INITIATION A LA PROGRAMMATION EN C

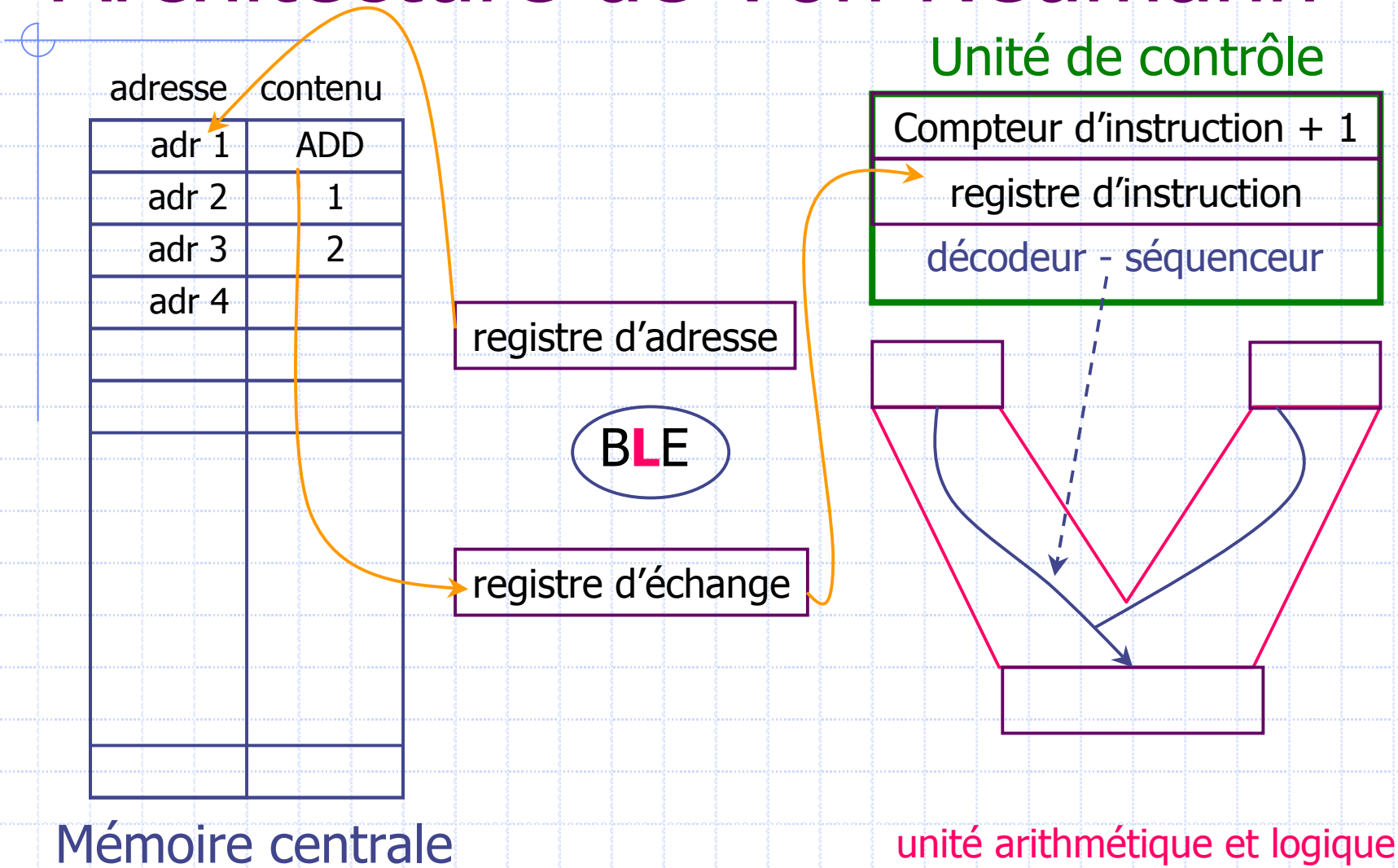
OBJECTIF : ETRE CAPABLE DE CRÉER OU MODIFIER
DE PETITS OUTILS DE TRAITEMENT D'IMAGE

Denis MARIANO-GOULART
Service de médecine nucléaire.
CHRU Lapeyronie. Montpellier.
http:\\scinti.etud.univ-montp1.fr

