COURS 3 - CONSTANTES, TABLEAUX, CHAÎNES DE CARACTÈRES ET ADRESSES

G. Bianchi, G. Blin, A. Bugeau, S. Gueorguieva, R. Uricaru

2015-2016

Programmation 1 - uf-info.ue.prog1@diff.u-bordeaux.fr

université de BORDEAUX

LES CONSTANTES

CONSTANTES

▷ `const int a = ...;` permet de dire que la variable `a` est en lecture seule

▷ `#define identificateur valeur` permet d'utiliser `identificateur` dans le code ; ce dernier sera remplacé avant la compilation par `valeur`

```
#define PI 3.14
#define SIZE_MAX 1024
#define YES 'y'
#define PROMPT "$>"
```

COMPORTEMENT

▷ Les macros sont remplacées ”brutalement” à la précompilation. Toutes les chaînes complètes correspondant à l’identificateur de la macro sont remplacées par la valeur de la macro

▷ Il y a un risque d’erreur (minimisé si toutes et uniquement les macros sont en majuscules)

  ▷ Eviter les identificateurs à un seul caractère

```
#define PI 3.14
int main(void){
    printf(”La valeur de Pi est %d\n”,PI);
    return EXIT_SUCCESS;
}
```

BONNE CONDUITE

- Toute constante numérique non triviale doit être définie!
  - Evite les modifications multiples
- Fournir un commentaire si nécessaire
  - Pour expliquer la constante
  - Pour éviter des modifications malheureuses

```
/** Error codes for I/O functions.
* They all must be negative. **/
#define FILE_NOT_FOUND -1
#define READ_FORBIDDEN -2
#define WRITE_FORBIDDEN -3
#define DISK_FULL -4
```

LES TABLEAUX

DÉFINITION

▷ Utile pour stocker plusieurs éléments de même type

▷ N'est pas utilisable pour stocker des informations de types différents

▷ Stocker de façon contiguë en mémoire (permet un accès rapide)

▷ Déclaration: `type nom[nb_elem];`

```
int t[10];
double cosinus[360];
```

▷ Avec initialisation:

```
char vowel[6]={'a','e','i','o','u','y'};
```

PROPRIÉTÉS

▷ Peut être défini pour tout type d'éléments

▷ Est numéroté de 0 à taille − 1

▷ L'accès au ième élément du tableau `tab` se fait à l'aide de `tab[i-1]`

▷ ⚠Un tableau ne connaît pas sa taille

▷ L'opérateur `sizeof` ne fonctionne que pour les tableaux dont la taille est connue à la compilation (pas magique...)

▷ C'est la responsabilité du programmeur de connaître la taille d'un tableau

TAILLE

▷ Trois solutions

▷ Stocker la taille grâce à une constante

▷ Transmettre à l'aide de variables la taille aux éléments du programme en ayant besoin (i.e., ceux qui vont le parcourir)

▷ Utiliser une convention de fin de tableau en insérant un élément particulier dans sa dernière case

DÉBORDEMENT

▷ Il n'y a aucun contrôle de débordement (i.e., accès en dehors du tableau) ; ni à la compilation, ni à l'exécution

▷ Un accès à une zone mémoire non utilisée par le programme va générer une erreur fatale engendrant l'arrêt de ce dernier

▷ Attention aux surprises

```
int t[10];
int j=12;
printf("%d",t[10]);
```

```
$> ./a.out
12
```

TABLEAUX À N DIMENSIONS

- `int t[100][16][45];`
- Chaque `t[i][j]` est un tableau de `45 int`
- `int t[2][3];`

t[0][0]	t[0][1]	t[0][2]	t[1][0]	t[1][1]	t[1][2]
@+0	@+4	@+8	@+12	@+16	@+20

- Attention à l'ordre de parcours
  - Pareil en théorie mais pas en pratique
  - Sur un tableau de 15000x15000 cases, on peut déjà constater un rapport de 8 sur le temps de parcours (4 vs 32 secondes)

LES CHAÎNES DE CARACTÈRES

CHAR[]

- Pas un type, mais une convention de stockage
  - Tableau de caractères dont le dernier est `'\0'`
- Délimitée par des guillemets
  - `char msg[]="Welcome";`
  - `char msg[]={'W','e','l','c','o','m','e','\0'};`

CARACTÉRISTIQUES

▷ Chaîne vide "" correspond à un tableau dont la première case contient '\0'

▷ La Longueur d'une chaîne correspond au nombre de caractères avant '\0'

▷ Accès au n^{eme} caractère

```
char s[]="wxyz";
char c=s[2]; //c='y'
```

LES ADRESSES

STOCKER DES ADRESSES

▷ Il est intéressant de pouvoir stocker des adresses de variables

  ▷ Par exemple pour la transmettre à une partie du code qui en a besoin

▷ Il faut connaître le type de la variable ...

