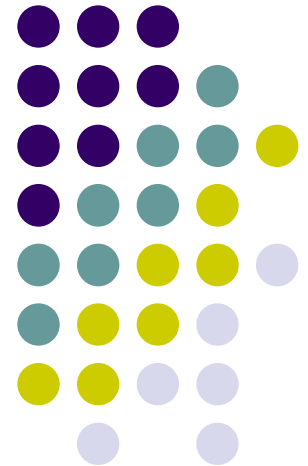


Le Langage C

F. Belabdelli
belabdelli@3il.fr





SOMMAIRE

- **Présentation générale du langage C**
- **Variables et traitements**
- **Les fonctions**
- **Tableaux et pointeurs**
- **Pointeurs et fonctions**
- **Type de données structurées**
- **Les fichiers de données**

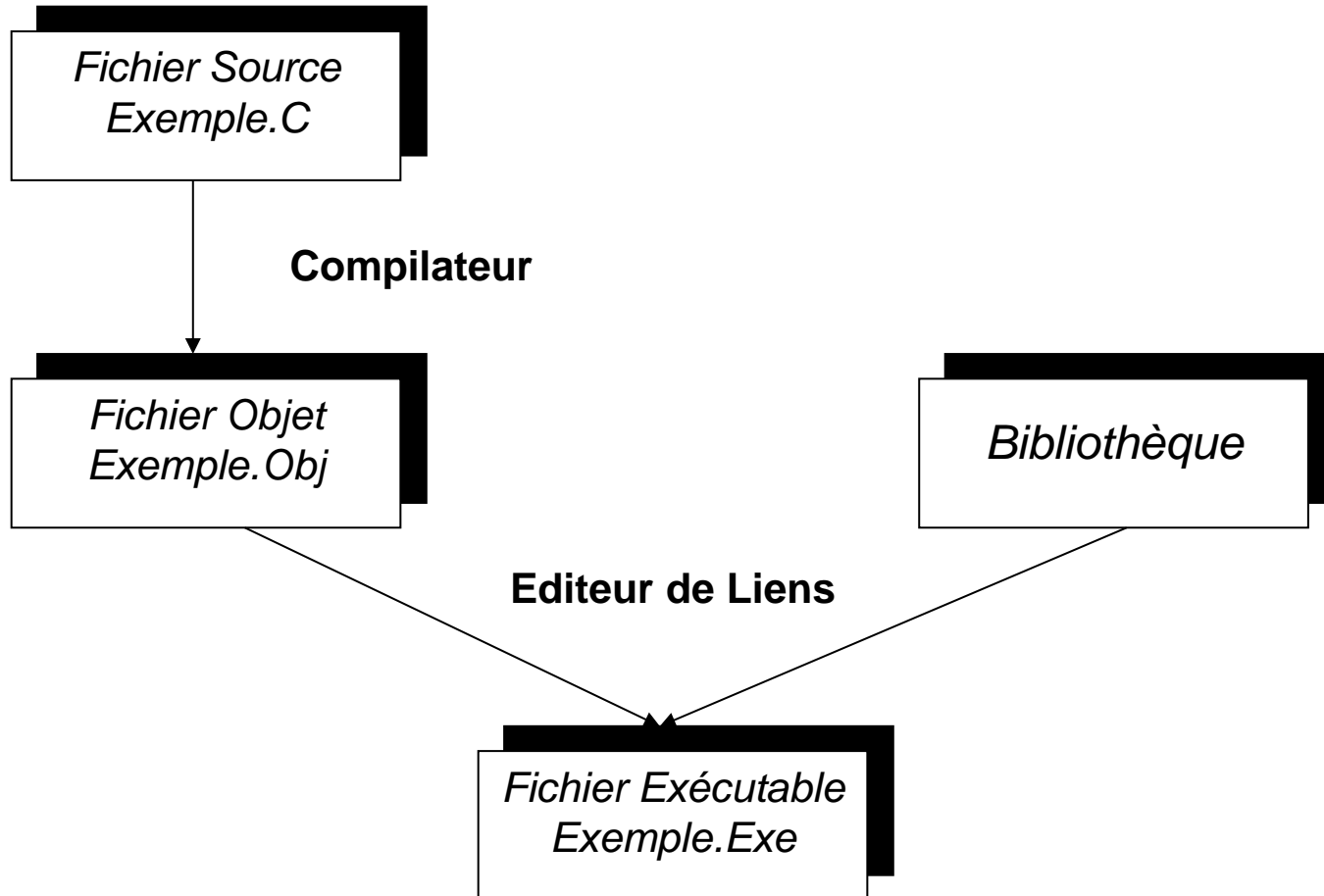


Historique

- **Algol 60** : projeté en 1960 par un comité international.
- **CPL : (Combined programming language)** projeté en 1963 à l'université de Cambridge.
- **BCPL : (Basic CPL)** projeté en 1967 par Martin Richards à Cambridge.
- **B** : Projeté en 1970 par Ken Thompson chez les laboratoires Bell.
- **C** : Projeté en 1972 par Dennis Ritchie laboratoires AT&T Américains



Structure d'un programme



Aspect général d'un programme



- Les variables globales
- Les fonctions
- Les fichiers
- La fonction point d'entrée → *main*
- Les commentaires → `//` ou `/* */`

Les éléments de base du langage



- Les séparateurs
- Les identificateurs
- Les mots réservés
- Les délimiteurs
- Les constantes
- Les opérateurs

Mots réservés



auto

break

case

char

const

continue

default

do

double

else

enum

extern

float

for

goto

if

int

long

register

return

short

signed

sizeof

static

struct

switch

typedef

union

unsigned

void

volatile

while

TABLE ASCII SIMPLE

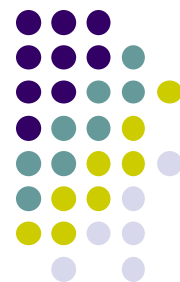
| Ctrl | Déc | Hex | Car | Code | Déc | Hex | Car | Déc | Hex | Car | Déc | Hex | Car |
|------|-----|-----|-----|------|-----|-----|-----|-----|-----|-----|-----|-----|----------------|
| ^@ | 0 | 00 | | NUL | 32 | 20 | ! | 64 | 40 | @ | 96 | 60 | ' |
| ^A | 1 | 01 | | SOH | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| ^B | 2 | 02 | | STX | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| ^C | 3 | 03 | | ETX | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| ^D | 4 | 04 | | EOT | 36 | 24 | \$ | 68 | 44 | D | 100 | 64 | d |
| ^E | 5 | 05 | | ENQ | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| ^F | 6 | 06 | | ACK | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| ^G | 7 | 07 | | BEL | 39 | 27 | , | 71 | 47 | G | 103 | 67 | g |
| ^H | 8 | 08 | | BS | 40 | 28 | (| 72 | 48 | H | 104 | 68 | h |
| ^I | 9 | 09 | | HT | 41 | 29 |) | 73 | 49 | I | 105 | 69 | i |
| ^J | 10 | 0A | | LF | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| ^K | 11 | 0B | | VT | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| ^L | 12 | 0C | | FF | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| ^M | 13 | 0D | | CR | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| ^N | 14 | 0E | | SO | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| ^O | 15 | 0F | | SI | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| ^P | 16 | 10 | | DLE | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| ^Q | 17 | 11 | | DC1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| ^R | 18 | 12 | | DC2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| ^S | 19 | 13 | | DC3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| ^T | 20 | 14 | | DC4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| ^U | 21 | 15 | | NAK | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| ^V | 22 | 16 | | SYN | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| ^W | 23 | 17 | | ETB | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| ^X | 24 | 18 | | CAN | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| ^Y | 25 | 19 | | EM | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| ^Z | 26 | 1A | | SUB | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| ^[| 27 | 1B | | ESC | 59 | 3B | ; | 91 | 5B | [| 123 | 7B | { |
| ^\ | 28 | 1C | | FS | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | |
| ^] | 29 | 1D | | GS | 61 | 3D | = | 93 | 5D |] | 125 | 7D | } |
| ^^ | 30 | 1E | ▲ | RS | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| ^- | 31 | 1F | ▼ | US | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | * [*] |

* Le code ASCII 127 correspond au code de suppression. Avec MS-DOS, ce code produit le même effet que ASCII 8 (BS). Le code de suppression peut être généré par la combinaison de touches CTRL + RET. ARR.



