

COURS 7

Programmation impérative

Types construits

- Structures
- Énumérations
- Unions

SOMMAIRE

- Informations pratiques
- Introduction
- Éléments de base
 - Programmer en Langage C – Compilation
 - Structure d'un programme / Règles d'écritures
 - Types de base
 - Constantes/Variables
 - Opérateurs
 - Instructions de contrôle
 - Pointeurs
 - Tableaux
- Fonctions
- Chaînes de caractères
- Pointeurs- Tableaux-Fonctions
- **Types Construits**
- Entrées – Sorties sur Fichiers
- Compilation séparée
- Implémentation de Types Abstraits de Données

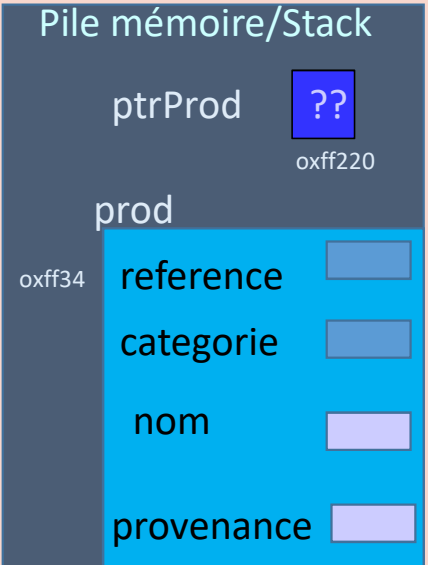
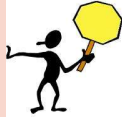
Types Construits / Structures

- Structures
 - Est composée de plusieurs champs et sert à représenter un objet réel ou un concept
 - Une structure rassemble plusieurs entités qui peuvent avoir des types différents
 - entités ont un lien entre elles
 - elles sont groupées sous un seul nom
 - Structure permet de traiter un groupe de variables liées entre elles comme un tout et non comme des entités séparées.
 - Notion différente de la notion de tableau qui permet de regrouper des objets de même type


Types Construits / Structures

Syntaxe	Syntaxe
<pre>struct nomStructure{ type1 champ1 ; type2 champ2 ; } ;</pre>	<p>nomStructure ⇔ étiquette de la structure</p> <p>struct indique une déclaration de structure constituée d'une liste de déclaration entre accolades</p> <p>La déclaration d'une structure ne réserve pas d'espace mémoire</p> <ul style="list-style-type: none">⇔ l'allocation se fera au moment de la définition d'une variable correspondant à ce modèle de structure⇔ réservation de mémoire sera faite comme pour une variable classique⇔ espace mémoire réservé = somme des espaces mémoires pour chaque champ

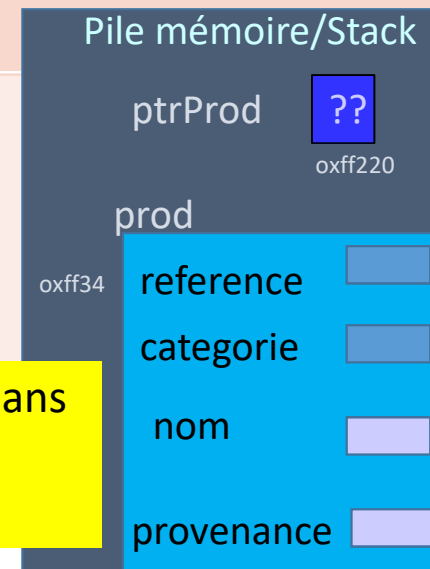
Types Construits / Structures

Modèle	Définition de variable
<pre>struct produit { int reference, categorie ; char *nom, *provenance ; };</pre> 	<p>définition d'une variable prod de type struct produit</p> <pre>struct produit prod;</pre> <p>définition d'une variable ptrProd de type pointeur sur struct produit</p> <pre>struct produit *ptrProd;</pre> <div style="background-color: yellow; padding: 5px;">  <p>Cette variable de type pointeur devra être initialisée avec une adresse existante ou allouée dynamiquement</p> </div> <p>⇔ ptrProd = (struct produit*) malloc(sizeof(struct produit));</p> <p>Même chose pour les pointeurs de caractères nom et provenance</p>

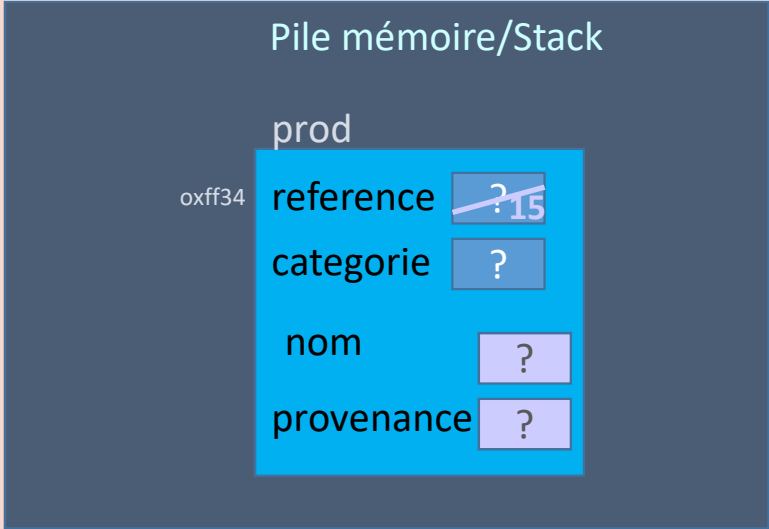
Types Construits / Structures

Définition d'un type à partir d'une structure	Définition de variable
<p>Utilisation de l'opérateur typedef</p> <p>typedef specifType new_nomType ;</p>	<p>typedef int entier ;</p> <p>⇔ entier devient synonyme de int</p> <p>entier x=15;</p>
<p>typedef struct produit {</p> <p> int reference, categorie ;</p> <p> char *nom, *provenance ;</p> <p>}produit ;</p>	<p>produit prod;</p> <p>produit *ptrProd;</p> 

Plus besoin de mettre struct dans les définitions/déclarations de variables, ou de paramètres