▷ Un pointeur est un type particulier ”dénoté par *” dont la valeur est une adresse* et dont le type ”cible” est spécifié

  ▷ e.g., `int *`, `char *`

*Rappel: une adresse = un entier

MANIPULER DES ADRESSES

▷ L'opérateur `&` permet de récupérer l'adresse d'une variable sous la forme d'un pointeur

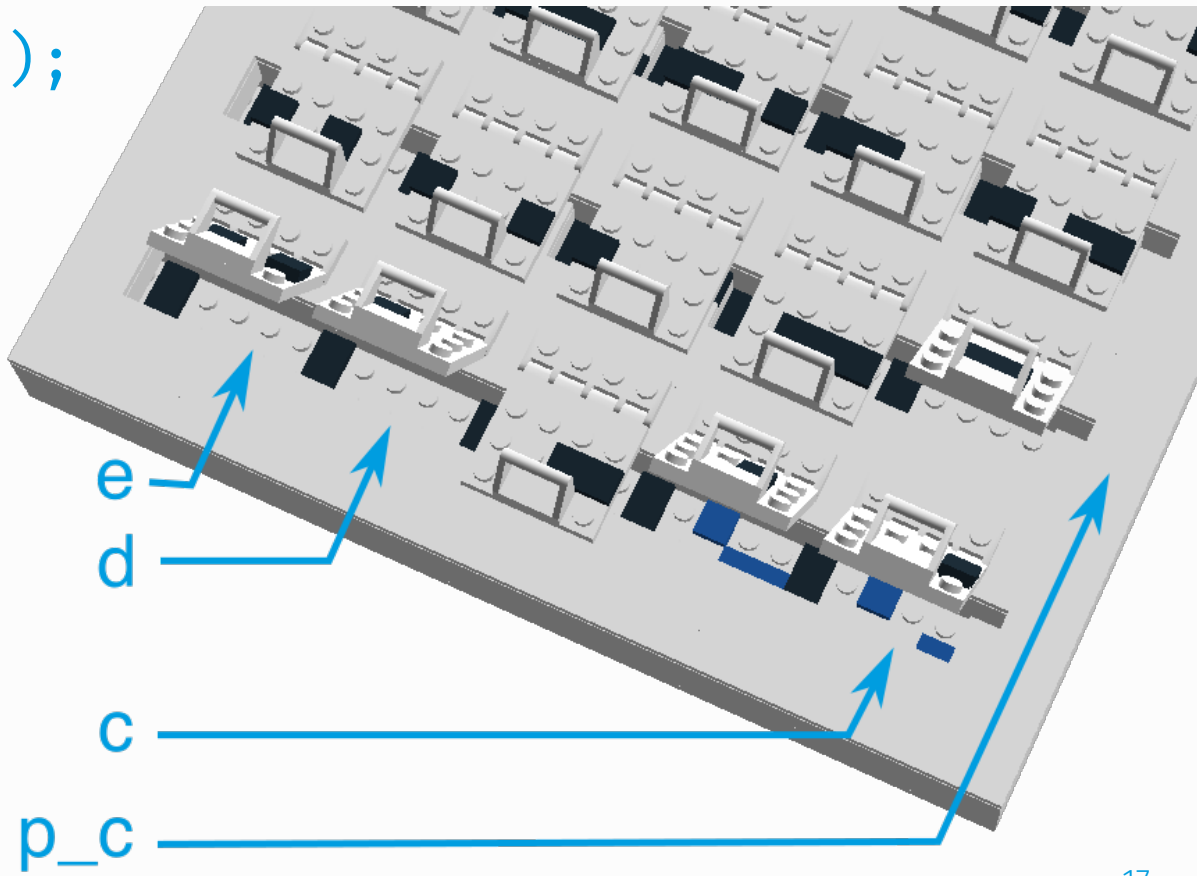
▷ L'opérateur `*` permet de récupérer le contenu de la mémoire situé à une adresse donnée par un pointeur

```
int b = 12;
int * p_b = &(b);
int c = *(p_b); // c=12
```

UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-mêmes, il suffit de connaître leur adresse

