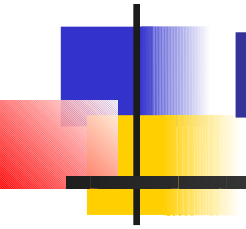
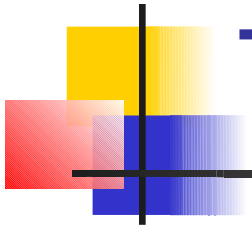


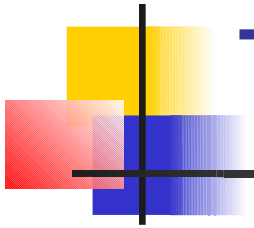
Rappel de C

Noël De Palma
Professeur Université Joseph Fourier
noel.depalma@inrialpes.fr



Types de bases

- Types d’entiers
  - short/int/long/float/double/long double
    - Signés par défaut
  - Unsigned
    - Pas de bits de signe
- Type caractère
  - char



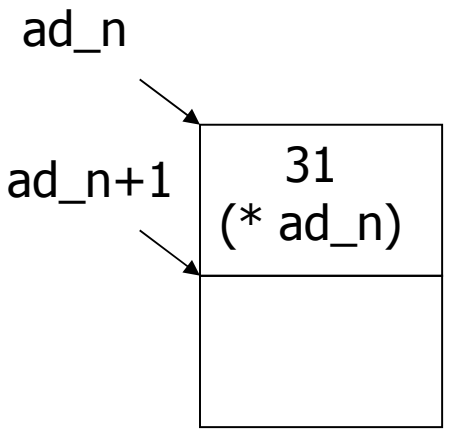
Tableau

- Déclaration (1ère forme)
  - *Type nomtableau [taille]*
  - Ex :
    - int t0[20]
      - 1 seule dimension
      - indice de 0 à 19. (80 octets pré alloués en mémoire)
      - t0[4] : accès au 5eme élément
    - char t1[10][20]
      - 2 dimensions
      - Indice 0 à 9 et 0 à 19 (800 octets pré alloués en mémoire)
      - T1[2][3]
    - Init :
      - int t0 [3] = {2,1,4}

Pointeurs

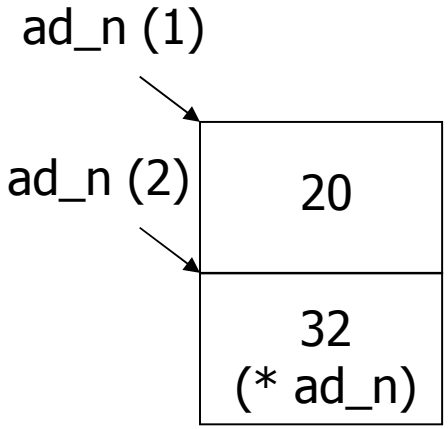
- Un pointeur = une adresse + un type
  - Déclaration
    - Type * nom;
      - int * p0; // ptr sur un int
    - Accès à la valeur pointé
      - *ptr
    - Exemple

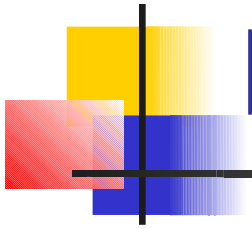
```
int n = 20;
int * ad_n;
ad_n = & n; // récupère l'@ de n
*ad_n = 30; // n=30
(*ad_n)++; // n=31
*ad_n++; // !!! Retourne valeur à l'adresse de &n + 2
```



Pointeurs

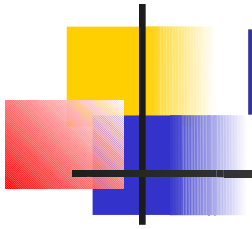
```
int n = 20
int * ad_n = &n; // (1)
*(ad_n +1) = 32; // (2)

// ad_n est un pointeur sur un int.
// ad_n+1 = @ de l'entier suivant !!!
// si ad_n  = ox00000000 alors
// ad_n+1 = ox00000004 (un int est codé sur 4
   octets !!!)
```





