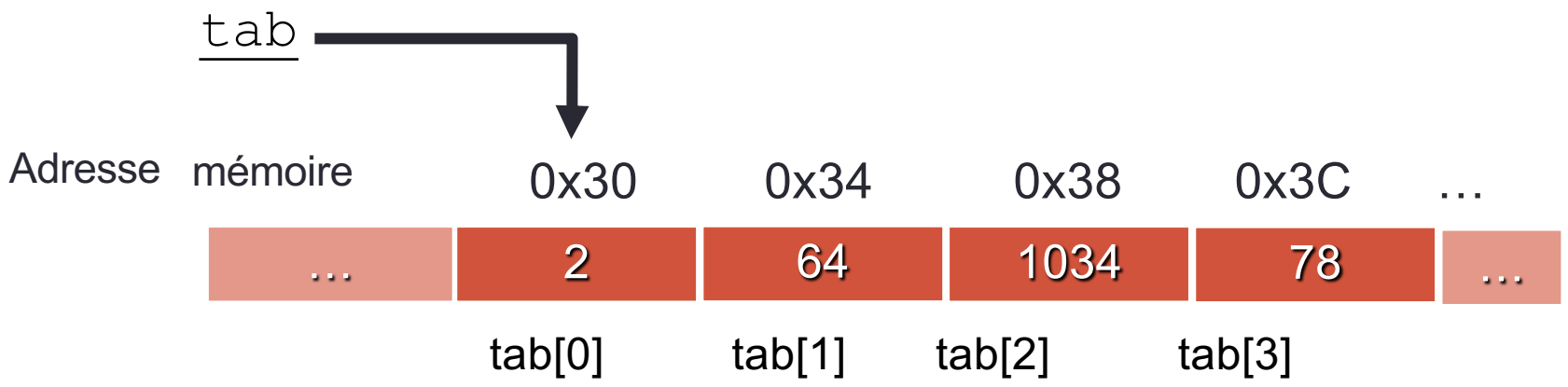
Sommaire chapitre 5

Tableau et Pointeur

L'identifiant (le nom) d'un tableau est un pointeur constant, sur l'adresse de son premier élément :

`tab` ⟷ `& tab[0]`

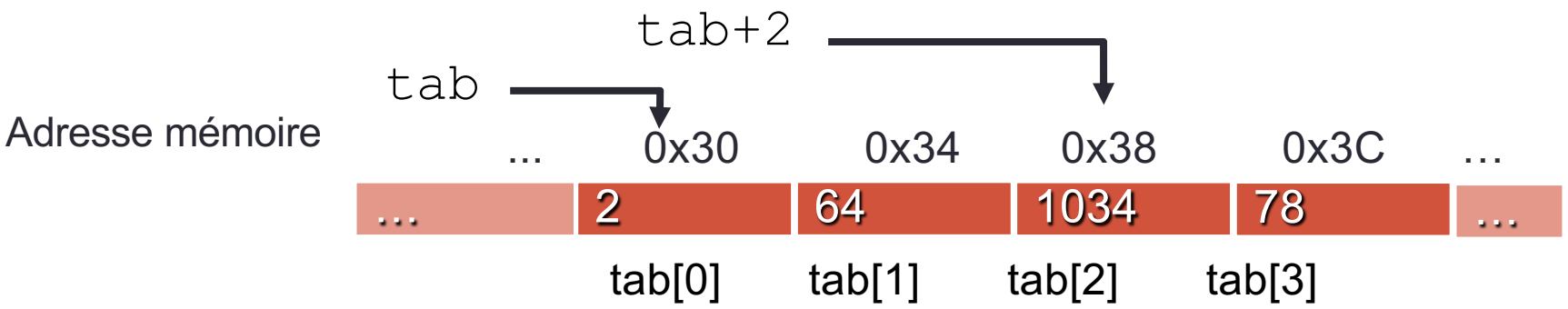
```
int tab[4];
```



Tableaux et arithmétique du pointeur

D'après l'arithmétique des pointeurs,
`tab+i` pointe sur le `i`ème élément suivant en mémoire de même type.