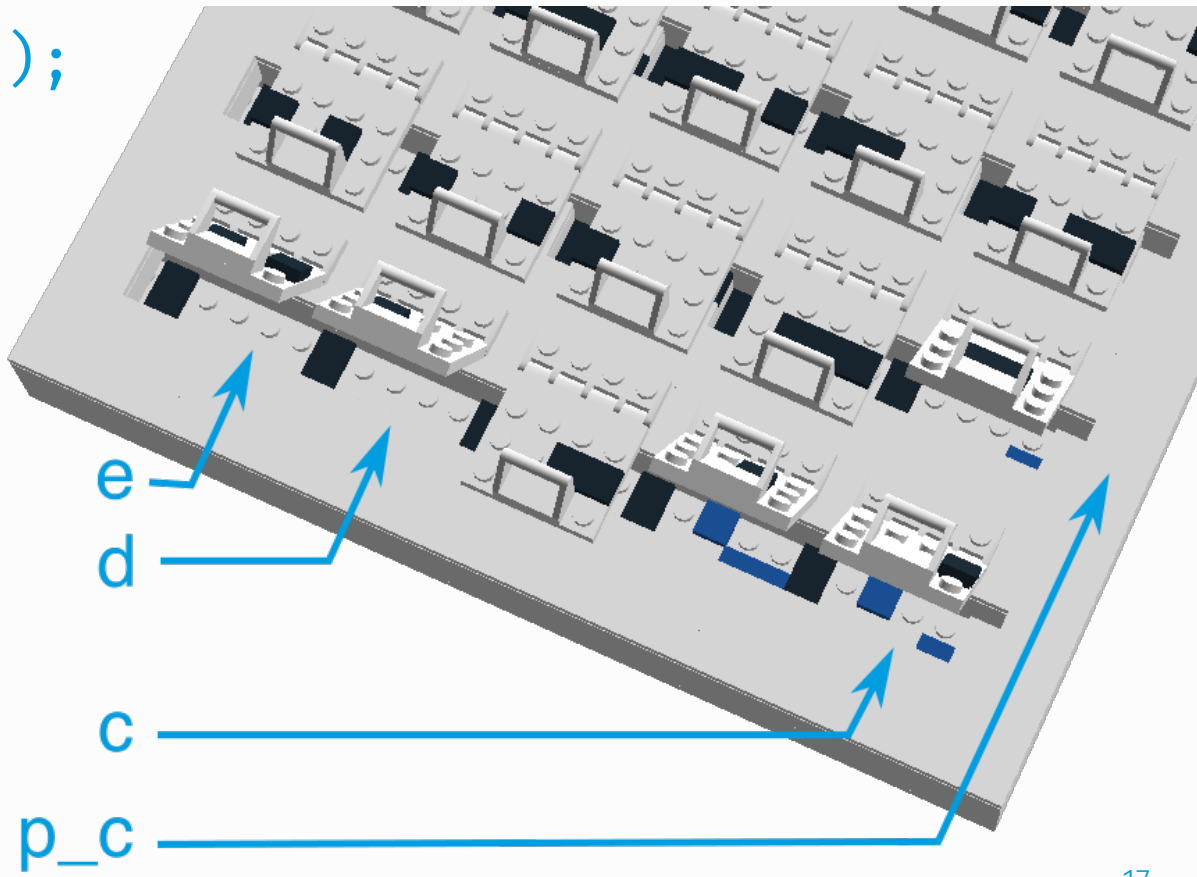
```
char c[]={'A','C'}; //'A' = 65
char d,e;
char * p_c = &(c);
d=*(p_c+1);
e=*(p_c)+1;
```



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-mêmes, il suffit de connaître leur adresse