TABLE ASCII ETENDUE

| Déc | Hex | Car | Déc | Hex | Car | Déc | Hex | Car | Déc | Hex | Car |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 128 | 80 | Ç | 160 | A0 | ā | 192 | C0 | Ł | 224 | E0 | α |
| 129 | 81 | ü | 161 | A1 | ī | 193 | C1 | ⌞ | 225 | E1 | β |
| 130 | 82 | ē | 162 | A2 | ō | 194 | C2 | ⌠ | 226 | E2 | Γ |
| 131 | 83 | â | 163 | A3 | ó | 195 | C3 | ⌡ | 227 | E3 | Π |
| 132 | 84 | ä | 164 | A4 | û | 196 | C4 | ⌢ | 228 | E4 | Σ |
| 133 | 85 | à | 165 | A5 | ñ | 197 | C5 | ⌣ | 229 | E5 | σ |
| 134 | 86 | å | 166 | A6 | ª | 198 | C6 | ⌤ | 230 | E6 | μ |
| 135 | 87 | ç | 167 | A7 | º | 199 | C7 | ⌥ | 231 | E7 | Υ |
| 136 | 88 | ê | 168 | A8 | ¿ | 200 | C8 | ⌦ | 232 | E8 | ϕ |
| 137 | 89 | ë | 169 | A9 | ¬ | 201 | C9 | ⌧ | 233 | E9 | Θ |
| 138 | 8A | è | 170 | AA | ½ | 202 | CA | ⌨ | 234 | EA | Ω |
| 139 | 8B | ì | 171 | AB | ¼ | 203 | CB | 〈 | 235 | EB | δ |
| 140 | 8C | î | 172 | AC | ¼ | 204 | CC | 〉 | 236 | EC | ∞ |
| 141 | 8D | ï | 173 | AD | • | 205 | CD | ⌫ | 237 | ED | Φ |
| 142 | 8E | Ä | 174 | AE | « | 206 | CE | ⌬ | 238 | EE | € |
| 143 | 8F | Å | 175 | AF | » | 207 | CF | ⌭ | 239 | EF | ∩ |
| 144 | 90 | É | 176 | B0 | ◊ | 208 | D0 | ⌮ | 240 | F0 | ≡ |
| 145 | 91 | æ | 177 | B1 | ▮ | 209 | D1 | ⌯ | 241 | F1 | ± |
| 146 | 92 | Œ | 178 | B2 | ■ | 210 | D2 | ⌰ | 242 | F2 | ≥ |
| 147 | 93 | ô | 179 | B3 | ▮ | 211 | D3 | ⌱ | 243 | F3 | ≤ |
| 148 | 94 | ö | 180 | B4 | ⌞ | 212 | D4 | ⌲ | 244 | F4 | ┌ |
| 149 | 95 | ò | 181 | B5 | ⌞ | 213 | D5 | ⌳ | 245 | F5 | └ |
| 150 | 96 | ù | 182 | B6 | ⌞ | 214 | D6 | ⌴ | 246 | F6 | ÷ |
| 151 | 97 | û | 183 | B7 | ⌞ | 215 | D7 | ⌵ | 247 | F7 | ≈ |
| 152 | 98 | ÿ | 184 | B8 | ⌞ | 216 | D8 | ⌶ | 248 | F8 | ◊ |
| 153 | 99 | ÿ | 185 | B9 | ⌞ | 217 | D9 | ⌷ | 249 | F9 | • |
| 154 | 9A | Ü | 186 | BA | ⌞ | 218 | DA | ⌸ | 250 | FA | · |
| 155 | 9B | ¢ | 187 | BB | ⌞ | 219 | DB | ■ | 251 | FB | √ |
| 156 | 9C | £ | 188 | BC | ⌞ | 220 | DC | ■ | 252 | FC | ³ |
| 157 | 9D | ¥ | 189 | BD | ⌞ | 221 | DD | ■ | 253 | FD | ² |
| 158 | 9E | ℔ | 190 | BE | ⌞ | 222 | DE | ■ | 254 | FE | ■ |
| 159 | 9F | ƒ | 191 | BF | ⌞ | 223 | DF | ■ | 255 | FF | |



Les types de données et variables



- Les entiers → *int*
- Les entiers de type → *short* et *long*
- Les entiers sans signe → *unsigned*
- Les caractères
- Les données en virgule flottante → *float*
- Les données en double précision → *double*



Format de données

- ***scanf*** (" %d %d %d ",&annee, &numero, &age) ;
- ***printf*** (" Année %d, Numéro %d, Age %d \n ",annee, numero, age) ;



Les structures de contrôle

- Les sélections

if (<Condition>) <Instruction_1> ;
else <Instruction_2> ;

| Opérateur | Signification | Exemple |
|-----------|---------------------|-----------------------------------|
| == | Egal à | <i>if</i> (a == b) ... ; |
| != | Différent de | <i>if</i> (a != b) ... ; |
| > | Supérieur à | <i>if</i> (a > b) ... ; |
| < | Inférieur à | <i>if</i> (a < b) ... ; |
| >= | Supérieur ou égal à | <i>if</i> (a >= b) ... ; |
| <= | Inférieur à | <i>if</i> (a <= b) ... ; |

Langage C



Les itérations

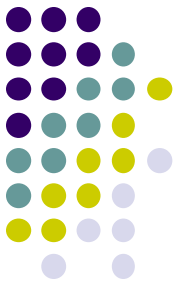
- ***while*** (**<Condition>**) {
 <Instruction> ;
}
- ***do*** {
 <instruction> ;
} ***while*** (**<condition>**) ;
- ***for*** (**<Initialisation>** ; **<Condition d'arrêt>** ; **<Incrémentation>**) {
 <Instruction>
}



Les sélections

- Les sélections *switch ... case ...* et *default*

```
switch ( <expression> )  
{  
  case <valeur_1> : <instruction_1> ;  
  case <valeur_1> : <instruction_1> ;  
  ...  
  case <valeur_n> : <instruction_n> ;  
  default : <instruction> ;  
}
```



Branchement inconditionnel

- **<étiquette> : <instruction> ;**
- ***goto* <étiquette> ;**



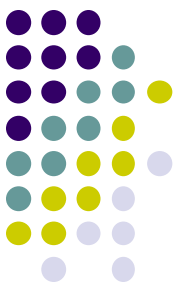
Les tableaux

- `<type> <nom_de_variable> [<Nombre_max>] ;`
int **tab**[5] ;
int **tab** [10][20] ;
int premiers[4] = { 3,5,7,11} ;
int matrice[2][4]={
 {1,2,3,4},
 {5,6,7,8},
 } ;
char ville[12]= "Montpellier" ;
char adresse[]= "499, rue de la croix verte" ;

