Séance 5

Structures de données; allocation dynamique; fichier en-tête.

Définition d'un type structuré:
```
typedef struct Individu
{
     char nom[64];
     char sexe;
     int age;
} Individu, *Pindividu;
```

Déclaration d'une variable de type structuré:
```
Individu x;
```

Utilisation:
```
strcpy(x.nom, "durand");
x.sexe = 'M';
x.age = 24;
```

Déclaration de pointeurs sur une variable de type structuré:
```
Pindividu p;
Individu * q;
```

Déclaration d'un tableau de types structurés:
```
Individu Tindividu[TAILLE];
```

Déclaration d'un tableau de pointeurs vers des types structurés:
```
Individu * Tindividu[TAILLE];
```

Allocation dynamique d'une structure:
```
p = (Individu *) malloc(sizeof(Individu));
```

Allocation dynamique de n structures:
```
p = (Individu *) calloc(n, sizeof(Individu));
```

Libération:
```
free(p);
```

Contenu d'un fichier en-tête:
```
// individu.h
#ifndef INDIVIDU_H
#define INDIVIDU_H
```

```
// définition de type structures
// déclaration de fonctions
#endif
```

Utilisation d'un fichier en-tête:

```
#include "individu.h"    // défini par le programmeur
#include <stdio.h>       // pré-défini par le C
```

Contenu d'un fichier .c:

```
// individu.c
// inclusion des fichiers en-tête
// ...
// définition de fonctions
// ...
```

Champs d'un type structuré de type pointeur sur un autre type structuré:

```
typedef struct Objet
{
     ...
} Objet;
typedef struct Individu
{
     ...
     Objet * monObjet;
} Individu;
```

Champs d'un type structuré de type pointeur sur le type structuré lui-même (utile pour programmer le père ou la mère d'un individu):

```
typedef struct Individu
{
     char nom[64];
     ...
     struct Individu * monPere;
     struct Individu * maMere;
} Individu;
```

Exercices

1) On va écrire un programme gérant une structure représentant un nombre complexe, allouée dynamiquement.

a) Déclarer une structure permettant de stocker un nombre complexe. Fournir un type `COMPLEXE` et pointeur sur un `COMPLEXE`.
b) Déclarer deux complexes `C1` et `C2`.
Écrire une fonction qui initialise un `COMPLEXE` connaissant son adresse.
Initialiser `C1` à `1.5+3.5i`, initialiser `C2` à `1.5-3.5i`
Écrire une fonction qui affiche un `COMPLEXE` connaissant son adresse. Afficher `C1` et `C2`.
c) Écrire une fonction qui additionne les valeurs de deux `COMPLEXE` connaissant leurs adresses et stocke le résultat dans le premier des 2.
Additionner `C1` et `C2` et stocker le résultat dans `C1`. Afficher `C1` et `C2`.
d) Déclarer dynamiquement un tableau `C` de `n` complexes.
Affecter aux éléments de `C` un complexe `a+ib` où a est croissant de `0` à `n-1` et `b` est décroissant de `n-1` à `0`. Afficher tous les éléments de `C`.
e) Calculer le complexe `S` égal à la somme de tous les éléments de `C` et afficher `S`.
f) Libérer `C`.

2) Soit le programme suivant gérant un tableau d'entiers alloué dynamiquement :

```
void main(){
     int A=1, B=2, C=3;
     int *p1=NULL, *p2, *p3=(int *)calloc(5,sizeof(int));
     p1 = &A ;
     p2 = &C ;
     *p1 = (*p2)++ ;
     p1 = p2 ;
     p2 = &B ;
     *p1 -= *p2 ;
     ++*p2 ;
     *p1 *= *p2 ;
     A = ++*p2**p1;
     p1 = &A ;
     *p2 = *p1/*p2 ;
     p3++ ;
     p3++ ;
     *p3++ = 4 ;
     free(p3);
}
```

Faites un schéma des cases mémoires pour chaque instruction du programme.

3) Commencer le projet: cf exécutable exemple `societe`.