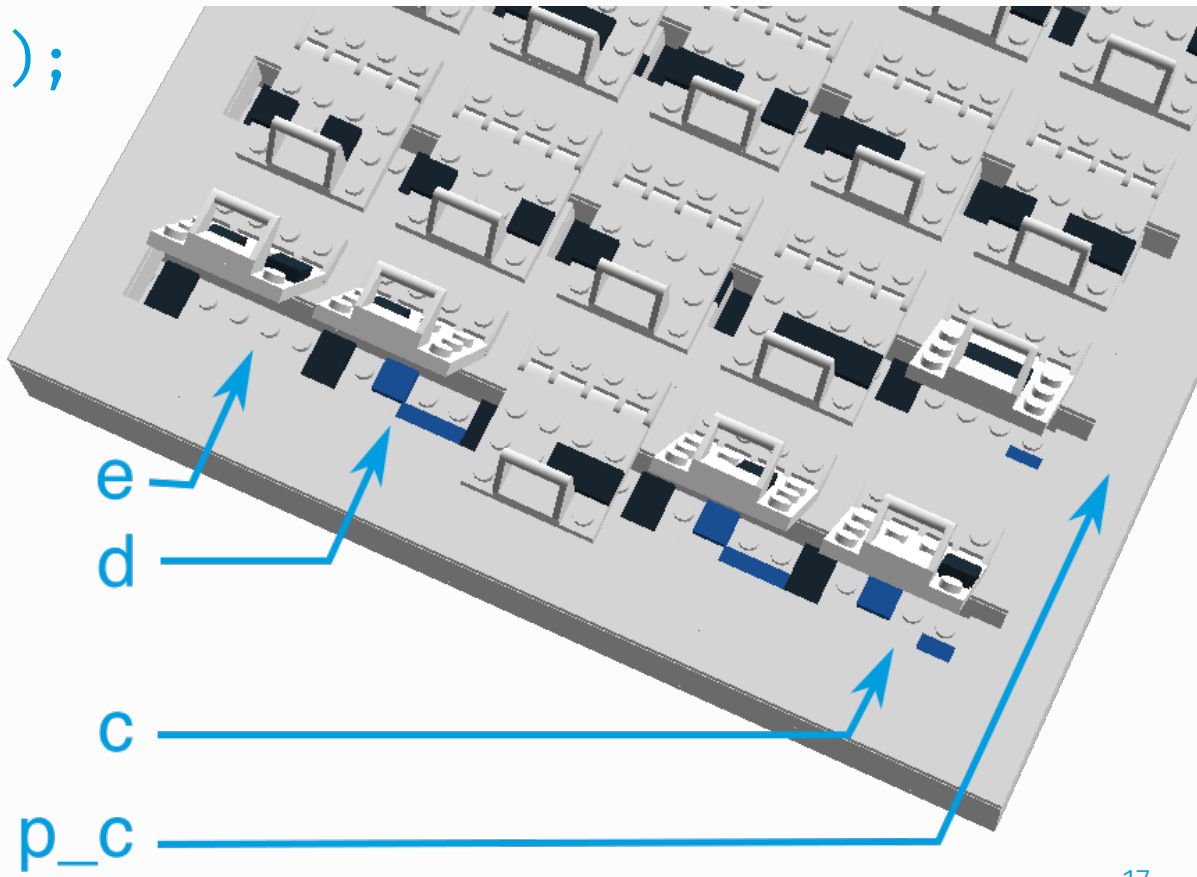
```
char c[]={'A','C'}; //'A' = 65
char d,e;
char * p_c = &(c);
d=*(p_c+1);
e=*(p_c)+1;
```



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-mêmes, il suffit de connaître leur adresse

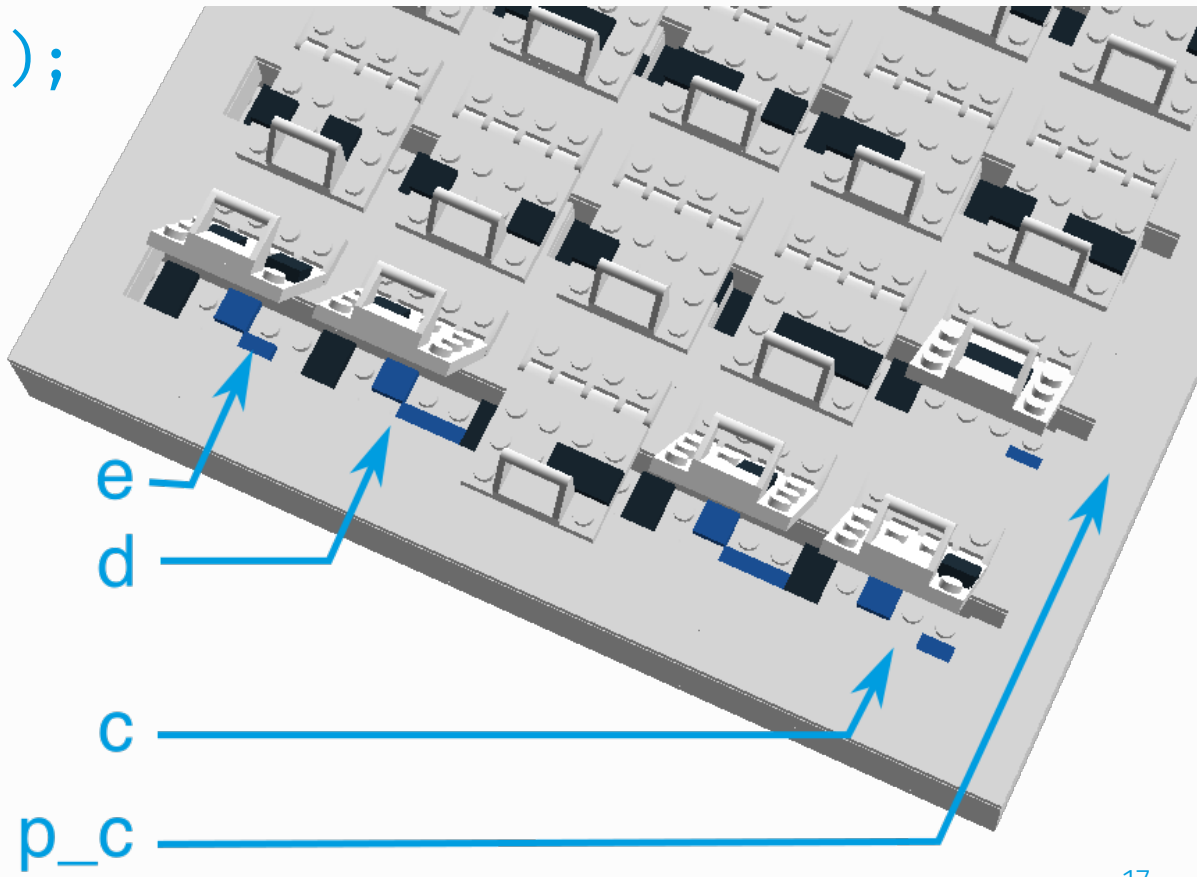
```
char c[]={'A','C'}; //'A' = 65
char d,e;
char * p_c = &(c);
d=*(p_c+1);
e=*(p_c)+1;
```



