tab[i];
        for (j = i; j < n; j++)
        {
            if (tab[j] < petit)
            {
                petit = tab[j];
                i_petit = j;
            }
        }
        tmp = tab[i_petit];
        tab[i_petit] = tab[i];
        tab[i] = tmp;
    }
}


int main(){
    int t[N] = {4,6,1,6, 9,0,-1,3};
    int i;
    tri_s(t, N);
              for(i = 0; i < N; i++){
                  printf("|%d|", t[i]);

      }
         printf("\n");

}
void tri_i(int *tab, int n){
    int i, j, tmp;

    for(i = 1; i < n; i++){
        j = i;
        while((j > 0) && (tab[j] < tab[j - 1])){
            tmp = tab[j - 1];
            tab[j - 1] = tab[j];
            tab[j] = tmp;
            j--;
        }
    }
}
```

Sortie : -1 0 1 3 4 6 6 9

Tri par insertion

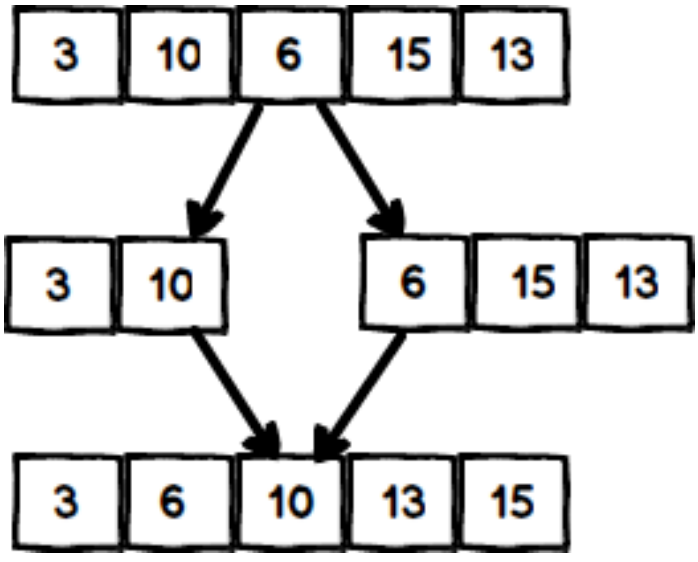
```
void tri_i(int *tab, int n){

int i, j, tmp;

    for(i = 1; i < n; i++){
        j = i;
        while((j > 0) && (tab[j] < tab[j - 1])){
            tmp = tab[j - 1];
            tab[j - 1] = tab[j];
            tab[j] = tmp;
            j--;
        }
    }
}
```

Sortie : - 1 0 1 3 4 6 6 9

Tri fusion

F_entrée un tableau d'entiers

On coupe le tableau en deux et on tris via le tri fusion, ce qui va donner un tableau coupé en deux ou la partie de gauche et la partie de droite sont triés.
A partir de la, il va y avoir une fusion entre les deux parties pour que à la fin le tableau soit trié.



T(n) = θ (n x log n)

Exemple :
// Trouver un code tri fusion car code du prof bug

Opérations sur les matrices

Modélisation d'une matrice en C peut se faire de plusieurs facon

- Tableaux deux dimensions a[i][j]
- Tableaux a[i * colonne + j]

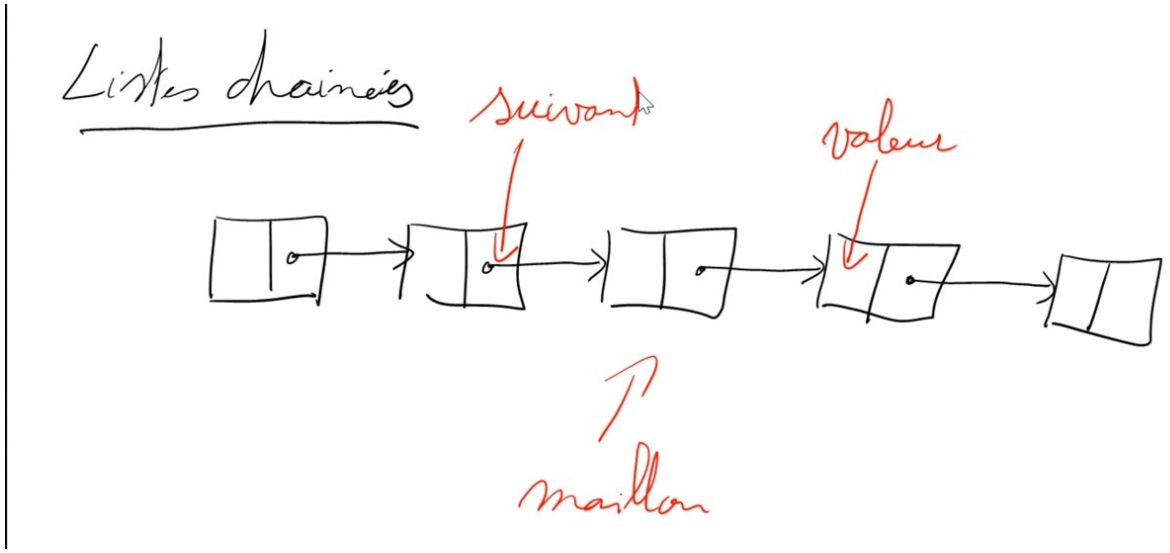
Ci,j = Ai,j + Bi,j

Cours du 7/10

Mutiplication de matrice