Conversions implicites et explicites



- Conversion implicite ou automatique :
 - *Le type dont le poids est le plus faible est converti au type dont le poids est le plus fort.*
 - *Dans une affectation, le type de la variable à droite sera converti en type de celle de gauche*
- Conversion explicite ou forcée (cast)
 - *(type souhaité) expression*



Entrées et sorties de données

| déclaration | lecture | écriture | format externe |
|---|--|--|--|
| int i ; int i ; int i ; unsigned int i ; | scanf ("%d",&i) ; scanf ("%o",&i) ; scanf ("%x",&i) ; scanf ("%u",&i) ; | printf ("%d",i) ; printf ("%o",i) ; printf ("%x",i) ; printf ("%u",i) ; | décimal octal hexadécimal décimal |
| short j ; short j ; short j ; unsigned short j ; | scanf ("%hd",&j) ; scanf ("%ho",&j) ; scanf ("%hx",&j) ; scanf ("%hu",&j) ; | printf ("%d",j) ; printf ("%o",j) ; printf ("%x",j) ; printf ("%u",j) ; | décimal octal hexadécimal décimal |
| long k ; long k ; long k ; unsigned long k ; | scanf ("%ld",&k) ; scanf ("%lo",&k) ; scanf ("%lx",&k) ; scanf ("%lu",&k) ; | printf ("%ld",k) ; printf ("%lo",k) ; printf ("%lx",k) ; printf ("%lu",k) ; | décimal octal hexadécimal décimal |
| float l ; float l ; float l ; | scanf ("%f",&l) ; scanf ("%e",&l) ; | printf ("%f",l) ; printf ("%e",l) ; printf ("%g",l) ; | point décimal exponentielle la plus courte des deux |
| double m ; double m ; double m ; | scanf ("%lf",&m) ; scanf ("%le",&m) ; | printf ("%f",m) ; printf ("%e",m) ; printf ("%g",m) ; | point décimal exponentielle la plus courte |
| long double n ; long double n ; long double n ; | scanf ("%Lf",&n) ; scanf ("%Le",&n) ; | printf ("%Lf",n) ; printf ("%Le",n) ; printf ("%Lg",n) ; | point décimal exponentielle la plus courte |
| char o ; char p[10] ; | scanf ("%c",&o) ; scanf ("%9s",p) ; scanf ("%9s",&p[0]) ; | printf ("%c",o) ; printf ("%s",p) ; | caractère chaîne de caractères |



Opérateurs

- Arithmétique : `*`, `/`, `+`, `-`, `%`
- Affectation : `=`
- Logique bit à bit : `&`, `|`, `^`, `~`, `<<`, `>>`
- Logique : `&&`, `||`, `!`
- Incrémentation : `++`
- Décrémentation : `--`
- Expression conditionnelle : `?:`

Priorité et associativité des Opérateurs



| Classe d'opérateur | Opérateur(s) | Associativité |
|----------------------------------|---|--------------------|
| Parenthésage | () | de gauche à droite |
| Appel de fonction Suffixes ou | () [] -> . ++ -- | de gauche à droite |
| Un-aires préfixes | & * + - ~ ! ++ -- sizeof sizeof() | de droite à gauche |
| Changement de type | (type) | de droite à gauche |
| Multiplicatifs | * / % | de gauche à droite |
| Additifs | + - | de gauche à droite |
| Décalages | << >> | de gauche à droite |
| Comparaisons | < <= > >= | de gauche à droite |
| Égalités | == != | de gauche à droite |
| et bit à bit | & | de gauche à droite |
| ou exclusif bit à bit | ^ | de gauche à droite |
| ou bit à bit | | de gauche à droite |
| et logique | && | de gauche à droite |
| ou logique | | de gauche à droite |
| Condition | ? : | de droite à gauche |
| Affectations | = += -= *= /= %= &= = ^= <<= >>= | de droite à gauche |
| Succession | , | de gauche à droite |



Les fonctions

- **<type> <nom_fonction> (<liste_des_arguments>)**
{

 ...

}
- Les fonctions doivent obligatoirement être définies à l'extérieur de toute autre fonction.
- Les fonctions peuvent faire l'objet d'une déclaration de leurs entête, et être appelées, puis vient la déclaration de leurs corps.
- Exemple

```
int somme (int debut, int fin)
{
    int i, som =0; /* se sont les variables locales de la fonction */
    for (i = debut ; i <= fin ; ++i) som+=i ;
    return (som) ;      /* valeur à renvoyer */
}
```

Appel de Fonctions



```
main()
{
    int val1, val2, total ;
    printf("Entrez deux nombres entiers :");
    scanf("%d%d",&val1,&val2);
    total=somme(val1, val2);
    printf("la somme de %d ...à ...%d = %d\n",val1, val2, total);
}
```

Classes d'allocation



- Classe ***auto***
- Classe ***Register***
- Classe ***static***
- Classe ***extern***



Classe d'allocation static

```
void ajout()  
{  
    int x = 0 ;  
    x += 5 ;  
    printf("x= %d \n", x) ;  
}  
void main()  
{  
    ajout() ;  
    ajout() ;  
}
```

```
ajout()  
{  
    static int x = 0 ;  
    x += 5 ;  
    printf("x= %d \n", x) ;  
}  
main()  
{  
    ajout() ;  
    ajout() ;  
}
```


Structures de blocs



```
main()
{
char c = 'A' ;
    {
        char c = 'B' ;
        {
            char c = 'C' ;
            printf("%c \n ", c) ;
        }
        printf("%c \n ", c) ;
    }
printf("%c \n", c) ;
}
```

```
affiche2() ;
{
    char c = 'C' ;
    printf("%c \n", c) ;
}
affiche1() ;
{
    char c = 'B' ;
    affiche2() ;
    printf("%c \n", c) ;
}
main()
{
    char c = 'A' ;
    affiche1() ;
    printf(" %c \n", c) ;
}
```



Les Fonctions récursives

- En informatique, un programme est dit récursif s'il s'appelle lui-même directement ou indirectement. Il s'agit donc forcément d'une fonction.
- Exemple : la factorielle, $n! = 1 \times 2 \times \dots \times n$ donc $n! = n \times (n-1)!$
- Puisqu'une fonction récursive s'appelle elle-même, il est impératif qu'on prévoit une condition d'arrêt à la récursion, sinon le programme ne s'arrête jamais!
- Attention : débordement de la pile !!

Exemples



- Factoriel

```
int factoriel(int n)
{
    if (n==0)
        return 1;
    return n * factoriel (n - 1);
}
```

```
int x = 5 , fact;
fact = factoriel (x);
```

- Suite de Fibonacci :

- Fibo = 1 si $n = 0$ ou $n = 1$
- $\text{Fibo} = \text{Fibo}(n-1) + \text{Fibo}(n-2)$ sinon

- Suite d'Ackerman :

- $A(m,n) = n+1$ si $m = 0$,
- $A(m,n) = A(m-1,1)$ si $n=0$ et $m > 0$
- $A(m,n) = A(m-1, A(m,n-1))$ sinon



Réversivité croisée

- La réversivité croisée consiste à écrire des fonctions qui s'appellent l'une l'autre.

```
bool estPair(int x)
{
    if(x==0)
        return true;
    return estImpair(x-1);
}
bool estImpair(int x)
{
    if(x==0)
        return false;
    return estPair(x-1);
}
```



Les pointeurs

- Un pointeur est une variable qui contient l'adresse d'une autre variable.

