Séance 6 : Les structures et chaînes de caractères

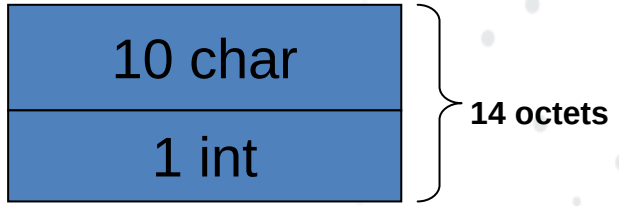
Langage C : séance 6

Les structures :

Une structure permet de définir un regroupement de données de types différents.

- Syntaxe 1 :

```
struct element {
        char nom[10];
        int age;
} ;
struct element e;   // e est une variable
```



- Syntaxe 2 :

```
struct element {
        char nom[10];
        int age;
}  e;     // e est une variable
```

Langage C : séance 6

Les structures :

- Syntaxe 3 : on utilise un typedef.

```
typedef struct {
        char nom[10];
        int age;
} element;    // element est un nouveau type
element e;   // e est une variable
```

- Utilisation : valable pour les 3 syntaxes précédentes.

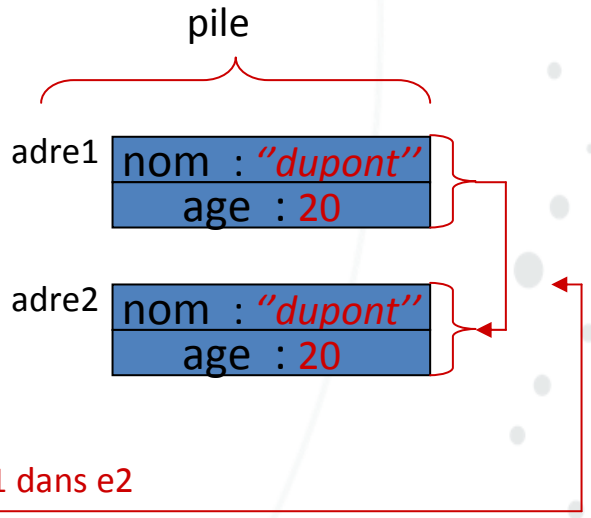
```
        e.age= 20;
Ou      scanf(''%d'',& e.age);
        printf(''%d'', e.age);

        strcpy(e.nom,''dupont'');
Ou      scanf(''%9s'',e.nom);    // e.nom est une chaîne de caractères
        printf(''%s'',e.nom);
```

Langage C : séance 6

Affectation entre variables structure :

On peut affecter une variable structure dans une autre de même type.

```
element e1 , e2 , t [3];   // réservation par le système dans la pile
e1.age= 20;
strcpy(e1.nom,"dupont");

e2=e1;
t[0] = e1 ;
```



Langage C : séance 6

Les pointeurs sur structures :

Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse

```
int main()
{
    element e1, e2 ;   // réservation par le système dans la pile
    … ;                //initialisation des variables e1 e2
    permutation (&e1,&e2) ;
}

 void permutation (element * a , element * b)
{
     element temp;
     temp = *a;
     *a = *b;
     *b = temp;
}
```

Langage C : séance 6

Les pointeurs sur structures :

Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse.

```
int main()
{
    element e1, e2 ;   // réservation par le système dans la pile
    init( &e1,&e2) ;
    …
}

 void init(element * a , element * b)
{
    (*a) .age = 0;              // parenthèses indispensable car priorité des opérateurs !!!
    strcpy( (*a) . nom, « inconnu » );
    b -> age  = 0;              // b->age   est  identique à   (*b).age
    strcpy( b->nom, « inconnu » );
}
```

L'écriture ***(*a).age*** *est identique à* ***a->age .***

On privilégiera l'utilisation de -> à celle de * .

Langage C : séance 6

Les pointeurs sur structures :

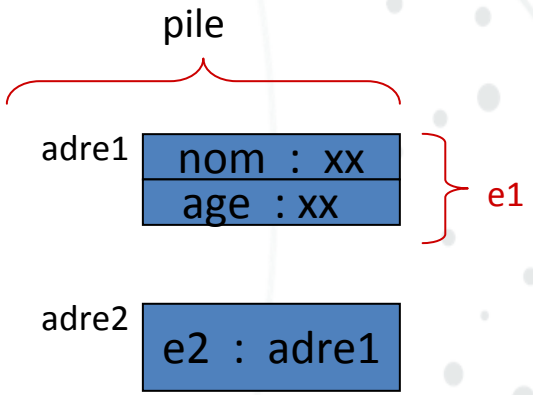
Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse

```
int main()
{
 element e1, e2 ;   // réservation par le système dans la pile
 saisir( &e1,&e2) ;
 …
}

 void saisir(element * a , element * b)
{    …
     scanf(« %d », &(*a) .age) ;
     scanf(« %s »,  (*a) . nom) ;
     …
     scanf (« %d », & b-> age ) ;
     scanf(« %s »,  b->nom );
}
```

Langage C : séance 6

Les pointeurs sur structure :

On peut déclarer un pointeur sur structure.



