

Algorithmique et structures de données - NF16

Connaissances de base sur le langage C

Mohamed Ali KANDI

mohamed-ali.kandi@hds.utc.fr

1. Qu'est-ce que c'est qu'un programme informatique?

1. Qu'est-ce que c'est qu'un programme informatique?

Un **programme informatique** est un ensemble d'instructions destinées à être exécutées par un ordinateur pour réaliser une tâche.

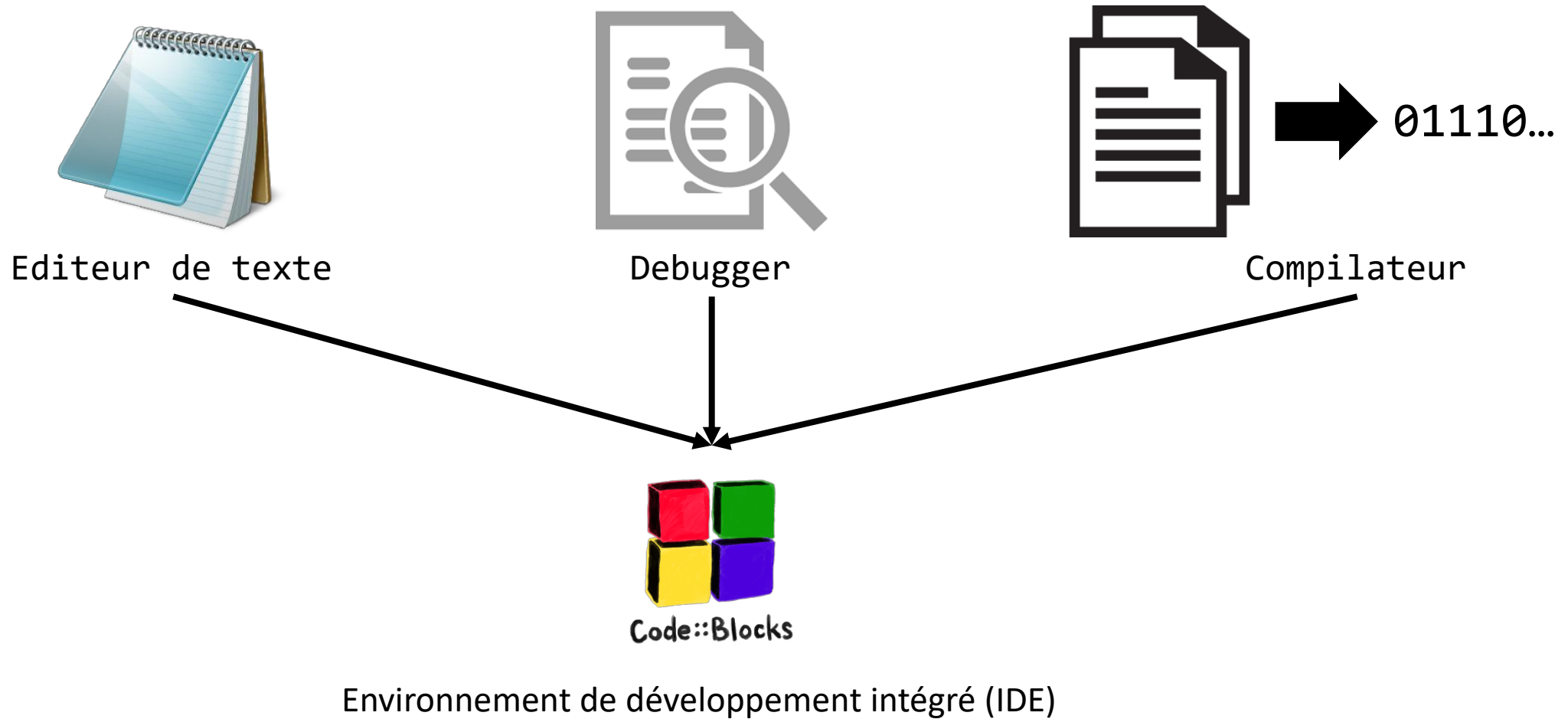


Problème: Un ordinateur ne parle pas le même langage que nous.

On doit donc utiliser des langages de programmation tels que le C.

1. Qu'est-ce que c'est qu'un programme informatique?

Pour cela, on a besoin de:



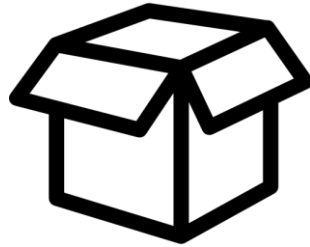
2. Les bases de la programmation C

2.1. Les variables

2. Les bases de la programmation C

2.1. Les variables

Les **variables** sont des éléments permettant de stocker les données nécessaires à l'exécution du programme.



- Nom: n
- Type: entier
- Valeur: 10

La valeur d'une variable peut varier au cours de l'exécution du programme.

2. Les bases de la programmation C

2.1. Les variables

Type de la variable	Déclaration
Entier	int / long int
Flottant	float / double
Caractère	char

Déclaration d'une variable

2. Les bases de la programmation C

2.1. Les variables

Opération	Signe
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Opération sur les variables

2. Les bases de la programmation C

2.1. Les variables

Type de la variable	Format
Entier	%d / %ld / %x(hexa) / %o(octa)
Flottant	%f / %lf / %g
Caractère	%c

2. Les bases de la programmation C

2.1. Les variables

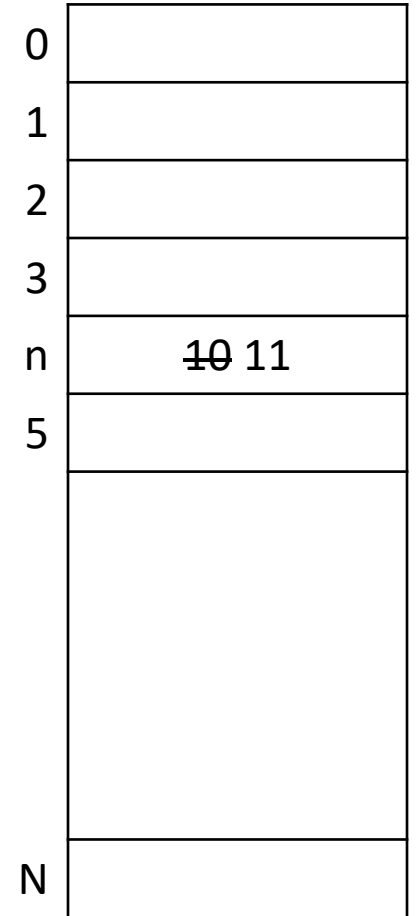
```
int main()
{
    int n;                //Déclarer la variable

    n=10;                 //Affecter une valeur à la variable

    n=n+1                 //Fair une operation sur la variable

    printf("%d",n);       //Afficher le contenu de la variable

    scanf("%d",&n);       //Demander à l'utilisateur la valeur de la variable
}
```



Mémoire

2. Les bases de la programmation C

2.2. Les conditions

2. Les bases de la programmation C

2.2. Les conditions

Les **conditions** permettent de vérifier les valeurs des variables moyennant des opérateurs de comparaison (`==`, `<>`, `>`, `<`, ...) et des opérateurs logiques (`&&`, `||`, `!`).

Par exemple:

- Est-ce que la valeur de `n` est supérieure à 0? ➔ `if(n>0)`
- Est-ce que la valeur de `n` est paire? ➔ `if(n%2==0)`
- Est-ce que la valeur de `n` est comprise entre 0 et `m`? ➔ `if (n>=0 && n<=m)`

Sans conditions, un programme informatique fait toujours la même chose.

