CORRIGES

Cahier de TD n° 5

Cahier de TD n° 5

THÈME 1 : VRAI/FAUX, DÉFINITION DE STRUCTURES

VRAI/FAUX

MANIPULATIONS DE BASE : LE '.' ET LA '→'MANIPULATIONS DE BASE : LE '.' ET LA '→'

THÈME 2 : FONCTIONS ET STRUCTURES

NOMBRES COMPLEXES Z = REEL + I IMAGINAIRE

Thème 1 : Vrai/faux, définition de structures

VRAI/FAUX

- Une structure est un type **VRAI**
- Une structure peut être un type défini par l'utilisateur **VRAI** (mot-clé **typedef)**
- Une structure a au moins un champ de type entier **FAUX**
- Un champ est une structure **FAUX**
- Un champ peut être de type structure **VRAI**
- Un champ peut comporter des structures **VRAI** si le champ est un tableau de structures
- Une structure comporte un ou des champs **VRAI**
- Un champ est défini uniquement par son type **FAUX**
- Un champ est défini uniquement par son nom **FAUX**
- Un champ est défini par son type et son nom **VRAI**
- Un champ peut être de n'importe quel type **VRAI**
- On ne peut pas définir de variable dont le type est une structure **FAUX**
- L'accès à un champ d'une structure se fait par la syntaxe : '*' **FAUX**
- L'accès à un champ d'une structure se fait par la syntaxe : '.' **VRAI**
- On peut définir un pointeur sur une structure **VRAI**

Manipulations de base : le '.' et la '->'

struct Personne {
char nom[128];
int age;
…
};
	struct Personne gilbert; // variable de type struct Personne
	struct Personne * p; // pointeur vers une structure de type struct Personne
	p = &gilbert; // ==> *p vaut gilbert
	gilbert.age = 20;
	(*p).age = 20;
	p->age = 21; // Gilbert a vieilli ...

choisissez, pour les exemples suivants :

les types et les noms des champs pour les objets listés, et écrivez la définition de la structure associée (pas plus de 5/6 champs par structure) :

- une adresse postale

```
struct AdressePostale {
        int numero;
        char rue[128]; // rue, avenue, boulevard …
        char ville[128]; // ville, lieu-dit …
        char pays[128];
};
```

- un CD musical ***(à faire)***
- un client d'une banque

```
struct ClientBancaire {
        char nom[128];
        char prenom[128];
        struct AdressePostale adrPost;
        struct DateNaissance dn;    // en supposant défini le type struct DateNaissance
        struct Compte compte;    // en supposant défini le type struct Compte
};
```

- un chien ***(à faire)***
- une émission télévisée ***(à faire)***
- un joueur de football

```
struct FootBalleur {
        char nom[128];
        char prenom[128];
        int age;
        struct AdressePostale adrPost;
        long salaire; // du lourd, un int ne suffit pas toujours...
        char club[128];
        char nationalite[64];
};
```

- une équipe de football

```
struct EquipeFootball {
        char nom[64];
        struct FootBalleur joueurs[30]; // un tableau de 30 variables de type struct FootBalleur
        FootBalleur capitaine;
};
```

Dans les exemples de programme suivants, indiquez quelles écritures sont correctes syntaxiquement.

```
typedef struct ordi
{
        long capa;          // capacité de la mémoire
        long freq;          // fréquence du proco
        char marque[30]; // marque du PC et du processeur
        long hd; // capacité du disque dur
} t_ordi;

int main()
{
        t_ordi mon_pc;
        long hd;

        cout << mon_pc << endl; // OK si l'opérateur << est surchargé
        t_ordi.capa = 512;
        // mon_pc.marque = "DELL AMD Athlon 2200+"; FAUX
        // impossible de modifier le pointeur constant pc_marque
        strcpy( mon_pc.marque, "DELL AMD Athlon 2200+");
        OU MIEUX
        strncpy( mon_pc.marque, "DELL AMD Athlon 2200+", 30);
        strncpy(mon_pc.marque,"DELL AMD Athlon 2200+",
                                        sizeof(mon_pc.marque)/sizeof(mon_pc.marque[0]));
        mon_pc.freq = 1800;
        hd = 120;
        mon_pc.hd = hd;
        return 0;
}
```

Utilisation du man

La fonction strcpy() copie la chaîne pointée par *src* (y compris l'octet nul « \0 » final) dans la chaîne pointée par *dest*. Les deux chaînes ne doivent pas se chevaucher. La chaîne*dest* doit être assez grande pour accueillir la copie.

La fonction **strncpy**() est identique, sauf que seuls les *n* premiers octets de *src* sont copiés. **Avertissement :** s'il n'y a pas d'octet nul dans les *n* premiers octets de *src*, la chaîne résultante dans *dest* ne disposera pas d'octet nul final.

Dans le cas où la longueur de *src* est inférieure à *n*, la fin de *dest* sera remplie avec des octets nuls.

Une implémentation simple de **strncpy**() pourrait être :

```
char *
strncpy(char * dest, const char * src, size_t n)
{
   size_t i;
   for (i = 0 ; i < n && src[i] !='\0' ; i++)
      dest[i] = src[i];
   for ( ; i < n ; i++)
      dest[i] = '\0';
   return dest;
```

}

Thème 2 : Fonctions et structures

Les fonctions sont très souvent associées aux structures, pour la simple et bonne raison qu'une structure ne se manipule pas comme une variable de type classique, et il faut donc prévoir des fonctions pour toutes les opérations classiques que l'on doit réaliser.