<type> * <nom_du_pointeur> ;

Exemple :

```
int x = 5;  
int *ptr ;  
ptr = &x ;  
*ptr = 10 ;
```

Les tableaux et les pointeurs



```
int tab [5], *ptr ;  
ptr = tab ;  
++ptr ; /** ptr pointe sur l'élément tab[1] */  
ptr +=3 ; /** ptr pointe sur l'élément tab[4] */
```

```
main()  
{  
    int i = 0 ;  
    int tab [] = {1,2,3,4,5} ;  
    while ( i < 5 )  
    {  
        printf ("%d, est %d, stocké à l'adresse %u \n", i , *(tab+i), tab+i) ;  
    }  
}
```

Mode de passage de paramètres : passage par valeur



```
void essai(int valeur)
{
    valeur++;
}
void main()
{
    int a=10;
    essai(a);
    printf("La valeur de a vaut : %d\n", a);
}
```

Mode de passage de paramètres : passage par adresse



```
void essai1(int *p)
{
    *p+=2;
}
void main()
{
    int x=10;
    essai1(&x);
    printf("x= %d\n", x);
}
```

```
void essai2(int *p)
{
    *p+=2;
    p++;
    *p+=3;
}
void main()
{
    int tab[2] = { 10,11 }, *p=tab;
    essai2(p);
    printf("tab : [%d,%d]\n", *p, *(p+1));
}
```


Allocation et libération dynamique de la mémoire



- Permet d'indiquer la taille de l'emplacement à réserver en mémoire

#include <stdlib.h>

- **void *malloc** (size_t taille) ;
- **Void free**(**void** *ptr_adresse) ;

```
int *ptr ;  
ptr = malloc (10 * sizeof (int)) ;  
if (ptr != NULL) {  
    printf ("Allocation réussie") ;  
    free(ptr) ;  
}  
else  
{  
    printf("Echec d'allocation de mémoire") ;  
}
```



Pointeurs et fonctions

```
char *message ()          /** fonction qui renvoie un pointeur de type char */
main()
{
    char *(ptr_message)() ;
    /** pointeur à une fonction qui renvoie un pointeur de type char */
    ptr_message = message ;
    /** affectation de l'adresse de la fonction au pointeur */
    printf(" %s ", message()) ;
    printf(" %s ",(*ptr_message)()) ;
}
char * message()
{
    return ("Bonjour") ;
}
```



Type de données structurées

```
struct personne
{
    char nom[20] ;
    char adresse[30] ;
    int age ;
};
```

```
struct personne pers;
```

```
typedef struct personne
{
    char nom[20] ;
    char adresse[30] ;
    int age ;
}personne ;
```

```
personne pers;
```



Tableaux de structures

```
struct personne
{
    char nom[20] ;
    char adresse[30] ;
    int age ;
};

main ()
{
    struct personne tab[3] ;
    strcpy(tab[0].nom , "Dupond" ) ;
    strcpy(tab[0].adresse , "Paris" ) ;
    tab[0].age = 30 ;
}
```



Pointeurs sur structures

```
main ()
{
    struct personne tab[3] , *ptr;
    strcpy(tab[0].nom, "Pierre") ;
    strcpy(tab[0].adresse, "Paris") ;
    tab[0].age = 20 ;
    strcpy(tab[1].nom , "François" );
    strcpy(tab[1].adresse , "Montpellier" );
    tab[1].age = 25 ;
    strcpy(tab[2].nom , "Claude" );
    strcpy(tab[2].adresse , "Limoges" );
    tab[2].age = 30 ;
    printf ("\n\t nom\t adresse\t age \n") ;
    for (ptr = tab ; ptr <= tab+2 ; ++ptr)
        printf ("\t %s \t %s \t %d \n",
            ptr->nom, ptr->adresse, ptr->age) ;
    getch() ;
}
```

```
struct personne
{
    char nom[20] ;
    char adresse[30] ;
    int age ;
};
```



L'opérateur typedef

```
typedef char caractere ;  
typedef char chaine[20] ;  
typedef int entier ;  
caractere c = 'r' ;  
chaine  nom, prenom ;  
entier  age ;
```

```
typedef struct  personne  
{  
    char nom[20] ;  
    char adresse[30] ;  
    int age ;  
} PERSONNE;
```

```
typedef enum{LUN, MAR, MER, JEU, VEN, SAM, DIM} MOIS ;
```

```
MOIS  var_mois ;  
struct  personne per1, per2, per3 ;  
PERSONNE per1, per2, per3 ;
```



Les unions

```
union exemple_union
{
    char carac [ 3 ] ;
    int ent ;
};

union exemple_union    mot ;

mot.ent = 546 ;
mot.carac [ 0 ] = 'e' ;
mot.carac [ 1 ] = 'u' ;
mot.carac [ 2 ] = 'i' ;
```



Fichiers bufférisés

- La gestion du fichier se fait par
 - un buffer
 - Un pointeur de type FILE
- Ouverture :
 - `FILE *fopen(const char *nom , const char *mode);`

```
#include <stdio.h>
FILE *fp;

if ((fp = fopen(« c:/Data/fichier1.txt","r")) == NULL)
{
    fprintf(stderr,"Impossible d'ouvrir le fichier données en lecture\n");
    exit(1);
}
```




Fichiers bufférés

- Mode d'ouverture

| Mode | Description |
|------|---|
| r | Ouverture en lecture seule sur un fichier existant |
| w | Ouverture en écriture seule. Si le fichier existe il est détruit. |
| a | Ouverture pour écriture à la fin du fichier. Création du fichier s'il n'existe pas |
| r+ | Ouverture en lecture ou écriture sur un fichier existant. |
| w+ | Ouverture en lecture ou écriture. Si le fichier existe il est détruit. Création du fichier s'il n'existe pas. |
| a+ | Ouverture en lecture ou écriture à la fin du fichier. Création du fichier s'il n'existe pas. |



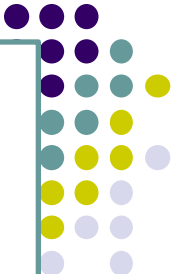
Fichiers bufférisés

- Fermeture
 - `int fclose(FILE *stream);`
 - Marque la fin de fichier (détermine la taille)
 - Réalise le transfert de données ($DD \leftrightarrow MC$)



Opérations d'Entrée/Sortie

- Ecriture binaire :
 - `size_t fwrite(void *ptr, size_t taille, size_t n, FILE *stream);`
- Lecture binaire :
 - `size_t fread(void *ptr, size_t taille, size_t n, FILE *stream);`
- Ecriture des tampons
 - `int fflush(FILE *stream);`



```
#define DIM 10000
void main()
{
    int i;
    double sum,tab1[DIM],tab2[DIM];
    FILE *fichier;
    // Remplissage du tableau
    for(i=0;i<DIM;i++)
        tab1[i]=i*atan(1);
    // Ecriture du fichier au format binaire
    fichier = fopen("essai2 bin essai2.bin","wb");
    if (fichier != NULL)
    {
        fwrite(tab1,sizeof(double),DIM,fichier);
        fclose(fichier);
    }
    // Lecture du fichier
    fichier = fichier = fopen("essai2 bin essai2.bin","rb");
    if (fichier != NULL)
    {
        fread(tab2,sizeof(double),DIM,fichier);
        fclose(fichier);
    }
}
```



Opérations d'Entrée/Sortie

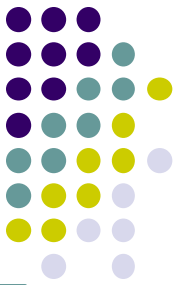
- Lecture d'un caractère
 - `int fgetc(FILE *stream);`
 - `int getc(FILE *stream);` /*macro identique à la fonction fgetc */
- Ecriture d'un caractère
 - `int fputc(int ch, FILE *stream);`
 - `int putc(int ch , FILE *stream);` /* macro identique à fputc */
- Lecture d'une chaîne :
 - `char *fgets(char *s, int n, FILE *stream);`
- Écriture d'une chaîne :
 - `int fputs(const char *s, FILE *stream);`