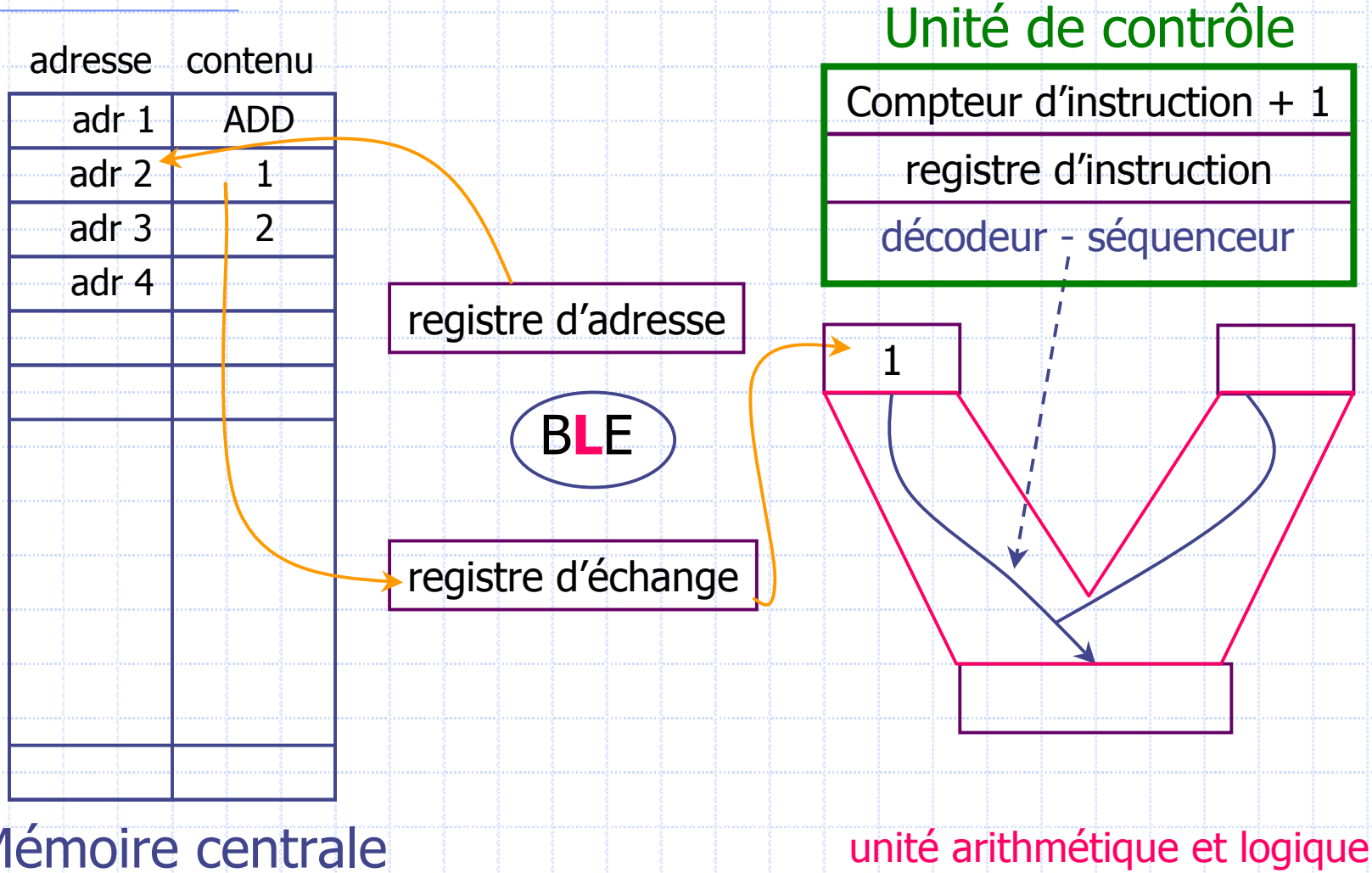
Plan

1. Architecture des ordinateurs
2. Création d'un programme exécutable
3. Le noyau du langage C
4. Quelques fonctions de la bibliothèque
5. Exemple d'application :
   Réalisation d'un filtrage linéaire d'image dicom

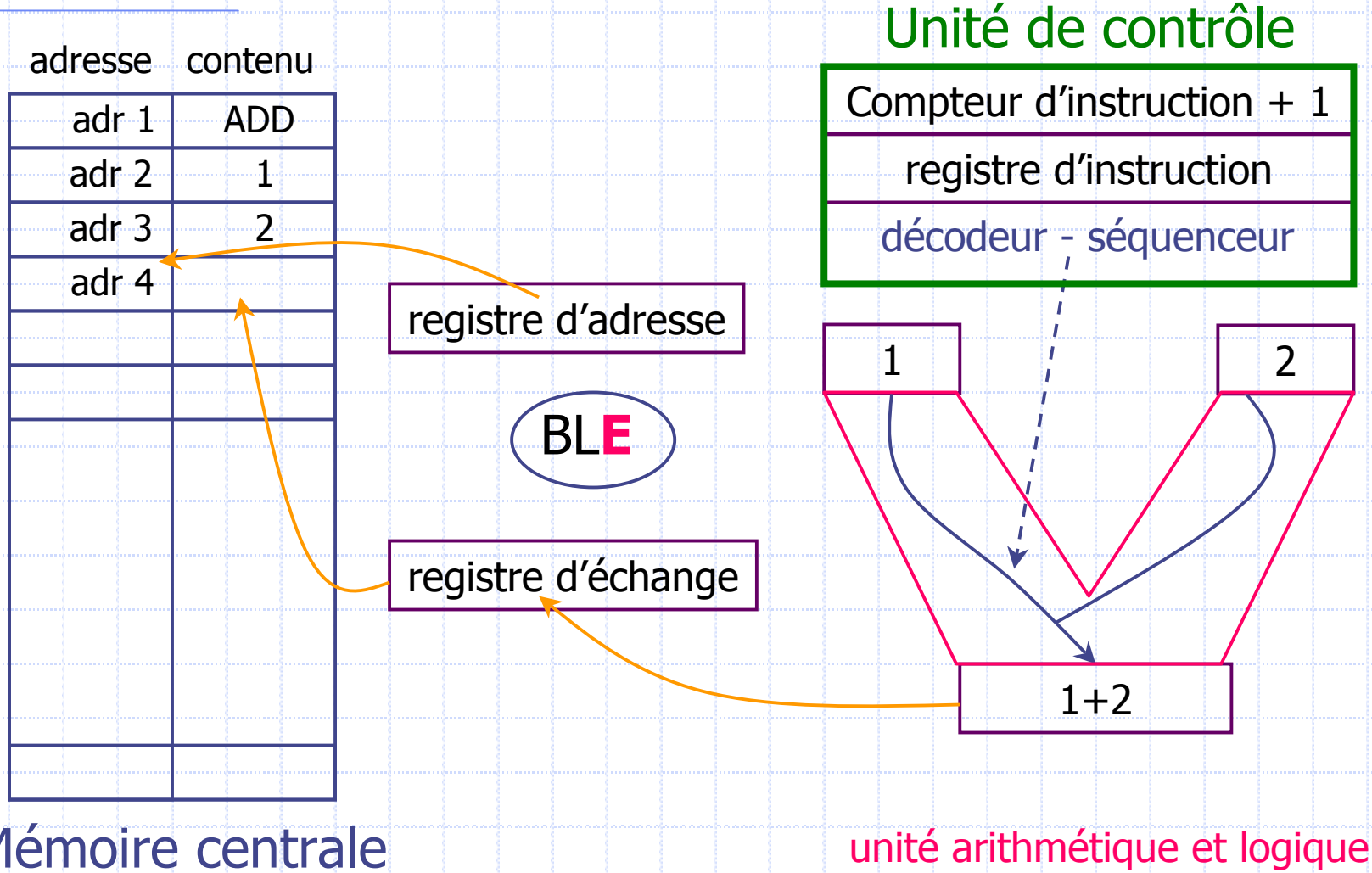
Architecture de Von Neumann



Architecture de Von Neumann



Architecture de Von Neumann



Codage des données et instructions

- indistinctement en mémoire centrale
- au moyens de dispositifs électroniques bistables : 2 états de tension
- codés par un bit $b \in \{0, 1\}$
- association de bits pour coder des données et des instructions variées.