Types Construits / Structures

Accès aux champs d'une structure	Exemple
<p>Opérateur ●</p> <p>Utilisé quand la variable n'est pas un pointeur sur la structure</p>	<pre>produit prod; prod.reference=15;</pre>  <p>The diagram illustrates the memory stack (Pile mémoire/Stack) containing a variable 'prod'. The 'prod' variable is represented as a box with four fields: 'reference' (containing the value 15), 'categorie', 'nom', and 'provenance'. The 'reference' field is highlighted with a blue box and a diagonal line, indicating it is the target of the assignment. The address '0xff34' is shown next to the 'prod' variable.</p>

Types Construits / Structures

Accès aux champs d'une structure	Exemple
<p>Opérateur -></p> <p>Utilisé quand la variable est un pointeur sur la structure</p>	<pre>produit *ptrProd; ptrProd = (produit*) malloc (sizeof(produit)); ptrProd->reference = 15;</pre> <p>The diagram shows a stack frame (Pile mémoire/Stack) with a pointer variable <code>ptrPprod</code> at memory address <code>0xff02</code>. The variable contains the value <code>0xff450</code>. An arrow points from this value to a structure located in the heap (Tas mémoire/Heap) at address <code>0xff450</code>. The structure has four fields: <code>reference</code> (value 15), <code>categorie</code>, <code>nom</code>, and <code>provenance</code>.</p>

Types Construits / Structures

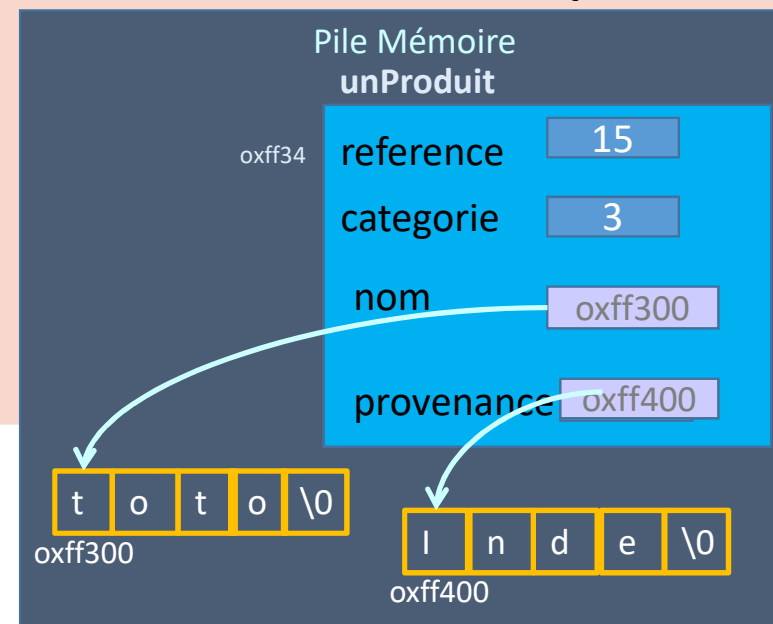
Initialisation

Au moment de la définition de variable
Par une liste de valeurs constantes

Exemple

produit

unProduit = {15, 3, "toto", "Inde"} ;



Types Construits / Structures

Initialisation

Au moment de l'exécution du programme
Par une liste de valeurs entrées au clavier
par l'utilisateur



Exemple

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define NB 40
typedef struct produit {
    int reference, categorie ;
    char *nom, *provenance ;
}produit ;

