Type structuré : enregistrement
[Type énuméré]
Notions de pointeurs

© J. Morinet Lambert , M. Cadot , L. Pierron
UHP

version : 13/12/04

Objectifs

- associer au sein d'une même catégorie des informations de types variés
- faciliter la gestion de données complexes
  - Cf. langage objet (classe, objet, propriétés)
- exemples
  - date : jour, mois, année
  - étudiant : nom , prénom, note1,note2, reçu, mention
    - nom , prénom : des chaînes de caractères
    - note1,note2 : des réels
    - reçu, mention : des booléens (entiers en C)
  - complexe : partie réelle, partie imaginaire

Traduction C

- **On déclare directement des variables dat, etud…**

```
struct { int jour;
         int mois;
         int an ; } dat;
struct {char nom[50];
        char prenom[50];
        float note1, note2;
        int recu, mention;} etud;
```

champs de structure : *type nom ;*

nom du type

- On déclare une catégorie de variable : **un type**

```
struct date { .... }; /* déclaration du type date */
```

- On utilise ce type pour déclarer les variables d1 et d2

```
struct date d1,d2; /* déclaration de variables */
```

Utilisation de structure

- Règle : on suffixe le nom de la variable par le nom du champs
- Exemple :

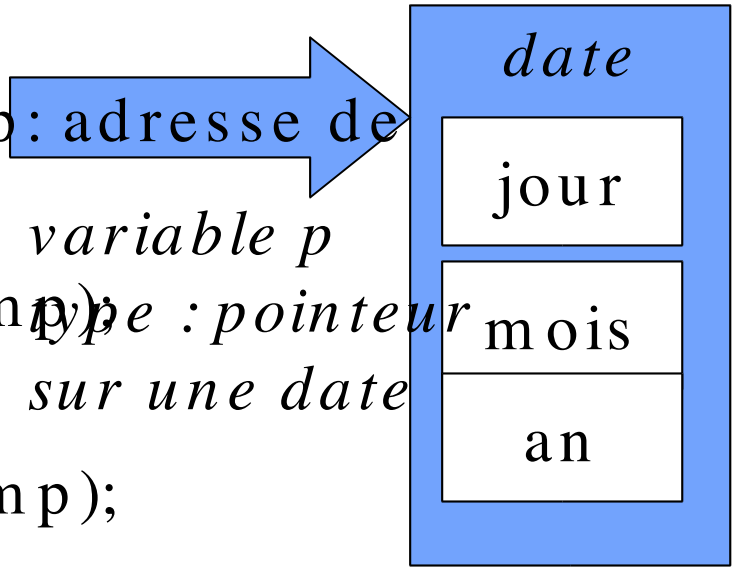
```
#include <stdio.h>
struct date{int jour, mois, an;} ;
int main (void)
{struct date today, d1, d2;
    today.jour=12;
    today.mois=11;
    today.an=2001;
     printf("on est le %i/%i/%i\n",today.jour,today.mois,today.an);
}
```

- pour affecter une valeur à une structure il faut affecter une valeur à chaque champs de la structure
- il faut développer des modules de saisie et d'affichage
- Affectation globale **possible** entre variables de même type : d2=d1;

Saisie de structure

```
void saisirDate(struct date *p)  p: adresse de
{int temp;
printf("le jour ");scanf("%i", &temp);
p->jour=temp;
printf("le mois ");scanf("%i",&temp);
p->mois=temp;
printf("l\'annee ");scanf("%i", &temp);
p->an=temp;
}
```

variable p type : pointeur sur une date



*variable pointée de type date notée : (*p) ou p->*

- **!!!! Notation** : p->jour est équivalent à (*p).jour

Afficher une structure

```
void afficherDate( struct date d)
   { /* afficher chaque champs */
      printf("le jour %i", d.jour);
      printf("le mois %i", d.mois);
      printf("l\'annee %i\n", d.an);
   }
```

- utilisation des modules d'entrée et sortie

```
struct date today;                  /* declaration  de today */
saisirDate(&today);       /* remplace scanf impossible si struct */
```