2. Les bases de la programmation C

2.2. Les conditions

```
int main()
{
    ① int n=10;

    ② if(n==0)
    {
        printf("La valeur de n est nulle");
    }
    ③ else if(n>0)
    {
        ④ printf("La valeur de n est positive");
    }
    else
    {
        printf("La valeur de n est negative");
    }
    ⑤ }
}
```

2. Les bases de la programmation C

2.2. Les conditions

```
int main()
{
    switch(expression)
    {
        case expression1:
            ...
            break;

        case expression2:
            ...
            break;

        ...

        default:
            ...
    }
}
```

2. Les bases de la programmation C

Les boucles


2. Les bases de la programmation C

2.3. Les boucles

Les **boucles** permettent d'exécuter un ensemble d'instructions de façon répétée. Une boucle se termine quand une condition est vérifiée.

```
int main()
{
    int n=0;

    while(n<10)
    {
        ...
        n=n+1;
    }
}
```



Cette instruction est répétée 10 fois


2. Les bases de la programmation C

2.3. Les boucles

Les **boucles** permettent d'exécuter un ensemble d'instructions de façon répétée. Une boucle se termine quand une condition est vérifiée.

```
int main()
{
    int n=0;

    do
    {
        ...
        n++;
    } while(n<10)
}
```



Ce bloc d'instructions est répété 10 fois

2. Les bases de la programmation C

2.3. Les boucles

Les **boucles** permettent d'exécuter un ensemble d'instructions de façon répétée. Une boucle se termine quand une condition est vérifiée.

```
int main()
{
    int n=0;

    for(n=0;n<10;n++)
    {
        ...
    }
}
```



Cette instruction est répétée 10 fois

2. Les bases de la programmation C

Les pointeurs

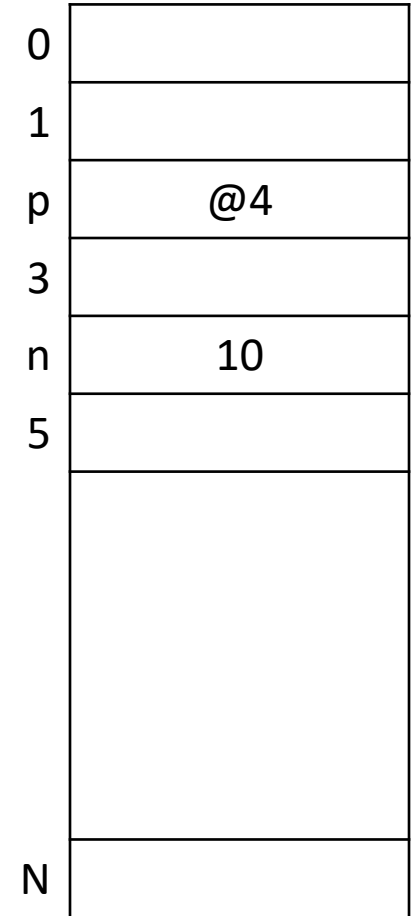
2. Les bases de la programmation C

2.4. Les pointeurs

Un **pointeur** est une variable contenant l'adresse d'une autre variable.

Soient `n` une variable et `p` un pointeur sur `n`:

- `n` signifie « La valeur de `n` » → 10
- `&n` signifie « l'adresse de `n` » → @4
- `p` signifie « l'adresse de `n` » → @4
- `*p` signifie « la valeur de `n` » → 10



Mémoire

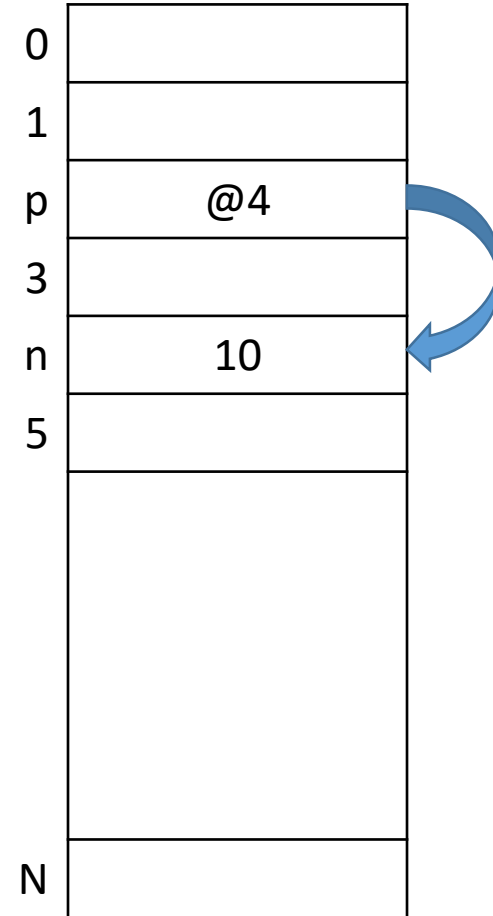
2. Les bases de la programmation C

2.4. Les pointeurs

```
int main()
{
    int n;
    int *p;

    n=10;
    p=&n;

    printf("%d",n);      // La valeur de n : 10
    printf("%p",&n);     // L'adresse de n : @4
    printf("%p",p);      // L'adresse de n : @4
    printf("%d",*p);     // La valeur de n : 10
}
```



Mémoire

2. Les bases de la programmation C

2.5. Les tableaux

2. Les bases de la programmation C

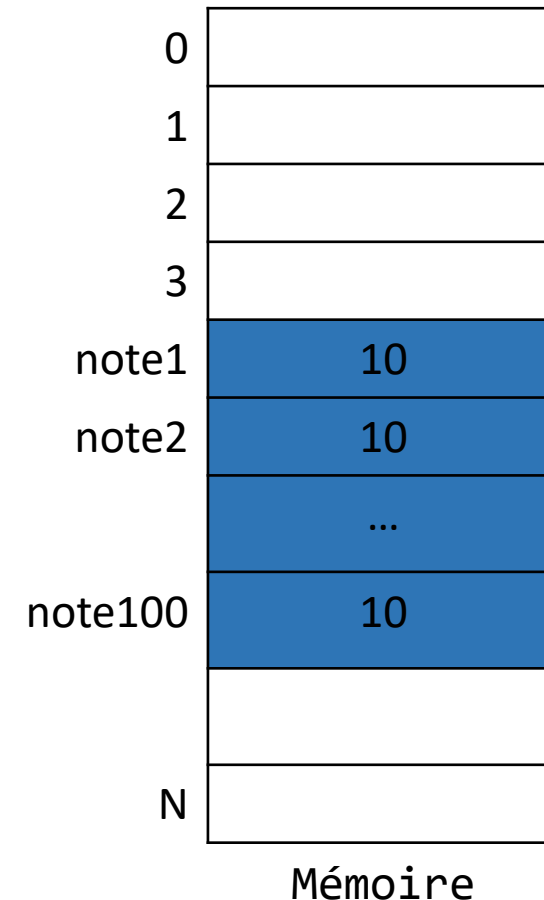
2.5. Les tableaux

Un **tableau** est une série de variables de même type qui sont situées dans un espace contigu en mémoire. **Exemple:** Stocker les notes de 100 étudiants.

