Corrigé série 12.1 : STRUCTURES COMPLEXES

EXERCICE 1 – MA PREMIERE STRUCTURE – [FACILE]

- Écrire une fonction ***afficher*** qui affiche les différents champs de la structure passée en argument.
- Écrire une fonction nommée ***RAZ*** permettant de remettre à zéro les 2 champs d'une structure de ce type. La transmission de l'argument doit se faire par adresse.

```
#include <stdio.h>
#include <stdlib.h>
// Il faut rendre le nouveau type "struct essai" visible dans tout le code
// pour cela on le met ici au début du code source HORS des fonctions
// (portée globale)

// Déclaration(définition) d'un nouveau type struct essai
struct essai
{
  int n;
  float x;
};

// prototypes des fonctions
void afficher(struct essai);
void RAZ(struct essai*);//il faut l'adresse(un pointeur sur) l'arg .pour modifier son contenu

int main(void)
{
   struct essai maVariable; //déclaration d'une VARIABLE maVariable  de type struct essai
   maVariable.n = 13;
   maVariable.x = -8.99;

   afficher(maVariable);
   RAZ(&maVariable);
   afficher(maVariable);
   return 0;
}

void afficher(struct essai toto)
{
   // toto est une variable, avec l'opérateur .  on accéde à ses champs
  printf("champ n: %d\n", toto.n);
  printf("champ x: %f\n", toto.x);
}

// on doit avoir l'adresse (un pointeur sur) l'argument pour modifier son contenu
void RAZ(struct essai* ptrToto)
{
  // on accède aux champs de la var. pointée par ptrToto avec l'opérateur -> (indirection)
  ptrToto->n = 0;
  ptrToto->x = 0.0;
}
```

EXERCICE 2 (a) - (e)

[MOYEN]

Déclarer les structures `Personne` et `Famille` , les utiliser pour mémoriser les membres de la famille Dupont {Jean 38 ans, Eva 36 ans, Emma 8 ans, Kevin 4 ans} dans une variable. Écrire une fonction **`AfficheAgeTotal`** qui affichera la somme des âges des membres de la famille, ainsi qu'une fonction **`UnAnDePlus`** qui ajoute un an à chacun des membres. Écrire un programme pour les tester et afficher aussi la taille en octets de la variable associée à la famille Dupont.

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

// On définit ici le type "struct Personne" pour le type "struct Famille"
struct Personne
{
  char *          prenom; // plus flexible que char prenom[32] par exemple
  unsigned short  age;
};

struct Famille
{
  char * nom;
  struct Personne pere;
  struct Personne mere;
  struct Personne fils;
  struct Personne fille;
};

// prototypes des fonctions
void affichageAgeTotal(struct Famille);
void unAnDePlus(struct Famille*);


int main(void)
{

   // déclaration et initialisation d'une VARIABLE de type struct Famille
   struct Famille uneFamille =  {
     "Dupont",         //champ nom de la Famille
     {"Jean", 38},     //champ pere (struct imbriquée --> accolade imbriquée)
     {"Eva",  36},     //champ mere
     {"Emma", 8},
     {"Kevin",4}               };

   affichageAgeTotal(uneFamille);     //appel avec la variable en argument
   unAnDePlus(&uneFamille);           //appel avec l'adresse de uneFamille
   affichageAgeTotal(uneFamille);

   printf("La famille %s ", uneFamille.nom);
   printf("occupe %d octets", sizeof(uneFamille));
   return 0;
}
// suite page suivante
```

```
L'age total de la famille Dupont est de 86 ans
L'age total de la famille Dupont est de 90 ans
La famille Dupont occupe 36 octets
```

```
// la structure  donnée en argument à l'appel de main est copiée --> copie
// Note: plus efficace de travailler en passant un pointeur si la structure est grande
void affichageAgeTotal(struct Famille copie )
{
    int ageTotal = copie.pere.age + copie.mere.age +
                   copie.fils.age + copie.fille.age;
    printf("L'age total de la famille %s est de %d ans\n", copie.nom,
  ageTotal);
}



// on doit avoir l'adresse (un pointeur sur) l'argument pour modifier son contenu
void unAnDePlus(struct Famille* ptrSurFamille)
{
  // puisque l'on a un pointeur, il faut employer l'opérateur ->

  ptrSurFamille->pere.age++;    // ou (*ptrSurFamille).pere.age++;
  ptrSurFamille->mere.age++;    // ou (*ptrSurFamille).mere.age++;
  ptrSurFamille->fils.age++;    // ou ...
  ptrSurFamille->fille.age++;
}
```

EXERCICE 2 (f) [BONUS]

(f) Réaliser un nouveau projet avec une famille composée d'une mère, d'un père et d'un nombre variable d'enfants (max.10) auxquels on accèdera par un tableau de pointeurs. Écrire et tester deux fonctions:

```
AjouterEnfant(...)
AfficherFamille(...)
```

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  ...
}
```

g) [BONUS] Monsieur et Madame Talus ont 4 fils, comment les appellent-ils?

Réponse : Jean, Jean, Jean et Jean ! Comme ça ils font des économies pour leur voiture!

