Cours n° B.3

Pointeurs

Pointeurs

- ☞ Un pointeur est une variable dont le contenu est une adresse-mémoire
- ☞ Un pointeur doit être défini en fonction du type de la donnée stockée à cette adresse mémoire :

```
char *  pc;   pointeur de caractères
 int *  pi;   pointeur d'entier
toto *  pt;   pointeur de toto
void *  pv;   pointeur de quelque chose
```

- ☞ **L'opérateur unaire «&» donne l'adresse-mémoire d'une donnée.**
  - ➠ Il ne s'applique qu'aux objets stockés en mémoire (variables, éléments de tableaux, etc) et donc pas aux expressions, constantes, etc
- ☞ **L'opérateur unaire «*» donne la donnée stockée à une adresse mémoire**.
  - ➠ C'est un opérateur d'indirection

`char c = 'A';` définition d'une variable

- ☞ de type `char` ;
- ☞ de nom `c` ;
- ☞ de valeur initiale 65 (ou `'A'`)
- ☞ d'adresse `&c`

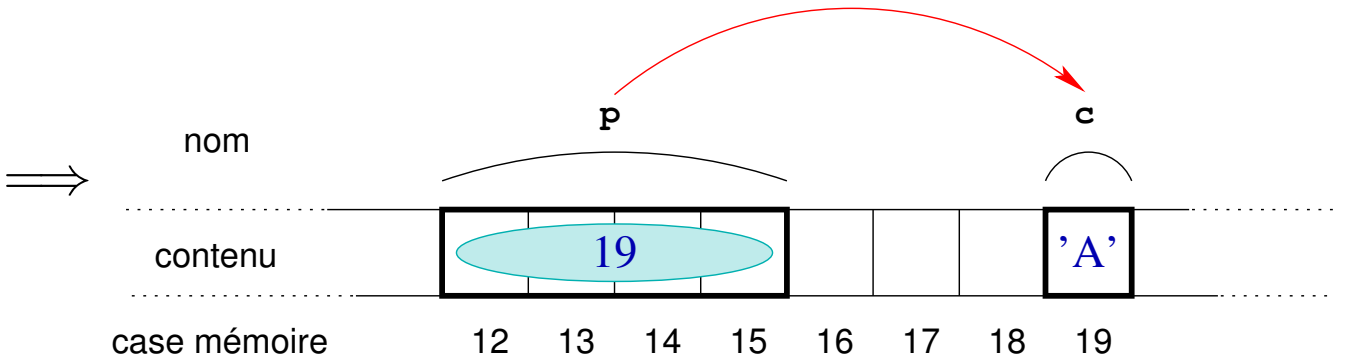
`char * p = &c;` définition d'un pointeur