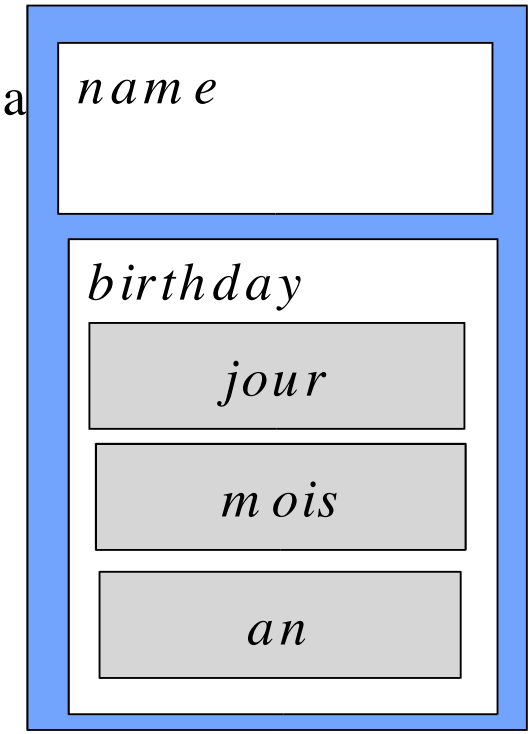
!!!!! *On passe bien l'adresse de la variable à modifier ! Ne pas confondre avec tableau dont le nom représente l'adresse du 1er élément*

```
afficherDate(today);      /* remplace printf impossible si struct */
```

Structures imbriquées

```
struct personne {char name[50];
                 struct date birthday;};
struct personne P1,P2; /*déclarations de variables */

    strcpy(P1.name, "Lagaffe Gaston"); /*utilisa
    P1.birthday.jour=28;
    P1.birthday.mois=2;
    P1.birthday.an=1957;
```



Initialisation de structure

- A la déclaration
- Exemple de déclaration de catégorie de variable"rectangle"

```
typedef struct {int longueur;
                int largeur;}rectangle;
```

- Déclaration de variable

```
rectangle R1;
```

- Déclaration et initialisation de structure

```
rectangle R2={16, 12};
```

- Attention
  - Respecter l'ordre des champs

Remarques

- il est possible de déclarer un type structuré de deux façons :

  struct **tnom1** {<*type_champs*> <*champs*>; ...};

  typedef **struct** {<*type_champs*> <*champs*>; ...} **tnom2** ;

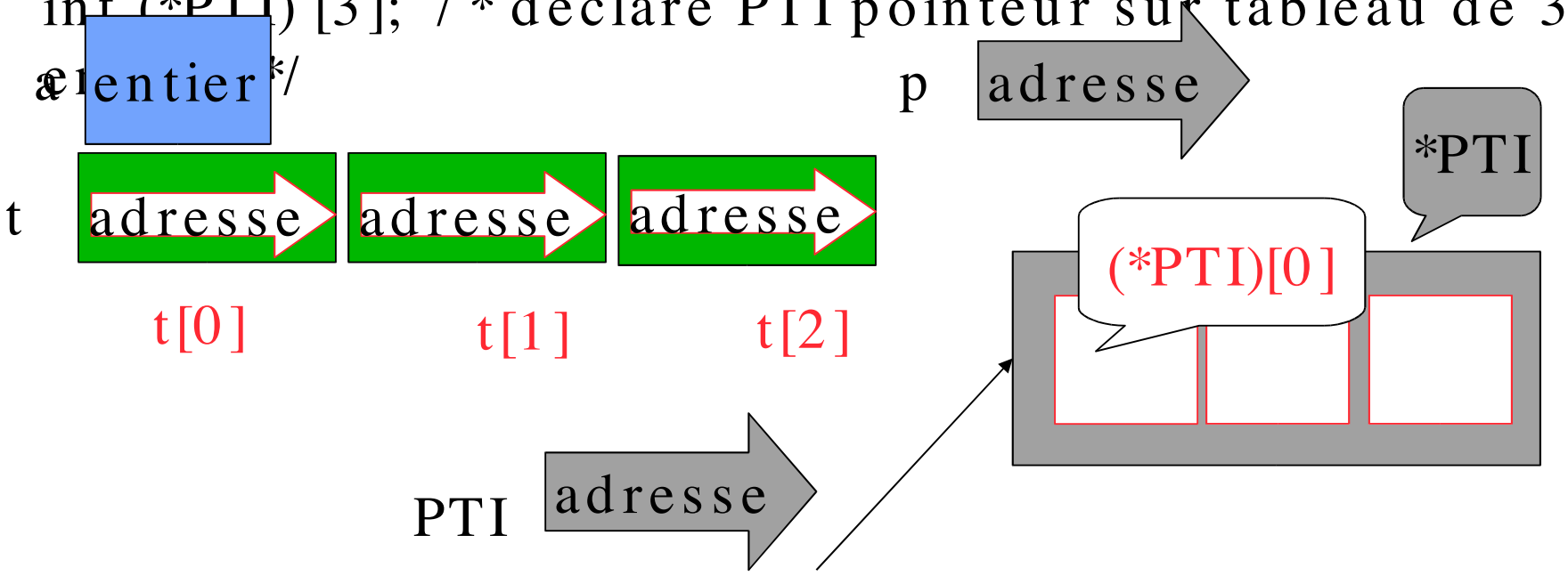
- dans le premier cas on doit toujours signaler qu'on travaille avec struct ce qui donne