Solution naïve:

```
int main()
{
    int note1;
    int note2;
    ...
    int note100;

    note1=10;
    note2=10;
    ...
    note3=10;
}
```



2. Les bases de la programmation C

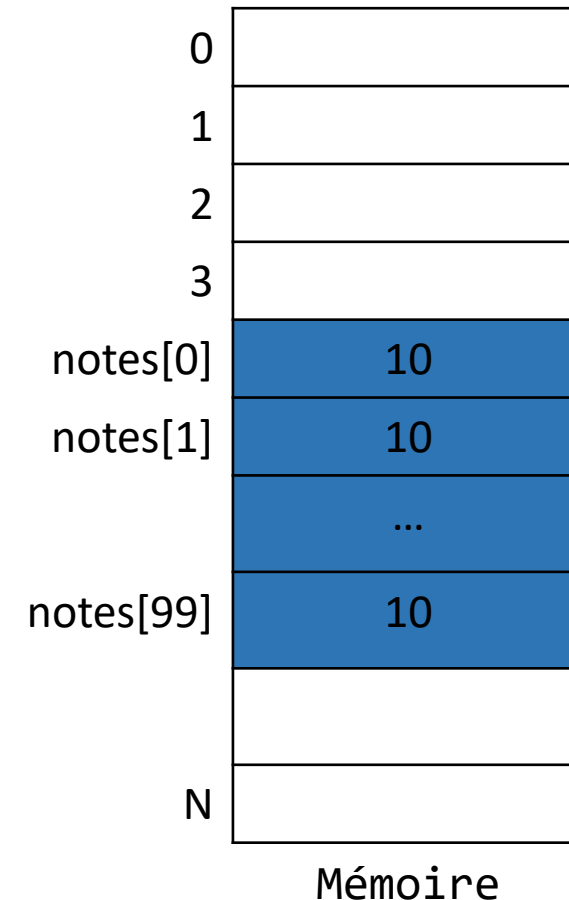
2.5. Les tableaux

Un **tableau** est une série de variables de même type qui sont situées dans un espace contigu en mémoire. **Exemple:** Stocker les notes de 100 étudiants.

Solution 2 (Approche statique):

```
int main()
{
    int notes[100];    // Déclaration du tableau

    int i=0;
    for(i=0;i<100;i++)
    {
        notes[i]=10;   //Parcours du tableau
    }
}
```



2. Les bases de la programmation C

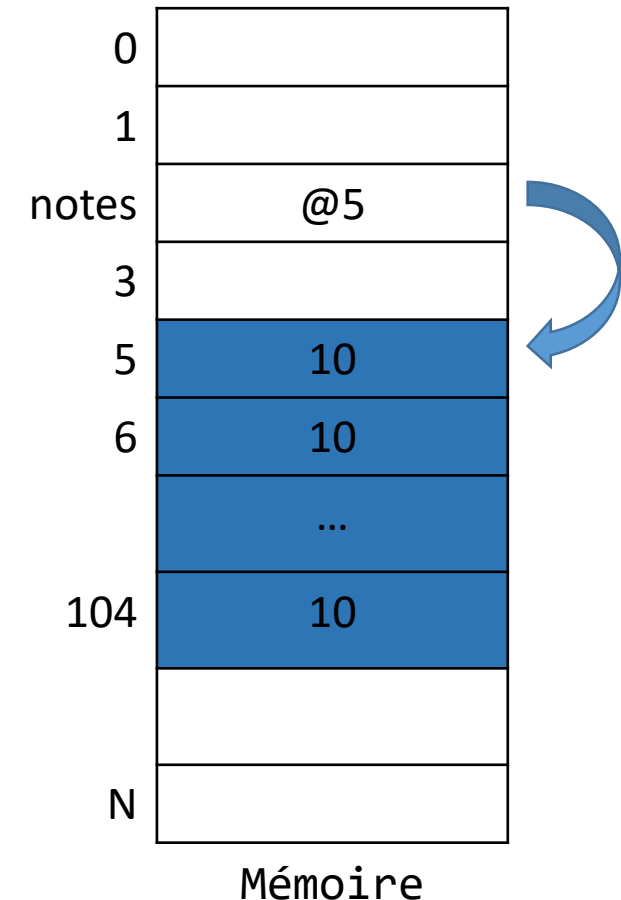
2.5. Les tableaux

Un **tableau** est une série de variables de même type qui sont situées dans un espace contigu en mémoire. **Exemple:** Stocker les notes de 100 étudiants.

Solution 3 (Approche dynamique):

```
int main()
{
    int *notes;                //Déclaration d'un pointeur
    notes=malloc(100*sizeof(int)); //Allocation de la mémoire

    int i=0;
    for(i=0;i<100;i++)        //Parcours du tableau
    {
        *(notes+i)=10;        // notes[i]=10
    }
    free(notes);              //Libération de la mémoire
}
```



2. Les bases de la programmation C

2.6. Les matrices

2. Les bases de la programmation C

2.6. Les matrices

Une **matrice** est un tableau à deux dimensions.

0	1
2	3
4	5

Matrice M

2. Les bases de la programmation C

2.6. Les matrices

Une **matrice** est un tableau à deux dimensions.

La mémoire de la machine étant sur une seule dimension, les lignes de la matrice sont stockées les unes après les autres.

0	1
2	3
4	5

Matrice M

M[0][0]	0
M[0][1]	1
M[1][0]	2
M[1][1]	3
M[2][0]	4
M[2][1]	5

Mémoire

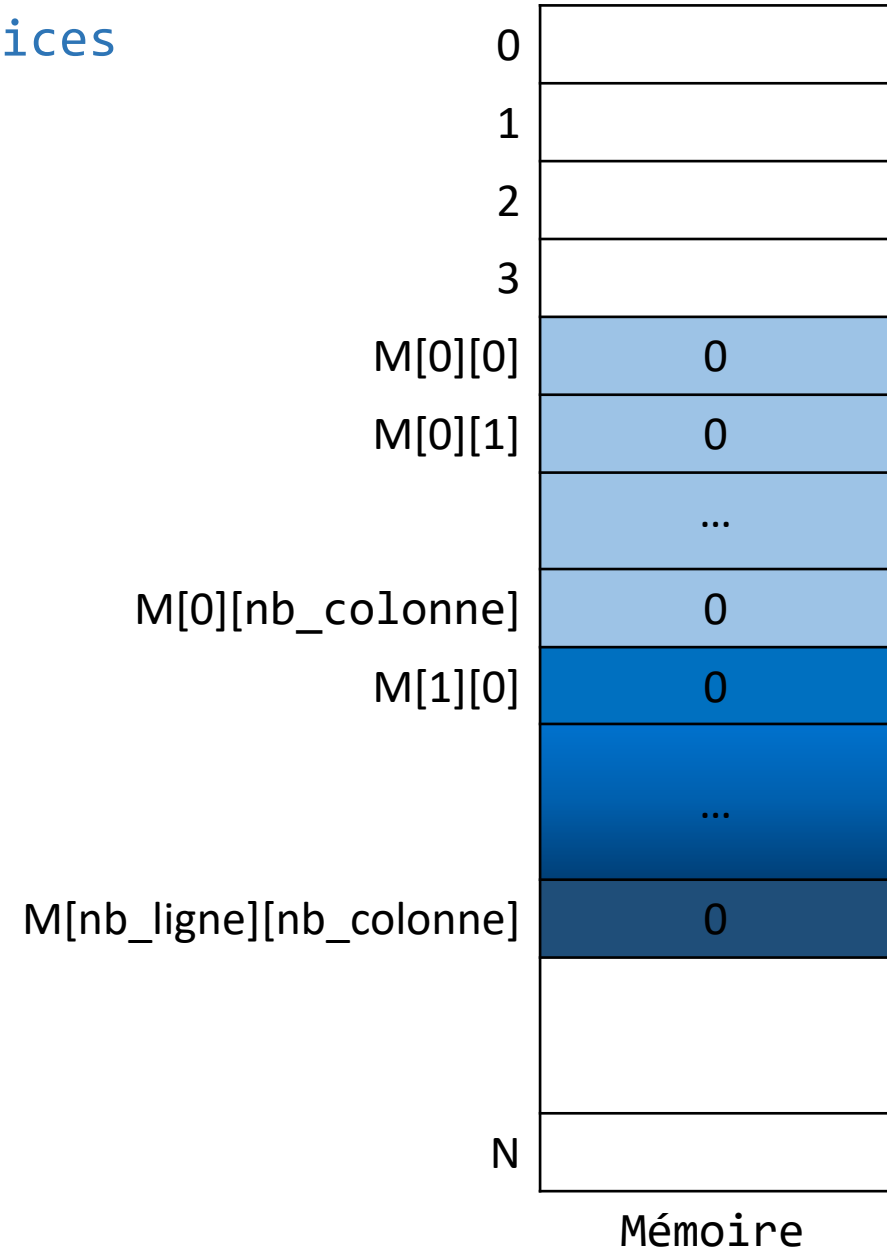
2. Les bases de la programmation C

2.6. Les matrices

1. Approche statique:

```
int main()
{
    int M[nb_ligne][nb_colonne];

    int i, j;
    for(i=0; i<nb_ligne; i++)
    {
        for(j=0; j<nb_colonne; j++)
        {
            M[i][j]=0;
        }
    }
}
```