Retour sur les tableaux

- Un nom de tableau est un pointeur
  - int t[20]
  - La notation ***t*** est équivalente à ***&t[0]***
  - t + 1 // &t[1]
  - t + i // &t[i]
  - t[i] // *(t+i)
  - *(t+2) = 3; // t[2] = 3



Retour sur les tableaux

Int t0[20];

Peut s'écrire :

```
int *t0;
t0 = malloc(20 * sizeof(int));
```

Accès :

```
*t0 = 3; // t0[0] = 3
*(t0+1) = 4; // t0[1] = 4
t0[2] = 5; // *(t0+2) = 5
```

Paramètres des fonctions

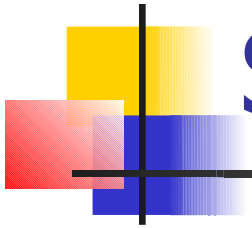
- Passage par valeur

```
void swap(int a, int b){
    int sv;
    sv = a; a = b; b = sv;
     printf(”swap a : %d, b : %d”, a,b);
}

main() {
    int a=5; int b = 10;

    printf(”a : %d, b : %d”, a,b); // a : 5, b : 10
     swap (a,b);                   // a : 10, b :5
    printf(”a : %d, b : %d”, a,b); // a : 5, b : 10
}
```

Paramètres des fonctions

- Passage par adresse

```
void swap(int *a, int *b){
    int sv;
    sv = *a; *a = *b; *b = sv;
     printf(”swap a : %d, b : %d”, *a,*b);
}

main() {
    int a=5; int b = 10;

    printf(”a : %d, b : %d”, a,b); // a : 5, b : 10
     swap (&a,&b);                          // a : 10, b :5
    printf(”a : %d, b : %d”, a,b); // a : 10, b : 5
}
```



Structure

- Un point

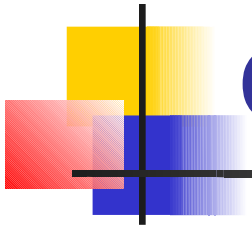
```
struct point {
    int x;
    int y; };
struct point p0;
p0.x = 2; p0.y = 3;
```

- Une liste de points

```
struct liste_point {
     struct point p0;
     struct liste_point *suivant } ;
struct liste_point *p ; (p->p0).x = 2; p= p->suivant;
```

Ou

```
struct liste_point {
    int x; int y;
    struct liste_point *suivant } ;
struct liste_point *p; p->x = 2; p = p->suivant;
```

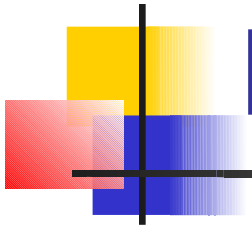


Conversions de pointeur

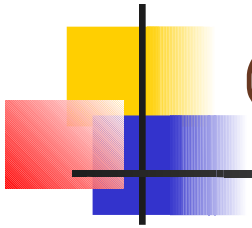
- struct liste_point *p ;

- p+4 ?
- (int)p + 4
- Plus généralement
  - (type) expression

Fichier d'en-tête



```
#ifndef __MEM_ALLOC_H
#define __MEM_ALLOC_H
#include <stddef.h>
void mem_init();
void *mem_alloc(size_t size);
void mem_free(void *zone, size_t size);
void mem_show(void (*print)(void *zone, size_t size));
#endif
```



Compilation séparée

- #Makefile de l'allocateur mémoire

- #compilateur
- CC = gcc

- #options générales de compilation
- CFLAGS = -Wall
- QUESTION : comment compiler memshell.c ????
- #pour compiler le shell de test de l'allocateur
- memshell: alloc.o memshell.o
- $(CC) $(CFLAGS) -o memshell alloc.o memshell.o

- #pour créer le fichier objet de l'allocateur
- alloc.o: alloc.c alloc.h
- $(CC) $(CFLAGS) -c alloc.c