int main () {
    char buffer[NB];
    int nbCar;
    produit prod;
    printf("Entrez la reference:\n");
    fflush(stdout);
    scanf("%d", &(prod.reference));
    printf("Entrez le nom: \n");
    fflush(stdout);
    scanf("%s", buffer);
    nbCar=strlen(buffer)+1;
    prod.nom = (char *)malloc(nbCar*sizeof(char));
    strcpy(prod.nom, buffer);
    return EXIT_SUCCESS ;
}
```

Types Construits / Structures

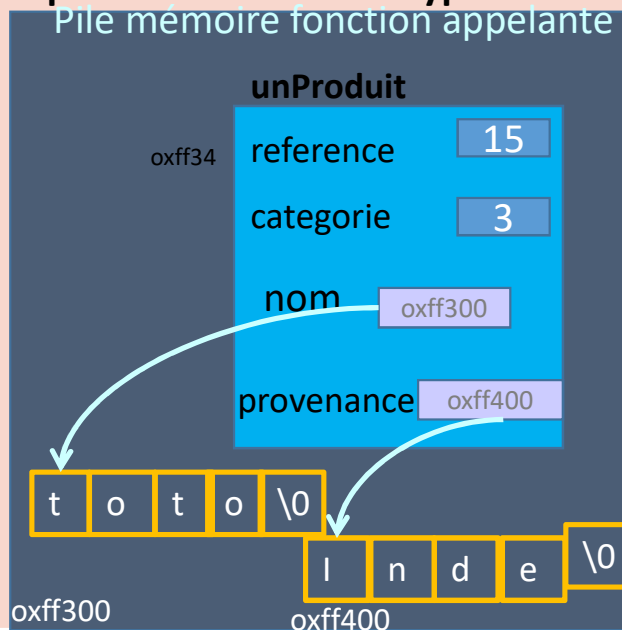
Initialisation	Exemple
<p>Au moment de l'exécution du programme Par une liste de valeurs entrées au clavier par l'utilisateur</p> <p>Pile mémoire/Stack</p> <p>ptrPprod 0xff450</p> <p>Tas mémoire/Heap</p> <p>reference 15</p> <p>categorie ?</p> <p>nom 0xff702</p> <p>provenance ?</p> <p>t o t o \0</p> <p>0xff702</p>	<pre> #include <stdio.h> #include <string.h> #include <stdlib.h> #define NB 40 typedef struct produit { int reference, categorie ; char *nom, *provenance ; }produit ; int main () { char buffer[NB]; int nbCar; produit *ptrProd; ptrProd=(produit *)malloc(sizeof(produit)); printf("Entrez la reference:\n"); fflush(stdout); scanf("%d", &(ptrProd->reference)); printf("Entrez le nom: \n"); fflush(stdout); scanf("%s", buffer); nbCar=strlen(buffer)+1; ptrProd->nom = (char *)malloc(nbCar*sizeof(char)); strcpy(ptrProd->nom, buffer); return EXIT_SUCCESS ; } </pre>

Types Construits / Structures

Passage de structure en paramètres d'une fonction

Passage de paramètre par valeur

Effectue une copie de la variable de type structure passée en paramètres



Exemple

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct produit {
    int reference, categorie ;
    char *nom, *provenance ;
}produit ;

void afficheProduit(produit p){
    printf("nom du produit: %s\n", p.nom);
    printf("provenance du produit: %s\n", p.provenance);
    printf("référence du produit: %d\n", p.reference);
    printf("catégorie du produit: %d\n", p.categorie);
}

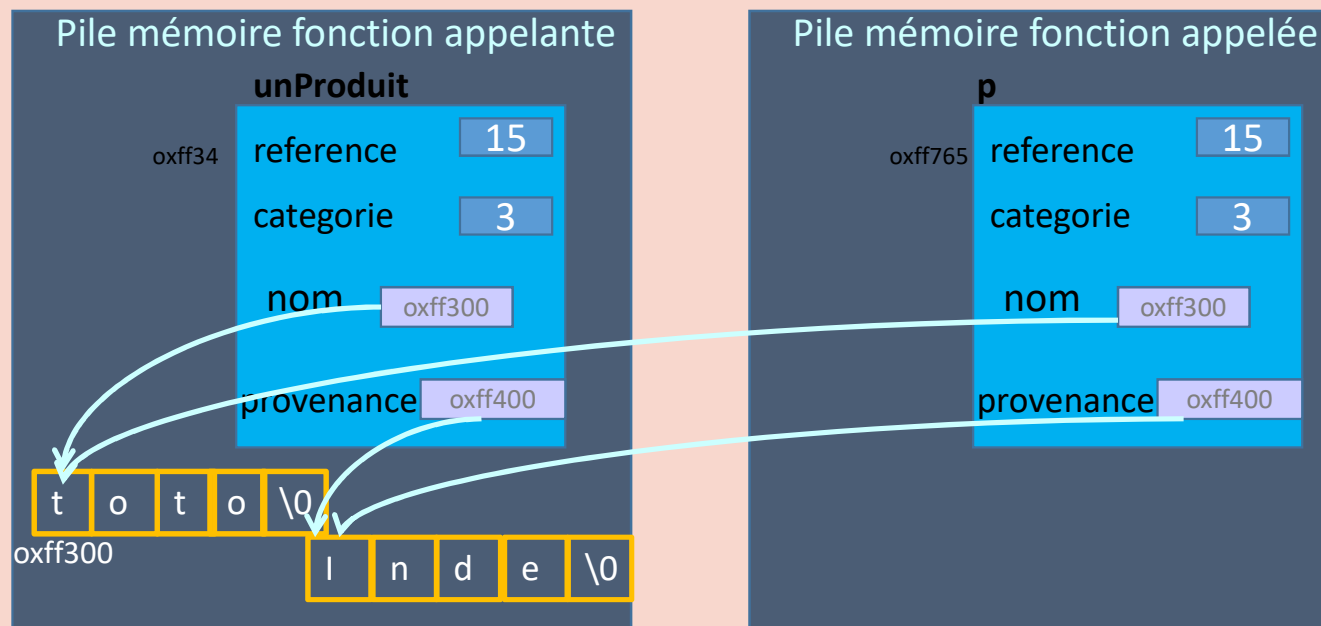
int main () {
    produit unProduit = {15, 3, "toto", "Inde"};
    afficheProduit(unProduit);
    return EXIT_SUCCESS ;
}
```

Types Construits / Structures

Passage de structure en paramètres d'une fonction

Passage de paramètre par valeur

Effectue une copie de la variable de type structure passée en paramètres



Types Construits / Structures

Passage de structure en paramètres d'une fonction

Passage de paramètre par adresse

Effectue une copie de l'adresse de la variable de type structure passée en paramètres

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define NB 40

typedef struct produit {
    int reference, categorie ;
    char *nom, *provenance ;
}produit