Codage des instructions et des nombres naturels

- 1 bit : 2 valeurs possibles : 0 ou 1
- 1 octet = 8 bits = 256 valeurs = 2^8

0	0	0	0	0	0	0	0

= 0 à

1	1	1	1	1	1	1	1

= 255

$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

0	0

F	F

$15.16^1 + 15.16^0$

- 1 mot = 2 octets = 16 bits
  - de 0 à $2^{16}-1 = 65\,535$

Codage des entiers signés

- Le premier bit sert à coder le signe
- 1 octet = 1 bit de signe + 7 bits
  - code un entier de –128 à + 127
- 2 octets = 1 bit de signe + 15 bits
  - Code un entier de –32 768 à 32767

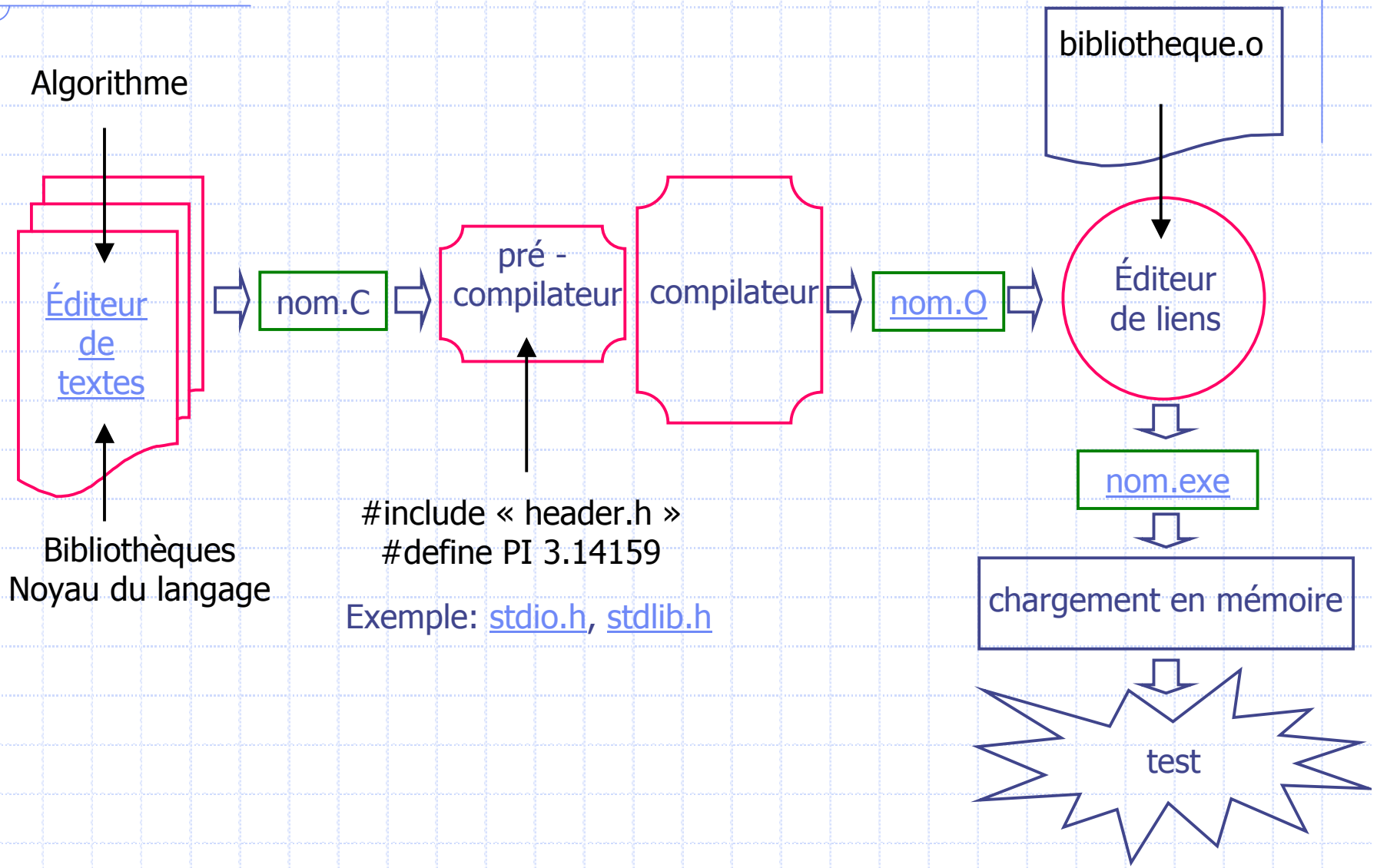
Codage des données numériques

Type	Type (C)	bits	Limites (de –L à L-1)
caractère/entier	char	8	L = 128
entier court	short int	16	L = 32768
entier (long)	(long) int	32	L = 2 147 483 648
nombre réel	float	32	$\pm 3{,}4.10^{\pm 38}$
réel double précision	double	64	$\pm 1{,}7.10^{\pm 308}$
réel très grande précision	Long double	80	$\pm 3{,}4.10^{-4932}$ à $\pm 1{,}1.10^{4932}$