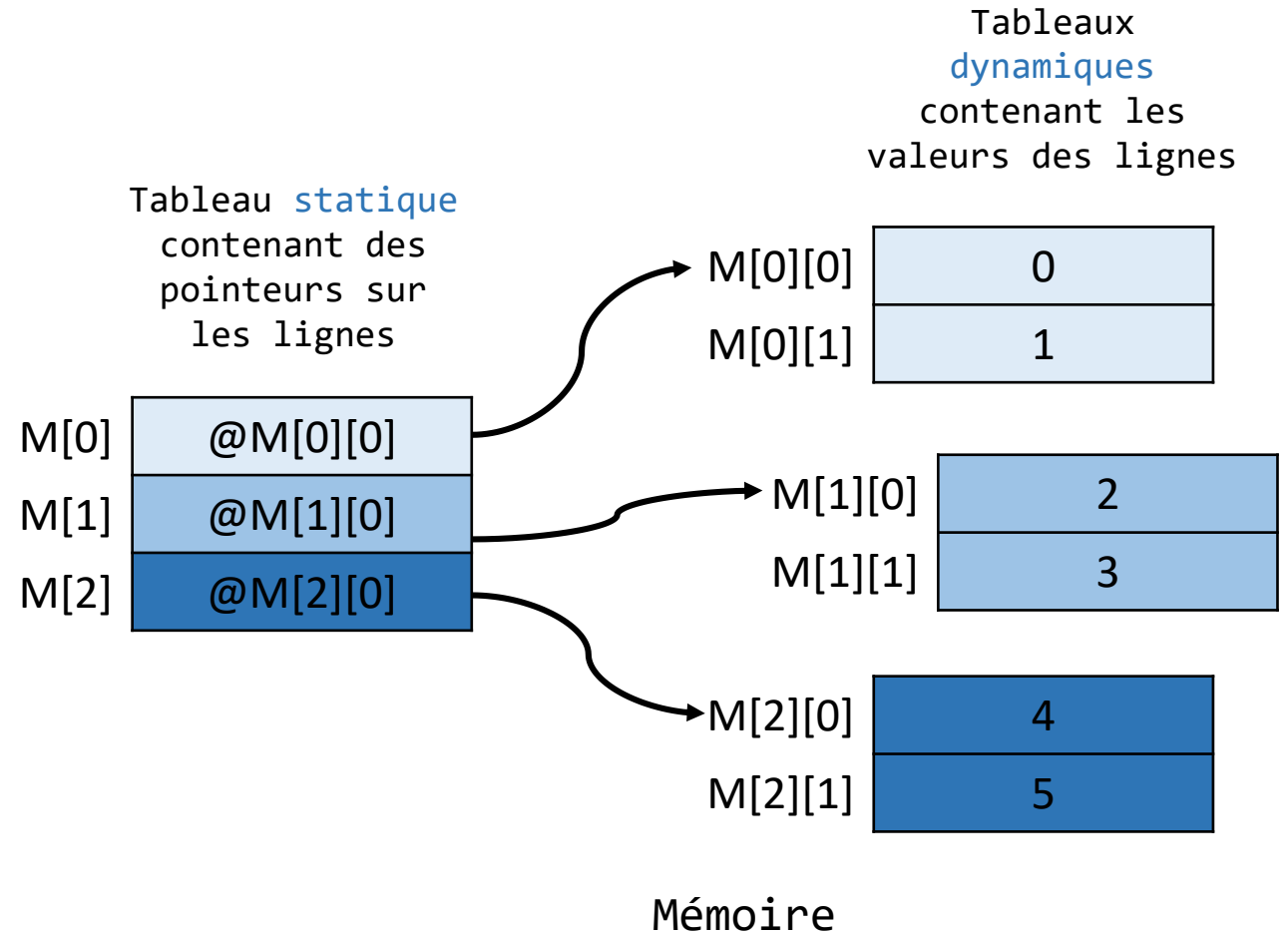
2. Les bases de la programmation C

2.6. Les matrices

2. Tableau de pointeurs

0	1
2	3
4	5

Matrice M



2. Les bases de la programmation C

2.6. Les matrices

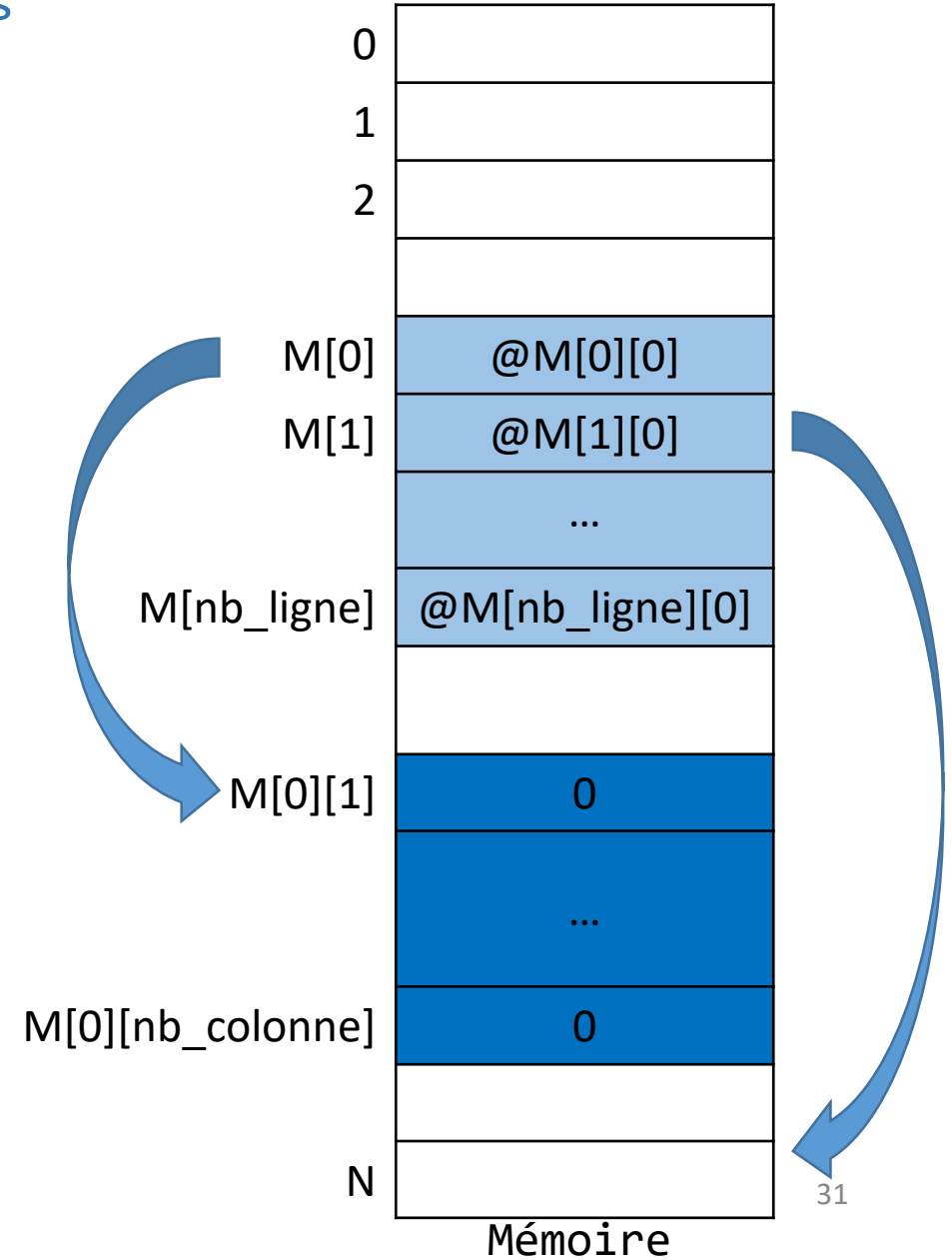
Tableau de pointeurs:

```
int main()
{
    int *M[nb_ligne];

    int i, j;
    for(i=0;i<nb_ligne;i++)
    {
        M[i]=malloc(nb_colonne*sizeof(int));
    }

    for(i=0;i<nb_ligne;i++)
    {
        for(j=0;j<nb_colonne;j++)
        {
            *(M[i]+j)=0;           // M[i][j]=0
        }
    }

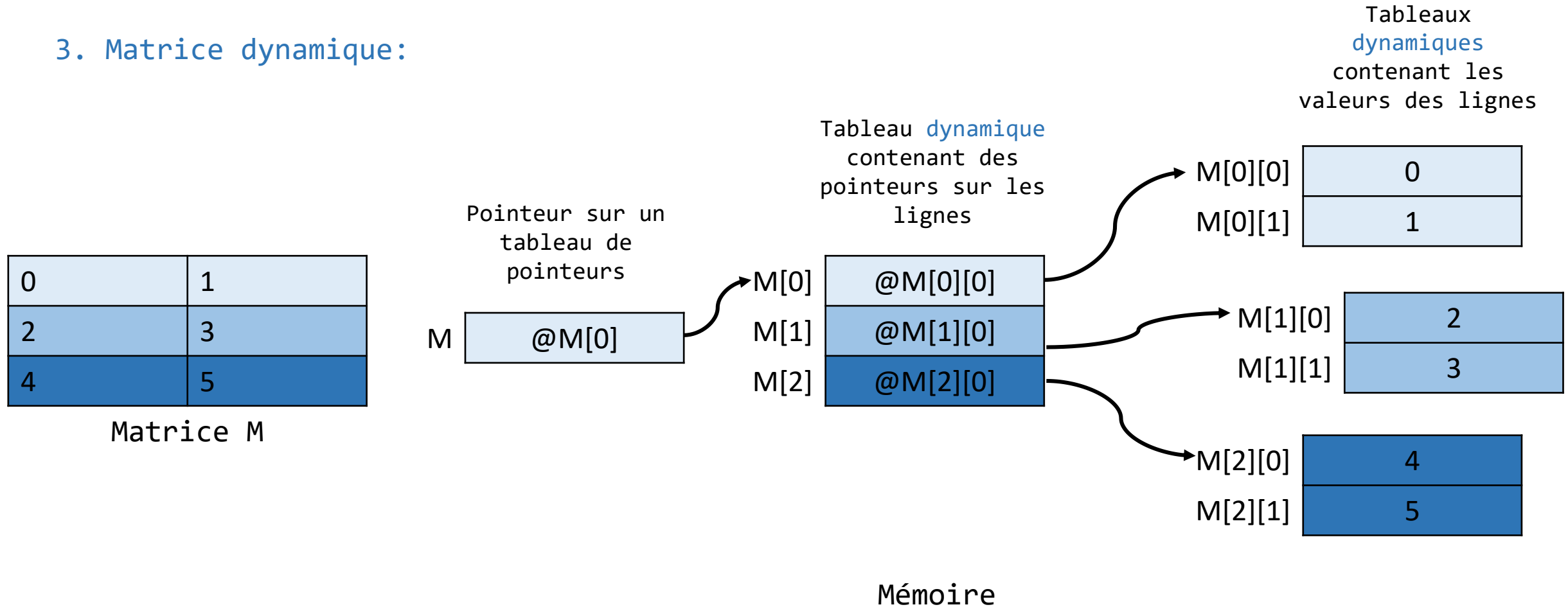
    for(i=0;i<nb_ligne;i++)
    {