Opérations d'Entrée/Sortie

- Écriture formatée dans un flux
 - `int fprintf(FILE *stream, const char *format, argument ...);`
- Lecture formatée dans un flux
 - `int fscanf(FILE *stream, const char *format, pointeur ...);`

Exemple : lecture à partir d'un fichier



```
#include <stdio.h>
void main()
{
    int i;
    double tab[20];
    FILE *fichier;
    // Ouverture du fichier en lecture grâce à "r"
    fichier = fopen("essai.txt","r");
    if (fichier != NULL)
    {
        for(i=0;i<20;i++)
            fscanf(fichier,"%lf\n",tab+i);
        fclose(fichier);
    }
    for( i=0;i<20;i++)
        printf("%lf\n",tab[i]);
}
```



Opérations d'Entrée/Sortie

- Position du pointeur de fichier
 - `int fseek(FILE *stream, long offset, int methode);`
 - `SEEK_SET (0)` : Positionnement à *offset* octet(s) du début du fichier.
 - `SEEK_CUR (1)` : Positionnement à la position courante *offset* octet(s).
 - `SEEK_END (2)` : Positionnement à la fin du fichier + *offset* octet(s).
- Position courante du fichier
 - `long ftell(FILE *stream);`



Opérations d'Entrée/Sortie

- Tester la fin de fichier
 - `int feof(FILE *stream);`
- Gestion des erreurs
 - `int ferror(FILE *stream);`
 - `int clearerr(FILE *stream);`
- Suppression du fichier
 - `int remove(const char *nom);`
- Renommer un fichier
 - `int rename(const char *anciennom , const char *nouveaunom);`

Example :



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

int main( int argc, char * argv[] ) {

    FILE * myFile = fopen( "anExistingFile", "r" );
    int rc = fputc( '!', myFile );

    printf( "Results == %d %d %d\n", rc, ferror( myFile ), errno );
    perror( "My message" );

    rc = fclose( myFile );

    return 0;
}
```

```
Results == -1 1 9
My message: Bad file descriptor
```



Arguments de la ligne de commande

```
main(int argc, char *argv[])
{
    int i;
    printf("Vous avez lance le programme : %s ",argv[0]);
    switch (argc)
    {
        case 1 :
            printf("sans argument \n");
            break;
        case 2 :
            printf("avec l'argument %s \n", argv[1]);
            break;
        default :

