les unions

Les unions se déclarent de la même manière que les structures. Elles possèdent donc elles aussi des champs typés. Mais on ne peut utiliser qu'un seul champ à la fois. En fait tous les champs d'une union se partagent le *même* espace mémoire. Les unions sont rarement nécessaires sauf lors de la programmation système.

- Syntaxe : voir poly p39

les énumérations

Les énumérations définissent un type par une liste de valeurs qu'il peut prendre.

- Syntaxe :

enum <nom>{<id_1>[,<id_i>]*}

<id_i> sont des identifiants respectant la syntaxe des identifiants ils ne peuvent être ni des valeurs numériques ni caractères ni chaîne. En réalité chaque <id_i> est numéroté à partir de zéro.

Exemples :


```c
#include <stdio.h>
int main(){
    enum vlrs {x3,y,b9cd,_xy_1};
    printf("%d\n",b9cd); // 2
    return 0;
}
```

On peut aussi choisir la numérotation :

```c
#include <stdio.h>
int main(){
    enum vlrs {a=23,bb=6,ccc=69,dddd=1};
    printf("%c,%d\n",ccc,ccc); // E,69
    return 0;
}
```


Attention :

```c
#include <stdio.h>
int main(){
   typedef enum  {a=23,bb=6,ccc=69,dddd=1}vlrs;
   vlrs x;
   x=bb; // ou x=6;
   printf("%d\n",x); // 6
   x=22; // il n'y a aucun contrôle
   printf("%d\n",x); // 22
   return 0;
}
```


Buffer clavier : Retour sur getchar()

```c
#include <ctype.h> //toupper
#include <stdlib.h> //EXIT_SUCCESS
#include <stdio.h>
#include <string.h> //strchr

unsigned long long facto(unsigned int); //  nom du paramètre formel facultatif

int main(){
  char rep;
  unsigned int nb;
  printf("Calcul de factorielle\n");
  do{
     printf("Entrez un entier : ");
     scanf("%d",&nb);
     printf("%llu\n",facto(nb));
     do{
        while (getchar()! = '\n'); // fflush(stdin);
        printf("un autre calcul ? (o/n)");
     }while (strchr("ON",rep = toupper(getchar())) == NULL) ;
     fflush(stdin) ; //while (getchar()!='\n');
  }while (rep == 'O') ;
  return EXIT_SUCCESS; }
```

```c
unsigned long long facto(unsigned int n){
    unsigned long long  r=1;
    unsigned int i ;
    for (i=1; i<=n; i++){
        r=r*i;
    }
    return r ;
}
```


fonctions avec un nombre variable de paramètres

Il est possible en C de définir des fonctions qui ont un nombre variable de paramètres. Cette fonctionnalité est indispensable dans certains cas, notamment pour les fonctions printf et scanf.

Une fonction possédant un nombre variable de paramètre doit posséder au moins un paramètre formel fixe. La notation ...(obligatoirement à la fin de la liste des paramètres d'une fonction) spécifie que la fonction possède un nombre quelconque de paramètres (éventuellement de types différents) en plus des paramètres formels fixes. Ainsi, une fonction ayant pour prototype

```c
int f(int a, char c, ...);
```

prend comme paramètre un entier, un caractère et un nombre quelconque d'autres paramètres. De même le prototype de la fonction printf est

```c
int printf(char *format, ...);
```

Exemple :

- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

```
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>  // macros : va_list , va_start , va_end

int add(int,...);

int main(void) {
    printf("\n %d", add(4,10,2,8,5));   // le premier paramètre
    printf("\n %d\n", add(6,10,15,5,2,8,10)); // indique le nb de val à
    return(EXIT_SUCCESS);                       // additionner
}
int add(int nb,...) {
    int res = 0, i;
    va_list liste_parametres;
    va_start(liste_parametres, nb);
    for (i = 0; i < nb; i++)
          res += va_arg(liste_parametres, int); // il faut connaître le
    va_end(liste_parametres);                   // type du paramètre
    return(res);    }
```


- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

les paramètres constants

```
#include <stdio.h>
#include <stdlib.h>
int f(int);
int g(const int);

int main(){
   printf("%d\n",f(5));
   printf("%d\n",g(6));
   return  EXIT_SUCCESS;
}
```

```
int f(int x){
   x*=x;
   return x+1;
}
int g(const int x){
   x*=x;
   return x+1;
}
```

warning: assignment of read-only location

- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

les Variables statiques

Voir poly p 62 à 64

```
int n = 10;
void fonction();

main() {
  int i;
  for (i = 0; i < 5; i++)
    fonction();
}

void fonction() {
  static int n;
  n++;
  printf("appel numero %d\n",n);
  return;
}
```


- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

les pointeurs de fonctions

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

int  f(int);
int  g(int);

int main(){
    int (*p)(int) ;
    p = f;
    printf("%d\n",p(5)); // 11
    p = g;
    printf("%d\n",p(5)); // 24
 return EXIT_SUCCESS;
}
```

```
int f( int n){
    return 2*n+1;
}
int g( int n){
    return n*n-1;
}
```


- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

Utilisation des pointeurs de fonctions

On l'utilise habituellement comme paramètre de fonction:

```
#include <stdlib.h> //EXIT_SUCCESS
#include <stdio.h>
int  f(int);
int  g(int);
int  h(int,int(*)(int));
```