  struct tnom1 s1,s2; /* déclaration de variables s1 et s2 */

- dans le 2ième cas il est inutile de spécifier struct ce qui donne

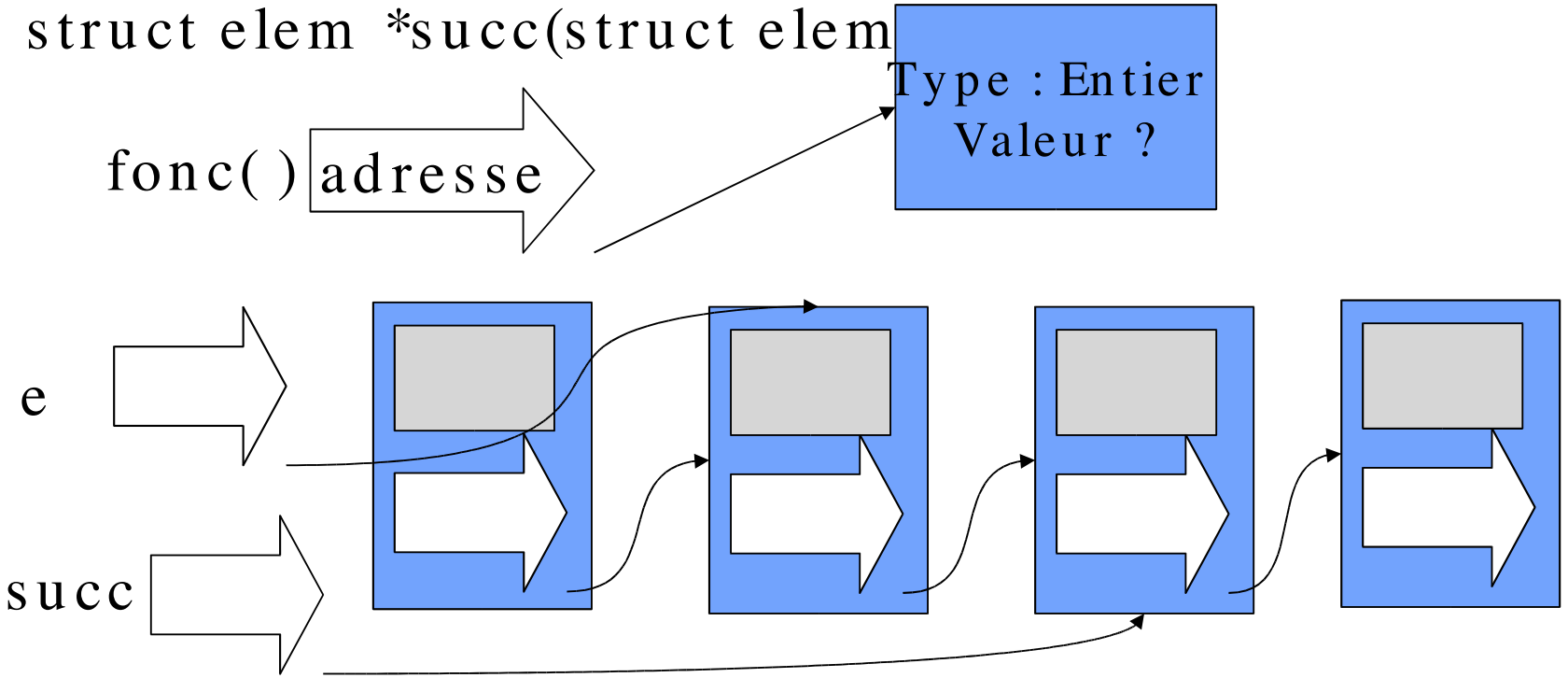
Notions de pointeurs

- int a; /* déclare a de type entier */
- int *p; /* déclare p : pointeur sur entier */
- int *t[3]; /* déclare t : tableau de 3 pointeurs sur entier */
- int (*PTI) [3]; /* déclare PTI pointeur sur tableau de 3 entier */



Rappels pointeurs suite

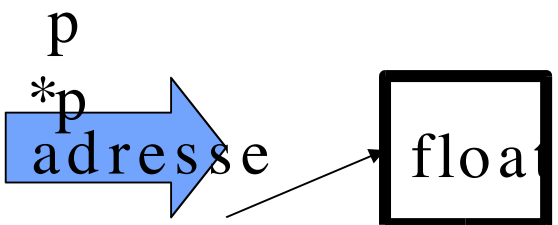
- int fonction1(...) /* fonction qui rend un entier */
- int *fonc(...) /* fonction qui a pour résultat un pointeur sur entier à savoir l'adresse d'un entier */
- struct elem *succ(struct elem



Variable dynamique (pointée par)

- A la déclaration : le pointeur

  float *P ;



- n'existe qu'une variable pointeur
- Pour créer la variable pointée par P

  malloc : permet d'allouer un espace mémoire pour un objet de taille connue

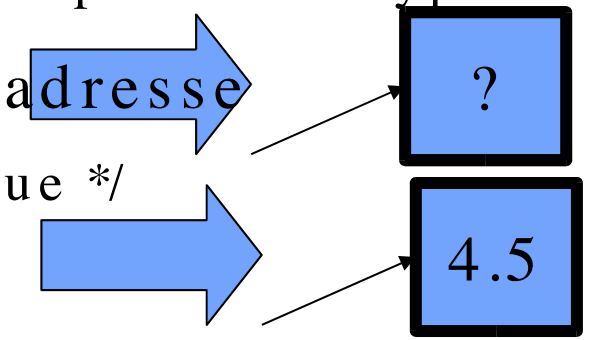
  l'adresse est placée dans un pointeur résultat de la fonction

  profil void *malloc(size_type size)

  librairie <stdlib.h>

  exemple : réservation pour une variable pointée de type float

```
t=sizeof(float);
p= malloc(t); /*(*p) variable dynamique */
if (p != NULL)
   {*p=4.5;printf("*p %f\n", (*p);}
```



Variables dynamiques

- créées dynamiquement lors de l'exécution du programme
- selon les besoins
- utilisation de la fonction malloc pour réserver la place requise
- Dés-allocation : free
- exemple : free(p); /* p de type pointeur sur réel : float *p */
- en réalité : la place mémoire est théoriquement disponible
  - cela dépend du système d'exploitation
  - de gros problèmes avec C++
- il n'existe pas en C de fonctions de gestion de mémoire ;

Utilisation de variables dynamiques

- application
  - comment représenter des listes d'information dont la taille est TRES variable
  - le nombre d'éléments peut varier de quelques à quelques dizaines

- solution 1 : réserver des tableaux surdimensionnés
  - représentation de listes en mode contigu
- solution 2 : créer des "cases" pour représenter les éléments de la liste au fur et à mesure des besoins
  - utilisation de variables dynamiques
  - représentation de liste en mode chaîné