            printf("avec les arguments suivant%s\ n");
            for ( i = 1; i <= argc ; i++)
                printf("%s\n", argv[i]);
    }
}
```

Langage C

Fonctions liées aux chaînes de caractères



- `char *strcpy(char *dest, char *src)`
- `char *strdup(char *src)`
- `char *strncpy(char *dest, char *src, size_t n)`
- `size_t strlen(char *src)`
- `char *strcat(char *dest, char *src)`
- `int strcmp(char *ch1, char *ch2)`
- `char *strcstr(char *ch, char *ss_ch)`

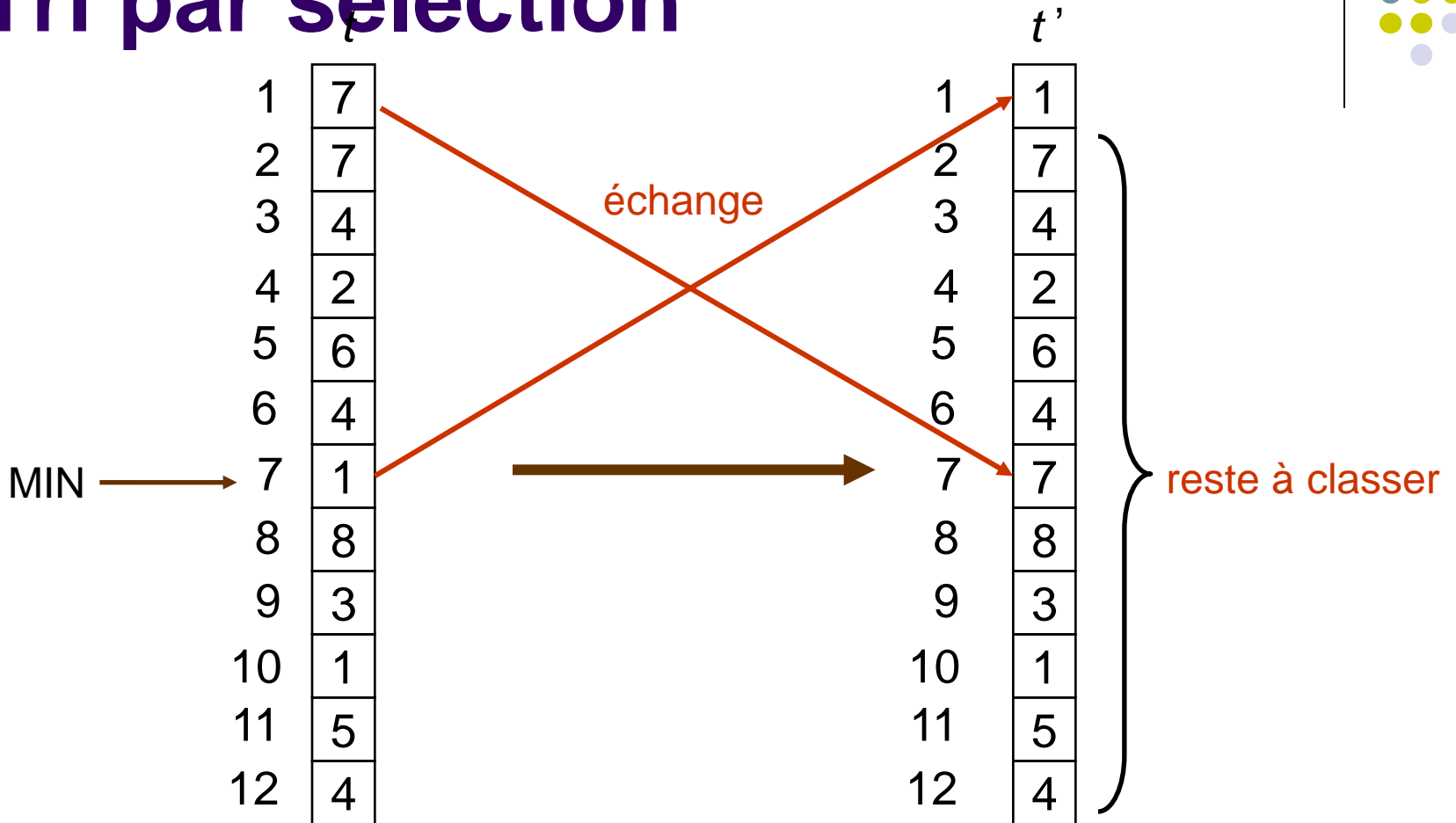


Tris

- Dichotomie
- Tris :
 - Par sélection
 - Par insertion
 - A bulle

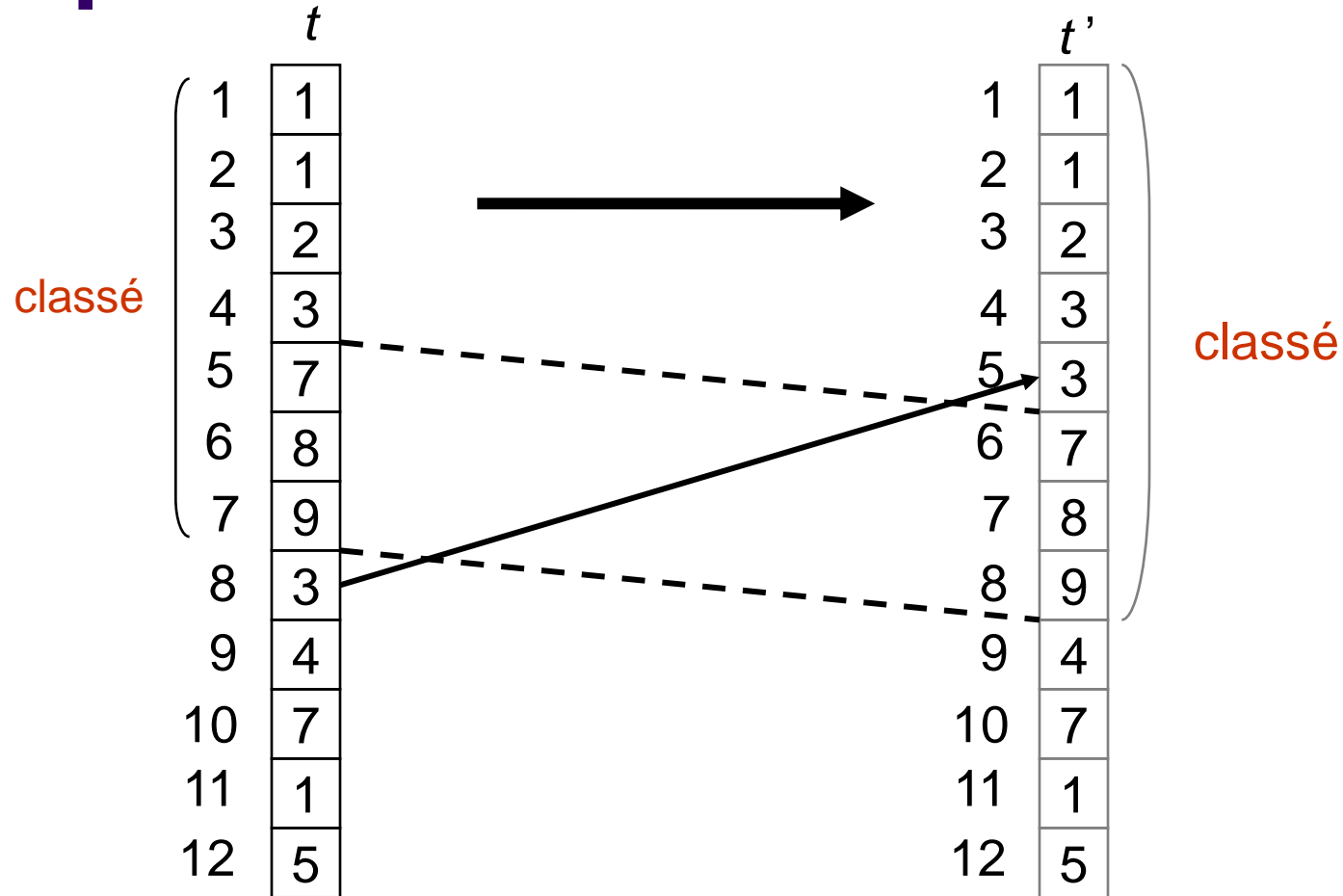


Tri par sélection



Recherche du minimum par balayage séquentiel

Tri par insertion



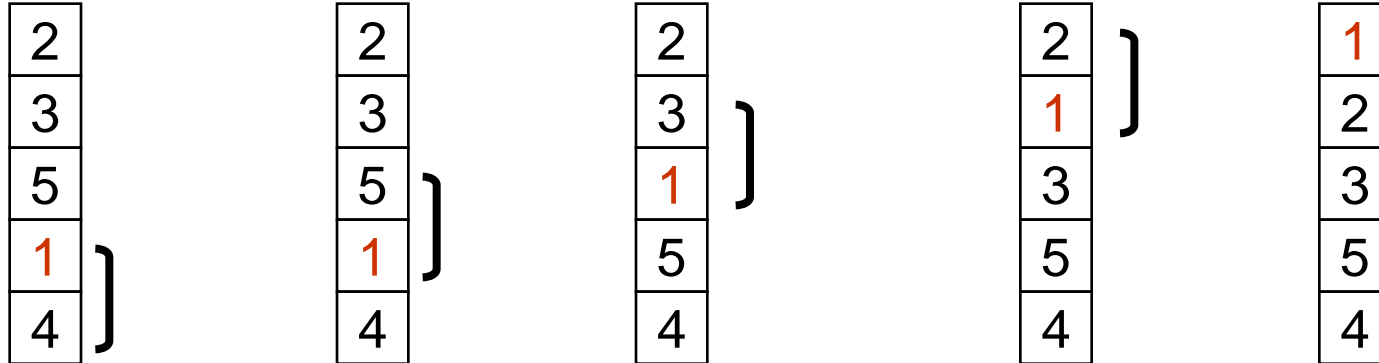
Point d'insertion

- recherche séquentielle
- recherche dichotomique

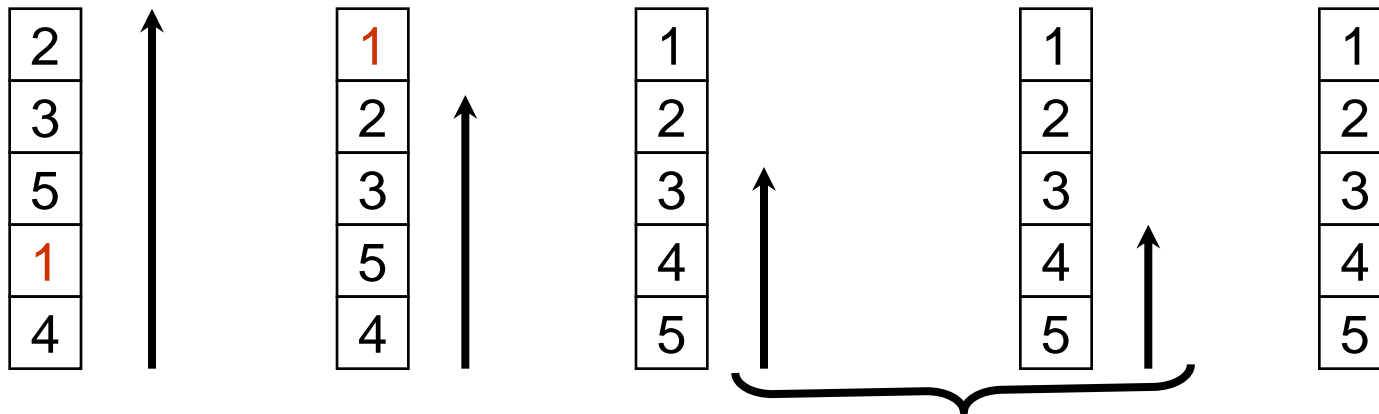
Tri à Bulles



Un balayage



Suite des balayages



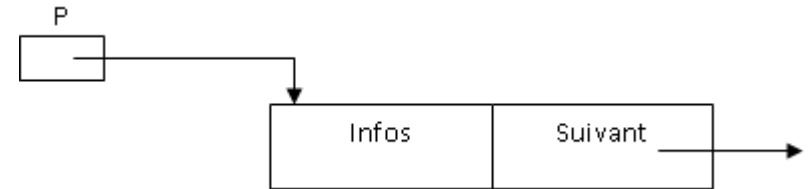
Language C
peuvent être éliminés



Les listes chaînées

- Une liste linéaire chaînée (ou liste chaînée) est constituée d'un ensemble de cellules chaînées entre elles. C'est l'adresse de la première de ces cellules qui détermine la liste. Cette adresse doit se trouver dans une variable (généralement appelée liste).

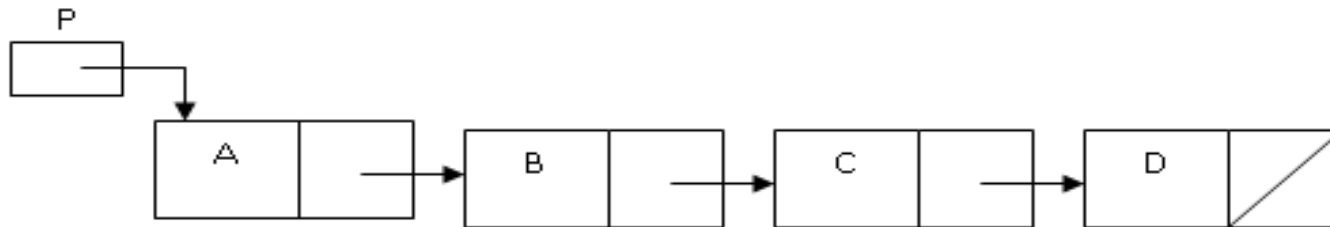
```
typedef struct Cellule
{
    <type> Infos
    struct Cellule *Suivant
}Cellule;
Cellule *p;
```



Opérations sur une liste chaînée



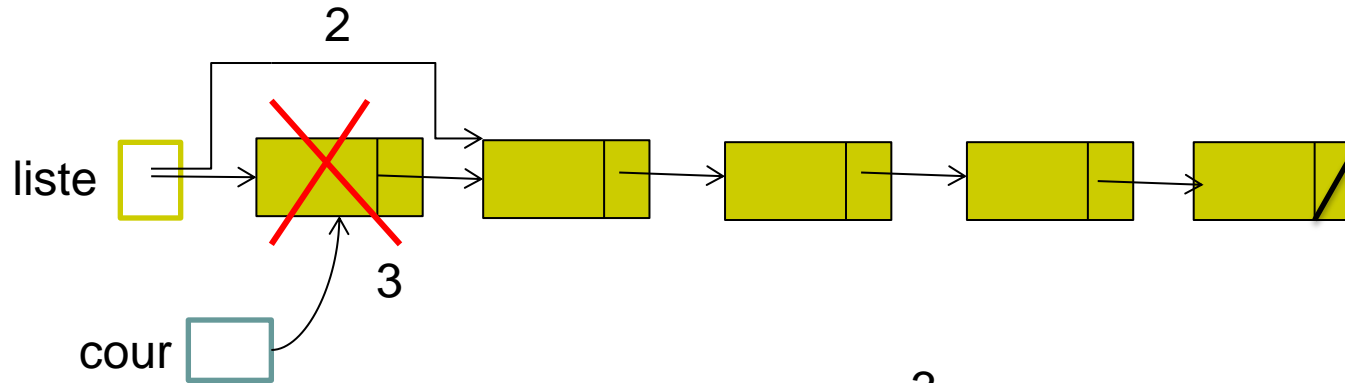
- Insertion en tête de liste
- Insertion en fin de liste
- Insertion en milieu de la liste (selon l'ordre)
- Supprimer tête de liste
- Supprimer queue de liste
- Supprimer en milieu de la liste



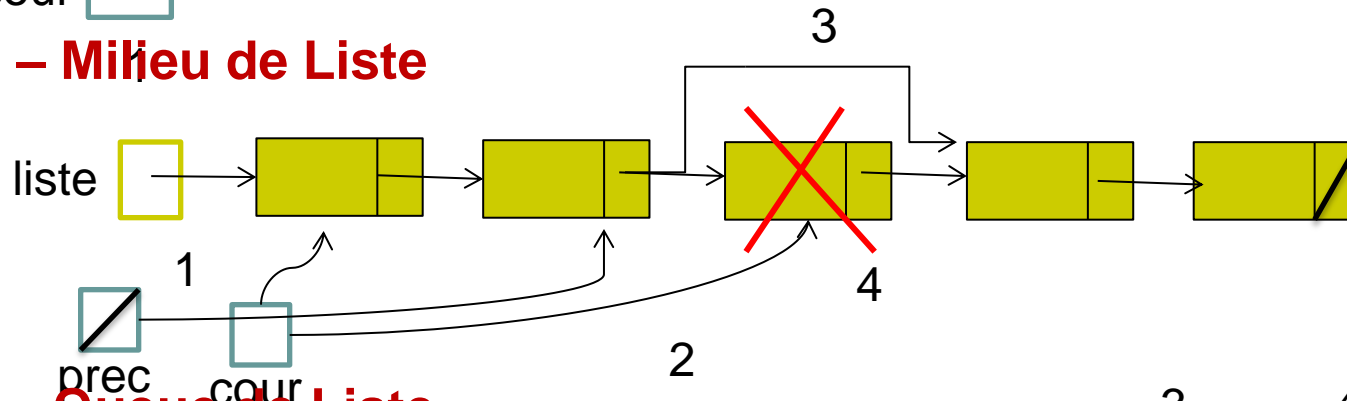
Supprimer un élément



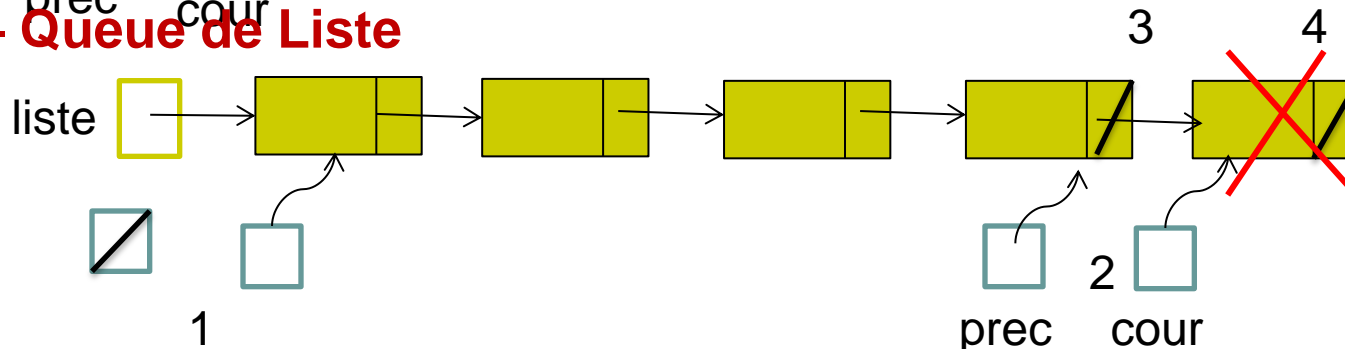
1 – Tête de Liste



2 – Milieu de Liste



3 – Queue de Liste





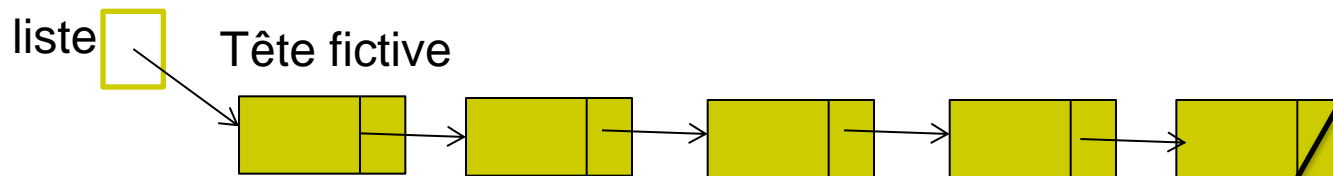
Variantes de Listes Chaînées

- ***Liste avec tête fictive***
- ***Liste chaînée circulaire***
- ***Liste doublement chaînée***
- ***Liste doublement chaînée circulaire***
- ***Liste triée***



Liste avec Tête Fictive

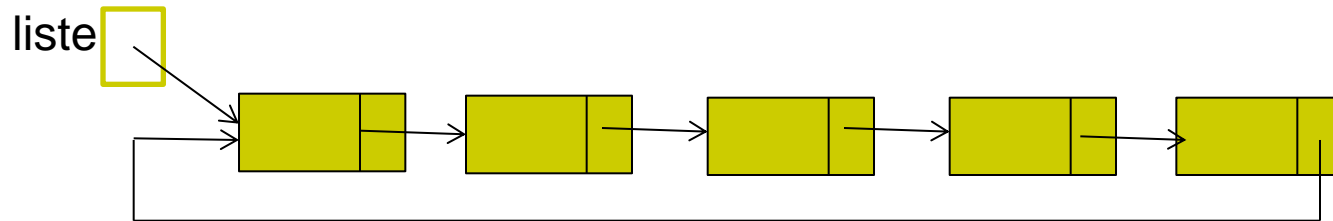
- Eviter d'avoir un traitement particulier pour le cas de la tête de liste (*opérations d'insertion et de suppression*)
 - Mettre en tête de liste une zone qui ne contient pas de valeur et reste toujours en tête





Liste Circulaire

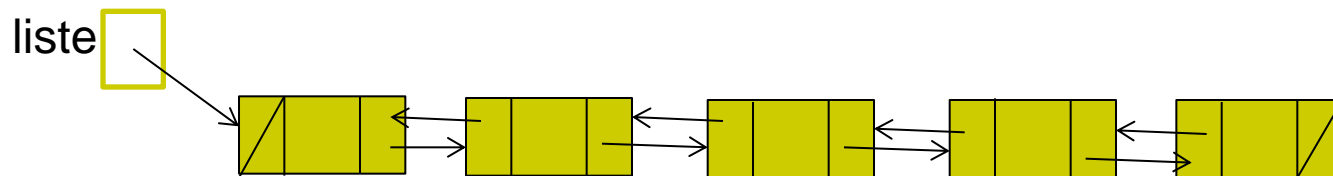
- Le suivant du dernier élément de la liste est le pointeur de tête





Liste Doublement Chaînée

- **Faciliter le parcours de la liste dans les deux sens**
 - utiliser un double chaînage ; chaque place repérant à la fois la place qui la précède et celle qui la suit



Programmation modulaire



- Un programme en langage C peut contenir plusieurs dizaines de milliers d'instructions !