- ☞ de type `char *` ;
- ☞ de nom `p` ;
- ☞ de valeur initiale `&c`

➥ `p` pointe sur `c`

Opérateurs sur les pointeurs

```
char     c;
char *   p;

c = 'A';
p = &c;
```



☞ p et &c ont la même valeur

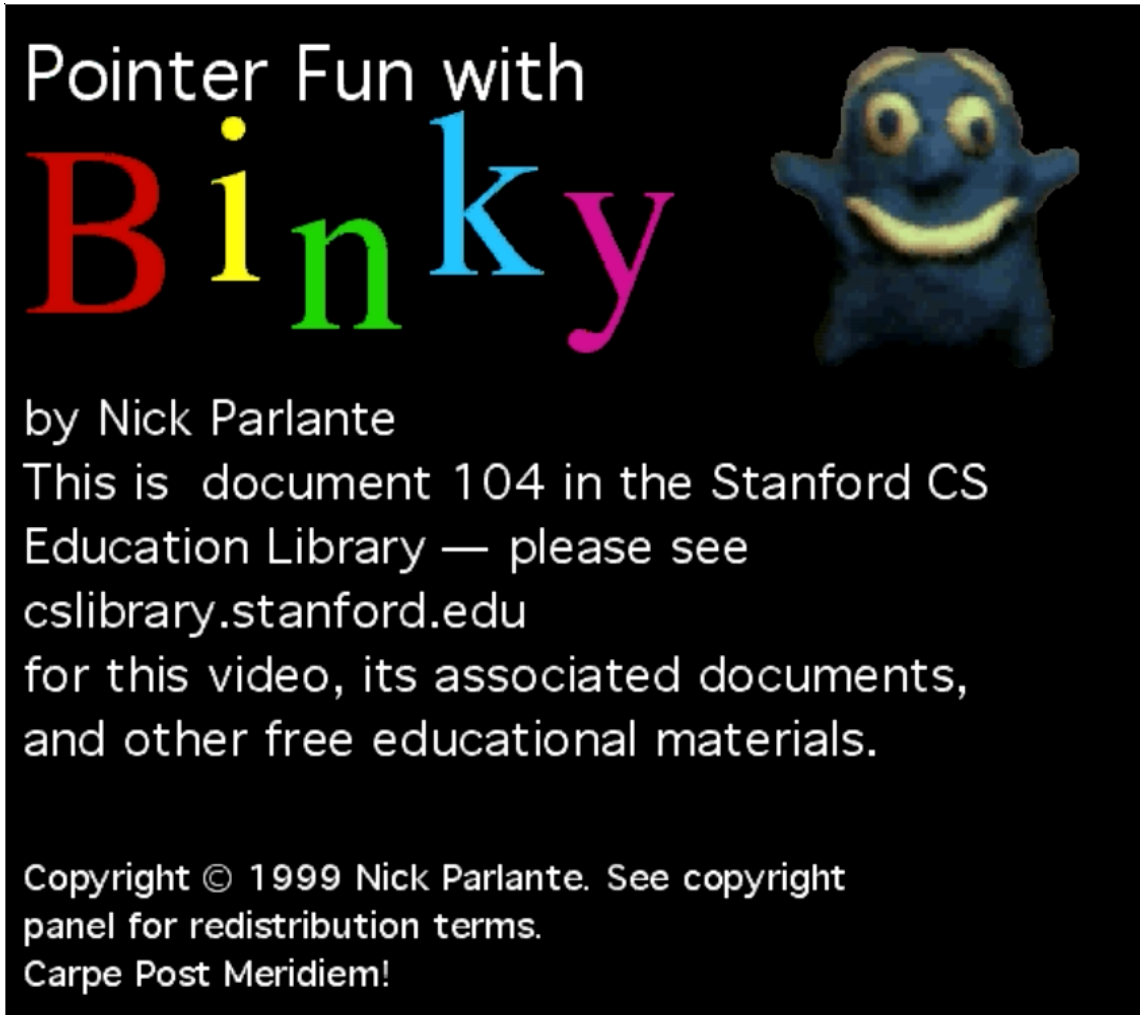
☞ *p et c ont la meme valeur : 65 (qui est représentable par 'A')

```c
#include <stdio.h>

int main (int argc, char ** argv)
{
  char c = 'A';

  char* p;

  p = &c;

  printf("c = %c\n", c);
  printf("c = %d\n", c);

  printf("\n");

  printf("*p = %c\n", *p);
  printf("*p = %d\n", *p);

  printf("\n");

  printf("p = %xd\n", p);

  printf("\n");

  printf("&c = %xd\n", &c);

  return 0;
}
```

```
int x=1;
int y=2;
int t[10];

int * pi; ..... pointeur d'entier

pi = &x; ..... pi pointe sur x

y = *pi; ..... y vaut 1

*pi = 0; ..... x vaut 0

pi = &t[5]; ..... pi pointe sur t[5]
```

Un pointeur ne peut pointer qu'un objet du type pour lequel il a été déclaré !

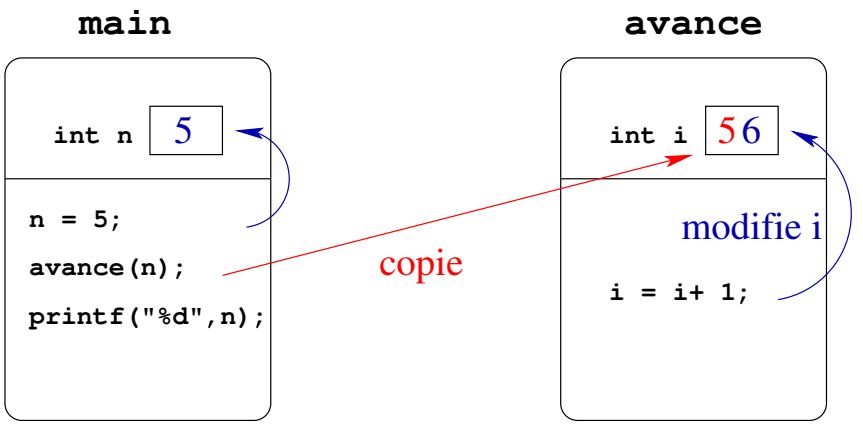
Binky !



