C

Wieslaw Zielonka
zielonka@irif.fr

exo 1

```
int tab[]={5,9,34,77,-33};

int len = ??? ; /* len : le nombre d'elements de tab*/
int tabinv[ len ];
int i,j;
for(  ???  ;   ???  ; ??? ){
      tabinv[i]=tab[j];
}
```

exo 1

```
int tab[]={5,9,34,77,-33};

int len = sizeof(tab)/sizeof(tab[0]);
int tabinv[ len ];
int i,j;
for( i=0, j=len-1  ;  j>= 0  ; i++, j-- ){
      tabinv[i]=tab[j];
}
```

Une autre possibilité :

int len = sizeof (tab) / sizeof(int);

Rappel : sizeof(tab) -- le nombre d'octets qu'occupe le vecteur tab

exo 2

```c
#include <stdio.h>

struct toto{

   int x;

    double t[2];

};

struct toto fun( struct toto s ){

  s.x +=1;

  s.t[0] += s.t[1];

  s.t[1] *= 2;

  return s;

}

int main(void){

  struct toto s, u;

  s.x = 20;

  s.t[0]=30.0;

  s.t[1]=40.0;
```

```c
u = fun(s);

 printf( "%d\n", s.x);  -> 20

 printf( "%f\n", s.t[0]);  -> 30

 printf( "%d\n", s.t[1]);  -> 40

 printf( "%d\n", u.x);  ->21

 printf( "%f\n", u.t[0]);  ->70

  printf( "%d\n", u.t[1]);   -> 80

  return 0;

}
```

Le paramètre **s** sert à initialiser la variable locale s de la fonction.
Donc **fun(s)** ne peut pas changer la valeur de **s.**

exo 8

```
int main(void){
  double t[]={0.1, -1.5, 2.2, -3.3, 4.4, -5.5,
              6.6, -7.7, 8.8, -9.9, 10.0, -11.1 };
  double *pa = &t[2];
  double *pb = &t[8];
  ptrdiff_t d = pa - pb;   la différence de pointeurs int * == le nombre d'objet int entre
                           les deux adresses  :  ici 2-8 == -6

  printf("%td\n" ,  d);
  pa++;                    incrémenter pa pour que pa contienne l'adresse de int suivant.
  printf("%f\n", *pa);   affiche t[3]  donc -3.3
  (*pb)++;                 incrémenter int pointé par pb, ici même chose que t[8]++
  printf("%f\n", *pb);   affiche 9.8

  double *v = &t[6];     v-3 l'adresse de int trois places avant t[6], donc v-3 l'adresse de t[3]
  printf("%f\n" , *(v-3) );  affiche la valeur int pointé, donc t[3] == 3.3

  v[-2]=101;             même chose que   *(v-2) = 101, v-2 l'adresse de t[4]
  printf("%f\n" , t[4] );  affiche t[4] donc 101
}
```

exo 3

```
void g( int n, int t[], int v[]){
  printf("%zu\n", sizeof( t ) );

  for(int i = 0; i < n; i++){
    int a = t[i]; t[i]=v[i] ; v[i] = a;
  }
  return;
}
int main(void){
  int p[]={1,2,3,4,5,6};
  printf("%zu\n", sizeof( p ) );
  int q[]={11,12,13,14,15,16};
  g(6,p,q);
  printf("%d\n", p[2] );
  printf("%d\n", q[3] );
  return 0;
}
```

t et v ne sont pas de vecteur !
le compilateur traduit cela en
void g(int n, int *t, int *v)
donc t est variable de type pointeur initialisé avec l'adresse du premier élément du vecteur **p**
sizeof(t) == sizeof(int *) ==
le nombre d'octet pour stocker une adresse

affiche la longueur de p mesurée en nombre d'octets **: 6*sizeof(int)**
donc l'affichage dépend de **sizeof(int)**