- Même si on fait un découpage en fonctions, il n'est pas recommandé de ne faire qu'un seul fichier source, car :
 - dur à maintenir
 - long à compiler : une modification d'une seule ligne oblige à recompiler tout le programme !
 - et si on veut réutiliser une partie du programme ?



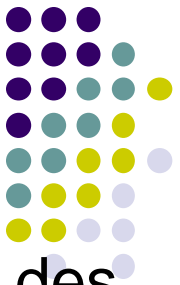
Programmation modulaire

- un module est un ensemble de fonctions (prototypes + définitions) que l'on met dans des fichiers à part : un programme sera donc un ensemble de fichiers qui seront réunis lors des différentes étapes de compilation et d'édition des liens.
- Les environnements de développement savent réunir les fichiers (Visual C++, Dev-C++, CodeBlocks), il est possible de le faire 'à la main' (environnements Unix/Linux avec le compilateur gcc).
- Plus facile à maintenir, possibilité de séparer les tâches lorsque l'on travaille en groupe, moins vulnérable, réutilisable.



Programmation modulaire

- Les fichiers utilisés pour générer un programme :
 - un fichier `.c` ne contenant que le programme principal, souvent nommé `main.c`
 - Des modules, qui sont des couples de fichiers `.c` et `.h`
 - `.c` : contient les définitions des fonctions (donc des instructions);
 - `.h` (h signifie header ou entête), contenant les prototypes des fonctions et des descriptions de constantes, de types, mais AUCUNE instruction.
- Un fichier `.c` se compile, un fichier `.h` ne se compile pas.



Contenu d'un fichier .c

- Le fichier .c d'un module contient toutes les définitions des fonctions ainsi que les informations que les autres modules n'ont pas besoin de connaître.

On y trouve donc :

- les prototypes des fonctions dites locales (utilisées par ce module uniquement)
- les définitions des types locaux
- les définitions des constantes locales

ainsi que les définitions de toutes les fonctions : les fonctions locales ET les fonctions exportées.

- C'est la directive `#include` qui permet d'inclure un fichier .h

Inclusion des fichiers .h et directives



Prenons l'exemple d'un programme réparti de la manière suivante :

- **un module point**, définissant le type **t_p3d** (point dans un espace à 3 dimensions, ce sont ses coordonnées nommées *x*, *y* et *z*), et permettant d'utiliser des fonctions **afficherPoint** et **saisiePoint**.
- **Un module transfos**, permettant de faire des calculs sur des points en 3D, en commençant par une fonction nommée **symetrique** calculant le point symétrique d'un point *P* par rapport à l'origine (0,0,0).
- Un programme principal qui fait la saisie d'un point *P*, l'affichage de ses coordonnées, le calcul du point *Q* symétrique de *P* et qui affiche les coordonnées de ce point *Q*.

Le module point



Le module point est constitué des fichiers point.c et point.h

point.h contient :

la définition du (ou des types) pour représenter un point, il s'agira d'un type composé nommé **t_p3d**.

les prototypes des fonctions **afficherP** et **saisirP**.