L'adresse unProduit envoyée
à la fonction saisieProduit

int main () {
    produit unProduit;
    saisieProduit(&unProduit);
    return EXIT_SUCCESS ;
}
```

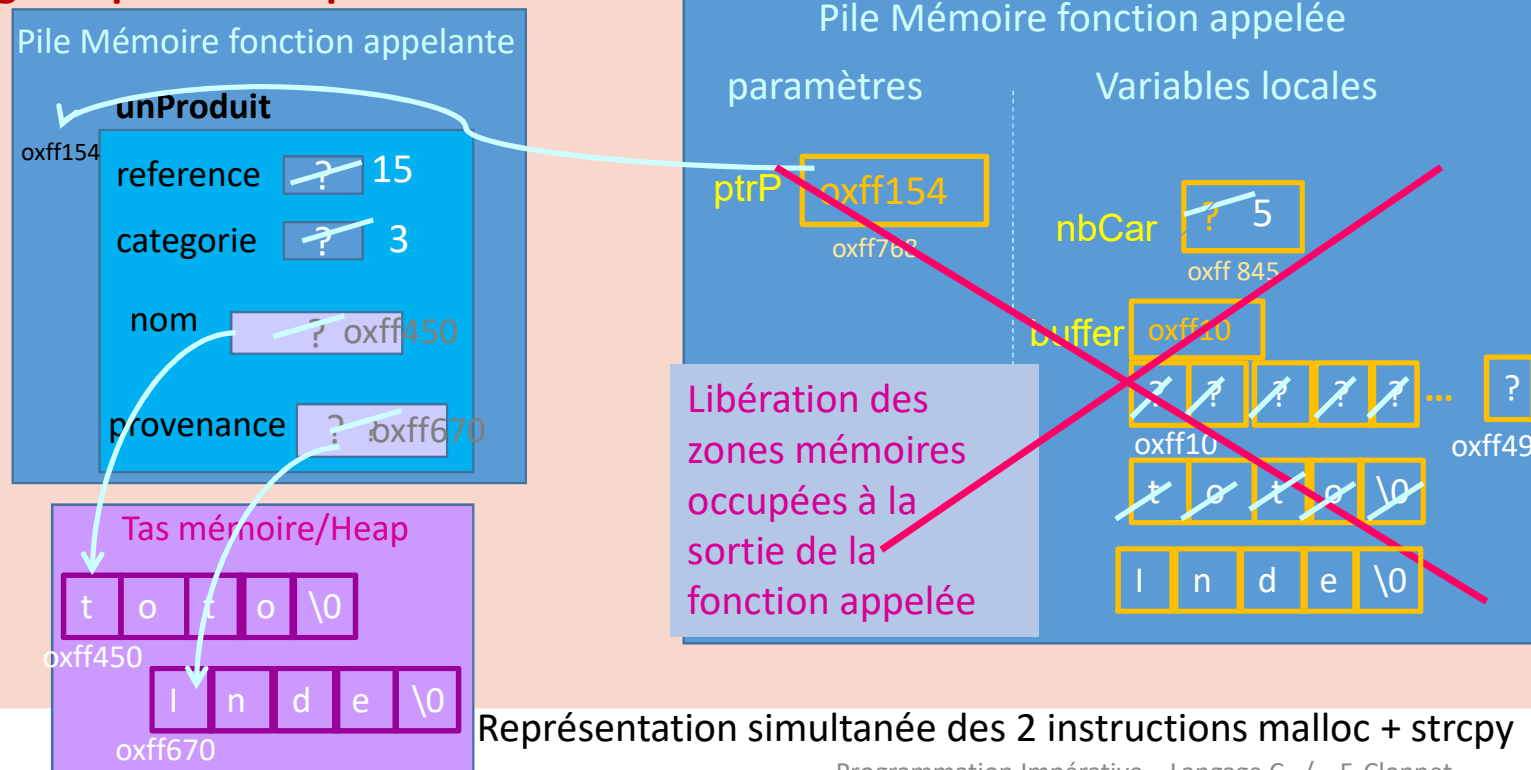
Exemple

```
void saisieProduit(produit *ptrP) { Utilisation d'un paramètre
    char buffer[NB];                de type pointeur pour
    int nbCar;                      stocker l'adresse
    printf("Entrez la reference (entier):");
    fflush(stdout);
    scanf("%d", &(ptrP->reference));
    printf("Entrez la catégorie (entier):");
    fflush(stdout);
    scanf("%d", &(ptrP->categorie));
    printf("Entrez le nom: ");
    fflush(stdout);
    scanf("%s", buffer);
    nbCar=strlen(buffer)+1;
    ptrP->nom = (char *)malloc(nbCar*sizeof(char));
    strcpy(ptrP->nom, buffer);
    printf("Entrez la provenance: ");
    fflush(stdout);
    scanf("%s", buffer);
    nbCar=strlen(buffer)+1;
    ptrP->provenance = (char *)malloc(nbCar*sizeof(char));
    strcpy(ptrP->provenance, buffer);
}
```

Types Construits / Structures

Passage de structure en paramètres d'une fonction

Passage de paramètre par adresse



Représentation simultanée des 2 instructions malloc + strcpy

Programmation Impérative – Langage C / F. Cloppet

Types Construits / Structures

Passage de structure en paramètres d'une fonction	Quand ?
Passage de paramètre par valeur	Si on fait seulement un accès aux champs et on ne modifie pas les valeurs des champs
Passage de paramètre par adresse	<ul style="list-style-type: none">• Si on modifie les valeurs des champs à l'intérieur de la fonction, et que ces modifications doivent être visibles à l'extérieur de la fonction• Si on ne fait pas de modification des champs, et que l'on souhaite éviter la duplication de la variable de type structure passée en paramètres <p>⇔ Duplication peut être coûteuse si la structure occupe une taille mémoire importante</p>

Types Construits / Structures

Retour d'une fonction	Exemple
Un type construit à partir d'une structure peut être un type de retour de fonction	<pre>produit fonction(){ produit p; return p; }</pre> <pre>//fonction appelante produit prod; prod = fonction();</pre> <p>Les valeurs des champs de p sont copiées dans les champs de prod</p>

Types Construits / Structures

Structure comme type de retour d'une fonction

```
produit saisieProd(){
    produit prod;
    char buffer[NB];
    int nbCar;
    printf("Entrez la reference (entier):");
    fflush(stdout);
    scanf("%d", &(prod.reference));
    printf("Entrez la catégorie (entier):");
    fflush(stdout);
    scanf("%d", &(prod.categorie));
    printf("Entrez le nom: ");
    fflush(stdout);
    scanf("%s", buffer);
    nbCar=strlen(buffer)+1;
    prod.nom = (char *)malloc(nbCar*sizeof(char));
    strcpy(prod.nom, buffer);
    printf("Entrez la provenance: ");
    fflush(stdout);
    scanf("%s", buffer);
    nbCar=strlen(buffer)+1;
    prod.provenance = (char *)malloc(nbCar*sizeof(char));
    strcpy(prod.provenance, buffer);
    return prod;
}
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define NB 40