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-mêmes, il suffit de connaître leur adresse

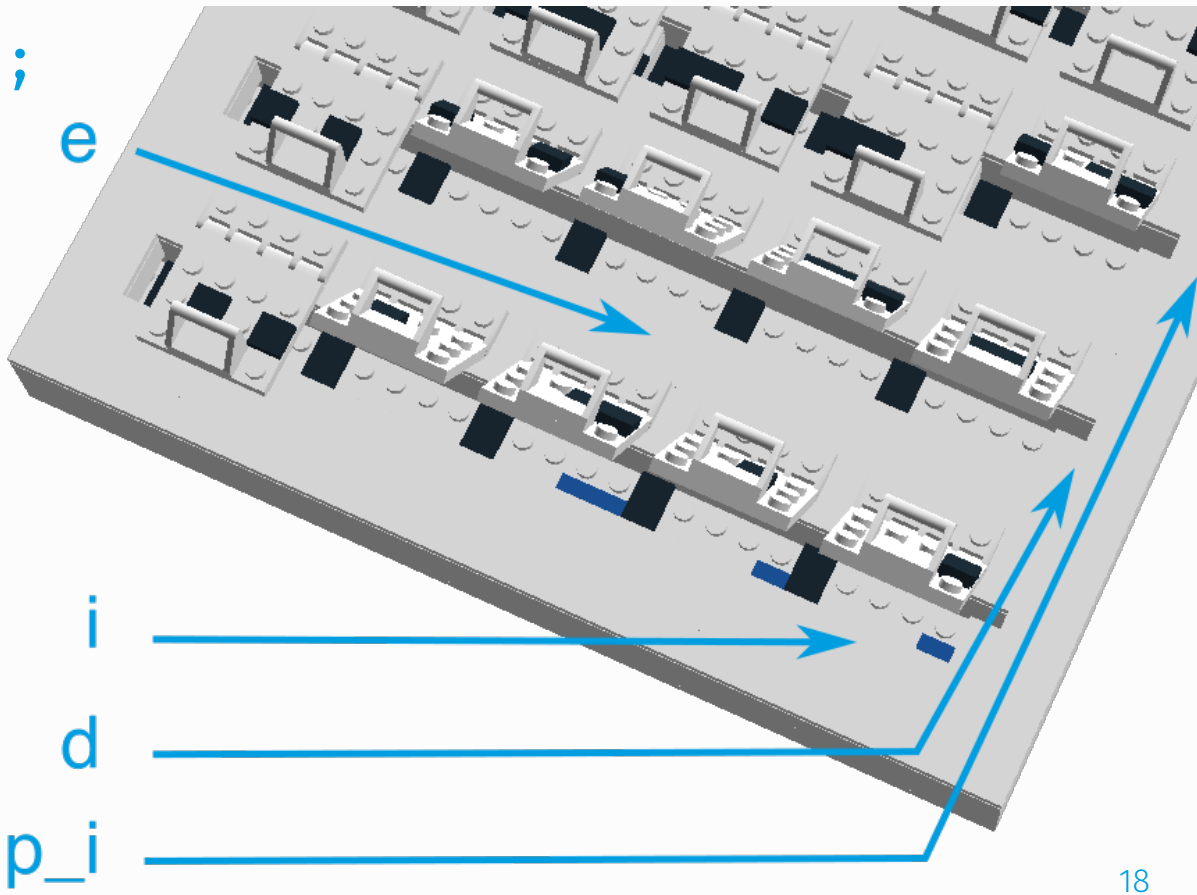
```
char c[]={'A','C'}; //'A' = 65
char d,e;
char * p_c = &(c);
d=*(p_c+1);
e=*(p_c)+1;
```



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-même, il suffit de connaître leur adresse