•déclaration 1:

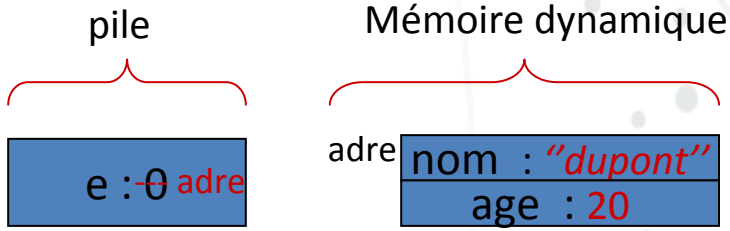
```
element e1;   // réservation par le système dans la pile
element *e2; // e2 est un pointeur sur structure
e2=&e1;       // e2 pointe sur la structure e1

(*e2).age= 20;                  // ou   e2->age=20;
scanf(''%d'',& (*e2).age); // ou scanf(''%d'',& e2->age);
printf(''%d'', (*e2).age);    //ou printf(''%d'', e2->age);

strcpy((*e2).nom,''dupont''); //ou  strcpy(e2->nom,''dupont'');
scanf(''%9s'', (*e2).nom);     // ou scanf(''%9s'',e2->nom);
printf(''%s'',(*e2).nom);    // ou printf(''%s'',e2->nom);
```

Rm : e1 n'est pas une adresse, e1 représente l'ensemble des données (age et nom).
&e1 correspond à l'adresse de la structure en mémoire (adre1 ici).

Langage C : séance 6

Les pointeurs sur structure :

On peut allouer une structure en mémoire dynamique.



•déclaration 2:

```
element *e=NULL;   // réservation par le système dans la pile
e=malloc(sizeof(element)); // allocation en mémoire dynamique

e->age= 20;  // ou   (*e).age=20;

strcpy(e->nom,''dupont'');  //ou  strcpy((*e).nom,"dupont");
…
free( e );   // libération en mémoire dynamique
e=NULL;
```

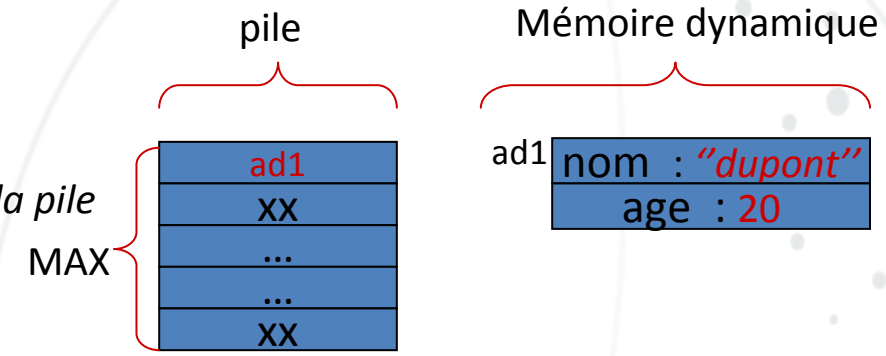
Langage C : séance 6

Tableau de pointeurs sur structure :

On peut déclarer un tableau de pointeurs sur structure.

•déclaration 3:

```
#define MAX 100
element * t[MAX];   // réservation par le système dans la pile

// allocation en mémoire dynamique :
t[0]=malloc(sizeof(element)); // ➔ ad1

t[0]->age= 20;
strcpy(t[0]->nom,''dupont'');
 …
free( t[0]);
t[0]=NULL;
```



Langage C : séance 6

Les priorités des opérateurs :

1	Fonction Tableau Champ de structure	( ) [ ] -> .
2	Negation booleenne In(De)crementation Indirection Adresse	! ++ -- * &
3	Multiplication Division Modulo	* / %
4	Addition Soustraction	+ -
5	Relation logique	< <= > >=
6	Egalite	== !=
7	Et booleen	&&
8	Ou booleen	||
9	Affectation	= += -=

Langage C : séance 6

Les Unions :

Une union est une structure qui ne peut avoir qu'un seul champ actif à la fois.

•Syntaxe :

Exemple :

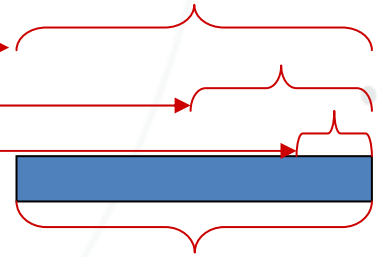
nom de l'union

```
union selection
        {
        short i;
        float f;
        char c;
} s;   //   s variable de type « union selection »
```

Liste des champs

pile

```
s.f=3.14;
s.i=10;
s.c='a';
```



Nbre d'octets du champ le plus grand

On a en fait : sizeof(s) = max (sizeof(i) , sizeof(f), sizeof(c))

Langage C : séance 6

Les chaînes de caractères : on utilise un tableau de caractères.