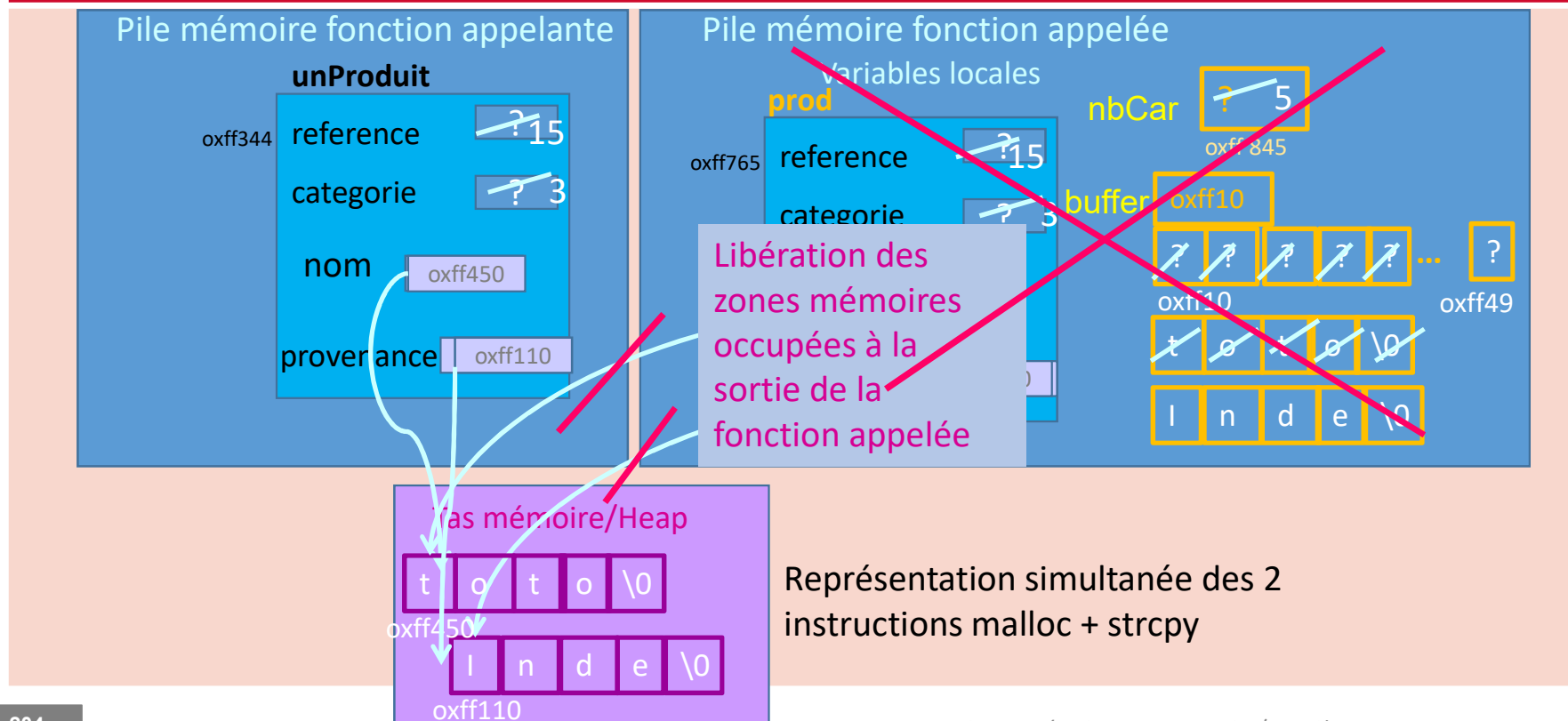
typedef struct produit {
    int reference, categorie ;
    char *nom, *provenance ;
}produit ;
```

```
int main () {
    produit unProduit;
    unProduit=saisieProd();
    afficheProduit(unProduit);
    return EXIT_SUCCESS ;
}
```

```
Entrez la reference (entier):15
Entrez la catégorie (entier):3
Entrez le nom: toto
Entrez la provenance: Inde
nom du produit: toto
provenance du produit: Inde
référence du produit: 15
catégorie du produit: 3
```

Types Construits / Structures

Structure comme type de retour d'une fonction

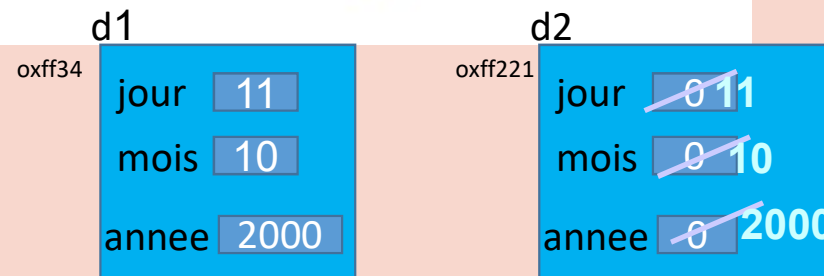


Types Construits / Structures

Affectation d'une variable de type structure
à une autre variable de même type est possible