        free(M[i]);
    }
}
```



2. Les bases de la programmation C

2.6. Les matrices

3. Matrice dynamique:



2. Les bases de la programmation C

Matrice dynamique:

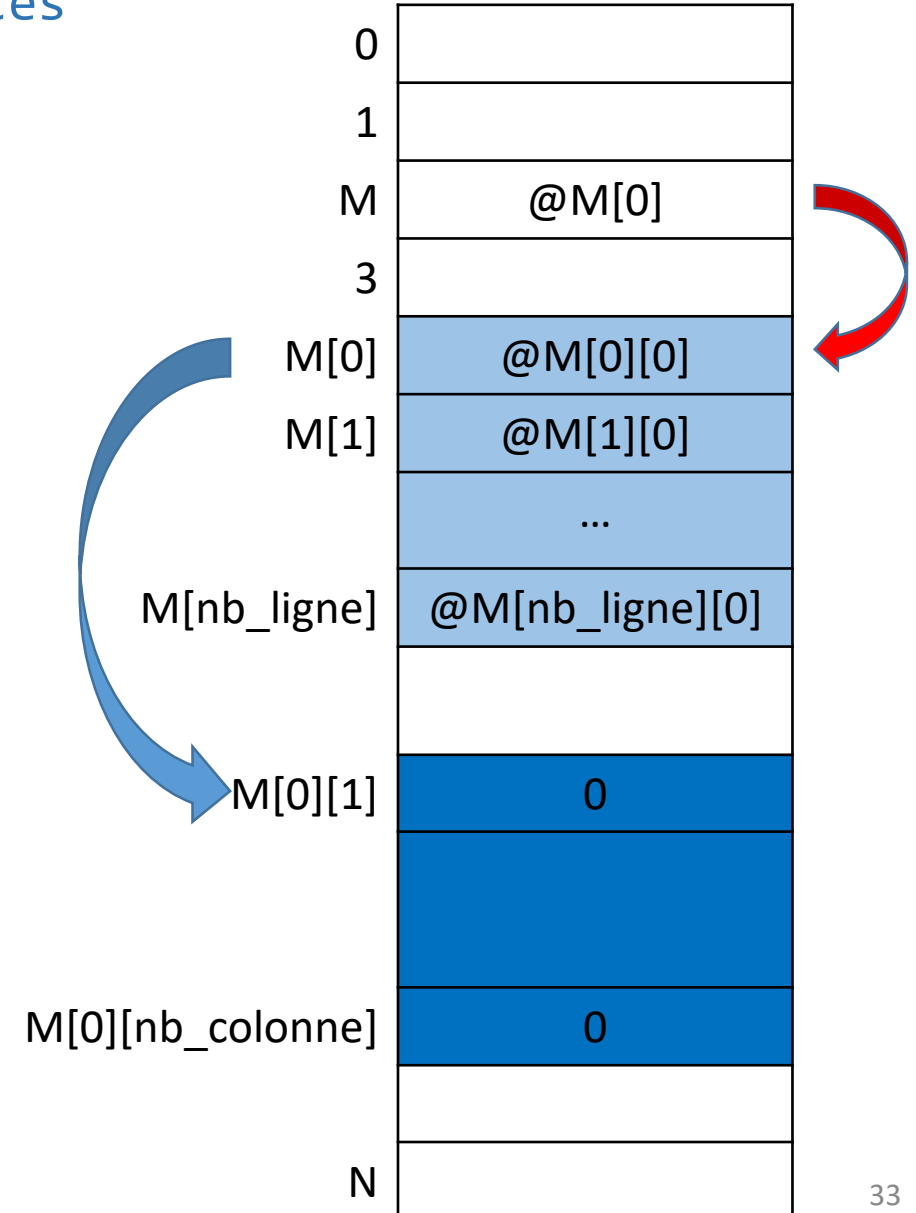
2.6. Les matrices

```
int main()
{
    int **M;

    int i, j;
    M=malloc(nb_ligne*sizeof(*M));
    for(i=0;i<nb_ligne;i++)
    {
        *(M+i)=malloc(nb_colonne*sizeof(**M));
    }

    for(i=0;i<nb_ligne;i++)
    {
        for(j=0;j<nb_colonne;j++)
        {
            (*(M+i)+j)=0;    // M[i][j]=0
        }
    }

    for(i=0;i<nb_ligne;i++)
    {
        free(*(M+i));
    }
    free(M);
}
```