This is document 104 in the Stanford CS Education Library. Please see http ://cslibrary.stanford.edu/ for this and other free educational materials.
Copyright Nick Parlante 1999.

Passage de paramètres (1)

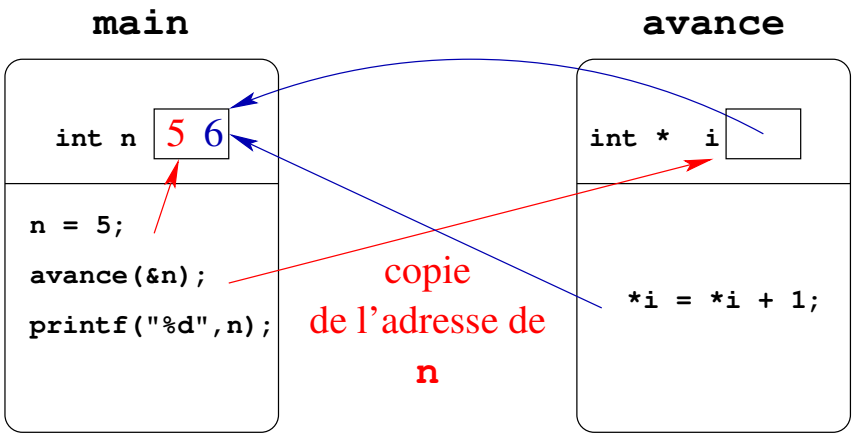
```c
#include <stdio.h>

void avance(int i)
{
  i = i + 1;
}

int main (int argc, char **argv)
{
  int n = 5;

  avance (n);

  printf ("%d\n", n);

  return 0;
}
```



➥ Passage d'une valeur

Passage de paramètres (2)

```c
#include <stdio.h>

void avance(int *i)
{
  *i = *i + 1;
}

int main (int argc, char **argv)
{
  int n = 5;

  avance (&n);

  printf ("%d\n", n);

  return 0;
}
```



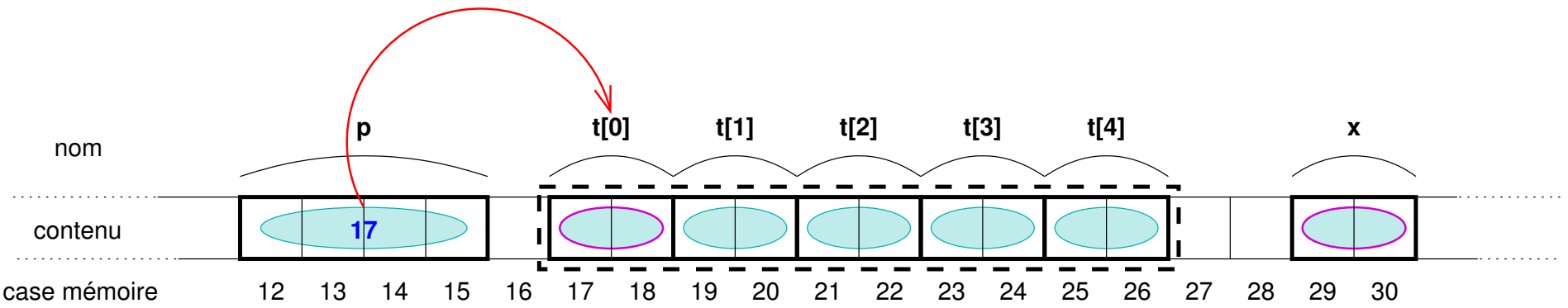
➥ Passage d'une référence

Pointeurs & tableaux (1)

`short t[5];` ..... définir un tableau de 5 entiers

`short * p;` ..... définir un pointeur d'entier

`short x;` ..... définir un entier



`p = &t[0];` ..... `p` pointe vers le premier élément de `t`

`x = *p;` ..... copie le contenu de `t[0]` dans `x`

Pointeurs & tableaux (2)

La valeur d'une variable de type tableau est l'adresse du 1er élément du tableau !