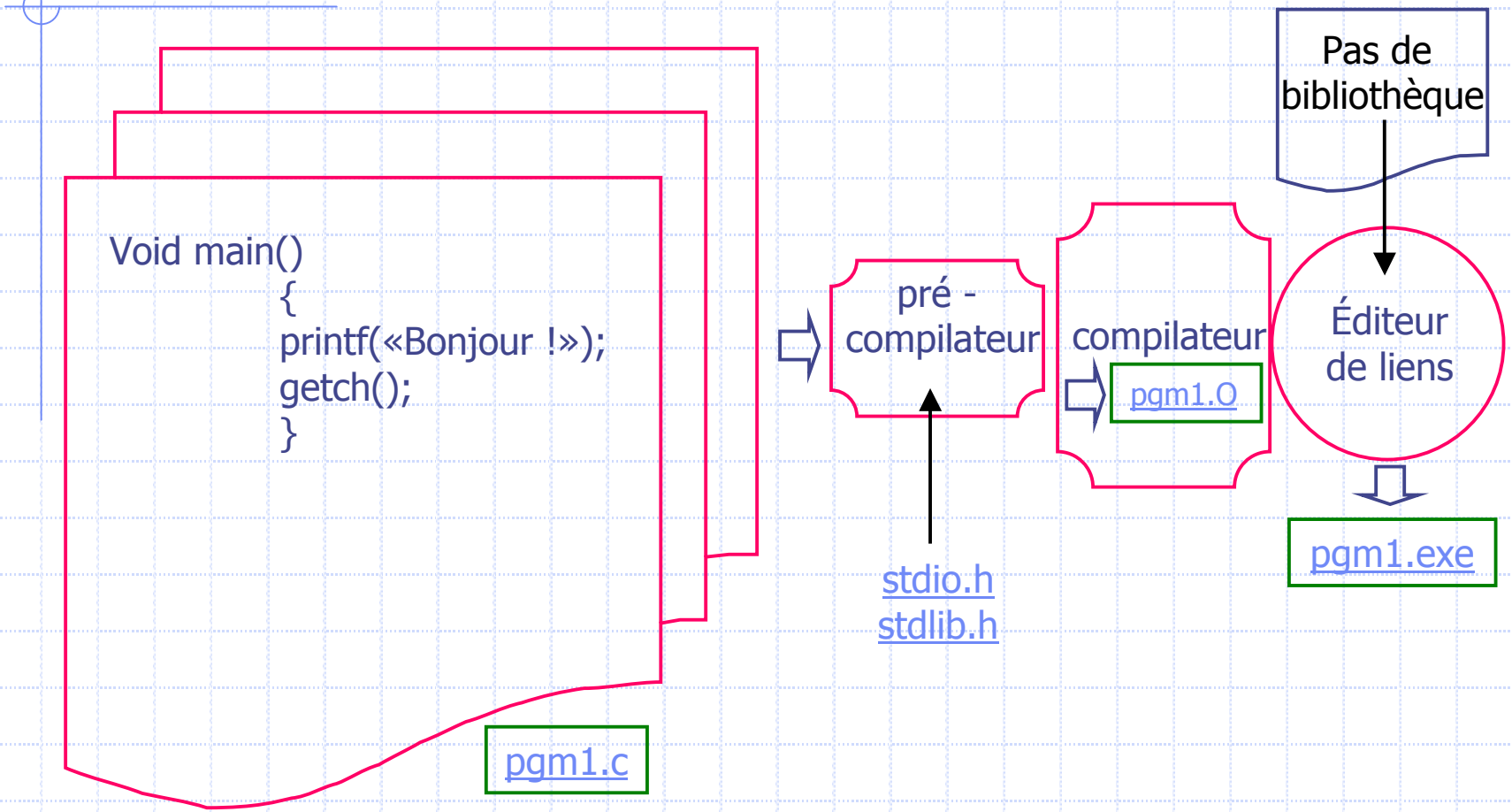
Point d'étape 1

- Instructions et données codées en binaire dans la mémoire centrale
- un programme et ses données sont illisibles par un être humain
- nécessité d'un langage de programmation lisible dont le texte sera ensuite traduit en binaire par un programme, le compilateur.

Etapes de programmation



Exemple de programmation



Syntaxe de base du C

- C
  - mot
    - opérateur
      - affectation : `a = 3;`
      - arithmétique : `++a; --a; a=10%3; a=-a; a=a+a;`
      - logique : `== != && ||`
      - sur bits : `& | ^ <<1 >>1 ~`
      - pointeurs : `*p; &a;`
      - structures : `->`
    - identificateur
      - mot réservé
        - type de variable: char, int, float…
        - structures de contrôle: tests, boucles
      - nom symbolique de variable, fonction, sous-programme
  - séparateur
    - `{ }` bloc d'instructions, structures, tableaux
    - `( )` arguments de fonction, priorités, conversion
    - `[ ]` tableaux
    - `,` liste d'arguments
    - `.` Élément d'une structure
    - `;` fin d'instruction
    - `:` étiquette

Déclaration des variables

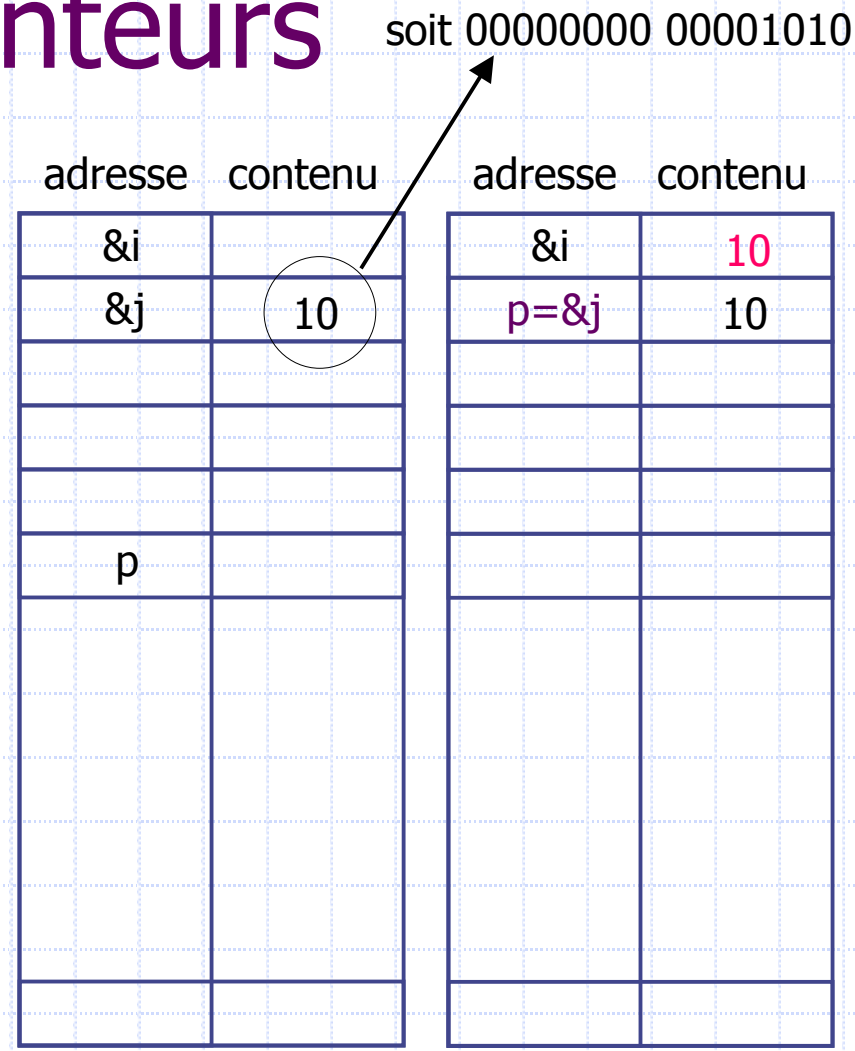
```
Void main()
 {
 short int i,j;
 float b;

 i=10;
 j=20;
 b=(float)(i)/(float)(j);

 printf("i =%d     j=%d\n",i,j);
 printf("reel : i/j = %f\n",b);
 printf("entier : i/j = %d\n",i/j);
 getch();
 }
```