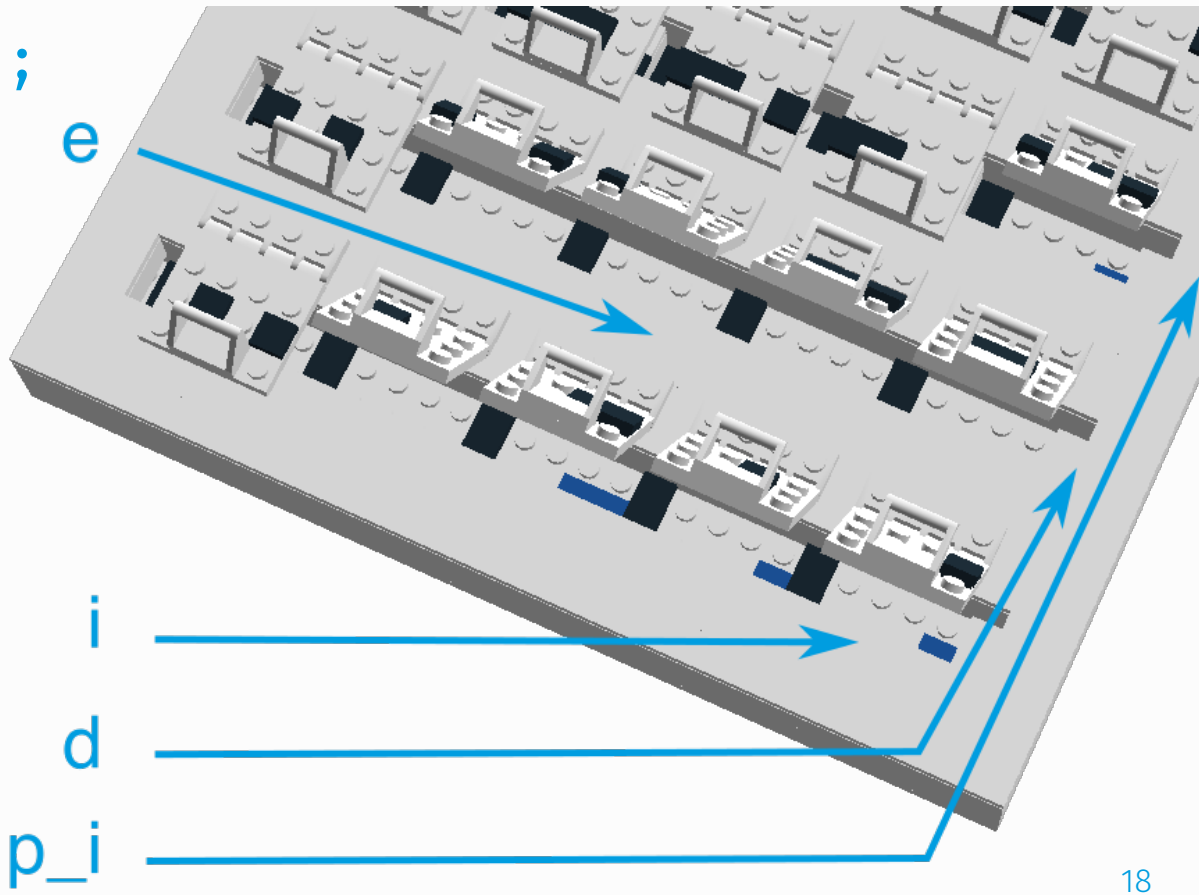
```
int i[]={257,3}; //int sur 2 octets
int d,e;
int * p_i = &(i);
d=*(p_i+1);
e=*(p_i)+1;
```



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-même, il suffit de connaître leur adresse

```
int i[]={257,3}; //int sur 2 octets
int d,e;
int * p_i = &(i);
d=*(p_i+1);
e=*(p_i)+1;
```