`p = &t[0]` $\equiv$ `p = t`

Un tableau est une constante

`p++` $\Rightarrow$ OK alors que `t++` $\Rightarrow$ INTERDIT

Exemple live

Pointeurs & tableaux (3)

```
char machaine[] = "bonjour";
```

définition d'un tableau de caractères de taille nécessaire pour contenir la chaîne avec lequel le tableau est initialisé.

➥ Il est possible de modifier le contenu du tableau mais **pas** l'espace pointé.

```
char * machaine = "bonjour";
```

définition d'un pointeur de caractère initialisé en pointant sur la chaîne constante `"bonjour"`.

➥ Il est possible de modifier le pointeur pour le faire pointer sur une autre zone mémoire mais il **n'est pas possible de modifier le contenu** de la chaîne constante.

Les pointeurs de structures (1)

```
struct point *ptr1, *ptr;
```
déclaration de pointeurs

```
ptr1 = &p1;
```
affectation de l'adresse

```
(*ptr).x = 5;
```
accès à un membre

```
ptr->y = 6;
```
autre notation

Les pointeurs de structures (2)

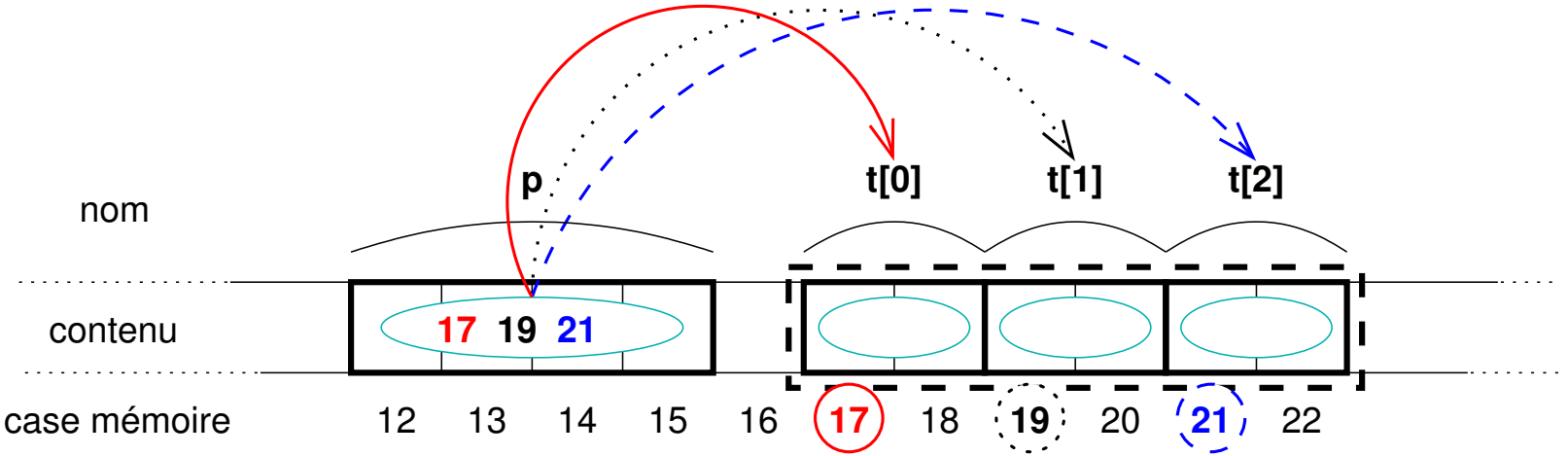
```
struct personne {
    char nom[9];
    int numero;
};

struct personne moi;
struct personne *per;

per = &moi;

scanf("%c",&(per->nom[0])); ...;
scanf("%c",&(per->nom[7]));
per->nom[8] = '\0';

per->numero = 402;

printf("%s %d\n", per->nom,per->numero);
```

Calcul sur les pointeurs

Certaines opérations sont définies sur les pointeurs :

- ☞ pointeur + entier
- ☞ pointeur – entier
- ☞ pointeur – pointeur
- ☞ comparaison de pointeurs (==, !=, <, <=, >, >=)

```
p = t;
p = p + 1;
p ++;
```



Par convention, on a l'habitude d'affecter `0` ou `NULL` à un pointeur qui ne pointe rien.