La déclaration permet
au compilateur
de réserver une adresse
et un espace mémoire
pour la variable

pgm2.c

Variables et pointeurs

```
void main()
 {
 short int i, j=10;
 short int *p;

 p=&j;
 i=*p;

 printf("%d %d\n",i,*&i);
 printf("%d %d\n",j,*&j);
 printf("adresse i = %d\n",&i);
 printf("p = %d\n",p);
 printf("adresse j = %d\n",&j);
 getch();
}
```



** = « contenu de l'adresse… » & = « adresse de la variable… »*

pgm3.c

Tableaux de variables (images)

- Allocation fixe:
  - short int a[9] ;
  - a[0]=1; a[1]=2;
- Allocation dynamique
  - short int *a;
  - a = malloc(9*sizeof(int));
  - *(a)=1; *(a+1)=2;
  - free(a);

adresse	contenu
a	1
a+1	2
a+2	
a+3	
a+4	
a+5	
a+6	
a+7	
a+8	

*NB: dans *(a+1), le déplacement de 1 est ajusté en unité –nombre de bits- correspondant au type de la variable : dans cet exemple, l'adresse se décale de 16 bits.*

Structures de contrôle

- if(condition) {instructions;} else {instructions;}

  if(b>a) { c=a; a=b ; b=c; } else c=0;

- while (condition) {instructions;}

  c=2; while(c<a) c=c*2;

- do {instructions;} while (condition);

  c=1; do c=c*2; while(c<a);

- for(initialisation, condition, incrément) {instructions;}

  for(i=0;i<100;i++) a[i]=3*i;

Fonctions et sous-programmes

```
/* Définition du prototype */
int somme(short int i, short int j);

/* Programme principal */
void main()
 {
 short int i, j, k1, k2;
 i=1;
 j=2;

 k1=somme(i,j);
 plus(i,j,&k2);

 printf("%d   %d\n",k1,k2);
 getch();
}
```

```
/* fonction */
int somme(short int x, short int y)
 {
 short int z;

 z = x+y;
 return(z);
 }
```

```
/* sous-programme équivalent */
void plus(short int x, short int y, short int *z)
 {
 *z = x+y;
 }
```

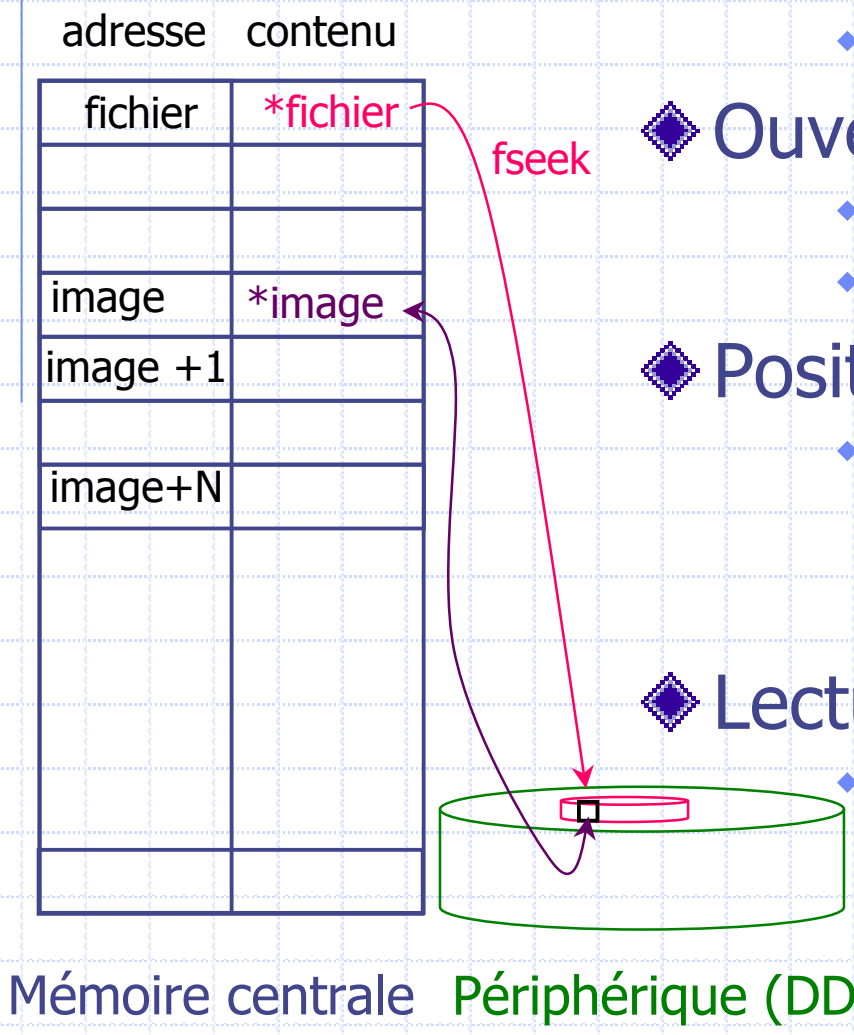
Attention à la nécessité de passer l'adresse de z au sous-programme, et non seulement son nom

pgm4.c

Entrées/sorties écran et clavier

- Affichage formatté à l'écran
  - int printf(const char *format, arguments)
  - ex : printf(« la variable vaut %3.3f\n », b);
- Entrée de texte au clavier
  - int scanf(const char *format, arguments)
  - ex : scanf(«%d»,&i);