- syntaxe :

  char machaine**[**Nbre_car**]**;

- dernier caractère de la chaîne : ‘\0’ (\zéro) de valeur 0 (zéro)

- initialisation :

  char tab[]={ ‘a’,’e’,’i’,’o’,’u’,’y’}; // taille 6, pas de ‘\0’ à la fin

  char tab[]=‘’aeiouy’’ ; // taille 7 car ‘\0’ placé à la fin

Langage C : séance 6

Saisie de chaînes de caractères avec « scanf » :

- Saisie limitée de caractères :

  scanf(‘’ %5s‘’ , ch);

  Avec 5, nombre max de caractères à saisir.

- Saisie comprenant un (ou plusieurs) espace(s) :

  scanf(‘’%[^\n]’’, ch);

- Saisie avec espace et 5 caractères max par exemple :

  scanf(‘’%5[^\n]’’, ch); // avec 5 nombre max de car.

Langage C : séance 6

La fonction « sscanf » :

- Permet d'extraire des données à partir d'une chaînes de caractères.
- Syntaxe : sscanf(*dans , comment , où*);

  Recherche l'information dans le premier argument, sous la forme du comment, et le range dans le où.

- exemple :

```
int entier;
float reel;
char tampon[51];
…
printf(''entrez le message \n'');          // on tape :  «12  3.14»
scanf(''%50[^\n]'', tampon);
sscanf(tampon,''%d %f'',&entier,&reel);     //  entier vaut 12, reel vaut 3.14
```

Langage C : séance 6

La fonction « sprintf » :

- Permet de convertir en chaîne de caractères des données numériques.
- Syntaxe : sprintf(chaine résultat *, format , variables*);

  Recherche l'information dans le(s) dernier(s) argument(s), sous la forme du format, et le stocke dans la chaîne résultat.

- exemple :

```
int entier=10;
float reel=3.14;
char tampon[50];
…
sprintf(tampon,''%d %f'',entier,reel);
printf(''%s'', tampon);  // affiche la chaîne ''10 3.14''
```

- Privilégier l'usage de **snprintf :** ← **à privilégier**

```
snprintf(tampon,50, ''%d %f'',entier,reel);   // évite le débordement du tampon (50 max ici)
```

Langage C : séance 6

Fonctions standards de traitement sur les chaines (string.h) :

- fonction pour calculer la longueur : **strlen**

```
char ch[]="dupont";        // ajout de '\0' en mémoire
int i=strlen(ch);          //  i vaut 6
```

La chaîne passée en paramètre à **strlen** doit se terminer obligatoirement par '\0'.

- fonction d'affectation d'une chaîne : **strcpy**

```
char ch1[20], ch2[20];
strcpy(ch1, "dupont" );        // ch1 reçoit « dupont »
strcpy(ch2, ch1);
```

- fonction d'affectation restreinte : **strncpy** ← **à privilégier**

```
char ch1[20], ch2[20];
strncpy(ch1, "dupont",4 );          // ch1 reçoit « dupo »
strncpy(ch2, ch1,3);                // ch2 reçoit « dup »
```

attention : pas d'ajout de '\0' par strncpy !

Langage C : séance 6

Fonctions standards de traitement sur les chaines (string.h) :

- fonction de concaténation : **strcat**

```
char ch1[20], ch2 []="dupont";
strcpy(ch1, "bonjour ");
strcat(ch1,ch2);            //  ch1 vaut "bonjour dupont"
```

- fonction de concaténation restreinte : **strncat** ← à privilégier

```
char ch1[20], ch2 []="dupont";
strcpy(ch1, "bonjour ");
strncat(ch1,ch2, 3);        //  ch1 vaut "bonjour dup"
```

Cette fonction rajoute un '\0' à la fin.

- fonction de comparaison : **strcmp**

strcmp(ch1,ch2) // comparaison dans l'ordre alphanumérique !

➔ retourne 0 (zéro) si les deux chaînes sont égales.

➔ retourne une valeur négative si ch1 < ch2 (ex: "dupantel" < "dupont")

➔ retourne une valeur positive si ch1 > ch2 (ex: "dupont" > "dupantel")

- fonction de comparaison restreinte : **strncmp** ← à privilégier

strncmp(ch1,ch2,n) // comparaison des n premiers caractères

➔ idem que strcmp

Langage C : séance 6

Les paramètres du « main » :

•**La déclaration du main :**

La fonction « main » avec paramètres permet de récupérer des valeurs lors de l'exécution du programme (lancement en ligne). Ces valeurs sont récupérées dans des chaînes de caractères.

•**Syntaxe :**

Nombre d'expressions de la commande

Tableau contenant chacune des expressions

```
int main(int argc, char * argv[])
        {
        /* argc : nombre total d'expression
          argv : tableau de chaînes de caractères
          argv[0] : nom de la commande
          argv[1] : 1er paramètre, …    */
        …..
        }
```

Langage C : séance 6

Les paramètres du « main » :

- **Exemple :**

Fichier « toto.c » :

```
int main(int argc, char * argv[]) {
        int i;
        printf(''nombre total =%d\n'',argc);
        for (i=0;i<argc;i++)
                printf(''%s\n'',argv[i]);
        return 0;
}
```

Exécution dans un terminal :

```
prompt #   ./toto  bonjour 1 2 3
nombre total =5
toto
bonjour
1
2
3
```