Toutes les réponses genre : **à l'appel g(6,p,q) le paramètre t devient une copie de vecteur p sont complètement fausses.**
p est un vecteur (pas une variable), tandis que **t** est une variable de type pointeur locale à lafonction g()

exo 9

```
int somme(int n, int t[]){
    int s = 0;
    for(int i = 0; i < n ; i++)
        s += t[i];
    return s;
}
int main(void){
   int t[] =
{9,8,6,9,4,6,4,5,3,6,-4,-6};
   int u = somme( ??? , ???);
    /* d'autres instructions */
}
```

int somme(int n, int *t)

dans u la somme de t[4],t[5],t[6], t[7]

int u = somme(4, &t[4]);

calculer la somme de 4 élément à partir de int à l'adresse &t[4]

une autre possibilité : **int u = somme(4, t+4);**

&t[4] - l'adresse de 4ème éléments de vecteur t

t + 4 - prendre l'adresse de début vecteur t et se déplacer 4 int plus loin dans la mémoire donc c'est la même chose que &t[4]

exo 10

```c
typedef struct{
  int degre;
  double coef;
}monome;

#define LEN 8
typedef struct{
  unsigned int n;
  monome p[LEN];
}polynome;

polynome sump(polynome u,
polynome v);
```

```c
int main(void){
  polynome u = { .n = 5, .p = {
      {.degre =1, .coef = 3.5 },
      {.degre =3, .coef = -4 },
      {.degre =5, .coef = 11.9 },
      {.degre =7, .coef = 103.5 },
      {.degre =9, .coef = -123.0 }
    }
  };

  polynome v = { .n = 6, .p ={
      {.degre =2, .coef = -5.5 },
      {.degre =3, .coef = 4 },
      {.degre =4, .coef = -121.9 },
      {.degre =5, .coef = 12.0 },
      {.degre =8, .coef = 121.0 },
      {.degre =9, .coef = 10.0 }
    }
  };
  polynome w = sump(u,v);
  afficher( w);
};
```

exo 10

```
polynome sump(polynome u, polynome v){
  size_t i=0,j=0,k=0;

  double s;
  polynome w;
  while( i < u.n && j < v.n && k < LEN ){
    if( u.p[i].degre == v.p[j].degre ){
      s = u.p[i].coef+v.p[j].coef;
      if( s != 0 ){
    w.p[k].degre = u.p[i].degre;
    w.p[k].coef  = s;
      }
      i++;
      j++;
    }else if( u.p[i].degre < v.p[j].degre ){
      w.p[k].degre = u.p[i].degre;
      s = w.p[k].coef  = u.p[i].coef;
      i++;
    }else{
      w.p[k].degre = v.p[j].degre;
      s = w.p[k].coef  = v.p[j].coef;
      j++;
    }
    if( s != 0 )
      k++;
  }
```

```
if( i >= u.n ){
  int m = k + (v.n - j);
  if( m >= LEN ){
      w.n = -1;
      return w;
  }
  /* copier les monômes restantes
de v vers w*/
   memmove( &w.p[k],
            &v.p[j],
       ( v.n-j ) * sizeof( monome ));
  w.n = m;
  return w;
}
```

m == le nombre total de monômes dans le résultat

il faut aussi un if symétrique :

u -> v

i -> j

j ->i

exo 10

```
/* le dernier cas, plus de place dans w :
    k >= LEN */
w.n = -1;

if( u.n - i != v.n - j)
  return w;

for( ; i < u.n ; i++, j++){
  if( u.p[i].degre != v.p[j].degre )
    return w;
  if( u.p[i].coef + v.p[j].coef != 0 )
    return w;
  }
  w.n = LEN;
  return w;
}/* fin sump */


void afficher(polynome u){
  for(int i = 0 ; i < u.n; i++){
    printf( " %+4.1f x^%d ", u.p[i].coef, u.p[i].degre );
  }
  printf("\n");    coef affiché sur 4 chiffres dont une après
                   virgule : %4.1f
                   %+4.1f  même chose mais avec le signe même pour
                   les nombres positifs
}
```