2. Les bases de la programmation C

2.7. Les chaînes de caractères

2. Les bases de la programmation C

2.7. Les chaînes de caractères

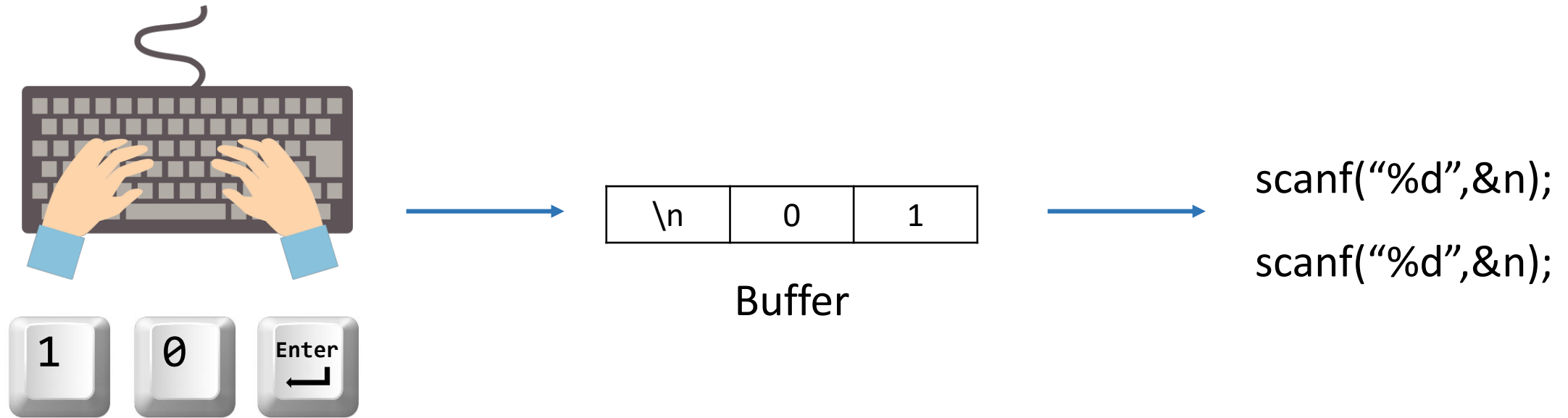
Une **chaîne caractère** est un tableau de caractères dont la fin est marquée par un caractère nul (`'\0'`).

B	o	n	j	o	u	r	\0
---	---	---	---	---	---	---	----

```
int main()
{
    char chaine[8];
    chaine[0]='B';
    chaine[1]='o';
    chaine[2]='n';
    chaine[3]='j';
    chaine[4]='o';
    chaine[5]='u';
    chaine[6]='r';
    chaine[7]='\0';
}
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères



2. Les bases de la programmation C

2.7. Les chaînes de caractères



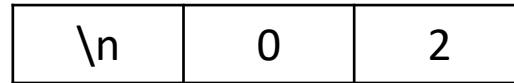
Buffer



```
scanf("%d",&n);// n←10  
scanf("%d",&n);
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères



Buffer



```
scanf("%d",&n);// n←10  
scanf("%d",&n);
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères



Buffer

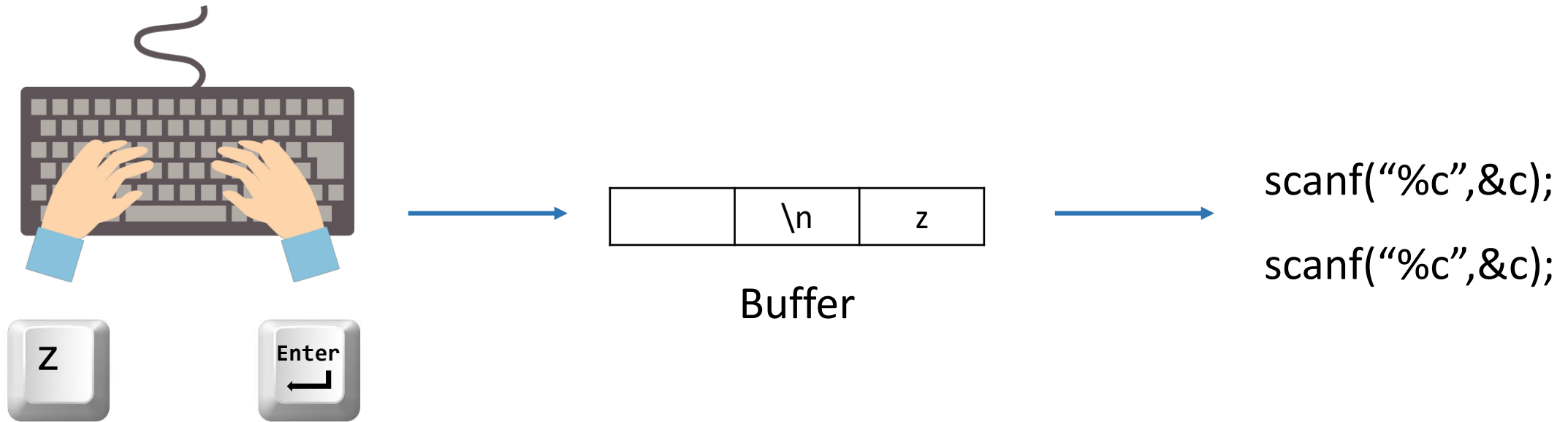


```
scanf("%d",&n);// n←10
```

```
scanf("%d",&n);// n←20
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères



2. Les bases de la programmation C

2.7. Les chaînes de caractères



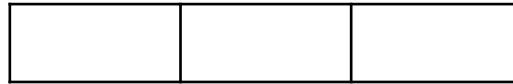
Buffer



```
scanf("%c",&c); //c ← 'z'  
scanf("%c",&c);
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères



Buffer



```
scanf("%c",&c); //c ← 'z'  
scanf("%c",&c); //c ← '\n'
```

2. Les bases de la programmation C

2.7. Les chaînes de caractères

```
scanf("%c%c",c);  
scanf("%c%c",c);
```

```
c=getchar();  
while (getchar () <> '\n') ;  
c=getchar();
```

```
c=getchar();  
fflush(STDIN);  
c=getchar();
```

2. Les bases de la programmation C

2.8. Les fonctions

2. Les bases de la programmation C

2.8. Les fonctions

Une **fonction** est une portion de code qui sert à faire quelque chose de précis.