Pour un tableau : `*(tab+i)` ↔ `tab[i]`
car `tab+i` ↔ `&(tab[i])`



Tableaux et pointeurs

```
*(tab+2)=-1;
```
↔
```
tab[2]=-1
```


```
*tab = *(tab+2);
/* parentheses! */
```
↔
```
tab[0]= tab[2]
```


```
*tab = *tab + 2 ;
/*no parenthese*/
```
↔
```
tab[0]= tab[0]+2
```

Sommaire chapitre 5

Allocation statique de mémoire

- Allocation d'un tableau statique :

```
int tab[3];
```

avec une constante pour la taille définie avant l'exécution.

La mémoire du tableau est alloué à sa création est n'est pas modifiable ensuite.

- Mais que faire si la taille n'est pas connue avant l'exécution ?
  - si la taille dépend du résultat d'une instruction
  - si l'utilisateur choisit la taille pendant l'exécution

    via `int taille; scanf("%d",&taille);`

    car `int tab[taille]` n'est pas autorisé, ni prévu en C ANSI !

Allocation dynamique de mémoire

- La solution: allocation dynamique de mémoire
  - demander au système d'allouer de la mémoire pendant l'exécution du programme avec la fonction `malloc` (memory allocation)

```
#include <stdlib.h>
        void* malloc (int size);
```

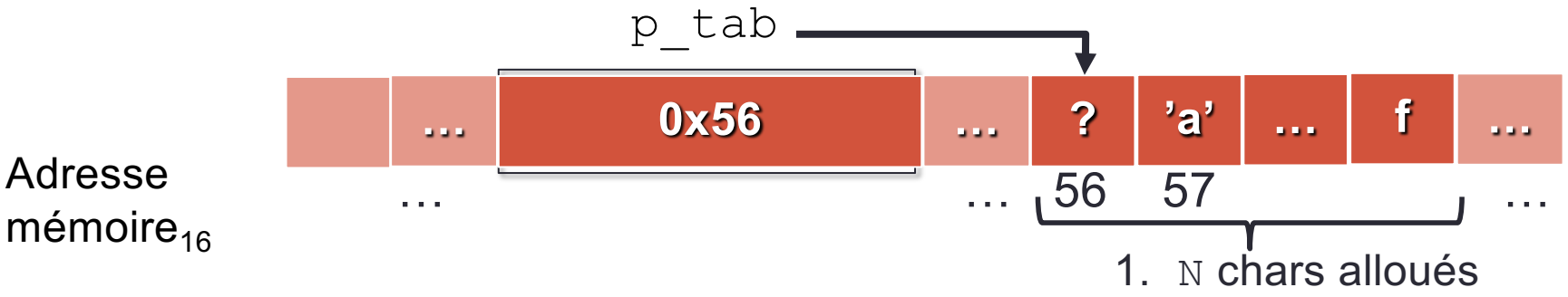
  - réserve un bloc contiguë de `size` octets en mémoire
  - retourne :
    - un pointeur générique (`void*`) sur l'adresse du bloc réservé
    - le pointeur `NULL` si il n'y a pas de mémoire disponible

Allocation dynamique d'un tableau

```
char *p_tab;
int N=3;
p_tab= (char*) malloc ( N*sizeof(char) );
```

1. Recherche et réservation d'un bloc de (N * taille d'un char) octets en mémoire

```
p_tab[1] ='a';
```



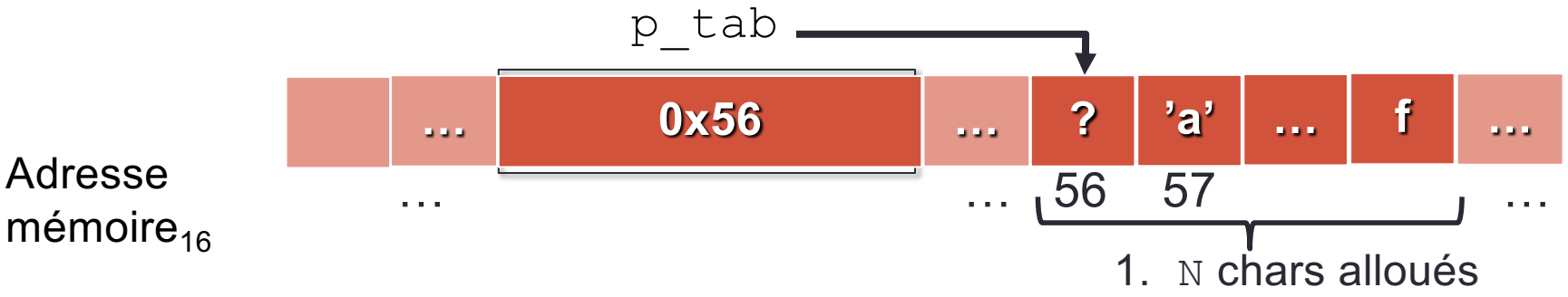
Allocation dynamique d'un tableau

```
char *p_tab;
int N=3;
p_tab= (char*) malloc ( N*sizeof(char) );
```