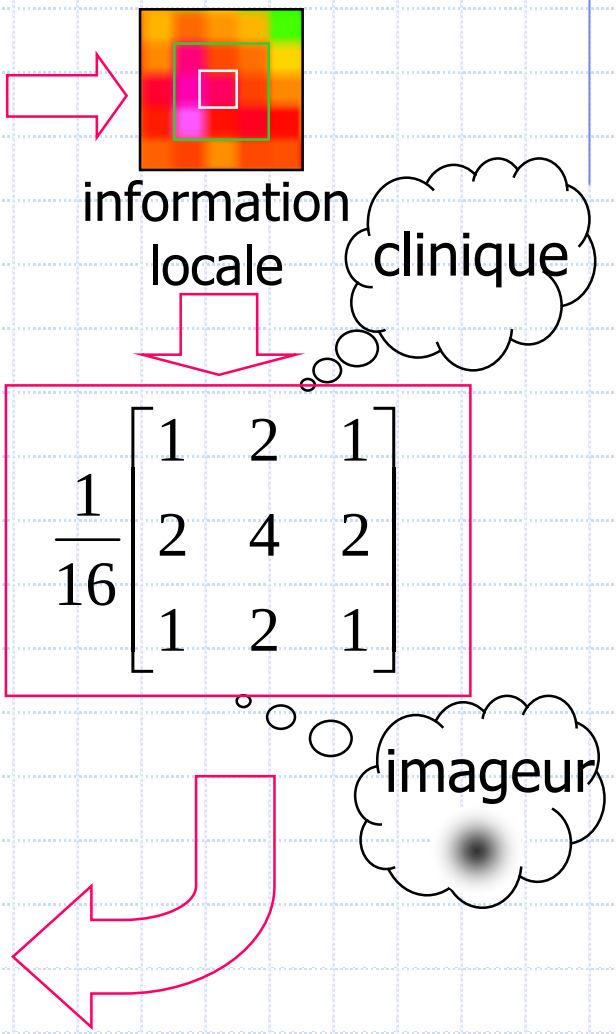
Lecture et écriture de fichiers



- Pointeur de fichier
  - FILE *fichier;
- Ouverture/fermeture d'un fichier
  - fichier = fopen(«nom_fichier», rb ou wb)
  - fclose(fichier)
- Positionnement dans un fichier
  - fseek(fichier, décalage, SEEK_XXX)
    - SEEK_SET (depuis le début),
    - SEEK_CUR (depuis la position courante)
    - SEEK_END (depuis la fin).
- Lecture/écriture dans un fichier
  - fread ou fwrite(image, taille, N, fichier)

Exemple d'application

ant.dcm



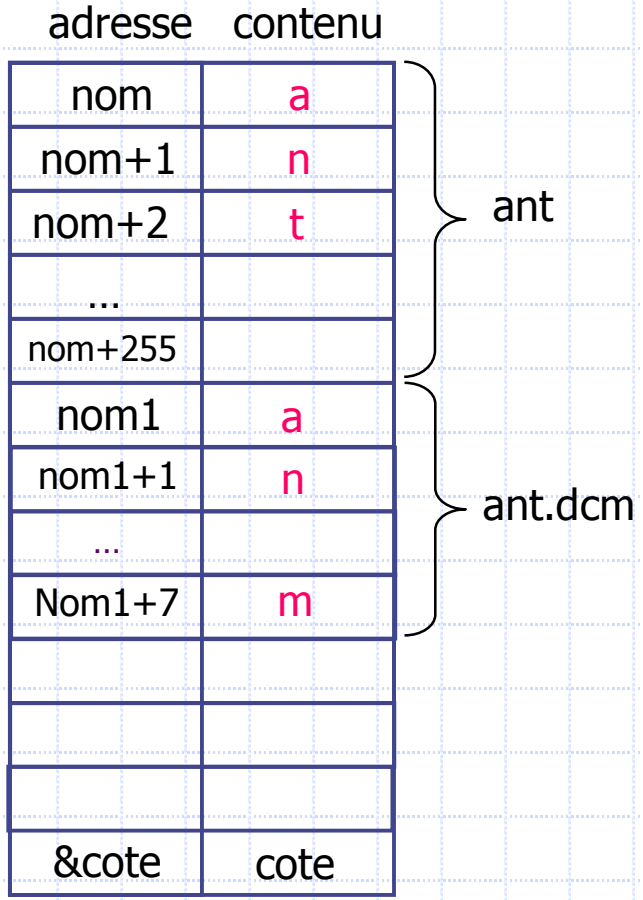
Etape 0 : plan de travail

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

int main()
    {
     /* Déclaration des variables */
    type nom ;
    /* Lecture du nom de l'image dcm */
    /* copie des pixels du DD vers la mémoire vive */
    /* filtrage des pixels dans la mémoire vive */
    /* recopie des pixels sur le disque dur */
    /* Fin de programme */
    }

void fonction()
    {
     /* action répétée */
    }
```

Etape 1 : Saisies clavier

instn1.c

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

int main()
    {
    char nom[256], nom1[256];
    int cote;

    printf("\nImage dicom à ouvrir : ");
    scanf("%s",nom);
    sprintf(nom1,"%s.dcm",nom);

    printf("\nTaille d'image  : ");
    scanf("%d",&cote);
    }
```