```
#include <stdio.h>
#include <stdlib.h>
typedef struct date {
    int jour,
        mois,
        annee;
}date;
int main () {
    date d1={11,10,2000}, d2={0,0,0};
    printf("Date d1: %d/%d/%d\n",d1.jour,d1.mois,d1.annee);
    printf("Date d2 avant affectation: %d/%d/%d\n",d2.jour,d2.mois,d2.annee);
    d2 = d1;
    printf("Date d2 après affectation: %d/%d/%d\n",d2.jour,d2.mois,d2.annee);
    return EXIT_SUCCESS ;
}
```

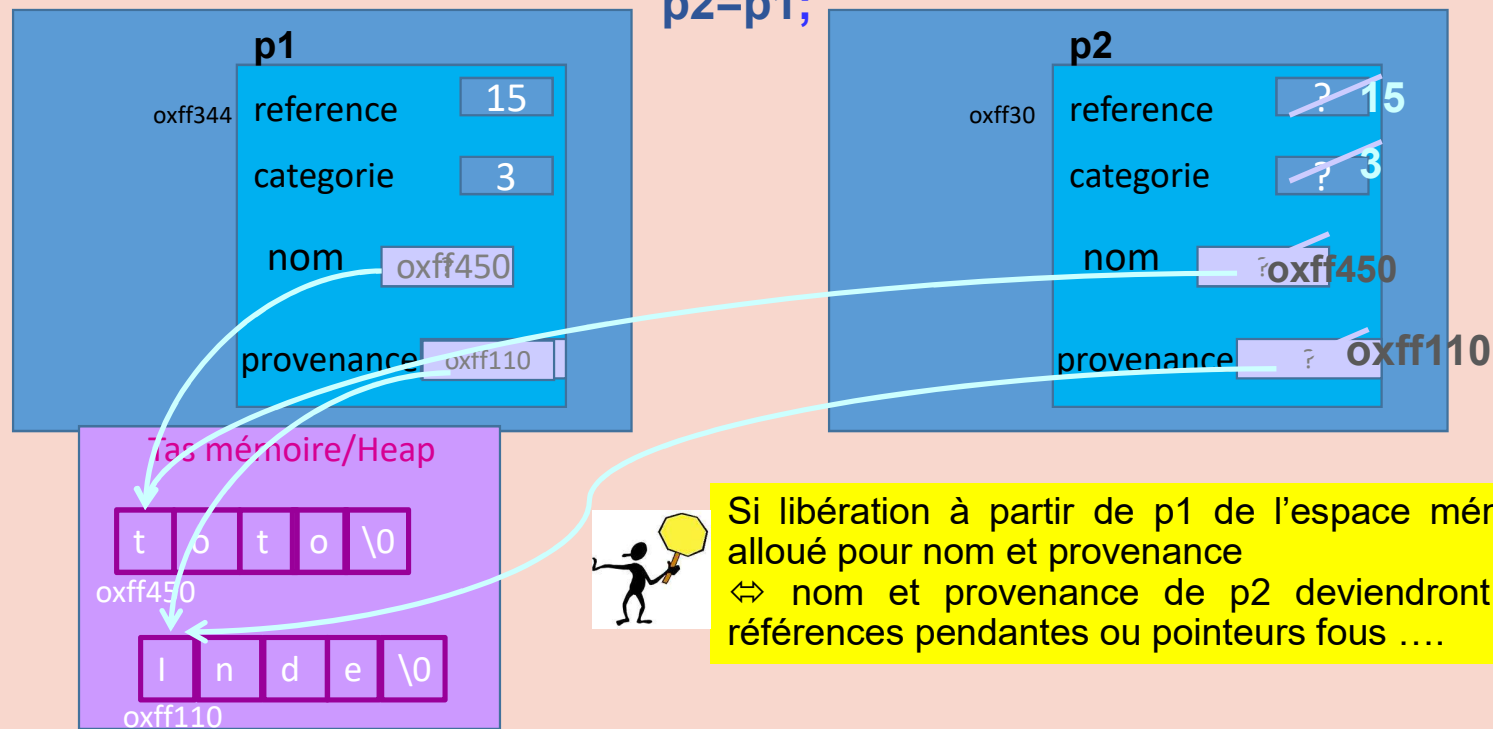
Date d1: 11/10/2000
Date d2 avant affectation: 0/0/0
Date d2 après affectation: 11/10/2000



Types Construits / Structures

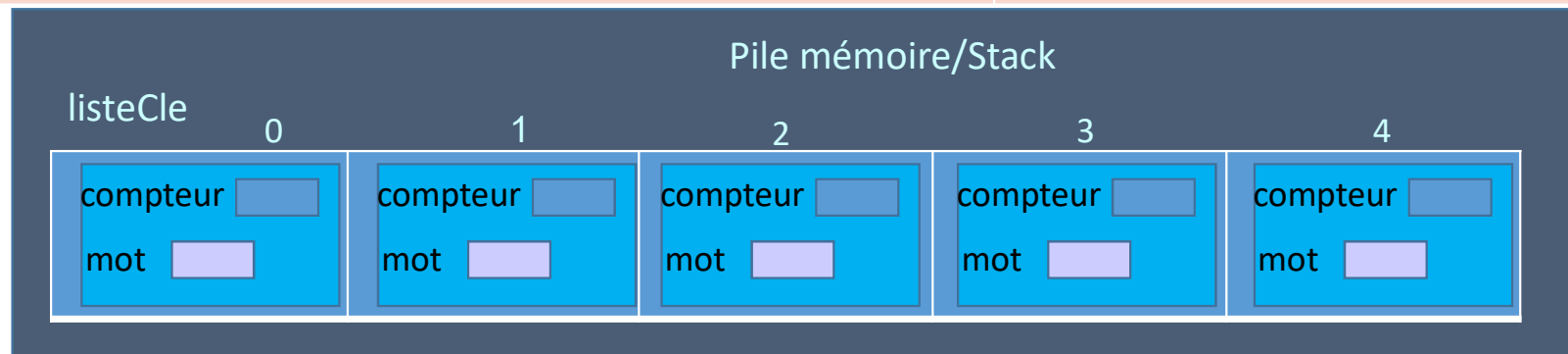
Affectation d'une variable de type structure à une autre variable de même type est possible

`p2=p1;`



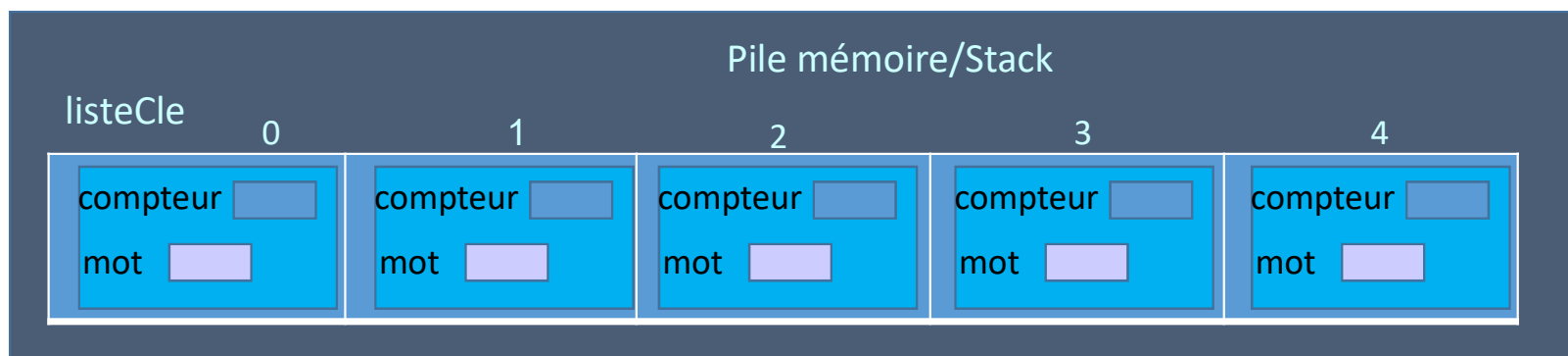
Types Construits / Structures

Tableau de structures	Exemple
<p>Une structure vue de l'extérieur est vue comme quelque chose d'homogène</p> <p>⇔ Il est possible de déclarer des tableaux de structures</p> <p>⇔ permet d'éviter de gérer n tableaux en parallèle</p>	<pre>#define NB 5 typedef struct cle { int compteur ; char *mot; }cle ; cle listeCle[NB] ;</pre>

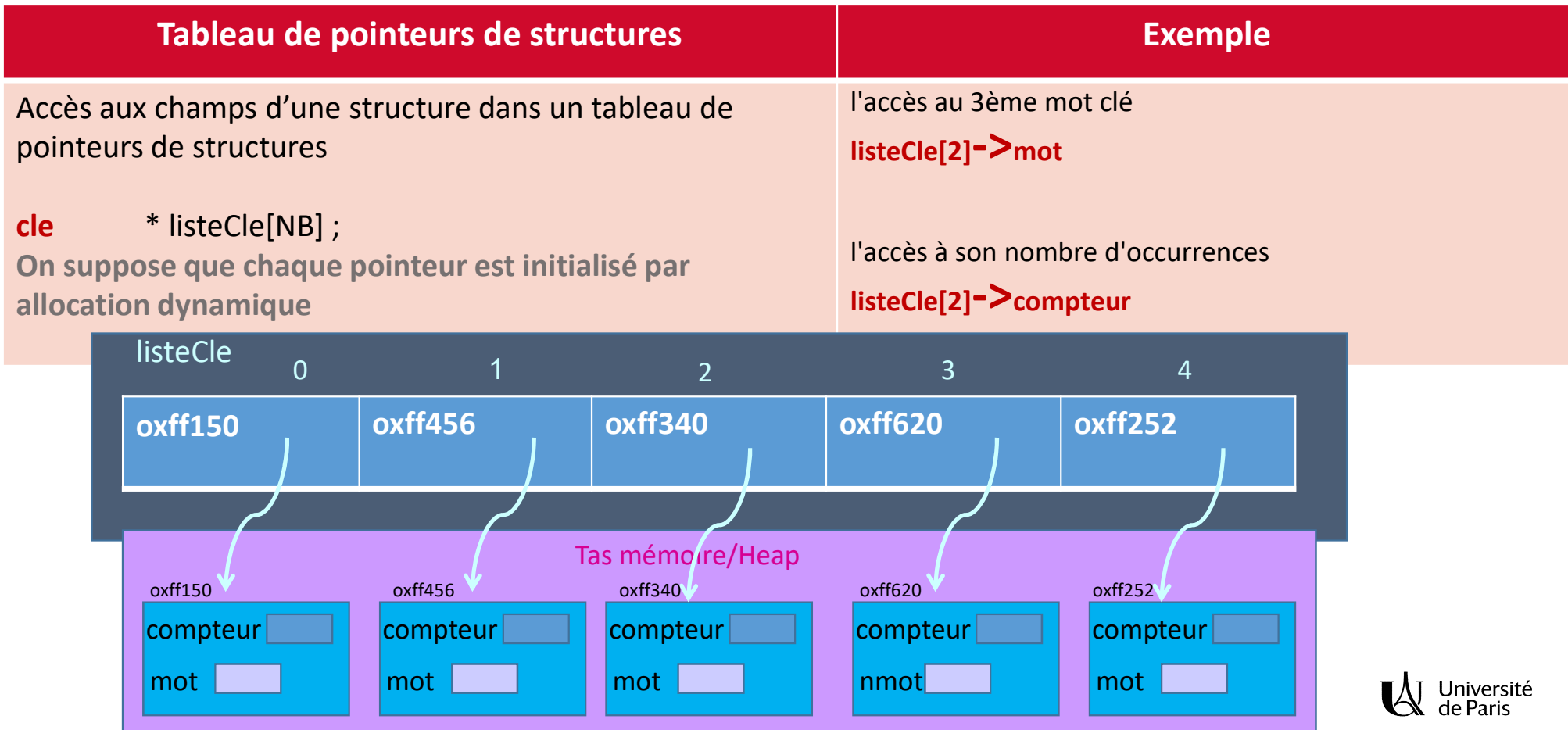


Types Construits / Structures

Tableau de structures	Exemple
Accès aux champs d'une structure dans un tableau de structures cle listeCle[NB] ;	l'accès au 3ème mot clé listeCle[2].mot l'accès à son nombre d'occurrences listeCle[2].compteur



Types Construits / Structures



Types Construits / Structures

Tableau de pointeurs de structures

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#define NB 3
#define NBCAR 130
```

```
typedef struct cle {
    char *mot ;
    int compteur ;
}cle ;
```

```
void calculOccurence(cle lCle[], char* mots[], int nb){
    int i,j;
    for(i=0;i<nb;i++){
        for(j=0;j<NB;j++){
            if(strcmp(mots[i], lCle[j].mot)==0)
                lCle[j].compteur++;
        }
    }
}
```