Arguments de ligne de commande

Un programme C peut être appelé avec des paramètres, dans ce cas, deux paramètres sont passés à la fonction `main` :

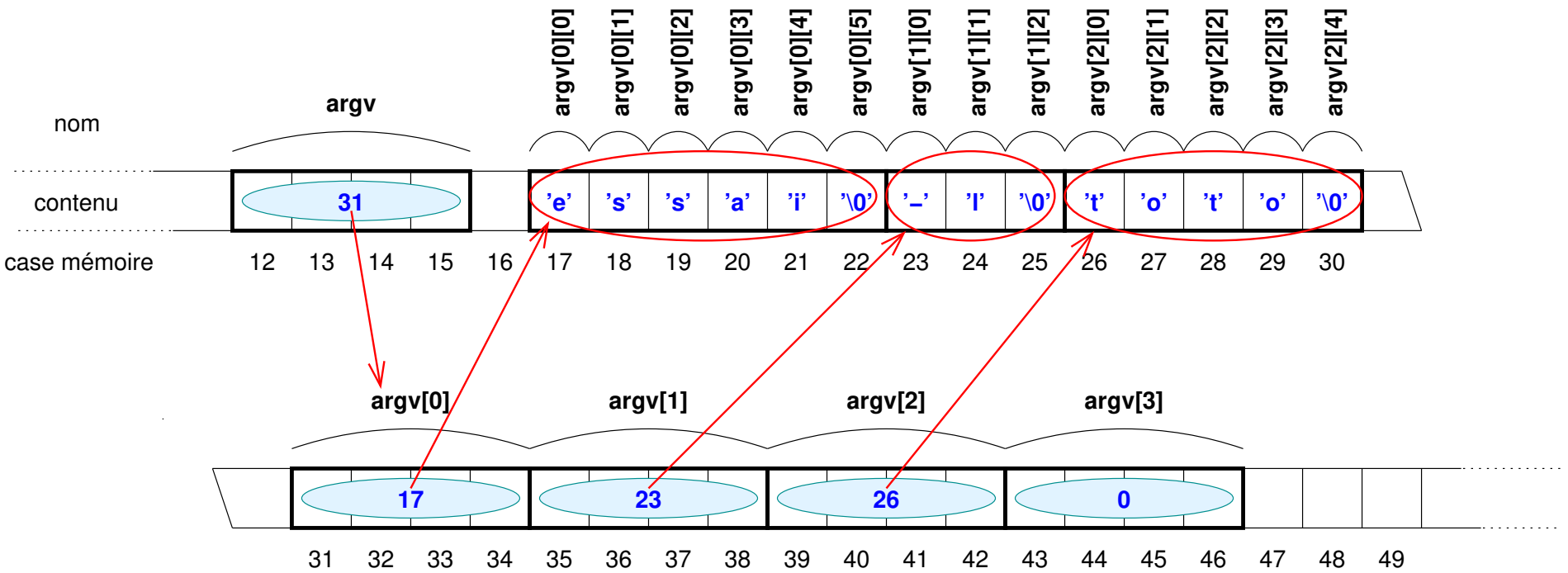
① `argc`, le nombre d'arguments («*mots*») de la ligne de commande ;

② `argv`, un tableau de chaînes de caractères qui contiennent les arguments (un argument ou «*mot*» par chaîne) ;

➥ `argv[argc]` vaut `NULL`.

```c
#include <stdio.h>

int main (int argc, char *argv[])
{
  while (argc >= 0)
    {
      printf("%s\n",argv[argc]);
      argc -= 1;
    }

   return 0;
}
```

```
$ essai -l toto
```


nom
contenu
case mémoire
argv
argv[0][0]
argv[0][1]
argv[0][2]
argv[0][3]
argv[0][4]
argv[0][5]
argv[1][0]
argv[1][1]
argv[1][2]
argv[2][0]
argv[2][1]
argv[2][2]
argv[2][3]
argv[2][4]
31
'e' 's' 's' 'a' 'i' '\0' '-' 'l' '\0' 't' 'o' 't' 'o' '\0'
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
argv[0]
argv[1]
argv[2]
argv[3]
17
23
26
0
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49