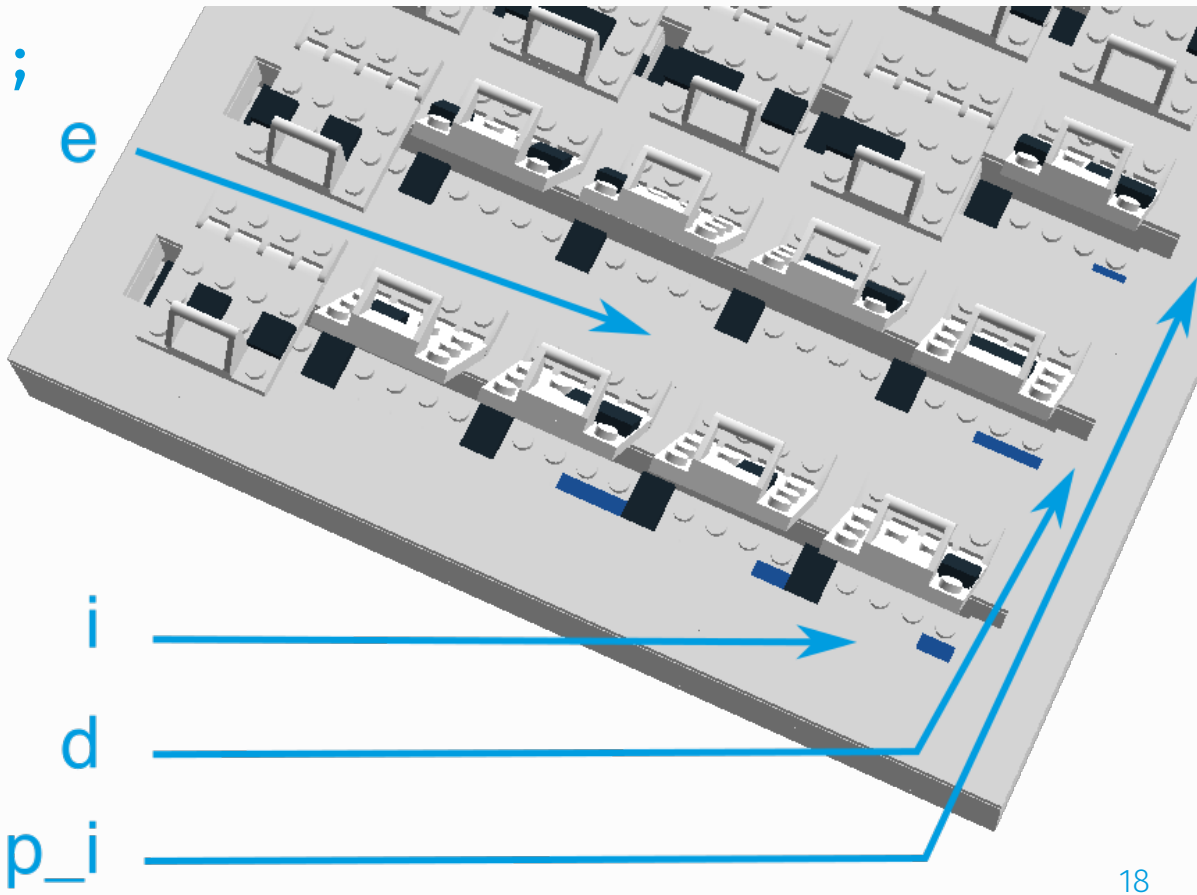


UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-même, il suffit de connaître leur adresse

```
int i[]={257,3}; //int sur 2 octets
int d,e;
int * p_i = &(i);
d=*(p_i+1);
e=*(p_i)+1;
```