```
int factorielle(int n)
{
    int i, fact;
    for(i=1; i<n; i++)
        fact*=i;
    return fact;
}

int maint()
{
    int fact;
    fact=factorielle(5);
}
```

2. Les bases de la programmation C

2.8. Les fonctions

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

```
int main()
{
    int n=3,p=5,C;
    int i, factn=1, factp=1, factnp=1;
    for(i=1;i<n;i++)
        factn*=i;
    for(i=1;i<p;i++)
        factp*=i;
    for(i=1;i<(n-p);i++)
        factnp*=i;
    C=factn/(factp*factnp);
}
```

```
int factorielle(int n)
{
    int i, fact=1;
    for(i=1;i<n;i++)
        fact*=i;
    return fact;
}

int main()
{
    int n=3,p=5,C;
    C=factorielle(n)/(factorielle(p)*factorielle(n-p));
}
```

2. Les bases de la programmation C

2.8. Les fonctions

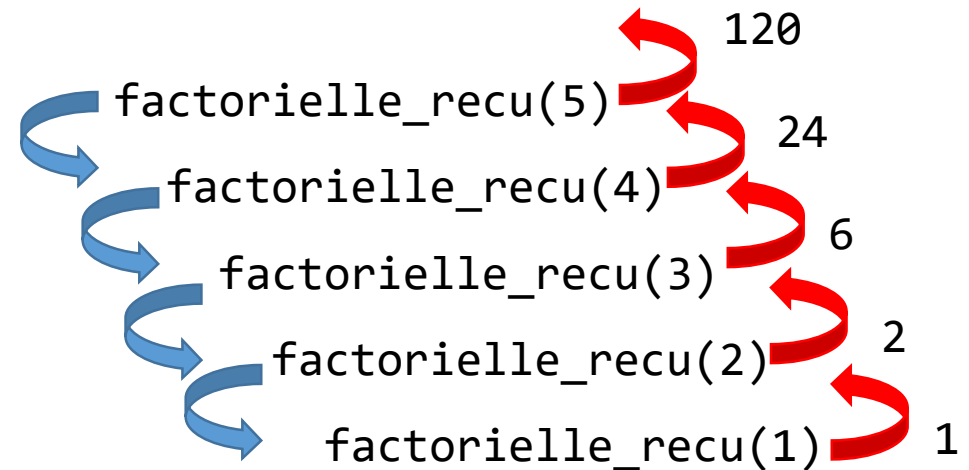
Fonction itérative.

```
int factorielle_iter(int n)
{
    int i, fact=1;
    for(i=1;i<n;i++)
        fact*=i;
    return fact;
}
```

It1: i=1, fact=1
It2: i=2, fact=2
It3: i=3, fact=6
It4: i=4, fact=24
It5: i=5, fact=120

Fonction récursive.

```
int factorielle_recu(int n)
{
    if(n==0 || n==1) return 1;
    return n*fact_rec(n-1);
}
```



2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

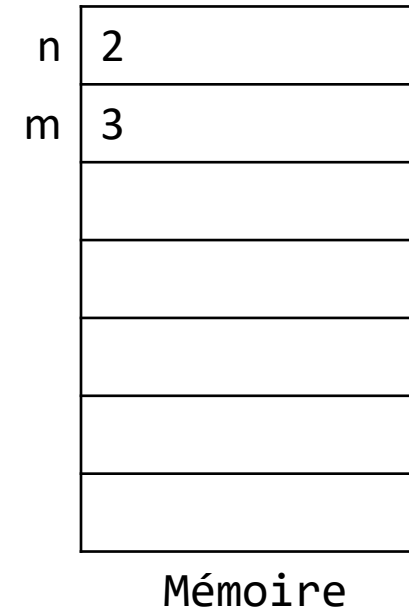
int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```

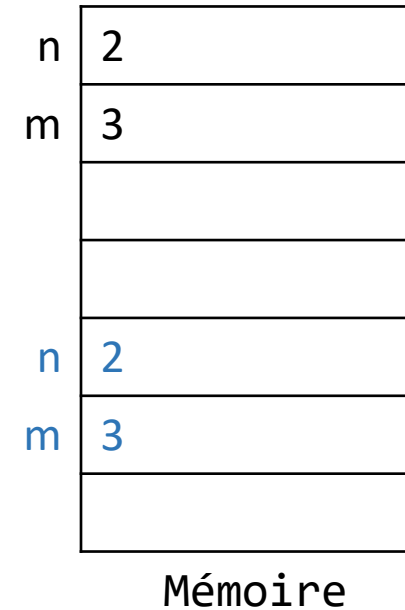


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```

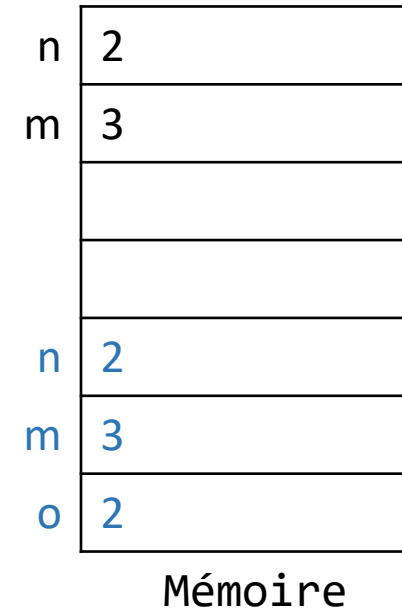


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```

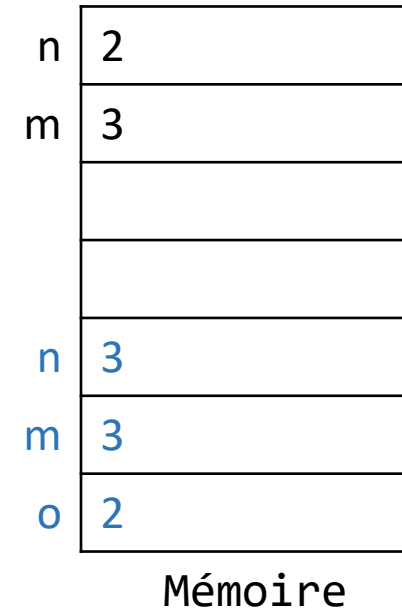


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```

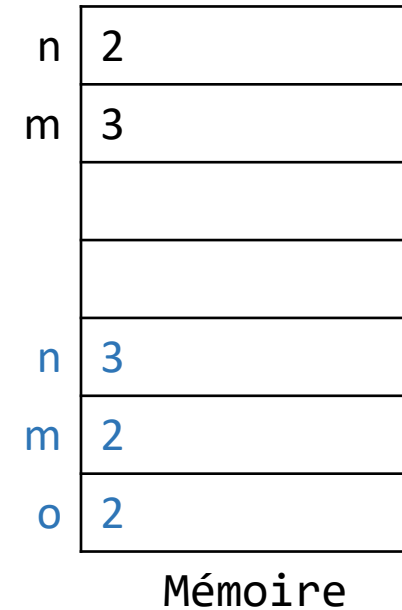


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```

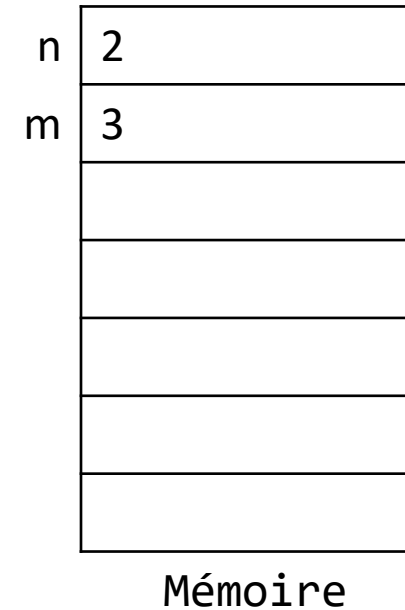


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int n, int m)
{
    int o = n;
    n = m;
    m = o;
}

int main()
{
    int n=2,m=3;
    echange(n,m);
    printf("%d %d", n, m);
}
```



