Langage C
Partiel du 13 février 2020, durée 2h

Documents autorisés et consignes

Tous les documents manuscrits sont autorisés. Livres interdits.
Votre code doit être écrit de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin de blocs de code (fin de boucles, etc.). Il est inutile d'écrire les `#include`.

1 Questions

Dans certains exercices on présente un fragment de code contenant des `printf`. Chaque `printf` affiche la valeur du deuxième paramètre. Tous les `printf` sont corrects donc si on vous demande si le code est correct la question ne concerne pas les `printf`. Quand dans une question on demande ce qu'un `printf` affiche il suffit juste écrire la valeur affichée, le détails d'affichage, comme par exemple le nombre d'espaces, le nombre de chiffre après le virgule, sont sans importance, c'est uniquement la valeur affichée qui compte.

Exercice 1 :

On considère le fragment de code suivant :

```
1    int tab[]={5,9,34,77,-33};
2
3    int len = ??? ;
4
5    int tabinv[ len ];
6
7    int i,j;
8    for(  ???  ;   ???  ; ??? ){
9      tabinv[i]=tab[j];
10   }
11
```

Vous devez remplacer ? ? ? par votre code, d'autre modifications sont interdites.
Dans la ligne 3 remplacer ? ? ? par une expression qui donne le nombre d'élément de `tab`. La réponse `int len = 5;` n'est pas appropriée, vous devez écrire l'expression qui ne changera pas même quand on décide d'ajouter d'autres éléments dans `tab` en modifiant la ligne 1.
Dans la ligne 6 modifier la boucle for en remplaçant les ? ? ? par un code approprié de telle sorte qu'à la sortie de la boucle le vecteur `tabinv` contient les élément de `tab` dans l'ordre inverse. Par exemple si `tab` contient les élements comme dans la ligne 1 alors `tabinv` prendra les valeurs $-33, 77, 34, 9, 5$. Bien sûr le changement de `tab` dans la ligne 1 ne doit pas provoquer de changement dans la ligne 8, c'est-à-dire `tabinv` doit toujours sortir de la boucle avec tous les éléments de `tab` dans l'ordre inverse.

Exercice 2 :

On considère le programme suivant.

```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;

  u = fun(s);

  printf( "%d\n", s.x);
  printf( "%f\n", s.t[0]);
  printf( "%d\n", s.t[1]);
  printf( "%d\n", u.x);
  printf( "%f\n", u.t[0]);
  printf( "%d\n", u.t[1]);
  return 0;
}

```

Est-ce que ce programme est correct ? (Correct dans le sens qu'il compile et s'exécute non pas qu'il soit intéressant.) Si ce n'est pas le cas expliquer succinctement ce qui pose un problème. Qu'est-ce qu'affiche chaque `printf` (soyez précis en indiquant chaque fois le numéro de la ligne avec le printf correspondant).

Exercice 3 :

```c
#include <stdio.h>
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};
```

```
12      printf("%zu\n", sizeof( p ) );
13      int q[]={11,12,13,14,15,16};
14      g(6,p,q);
15      printf("%d\n", p[2] );
16      printf("%d\n", q[3] );
17      return 0;
18    }
19
```

Est-ce que les `printf` dans les lignes 3 et 12 affichent la même valeur ?
En fait quelle(s) information(s) affichent les deux `printf` ? Il ne s'agit pas de donner les valeurs affichées mais plutôt de dire que le `printf` affiche la taille de `???` mesurée en `???`.
Quelles valeurs affichent les printfs dans les lignes 15 et 16 ?

Exercice 4 :

On définit :

```
1   enum sex {HOMME, FEMME};
2   struct personne{
3     unsigned int numero;
4     enum sex gender;
5     unsigned int age;
6   };
7
```

Définir une variable `x` de type `struct personne` et l'initialiser *au moment de la déclaration* (déclaration et initialisation dans la même inscruction) de façon à ce que les champs `age`, `numero` et `gender` prennent respectivement les valeurs 33, 1054 et `FEMME`.