```
int f (n) {return 2*n+1;}
int g (m) return m*m-1;
/* les {} sont obligatoires
en ansi, de même les //
sont interdits en ansi */
```

```
int main(){
    printf("%d\n",h(5,f));
    printf("%d\n",h(5,g));
    return EXIT_SUCCESS; }
int h(int x, int (*k)(int)) {return k(x);}
```


- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

Les tris (rappel) : On défini 3 grands types de tri

- Insertion
  - Soit le tableau $[t_0,\ldots,t_i,t_{i+1},\ldots t_n]$supposé trié entre 0 et i
  - l'étape suivante consiste à insérer t_{i+1} à sa place entre $t_0,\ldots,t_i$
- Sélection
  - Soit le tableau $[t_0,\ldots,t_i,t_{i+1},\ldots t_n]$supposé trié entre 1 et i et tels que $t_j \leq t_k \ \forall\ 0 \leq j \leq i$ et $i+1 \leq k \leq n$
  - l'étape suivante consiste à rechercher le plus petit élément entre i+1 et n et à le permuter avec t_{i+1}
- Fusion

Les tris (rappel)

- Fusion : c'est une méthode essentiellement récursive :
  - On coupe le tableau en 2
  - on trie chaque moitié (grâce au tri fusion : récursivité)
  - On fusionne les 2 parties en intercalant les élément de l'un à leur place entre les éléments de l'autre.
- Ce tri est en $n.\log_2(n)$ donc très efficace mais l'utilisation de tableaux intermédiaire et de recopies lui font perdre son efficacité

Les tris (rappel)

- Le quicksort fonctionne un peu sous le principe du tri fusion :
  - On sélectionne un élément du tableau (que l'on appelle pivot)
  - On le positionne dans le tableau de telle sorte que tous les éléments plus petit que lui sont à sa gauche et les plus grands à sa droite.
  - On recommence en appliquant le quicksort aux 2 parties du tableau.
- L'efficacité du tri dépend de la sélection du pivot : elle sera maximale lorsque la place du pivot (rangé) sera en plein milieu du tableau.

Tri = comparaison

- Trier un tableau consiste à définir une fonction de comparaison entre 2 éléments du tableau afin de comparer les éléments 2 à 2 pour pouvoir mettre le plus « petit » avant le plus « grand »
- C'est cette fonction de comparaison qui sera passé en argument à la fonction de tri afin de pouvoir ordonner le tableau selon divers critères : croissant, décroissant etc.

Application :

Le type T_etudiant est une structure comportant :

Un nom , un prénom, un nip, et 6 notes :

BDD, CLA, COE , COO, EEO, PDC

à 9 champs dont on précisera le type.

On souhaite classer les étudiants selon 9 critères : nom, nip et ordres de mérite dans chaque matière ainsi que selon la moyenne générale (on supposera que toutes les notes ont le même poids). On définira 9 fonction d'ordre. On utilisera par exemple la méthode de tri par insertion.

Dans un deuxième temps, on remplacera la fonction de tri personnelle par celle qui est définie dans stdlib.h

```
void qsort(void *base, size_t nel, size_t width ,
int (*compar)(const void *, const void *));
```

Exemple d'utilisation :

```
int main(){
   int i , tb[]={7,5,9,2,8,3,4,7,2,6,1,5};
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int),croissant);
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int),decroissant);
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   return EXIT_SUCCESS;
}
```


Les fonctions de comparaison définies ainsi :

```
#include <stdlib.h>
#include <stdio.h>
int croissant(int *, int *);
int decroissant(int * , int *);
int main(){…}
int croissant(int *a, int *b){    return *a-*b;  }
int decroissant(int *a, int *b){    return *b-*a; }
```

warning: passing arg 4 of `qsort' from incompatible pointer type car le 4ème argument de qsort de type

```
int (*compar)(const void *, const void *)
```

- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

On utilise donc un cast :

```
#include <stdlib.h>
#include <stdio.h>
int croissant(int *, int *);
int decroissant(int * , int *);
typedef int (*compar_t)(const void *, const void *);
int main(){
   int i , tb[]={7,5,9,2,8,3,4,7,2,6,1,5};
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int),(compar_t)croissant);
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int), (compar_t)decroissant);
    for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   return EXIT_SUCCESS;
}

int croissant(int *a, int *b){    return *a-*b;  }
int decroissant(int *a, int *b){  return *b-*a;  }
```


- Unions
- Enumérations
- Buffer Clavier
- Fonctions : nombre variable de paramètres
- Paramètres constants
- Variables statiques
- Pointeurs de fonctions
- Tris: Rappel
- Application
- stdlib : qsort

Autre façon de caster :

```
#include <stdlib.h>
#include <stdio.h>
int croissant(const void *,const void  *);
int decroissant(const void * ,const void *);
int main(){
   int i , tb[]={7,5,9,2,8,3,4,7,2,6,1,5};
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int),croissant);
   for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   qsort(tb,12,sizeof(int), decroissant);
    for(i=0;i<12;i++) printf("%d,",tb[i]);printf("\n");
   return EXIT_SUCCESS;
}
int croissant(const void *a,const void *b) {
   return (*(int *) a) - (*(int *) b) ;}
int decroissant(const void *a,const void  *b) {
   return (*(int *) b) - (*(int *) a) ;}
```