Version I : Pour les enfants : Tableau dynamique de Personne

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

struct Personne
{
     char prenom[20];
     int age;
};

struct Famille
{
    char nom[20];
     struct Personne pere, mere;
     int nombreEnfants;
     struct Personne *listEnfants; //Pointeur sur une liste allouée
dynamiquement
};
```

```
void afficheAgeTotal(struct Famille f)
{
    int i,ageTotal=0;
    ageTotal += f.pere.age;
    ageTotal += f.mere.age;
    for(i=0; i< f.nombreEnfants;i++)
        ageTotal += f.listEnfants[i].age;
    printf("Age total: %d\n",ageTotal);
}

void afficheFamille(struct Famille f)
{
    int i;
    struct Personne *ptr;

    // Affichage pere et mere AVEC POINTEUR
    ptr=&(f.pere);
    for(i=0;i<2;i++)
    {
        printf("Parent: %s %d\n", ptr->prenom, ptr->age);
        ptr++;  // Passe à la personne suivante
    }
    // Affichage enfants
    ptr=f.listEnfants;
    for(i=0;i<f.nombreEnfants;i++)
    {
        //formalisme tableau
        printf("Enfants: %s %d\n", f.listEnfants[i].prenom,
                                   f.listEnfants[i].age);
        // acces pointeur
        printf("Enfants: %s %d\n", ptr->prenom, ptr->age );
        ptr++;
    }
}

void unAnDePlus(struct Famille *f)
{
    int i;
    (*f).pere.age++;
    f->mere.age++;  // Tous ces accès sont équvalents
    for(i=0;i<f->nombreEnfants;i++)
        f->listEnfants[i].age++;
 }
```

```
int main()
{

    struct Famille uneFamille =  {
         "Dupont",      // champ nom de la Famille
         {"Jean", 38},  // champ Pere (struct imbriquée --> accolade
imbriquée)
         {"Eva",  36},  // champ Mere
    };
    //Allocation dynamique de 3 enfants
    uneFamille.nombreEnfants = 3;
    uneFamille.listEnfants   = (struct Personne*) malloc(
                uneFamille.nombreEnfants * sizeof(struct Personne) );

    strcpy( uneFamille.listEnfants[0].prenom, "Paul");
    uneFamille.listEnfants[0].age = 100;
    strcpy( uneFamille.listEnfants[1].prenom, "Emile");
    uneFamille.listEnfants[1].age = 101;
    strcpy( uneFamille.listEnfants[2].prenom, "Victor");
    uneFamille.listEnfants[2].age = 102;

    afficheFamille(uneFamille);
    unAnDePlus(&uneFamille);
    afficheFamille(uneFamille);
    afficheAgeTotal(uneFamille);

    printf("taille  famille  %d\n",sizeof(uneFamille));
    // -> 116 = 20 + 2*(20+4) + adresse(8) = 76

    free(uneFamille.listEnfants);
    uneFamille.listEnfants=NULL;
    uneFamille.nombreEnfants = 0;


    return 0;
}
```

EXERCICE 3 – TYPE SYNONYME COMPLEXE – [MOYEN]

Compléter le programme donné:

- **définir le type `ComplexTy`** avec 2 membres de type réel: `reelle`, `imaginaire`
- ajouter le code à scanf pour que la saisie d'un nombres complexe ne se fasse qu'au format (r,i) et encapsuler ce code dans une fonction `Saisie`.
- **ajouter les fonctions** appelées pour calculer la somme, et la différence de deux nombres complexes. Voir code plus bas.
- implémenter une fonction **`affiche_complexe(…)`** pour afficher un nombre complexe sous la forme **(r, i)**, partie réelle et partie imaginaire.

Note: on demande d'écrire les **déclarations** et les **fonctions** manquantes.

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

typedef struct
{
  float reelle ;
  float imaginaire;
} ComplexeTy;

// Prototypes des fonctions
ComplexeTy saisie();
ComplexeTy somme_complexe(ComplexeTy a, ComplexeTy b);
ComplexeTy difference_complexe(ComplexeTy a, ComplexeTy b);
void       affiche_complexe(ComplexeTy c);


int main()
{
  ComplexeTy c1, c2, resultat;
  c1=saisie();
  c2=saisie();

  resultat = somme_complexe(c1, c2);

  printf("\nSomme des deux nombres      : ");
  affiche_complexe(resultat);
  resultat = difference_complexe(c1, c2);
  printf("\nDifference des deux nombres : ");
  affiche_complexe(resultat);

  system("Pause");
  return 0;
]

ComplexeTy saisie()
{
  ComplexeTy nouveau;
  int status;
  do
  {
    printf("Donner la valeur d'un nombre complexe (r,i): ");
```

```
    status = scanf("(%f, %f)", &nouveau.reelle, &nouveau.imaginaire);
    fflush(stdin);
  }
  while (status != 2);
  return nouveau;       // COPIE de la variable renvoyée à main
}

ComplexeTy somme_complexe(ComplexeTy a, ComplexeTy b)
{
  ComplexeTy c;

  c.reelle     = a.reelle     + b.reelle;
  c.imaginaire = a.imaginaire + b.imaginaire;

  return c;
}

ComplexeTy difference_complexe(ComplexeTy a, ComplexeTy b)
{
  ComplexeTy c;

  c.reelle     = a.reelle     - b.reelle;
  c.imaginaire = a.imaginaire - b.imaginaire;

  return c;
}

void affiche_complexe(ComplexeTy c)
{
  printf("(%f, %f)\n", c.reelle, c.imaginaire);
}
```