Exercice 5 :

On assule `struct personne` de l'exercice précédent. Comment déclarer un nouveau type `personne` pour qu'on puisse déclarer une variable

```
1   personne a;
2
```

sans `struct` devant ?

Exercice 6 :

Est-ce que le code suivant est correct :

```
1   int[10] fun(int len){
2     int t[10];
3     for(int i = 0; i<10;i++){
4       t[i]=i*i;
5     }
6     return t;
7   }
8
9   int main(void){
```

```
      int[] tab = fun();
      /* d'autres instructions */
      return 0;
    }

```

Si vous pensez que le code contient des erreurs expliquez (très brièvement, une phrase suffit) quel est le problème.

Exercice 7 :

Dans le fragment de code suivant

```
    int x;
    if( f(1) >=0 )
       x = g( 1 );
    else
       x = h( 1 );

```

`f, g, h` sont des fonctions qui prennent un argument int et retournent un int.
Remplacer ? ? ? dans l'instruction

```
     int x = ???   ;

```

pour obtenir le même résultat que dans le fragment précédent, c'est-à-dire `x` prend soit la valeur `g(1)` soit `h(1)` en fonction de signe de `f(1)`.

Exercice 8 :

Dans le programme suivant indiquer ce qu'affiche chaque `printf`.

```
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;
  printf("%td\n" ,  d);

  pa++;
  printf("%f\n", *pa);

  (*pb)++;
  printf("%f\n", *pb);

  double *v = &t[6];
  printf("%f\n" , *(v-3) );

  v[-2]=101;
  printf("%f\n" , t[4] );
}
```

Exercice 9 :

On considère le programme suivant :

```
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 */
}

```

Remplacer ? ? ? dans la ligne par votre code de telle sorte que `u` reçoive la somme de `t[4], t[5], t[6], t[7]`.

2 Programme

Exercice 10 :

On définit

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

#define LEN 8

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

```

Un élément de type `monome` représente un monôme dans le sens d'algèbre. Par exemple la structure `monome m` avec `m.degre = 5` et `m.coef = 18.6` représente un monôme $18.6x^5$.
Un polynôme est représenté par la structure `polynome` composée d'un entier `n` et d'un vecteur de monômes `p`. Puisque le nombre d'éléments dans `p` est limité par `LEN` les polynômes que nous construisons peuvent avoir au plus `LEN` monômes. Le champ `n` donne le nombre de monômes dans le polynôme, c'est-à-dire uniquement le monômes `p[0],...,p[n-1]` constituent le polynôme, d'autres éléments du tableau `p` ne sont pas utilisé.
Par exemple, supposons que `n==4` et le vecteur `p` contient :

0	3	8	9				
9.2	-1.9	9	-2.8				

Dans ce tableau la rangée en haut donne les degrés de monômes et la rangé en bas les coefficients, seulement le 4 premiers éléments sont montré puisque `n==4`.

Cet exemple correspond au polynôme $9.2x^0 - 1.9x^3 + 9x^8 - 2.8x^9$.
On suppose que notre implementation de polynômes doit satisfaire deux conditions :

(1) On garde toujours uniquement les monômes avec le coefficients différents de 0, c'est-à-dire `t[i]<>0` pour $i < n$.

Notons que cela implique que le polynôme 0 qui a tous les coefficients 0 est représenté par la structure avec `n==0`.

(2) Les monômes sont stockés dans l'ordre croissant de degrés.

Écrire la fonction

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

qui calcule et retourne la somme de polynômes `u` et `v`.
Rappelons que la somme de deux polynômes s'obtient en prenant la somme de coefficients de monômes de même degré. Par exemple la somme de $u(x) = 2x^0 + 6x^2 - 5x^3 + 2x^5$ et $v(x) = 3x^1 - 6x^2 + 2x^3 + 11x^6$ est le polunôme $2x^0 + 3x^1 - 3x^3 + 2x^5 + 11x^6$. Notons que le monôme x^2 disparaît dans cette somme parce que son coefficient est 0. La somme de deux polynômes peut avoir trop de monômes non nuls (plus que `LEN`), dans ce cas vous mettrez -1 dans le champ `n` du résultat.