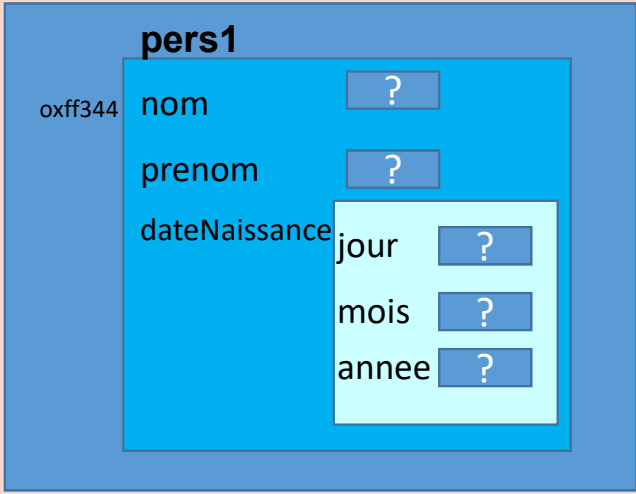
```
void afficheListeCle(cle lCle[]){
    int j;
    for(j=0;j<NB;j++){
        printf("mot cle: %s\t", lCle[j].mot);
        printf("nb d'occurrences: %d\n", lCle[j].compteur);
    }
}
```