```c
void multi_m(int *a, int *b, int *c, int l, int col){

      int i,j,k;

        for(i = 0; i < l; i++){
                for(j = 0; j < col; j++){
                        c[i * col + j] = 0;
                        for(k = 0; k < col; k++){
                                c[i * col + j] += a[i * col + k] * b[k * l + i];
                        }
                }
        }
}
```

Liste chaînées



Le premier maillon est la la « tête », on ne peut accéder a une liste chaînées uniquement avec la tête.
Le dernier maillon est la « queue » de la liste.
On accède à la tête de la liste via son adresse via un pointeur.

Modèle

Exemple :

```
int liste_vide(struct maillon *tete){
        return (tete == NULL);
}



struct maillon{
        int valeur;
        struct maillon *suivant;
};
```

Ajout d’un élément dans une liste chaînée

Exemple de code

```
 struct maillon{
        int valeur;
        struct maillon *suivant;
};


void ajout(struct maillon** tete, int e){
        struct maillon *ptr;
        ptr = (struct maillon*)malloc(sizeof(struct maillon));
        ptr->valeur = e;
        ptr->suivant = *tete;
        *tete = ptr;
}

int main(){

        struct maillon *l;
        l = NULL;
        ajout(&l, 17);


}
```

Cours du 12/10

Rechercher un élément dans la liste chaînées

```
int recherche(struct maillon *t, int e){
        struct maillon *ptr;
        ptr = t;
        while(ptr != NULL){
                if(ptr->valeur == e)
                        return 1;
                ptr = ptr->suivant;
        }
        return 0;
}
```

Supprimer un élément dans la liste chaînées

```
void supprime(struct maillon **l, int e){

       struct maillon *ptr, *prec = NULL;

        if(ptr != NULL){
                ptr = *l;
                while((ptr != NULL) && (ptr->valeur != e)){
                        prec = ptr;
                        ptr = ptr->suivant;
                }

                if(ptr != NULL){
                        if(prec == NULL){
                                *l = ptr->suivant;
                        }else {
                                prec->suivant = ptr->suivant;
                        }

                free(ptr);
          }
   }

}
```

Insertion en queue

```
void inser_q(struct maillon **l, int e){
        struct maillon *ptr, *new;
        ptr = *l;
        new = (struct maillon*)malloc(sizeof(struct maillon));
        new->valeur = e;
        new->suivant = NULL;

        if(ptr == NULL){
                *l = new;
        }else{
                while(ptr->suivant != NULL ){
                        ptr = ptr->suivant;
                }
                ptr->suivant = new;
        }
}
```

Cours du 19/10

Liste triée

- Trouver le bon endroit : ptr→ suivant → valeur > e
- Insertion en tête : ptr → valeur > e
- Insertion en queue : ptr → suivant == NULL
- Liste vide : tete == NULL

```
void insere_t_c(struct maillon **t, int e){
  struct maillon *new, *ptr;
        new = (struct maillon*)malloc(sizeof(struct maillon)); new->valeur = e;
        ptr = *t;

        if((ptr == NULL) || (ptr->valeur > e)){
                new->suivant = *t;
                *t = new;
        }
        else{
                while((ptr->suivant != NULL) && (ptr->suivant->valeur < e))
                        ptr = ptr->suivant;
                        if(ptr->suivant == NULL){
                                new->suivant = NULL;
                                ptr->suivant = new;
                        }else{
                                new->suivant = ptr->suivant;
                                ptr->suivant = new;

                }
        }
}
```

Pils et fils

Il existe deux « type » de pils :
- LIFO pour Last In First Out
- FIFO pour First In First Out

Empiler :

```
int empile(struct maillon **sommet, int e){
        struct maillon* new;
        new = (struct maillon*)malloc(sizeof(struct maillon));

        if(new == NULL)
                return 0;

        new->suivant = *sommet;
        *sommet = new;
        new->valeur = e;
        return 1;
}
```

Depiler

```
int depiler(struct maillon **sommet){
        struct maillon *ptr;
        int val;

        if(*sommet != NULL){
                ptr = *sommet;
                *sommet = ptr->suivant;
                val = ptr->valeur;
                free(ptr);
                return val;
        }

}
```

Enfiler

```
int enfile(struct maillon **queue, struct maillon **tete, int e){
        struct maillon *ptr;
        ptr = (struct maillon*)malloc(sizeof(struct maillon));
        if(ptr == NULL)
                return 0;
        ptr->valeur = e;
        ptr->suivant = NULL;
        if(*queue != NULL)
        {
                (*queue)->suivant = ptr;
                *queue = ptr;
        }
        else{
                *tete = ptr;
                *queue = ptr;
        }

        return 1;
}
```

Cours du 26/10 – Dernier cours

Algo sur des chaînes de caractères

```
void modif(char *chaine){
    int i;
    i = 0;
    while(chaine[i] != '\O'){
        if(chaine[i] == 'e')
            chaine[i] == 'E';
        i += 1;
    }
}
```

Recherche d'une sous-chaîne

```
#include <string.h>
#include <stdio.h>

int cherche_str(char *c, char *k){
    int ic, ik, lc, lk;
    lc = strlen(c);
    lk = strlen(k);

    for(ic = 0; ic <= (lc - lk); ic++){
        ik = 0;
        while( (ik < lk) && (k[ik] == c[ic+ik]) ){
            ik++;
        }
        if(ik==lk)
            return ic;
    }

    return -1;
}

int main(){
    char *ch1 = "J'aime la choucroute";
    char *ch2 = "choucroute";
    printf("Le résultat est %d \n", cherche_str(ch1, ch2));
}
```