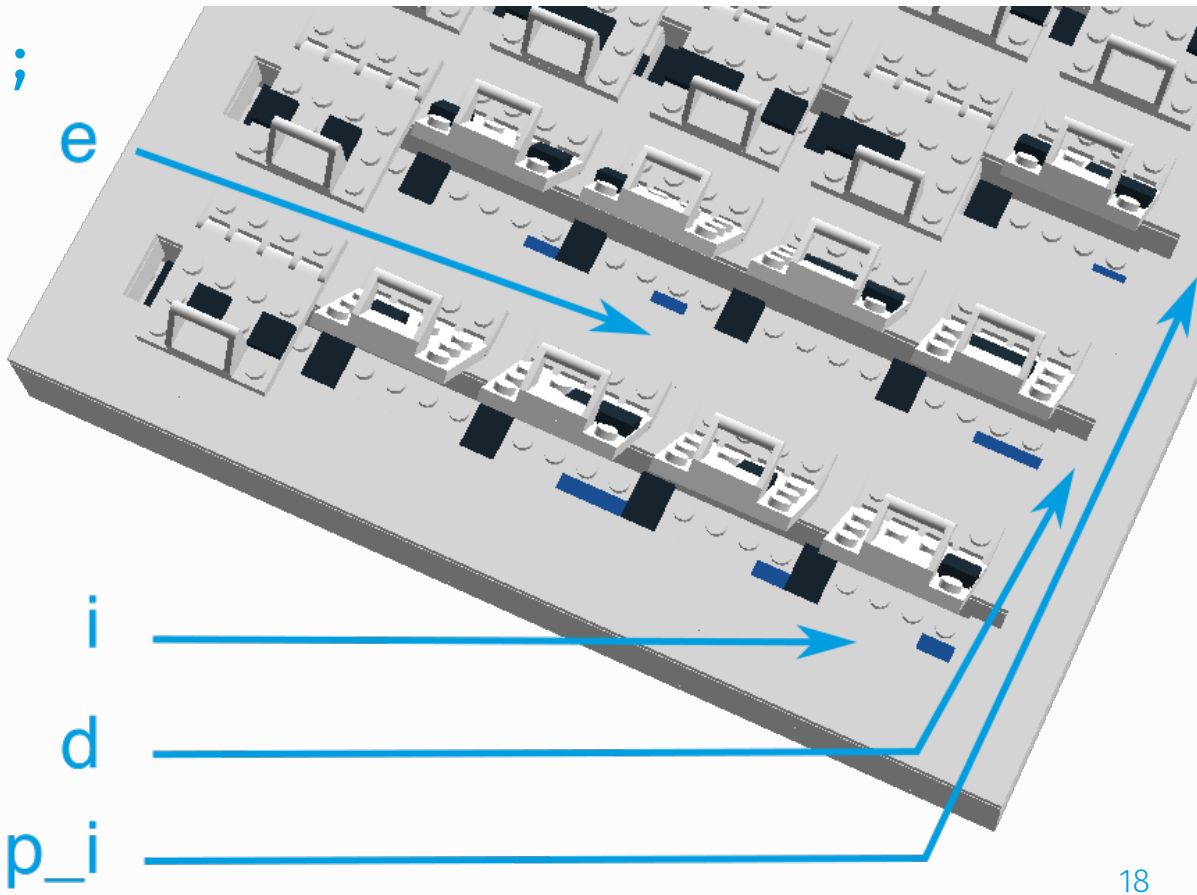
⚠



UTILITÉ DES ADRESSES

▷ Pas besoin de faire des copies des données elles-même, il suffit de connaître leur adresse

```
int i[]={257,3}; //int sur 2 octets
int d,e;
int * p_i = &(i);
d=*(p_i+1);
e=*(p_i)+1;
```



ARITHMÉTIQUE DES POINTEURS

▷ Une opération arithmétique au sein de l'opérateur & est liée à la taille du type cible

```
type x,k;
int j=2;
type * p_x = &(x);
k=*(p_x+j);
// k = contenu a l'adresse &(x)+j*sizeof(type)
```

▷ Intérêt ?

RETOUR SUR LES TABLEAUX

- Un tableau ne connaît pas sa taille
- Tableau = adresse de son premier élément et un type (comme les pointeurs)
- `int tab[2]` et `int* p_t` sont équivalents
  - L'adresse d'un élément de type `int`
  - Différences ?

RETOUR SUR LES TABLEAUX

- Un tableau ne connaît pas sa taille
- Tableau = adresse de son premier élément et un type (comme les pointeurs)
- `int tab[2]` et `int* p_t` sont équivalents
  - L'adresse d'un élément de type `int`
  - Différences ?

L'espace alloué n'est pas le même: `tab` correspond à une adresse où 2 entiers peuvent être stockés; `p_t` correspond à une adresse où une adresse peut être stockée.

RETOUR SUR LES TABLEAUX

▷ Un tableau ne connaît pas sa taille

▷ Tableau = adresse de son premier élément et un type (comme les pointeurs)

▷ `int tab[2]` et `int* p_t` sont équivalents

  ▷ L'adresse d'un élément de type `int`

  ▷ Différences ?

L'espace alloué n'est pas le même: `tab` correspond à une adresse où 2 entiers peuvent être stockés; `p_t` correspond à une adresse où une adresse peut être stockée.

  ▷ `tab[1]` et `*(&(tab)+1)` sont égaux – à savoir la valeur du 2ème élément du tableau `tab`

▷ Quid de `t[j]` ?

RETOUR SUR LES CHAÎNES DE CARACTÈRES

▷ Donc `char tab[]` et `char * tab` c'est pareil ... ⚠

```
int main (void)
{
    char *tab="Cabri";
    tab[0]='L';
    return 0;
}
```

```
int main (void)
{
    char tab[]="Cabri";
    tab[0]='L';
    return 0;
}
```

▷ Dans le 1er cas, `"Cabri"` est une chaîne constante stockée en mémoire (dans une zone protégée en écriture car commune pour tout le programme) et `tab` contient juste son adresse

▷ Dans le 2nd cas, c'est le contenu de la chaîne qui a été recopié caractère par caractère dans `tab`, elle n'est pas stockée ailleurs

ADRESSES INVALIDES

▷ Par défaut, la valeur d'un pointeur non initialisé est égale à une constante symbolique notée `NULL` qui est définie dans `stdio.h`

▷ Cette constante représente, par convention, une adresse invalide

▷ L'accès à l'adresse `NULL` provoque l'interruption du programme avec un `Segmentation fault`

DOGGY BAG

TO TAKE AWAY ...

▷ Tableau = adresse de son premier élément + le type des éléments contenus

▷ Tableau et pointeur sont étroitement liés

▷ Un tableau ne connaît pas sa taille (⚠ débordement)

▷ Les opérateurs `&` et `*` permettent de manipuler des adresses

▷ Les opérations arithmétiques sur les pointeurs tiennent compte du type cible

Questions?