1. Recherche et réservation d'un bloc de (N * taille d'un char) octets en mémoire

2. cast: conversion du pointeur/adresse (void*) retourné en (char*)

```
p_tab[1] ='a';
```



Allocation dynamique d'un tableau

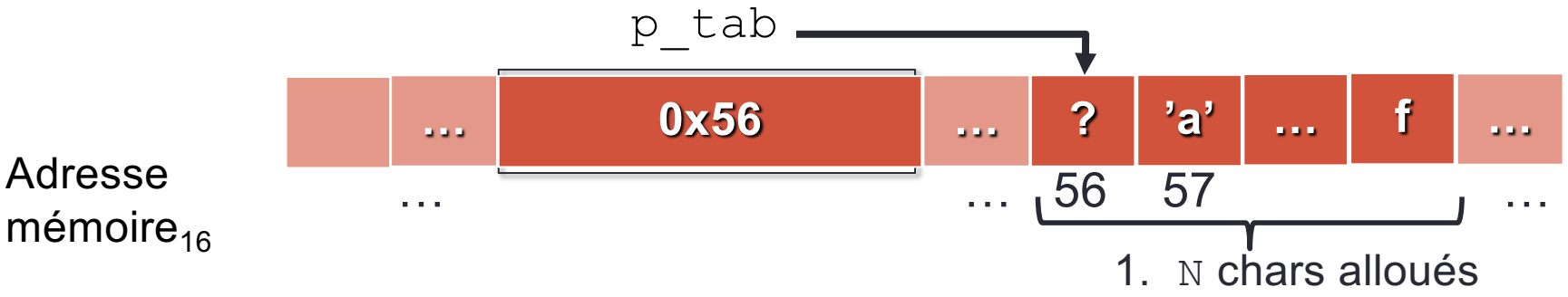
```
char *p_tab;
int N=3;
p_tab= (char*) malloc ( N*sizeof(char) );
```

1. Recherche et réservation d'un bloc de (N * taille d'un char) octets en mémoire

2. cast: conversion du pointeur/adresse (void*) retourné en (char*)

3. l'adresse du bloc est mémorisé dans notre pointeur pour usage ultérieur

```
p_tab[1] ='a';
```



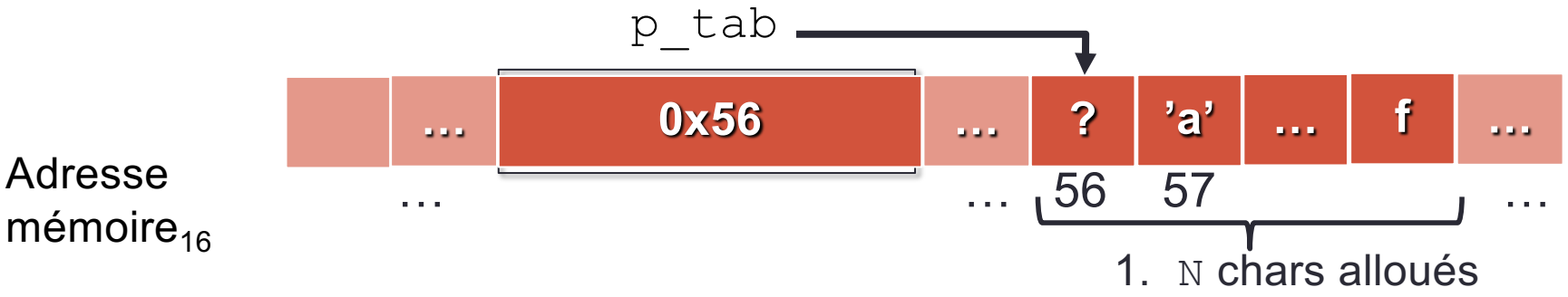
Allocation dynamique d'un tableau

```
char *p_tab;
int N=3;
p_tab= (char*) malloc ( N*sizeof(char) );
```

1. Recherche et réservation d'un bloc de (N * taille d'un char) octets en mémoire
2. cast: conversion du pointeur/adresse (void*) retourné en (char*)
3. l'adresse du bloc est mémorisé dans notre pointeur pour usage ultérieur

```
p_tab[1] ='a';
```

On utilise le pointeur comme un tableau classique!