On suppose que l'on dispose de la fonction decoupePhrase qui saisit la phrase et la découpe en mots stockés dans un tableau de chaînes de caractères

```
int main () {
    char *motsPhrase[NBCAR];
    cle lCle[NB]={{ "le",0},{ "la",0},{ "les",0}};
    int n;
    decoupePhrase(motsPhrase,&n);
    calculOccurence(lCle, motsPhrase, n);
    afficheListeCle(lCle);
    return EXIT_SUCCESS ;
}
```

```
Entrez le texte (nb de caractères <130:)
le chat et le chien sont les meilleurs amis de la terre.
mot cle: le      nb d'occurrences: 2
mot cle: la      nb d'occurrences: 1
mot cle: les     nb d'occurrences: 1
```

Types Construits / Structures

Composition de structures	Représentation
<p>Un type construit à partir d'une structure peut être utilisé comme type d'un champ d'une autre structure</p> <pre>typedef struct date { int jour, mois, annee; }date ; typedef struct personne { char *nom,*prenom; date dateNaissance; }personne ;</pre>	<pre>personne pers1;</pre> 

Types Construits / Structures

Composition de structures	Accès aux champs
<p>Un type construit à partir d'une structure peut être utilisé comme type d'un champ d'une autre structure</p> <pre>typedef struct date { int jour, mois, annee; }date ; typedef struct personne { char *nom,*prenom; date dateNaissance; }personne ;</pre>	<pre>personne pers1;</pre> <p>accès au champ jour de la date de naissance de <code>pers1</code></p> <pre>pers1.dateNaissance.jour</pre> <pre>personne *pers2;</pre> <p>..... On suppose <code>pers2</code> allouée dynamiquement</p> <p>accès au champ jour de la date de naissance de <code>pers2</code></p> <pre>pers2->dateNaissance.jour</pre>

Types Construits / Structures

Structures récursives ou autoréférentielles

- structure récursive quand un des champs est du type de la structure qu'on est en train de définir
- Il est **INTERDIT** de définir **une structure contenant une instance d'elle-même** **MAIS** il n'est pas interdit qu'un champ soit du type **pointeur sur la structure**.

```
typedef struct individu{  
    char        nom [40];  
    char        prenom [40];  
    int         age ;  
    struct individu *pere,  
                *mere ;  
}individu ;
```

écriture correcte

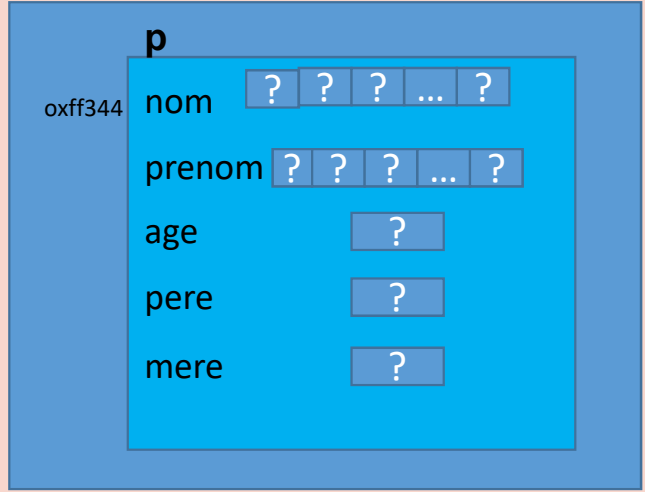
```
typedef struct individu{  
    char        nom [40];  
    char        prenom[40] ;  
    int         age ;  
    struct individu pere,  
                mere ;  
}individu ;
```

écriture incorrecte



Allouer les pointeurs pere et mere avant de remplir leurs champs

Types Construits / Structures

Structures récursives	Représentation
<pre>typedef struct individu { char nom[40],prenom[40]; int age; struct individu *pere,*mere; }individu;</pre>	<p>individu p;</p> 

Types Construits / Structures

Structures récursives	Exemple
<p>Soit individu p; Supposons les allocations dynamiques, et les initialisations des champs faites</p> <ul style="list-style-type: none">• accéder au père de p, à la mère de p?• accéder au grand-père paternel de p, à la grand-mère maternelle de p?• accéder à l'arrière grand-père paternel de p, à l'arrière grand-mère paternelle de p?	<p>individu p;</p> <p>p.pere p.mere</p> <p>p.pere->pere p.mere->mere</p> <p>p.pere->pere->pere p.pere->mere->mere</p>

Types Construits / Structures

Structures récursives ou autoréférentielles

```
void SaisiePersonne(Individu *personne){
    char rep;
    printf("Nom de la personne? ");
    fflush(stdout);
    scanf("%s", personne->nom);
    printf("Prénom de la personne? ");
    fflush(stdout);
    scanf("%s", personne->prenom);
    printf("\nAge de la personne? ");
    fflush(stdout);
    scanf("%d", &personne->age);
    fflush(stdin);
    do{
        printf("\nVoulez-vous saisir les informations concernant son père? (O/N)");
        fflush(stdout);
        scanf("%c",&rep);
        fflush(stdin);
    }while(rep!='O' && rep!='o' && rep!='N' && rep!='n');
    if(rep=='o' || rep=='O'){
        personne->pere = (Individu *)malloc(sizeof(Individu));
        if(personne->pere!=NULL)
            SaisiePersonne(personne->pere);
    }else
        personne->pere=NULL;
    //meme chose pour les infos concernant la mere
}
```

Types Construits / Structures

Structures récursives ou autoréférentielles

```
void AffichePersonne(Individu personne){
    printf("\n***** Infos personne *****");
    fflush(stdout);
    printf("\nNom: %s ", personne.nom);
    fflush(stdout);
    printf("\nPrénom: %s ", personne.prenom);
    fflush(stdout);
    printf("\nAge: %d ", personne.age);
    fflush(stdout);
    if(personne.pere!=NULL){
        printf("\n    ++++ infos Père ++++");
        fflush(stdout);
        AffichePersonne(*personne.pere);
    }
    //meme chose pour les infos concernant la mere
}

int main(){
    Individu p;
    SaisiePersonne(&p);
    AffichePersonne(p);
    return EXIT_SUCCESS;
}
```