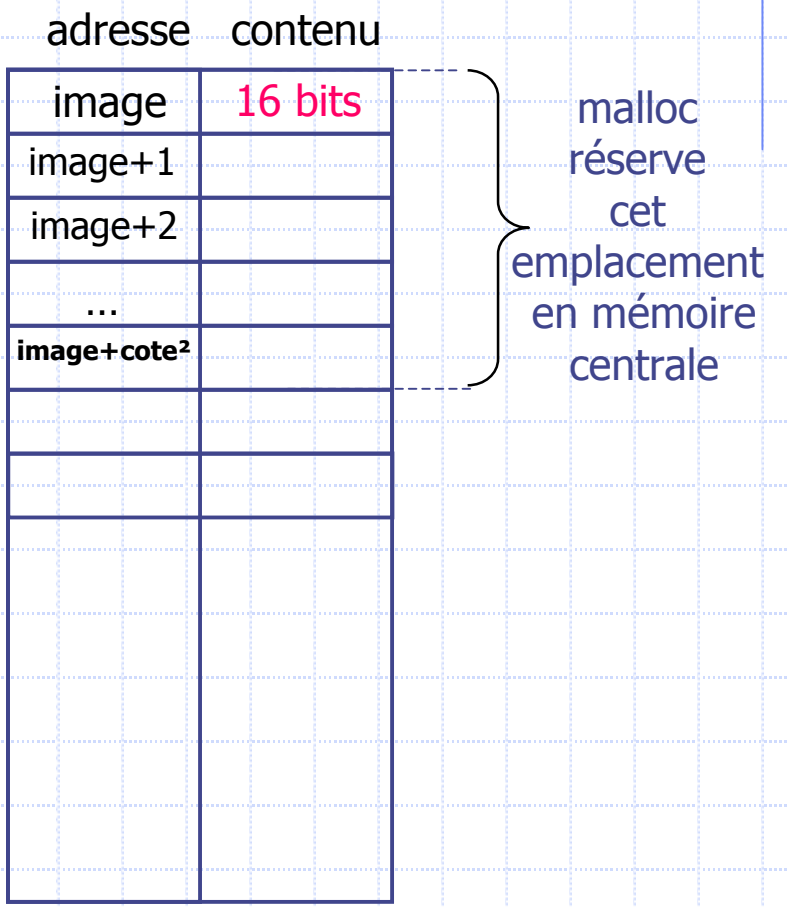


Mémoire centrale

Etape 2 : allocation

instn2.c

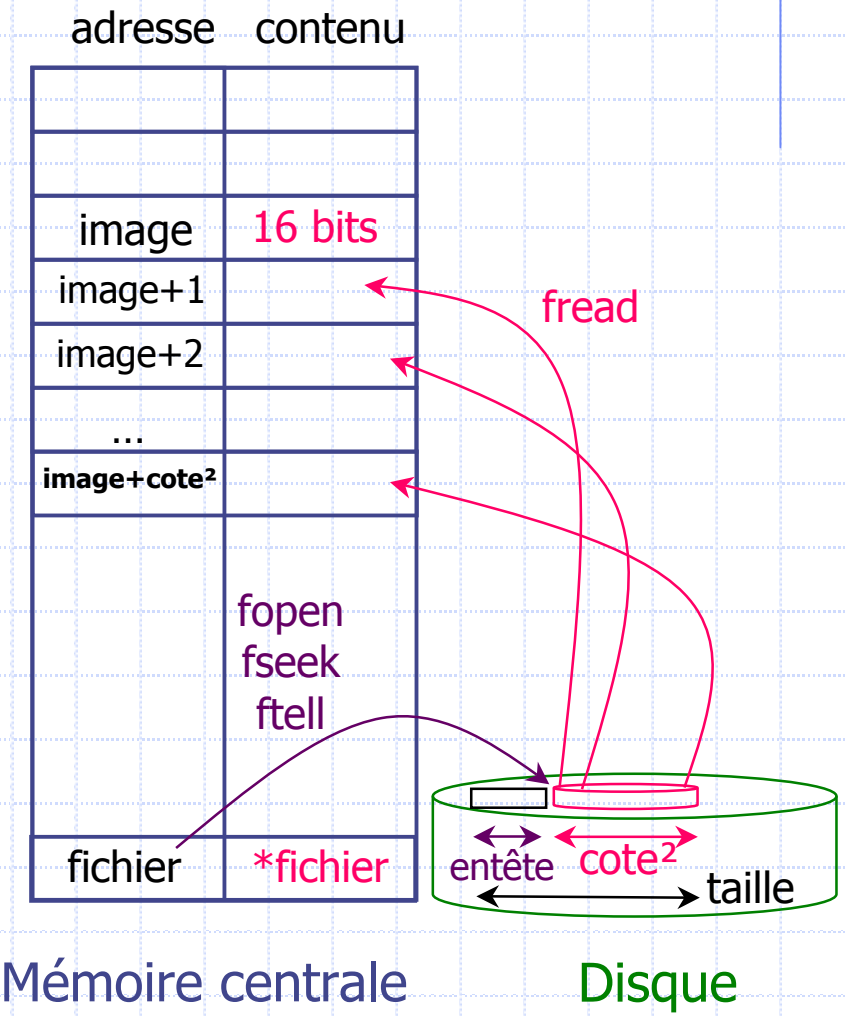
```
#include <stdio.h>
#include <stdlib.h>

void message(char *information);

int main()
    {
    short int *image;
    /* ... */
    image=malloc(cote*cote*sizeof(short int));
    if(image==NULL)
        {
        sprintf(information, « problème !\n »);
        message(information);
        }
    }

void message(char *information)
    {
     printf(information);
     getch();
     exit(1) ;
     }
```



instn3.c

Etape 3 : transfert en mémoire vive

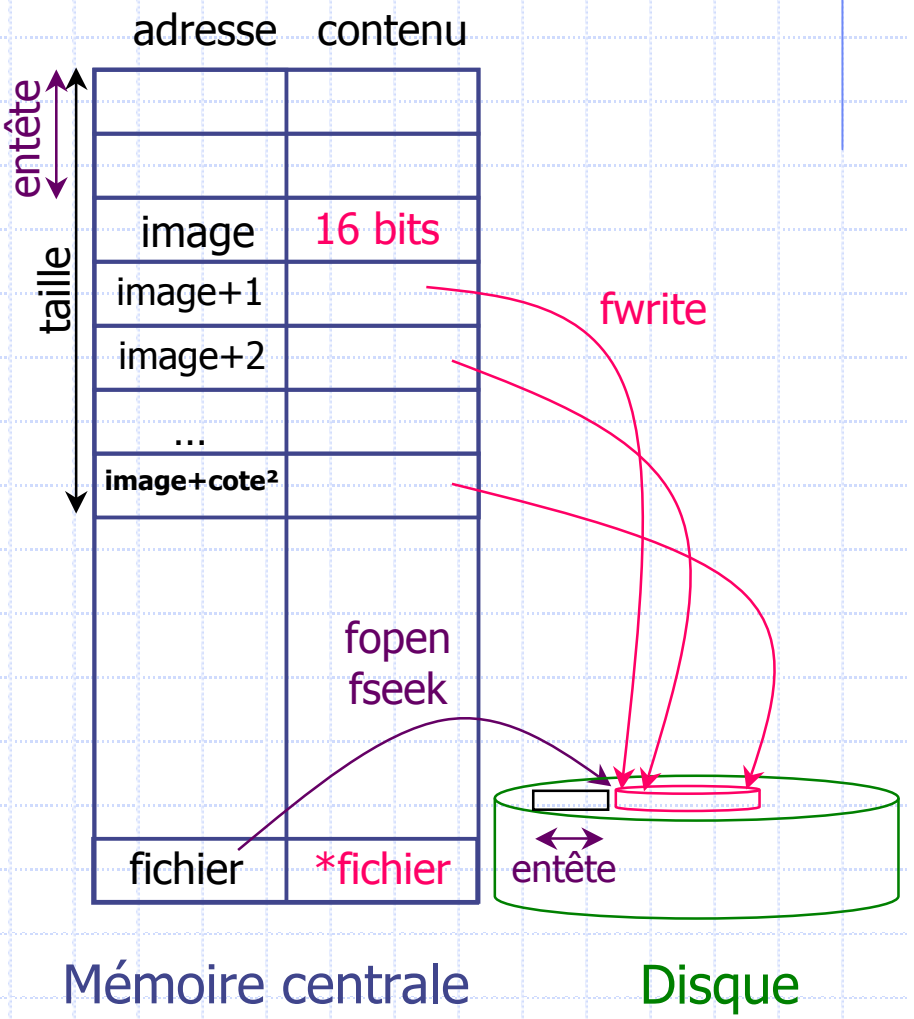
```
#include <stdio.h>
#include <stdlib.h>

#define MOT 2

void message(char *information);
void lecture_ecriture_image(short int *image, int
cote, char *nom1, char *nom2, int option);

int main()
    {
    short int *image; char nom2[256];

    /* ... */

    image=malloc(cote*cote*sizeof(short int));

   lecture_ecriture_image(image,cote,nom1,nom1,0);


   /* ... */
   }
```