```
typedef struct s_p3d
{
    double x,y,z;
} t_p3d, *p_p3d;

extern void saisirPoint(p_p3d);
extern void affichePoint(t_p3d);
```



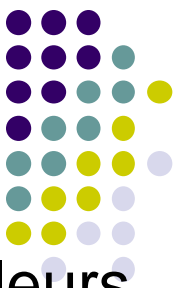
Le module point

Le fichier point.c contient les définitions des fonctions. Puisque les fonctions utilisent les types `t_p3d` et `p_p3d` définis dans `point.h`, il faut inclure ce fichier.

```
#include "point.h"

void saisirP(p_p3d ptr_point)
{
    scanf("%lf", &(ptr_point->x));
    scanf("%lf", &(ptr_point->y));
    scanf("%lf", &(ptr_point->z));
}

void afficherP(t_p3d point)
{
    printf("[%6.3lf | %6.3lf | %6.3lf]\n", point.x, point.y, point.z);
}
```



Le module point

- On veut par contre avoir une saisie sécurisée des valeurs entrées, en utilisant une fonction **saisieSec**, que l'on doit programmer. Cette fonction permettra de s'assurer que l'on a bien saisi une valeur à virgule, et pas un caractère ou du texte.
- Cette fonction appartient au module point, mais n'a pas besoin d'être exportée, car seule la fonction **saisirPoint** va l'utiliser.
- Il s'agira donc d'une fonction locale au module point, donc son prototype et sa définition se trouveront dans le fichier point.c. aucun autre module n'a besoin de connaître son existence.

Le module point



```
// prototype de la fonction locale

double saisieSec(void);

// définition des fonctions locales

double saisieSec(void)
{
    double val;

    do
    {
        printf("Entrez une valeur à virgule :");
        fflush(stdin);
    }
    while ( scanf("%lf",&val) !=1);

    return val;
}
```