Allocation dynamique de la mémoire

- Toujours tester le succès de l'allocation de mémoire :
  - pour éviter une erreur de segmentation

```
if (p_tab == NULL)
   {  printf("echec de l'allocation");
      return (-1);          /*on retourne ou quitte */
   }
```

- Libération de la mémoire allouée par `malloc()` :

```
free(void *ptr);
```

- libère le bloc mémoire d'adresse `ptr` alloué par un `malloc`.
- la mémoire libérée peut être réutilisée

Allocation dynamique d'un tableau d'int

```
#include <stdlib.h>                          /*prototype de malloc() */

int main(void)
{   int n, *tab;

    printf ("Taille désiree? ");
    scanf ("%d",&n);
    tab=(int*) malloc (n*sizeof(int));              /*allocation*/
```

Allocation dynamique d'un tableau d'int

```
#include <stdlib.h>                          /*prototype de malloc() */

int main(void)
{   int n, *tab;

    printf ("Taille désiree? ");
    scanf ("%d",&n);
    tab=(int*) malloc (n*sizeof(int));              /*allocation*/

    if (tab==NULL)
    { printf("erreur allocation tab"); return -1;
    }
```

Allocation dynamique d'un tableau d'int

```
#include <stdlib.h>                          /*prototype de malloc() */

int main(void)
{   int n, *tab;

    printf ("Taille désiree? ");
    scanf ("%d",&n);
    tab=(int*) malloc (n*sizeof(int));              /*allocation*/

    if (tab==NULL)
    { printf("erreur allocation tab"); return -1;
    }

    tab[n-1]=1;
```

Allocation dynamique d'un tableau d'int

```
#include <stdlib.h>                          /*prototype de malloc() */

int main(void)
{   int n, *tab;

    printf ("Taille désiree? ");
    scanf ("%d",&n);
    tab=(int*) malloc (n*sizeof(int));              /*allocation*/

    if (tab==NULL)
    { printf("erreur allocation tab"); return -1;
    }

    tab[n-1]=1;

    free(tab);                                  /* libérer la mémoire*/
    return 0; }
```

TABLEAU ET FONCTION

Premiers pas !

Tableaux : passage à une fonction

- On peut passer un tableau statique ou dynamique à une fonction.
- L'argument formel à utiliser dans la fonction est alors de la forme

  `type* name` **ou** `type name[]`

  avec `type` celui des éléments du tableau.

```
void affiche ( int *tab, int n )
  ou
void affiche ( int tab[], int n )
 {     int i;
       for (i=0; i<n ; i++)
            {  printf("%d\n",tab[i]); }
 }
```

Tableaux : passage à une fonction

- Exemple 1 : fonction d'affichage d'un tableau

```
void affiche ( int *tab, int n )
 {     int i;
       for (i=0; i<n ; i++)
            {  printf("%d\n",tab[i]); }
 }

int main(void)
{    int u[3]={1,-2,7};
     affiche(u,3);
     return(0);
}
```

Tableaux : passage à une fonction

- Exemple 2 : fonction de calcul de la somme des éléments

```
double somme_elts ( double *tab, int n )
{    int i; double somme=0;
        for (i=0; i<n ; i++)
       { somme += tab[i]; }
       return (somme);
}

int main(void)
{    double u[3]={1,-2,7};
     double sum_u;
     sum_u=somme_elts(u,3);
     return(0);
}
```

Tableaux : passage à une fonction

- Un tableau n'est pas passé par valeur, mais par adresse (voir futur CI).
- **Conséquence : toute modification d'un tableau dans une fonction est conservé après retour de la fonction**
  (toujours vrai pour un tableau mais faux pour une variable !)

```
double mod ( double *tab, int n )
{      tab[1]=4;
}

int main(void)
{    double u[3]={1,-2,7};
     mod(u,3);
     printf("%d\n", u[0]);        ← affichera 4
     return(0);
}
```