  - cf. cours représentation de liste en mode chaîné

Java et gestion de la mémoire

- évolution des langages
- machine virtuelle java
  - ne plus dépendre des compilateurs sur divers types de station
- ne plus gérer la mémoire
  - source de difficulté
  - de nombreuses erreurs
  - de nombreux plantages
  - gestion "floue" répartie entre programmeur et système d'exploitation
  - ramasse miettes peu efficace
  - règle : même si le système gère mal la mémoire, le programmeur développe correctement : ne pas oublier de dés-allouer l'espace requis pour des variables qui n'ont plus de raison d'être

/* maîtriser les chaînes de caractères */

```
#include <stdio.h>
#include <string.h>
typedef char tch[128];
int main(void)
{tch ch1,ch2="bonjour"; /* déclaration de chaîne avec init */
char *p; /* pointeur sur char */
//ch1="toto" ; /* interdit car ch1 représente une adresse */
strcpy(ch1,"toto"); /*procédure passage de l'adresse de la chaine */
printf("chaîne ch1 %s\n",ch1);
p="titi"; printf("chaîne p %s\n",p);  /* affectation directe */
printf("valeur ch1 %lu\n", ch1);      /* adresse :1476 */
printf("valeur ch1[0] %c\n",ch1[0]);  /* cfhar : 't' */
printf("adresse ch1[0] %lu\n", &ch1[0]); /* adresse : 1476 */
printf("adresse ch2[0] %lu\n", &ch2[0]); /* adresse : 1604 */
p=ch2; /* affectation d'adresse  */
printf("valeur p au format lu %lu\n", p); /* adresse : 1604 */
printf("valeur p au format s %s\n", p);  /* chaîne : "bonjour"
}
```

Pointer sur une structure

- Une variable structurée

  rectangle R1; /* déclaration de la variable */
- Un pointeur sur cette catégorie de variable

  rectangle *pr; /* déclaration du pointeur */
- On affecte l'adresse de R1 à pr

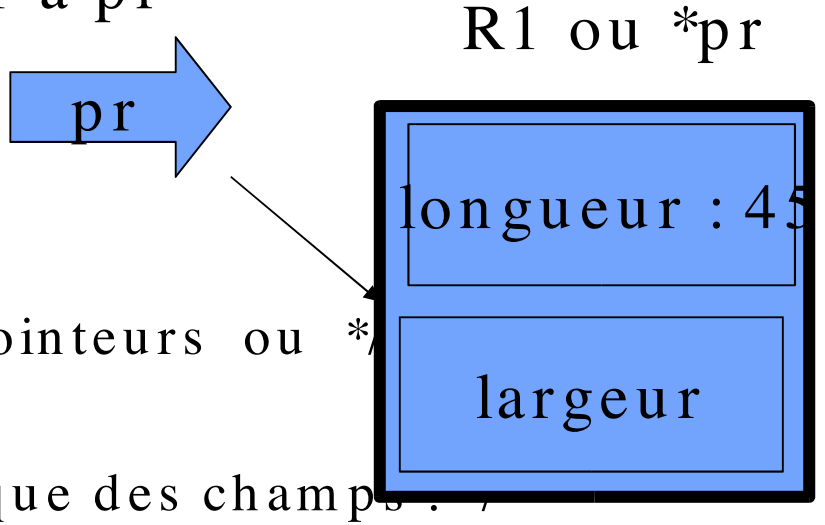
  pr = &R1;
- Deux façon de travailler

  pr->longueur = 45;

  /* -> uniquement avec des pointeurs ou */

  (*pr).longueur = 45 ;

  /* on utilise la syntaxe classique des champs */



Type énuméré

- représente une liste de valeurs différentes
- liste de valeurs symboliques
- déclarations

  typedef **enum** {<*liste_enumeree*>}< *nom_type*> ;
- exemple :

  typedef **enum** {bleu, blanc,rouge}couleur; /* type */

  couleur c1, c2 ; /*définition de variables */

  enum {feminin=2, masculin=1}sexe;
- fonctionnement

  sans indication particulière entiers codés : 0,1,2…

  avec la valeur d'initialisation

Types qui n'existent pas en C

- type booléen
  - Valeur 0 pour faux
  - printf("est ce que 3 est équivalent a 2 ? %i\n", 3==2)
  - Résultat : on affiche 0 (faux)
- type intervalle
  - Pour limiter les valeurs possibles
  - Exemple mois 1..12
  - Et contrôler automatiquement lors de la saisie