instn4.c

Etape 4 : test de ré-écriture

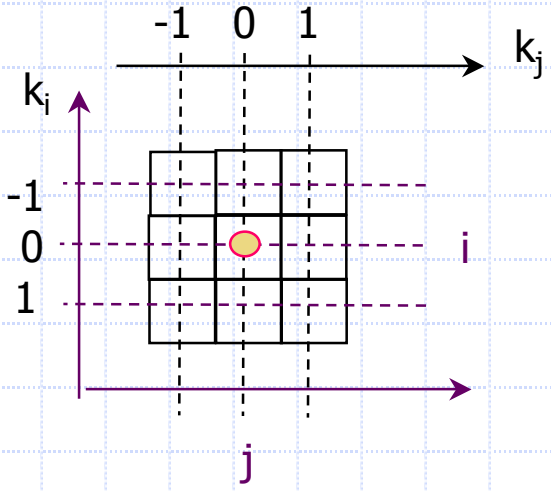
```
#include <stdio.h>
#include <stdlib.h>

#define MOT 2

void message(char *information);
void lecture_ecriture_image(short int *image, int
cote, char *nom1, char *nom2, int option);

int main()
    {
    short int *image;

    image=malloc(cote*cote*sizeof(short int));

lecture_ecriture_image(image,cote,nom1,nom1,0);

sprintf(nom2, « %s-filtre.dcm »,nom);
lecture_ecriture_image(image,cote,nom1,nom2,1);
    }
```



instn5.c

Etape 5 : moyenne pondérée

```
int main()
    {
    short int tampon;
    float calcul:
    int ki,kj;

    tampon = malloc(cote*cote*sizeof(short int)) ;

    for(i=1; i<cote-1; i++) for(j=1; j<cote-1; j++)
            {
            calcul = 0.0;
            for(ki=-1;ki<=1;ki++) for(kj=-1;kj<=1;kj++)
            calcul +=  (float)(*(image + j+kj + (i+ki)*cote)) ;

            *(tampon+j+i*cote) = (short int)( calcul/9.0 + 0.5);
            }

    for(i=1; i<cote-1; i++) for(j=1; j<cote-1; j++)
        *(image +j+i*cote) = *(tampon+j+i*cote);

free(tampon);
}
```



ant.dcm

Etape 6 : Pondérations

instn6.c

```
int main()
    {
    float somme, calcul;

   float coef[3][3]={1.,2.,1.,2.,4.,2.,1.,2.,1.} ;

   somme = 0.0; for(i=0;i<3;i++) for(j=0;j<3;j++) somme += coef[i][j];

    tampon = malloc(cote2*sizeof(short int)) ;

    for(i=1; i<cote-1; i++) for(j=1; j<cote-1; j++)
                {
                calcul = 0.0;
                for(ki=-1;ki<=1;ki++) for(kj=-1;kj<=1;kj++)
                                calcul += (coef[ki+1][kj+1] * (float)(*(image + j+kj + (i+ki)*cote))) ;

                *(tampon+j+i*cote) = (short int)( calcul/somme + 0.5);
                }
    for(i=1; i<cote-1; i++) for(j=1; j<cote-1; j++)  *(image +j+i*cote) = *(tampon+j+i*cote);

   free(tampon);
   }
```

ant.dcm

Etapes suivantes : à vous de jouer !

Modifiez le programme pour réaliser :

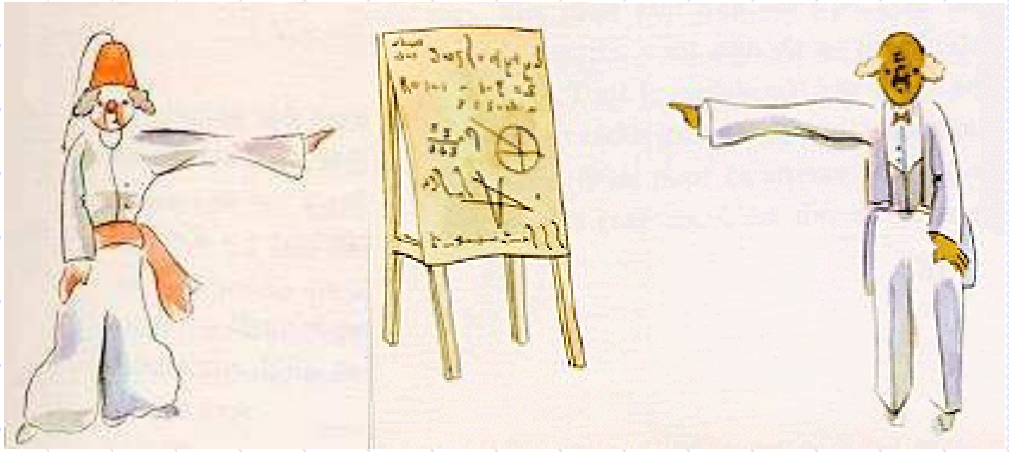
- des seuillages
- un lissage sur masque adapté
- une érosion, dilatation, ouverture, fermeture etc…

Il suffit de modifier la boucle :

```
for(i=1; i<cote-1; i++) for(j=1; j<cote-1; j++)
        {
        calcul = 0.0;
        for(ki=-1;ki<=1;ki++) for(kj=-1;kj<=1;kj++)
                calcul += (coef[ki+1][kj+1] * (float)(*(image + j+kj + (i+ki)*cote)));
        *(tampon+j+i*cote) = (short int)( calcul/somme + 0.5);
        }
```

Bibliographie

- Le langage C: norme ANSI
  - B.W. Kernighan et D.M. Ritchie. Dunod
- www.ensta.fr/~enstar/doc/c++/courscpp/index.html
- Introduction à turbo C++
- Turbo C++ par la pratique
  - T. Lachand-Robert. Sybex



Merci de votre attention…

http:\\scinti.etud.univ-montp1.fr
d-mariano_goulart@chu-montpellier.fr