2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

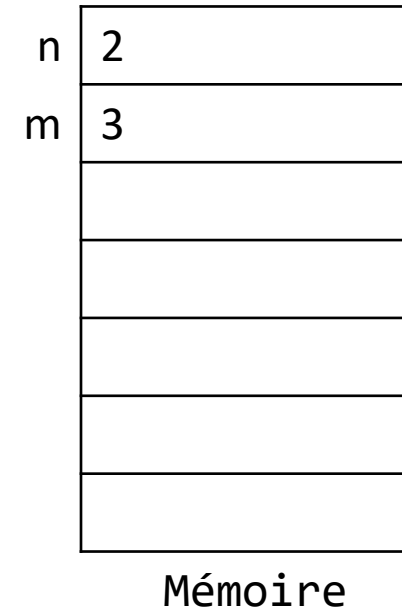
int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

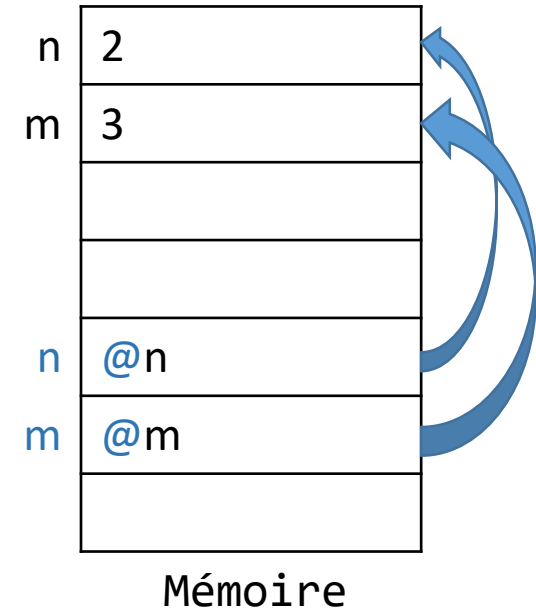


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

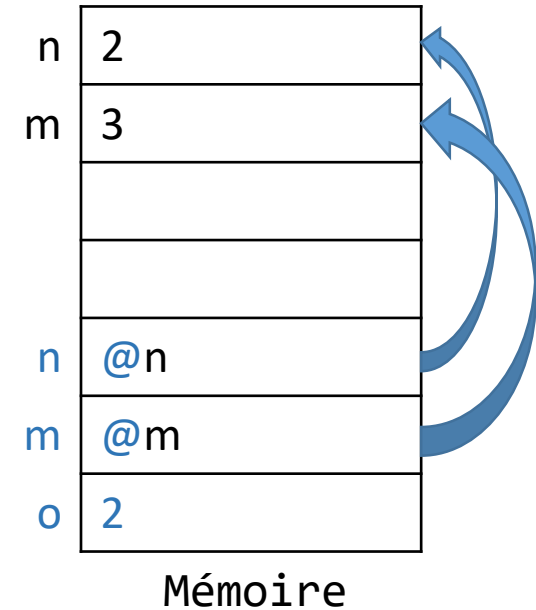


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

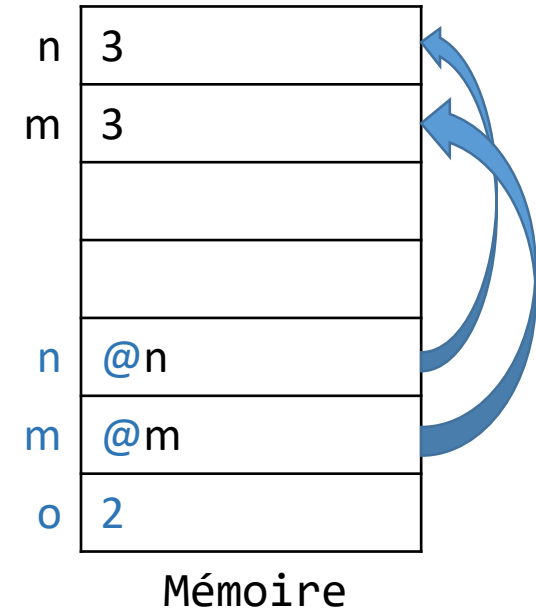


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

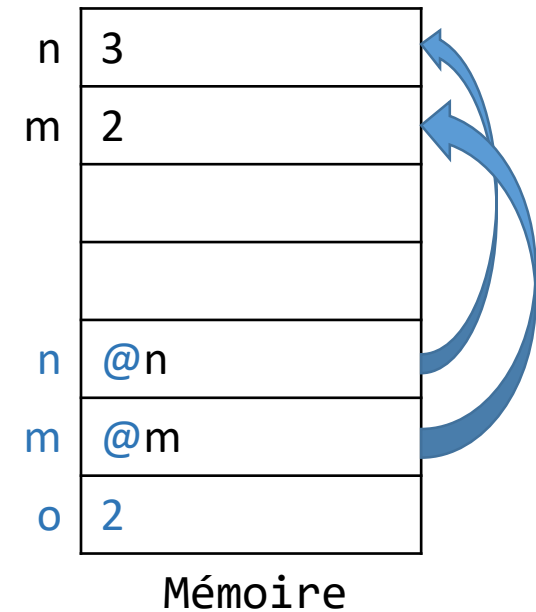


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```

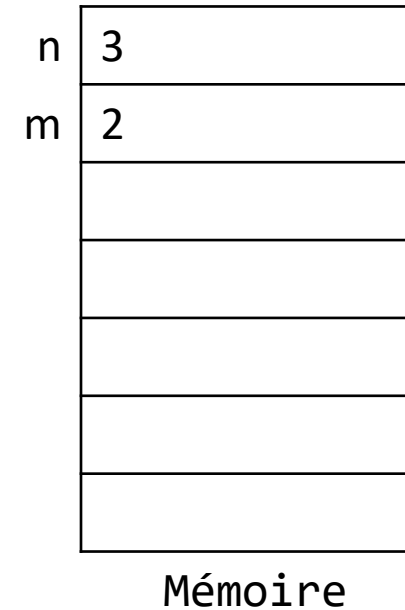


2. Les bases de la programmation C

2.8. Les fonctions

```
void echange(int *n, int *m)
{
    int o = *n;
    *n = *m;
    *m = o;
}

int main()
{
    int n=2,m=3;
    echange(&n,&m);
    printf("%d %d", n, m);
}
```



2. Les bases de la programmation C

2.9. Les structures

2. Les bases de la programmation C

2.9. Les structures

Une **structure** est un ensemble de variables pouvant être de différents types.

Exemple: Stocker des informations de 100 étudiants (Nom, prénom, matricule...etc).

Solution naïve

```
int main()
{
    char *noms[100];
    char *prenoms[100];
    int matricules[100];

    matricules[0]=1;
}
```

Solution avec structure

```
typedef struct
{
    char *nom;
    char *prenom;
    int matricule;
}etudiant;

int main()
{
    etudiant et[100];
    et[0].matricule=1;
}
```