Nombres complexes z = reel + i imaginaire

On veut définir un nouveau type pour représenter et manipuler des nombres complexes.

1) Définir le type Complexe qui permette de représenter un nombre complexe comme une structure ayant deux champs de type doublenommés reelet imaginaire.

Les fonctions demandées ci-après doivent avoir en paramètre un pointeur sur une structure de type Complexe.

2) Définir une fonction d'initialisation d'un complexe **initialiserComplexe** (z = (0, 0)).
3) Définir une fonction de saisie au clavier d'un complexe **remplirComplexe**.
4) Définir une fonction d'affichage d'un complexe **afficherComplexe**.
5) Définir une fonction **cmpComplexe**qui compare deux complexes et renvoie 0 s'ils sont égaux et −1 sinon.
6) Définir une fonction **sommeComplexe** qui effectue la somme de deux complexes.
7) Définir une fonction **produitComplexe**qui effectue le produit de deux complexes.
8) Tester les fonctions dans la fonction principale de votre programme.
9) Définir des fonctions d'initialisation, de saisie, d'affichage pour un tableau de 10 complexes alloués statiquement.

```
#include <iostream>
#include <float.h> // pour DBL_EPSILON
#include <math.h>

using namespace std;

typedef struct {
   double reel;
   double imaginaire;
} Complexe;

void initialiserComplexe(Complexe * z);
void remplirComplexe(Complexe * z);
void afficherComplexe(const Complexe * z);
int cmpComplexe(const Complexe * z1, const Complexe * z2);
void sommeComplexe(const Complexe * z1, const Complexe * z2, Complexe * z3);
void produitComplexe(const Complexe * z1, const Complexe * z2, Complexe * z3);

void initialiserTabComplexes(Complexe * complexes, int taille);
void remplirTabComplexes(Complexe * complexes, int taille);
void afficherTabComplexes( const Complexe * complexes, int taille);

int main()
{
        Complexe z;
        Complexe z1, z2, z3, z4;
        Complexe complexes[10];

        cout << "Nombre complexe z" << endl;
        initialiserComplexe(&z);
        afficherComplexe(&z);
        remplirComplexe(&z);
        afficherComplexe(&z);

   z1.reel = 3;
   z1.imaginaire = 6;
   z2.reel = 2;
   z2.imaginaire = 4;
   z3.reel = 3;
   z3.imaginaire = 6;
   cout << "egalite z1 et z2 " << cmpComplexe(&z1, &z2) << endl;
   cout << "egalite z1 et z3 " << cmpComplexe(&z1, &z3) << endl;

   cout << " somme de z1 et z2 : " << endl;
   sommeComplexe(&z1, &z2, &z4);
        afficherComplexe(&z4);

   cout << " produit de z1 et z2 : " << endl;
   produitComplexe(&z1, &z2, &z4);
   afficherComplexe(&z4);


        cout << "\n\n INITIALISER  TABLEAU DE COMPLEXES" << endl;
        initialiserTabComplexes(complexes, sizeof(complexes)/sizeof(complexes[0]));

        cout << "\n\n AFICHER TABLEAU DE COMPLEXES" << endl;
        afficherTabComplexes(complexes,  sizeof(complexes)/sizeof(complexes[0]));

        cout << "\n\n REMPLIR UN TABLEAU DE COMPLEXES" << endl;
        remplirTabComplexes(complexes,  sizeof(complexes)/sizeof(complexes[0]));
```

```
        cout << "\n\n AFICHER TABLEAU DE COMPLEXES" << endl;
        afficherTabComplexes(complexes,  sizeof(complexes)/sizeof(complexes[0]));

        return 0;
}


void initialiserComplexe(Complexe * z)
{
   z->reel = 0;
   z->imaginaire = 0;
}

void remplirComplexe(Complexe * z)
{
   cout << "partie reelle: "; cin >> z->reel;
   cout << "partie imaginaire: "; cin >> z->imaginaire;
}

void afficherComplexe(const Complexe * z)
{
   cout << "(" << z->reel << ", " << z->imaginaire << ")" << endl;
}

int cmpComplexe( const Complexe * z1, const Complexe * z2)
{
   if (fabs(z1->reel - z2->reel) < DBL_EPSILON  && (fabs(z2->imaginaire - z2->imaginaire)
                                                                    < DBL_EPSILON))
      return 0;
   return -1;
}

void sommeComplexe( const Complexe * z1, const Complexe * z2, Complexe * z3)
{
   z3->reel =  z1->reel +  z2->reel;
   z3->imaginaire = z1->imaginaire + z2->imaginaire;
}

void produitComplexe( const Complexe * z1, const Complexe * z2, Complexe * z3)
{
   z3->reel =  z1->reel *  z2->reel - z1->imaginaire * z2->imaginaire;
   z3->imaginaire = z1->reel * z2->imaginaire + z2->reel * z1->imaginaire;
}

void initialiserTabComplexes(Complexe * complexes, int taille)
{
    int i;
   for (i = 0; i <taille; i++)
   {
       remplirComplexe(&complexes[i]);
   }
}

void remplirTabComplexes(Complexe * complexes, int taille)
{
    int i;
   for (i = 0; i < taille; i++)
   {
       remplirComplexe(&complexes[i]);
```

```
    }
}

void afficherTabComplexes(const Complexe * complexes, int taille)
{
    int i;
   for (i = 0; i < taille; i++)
    {
        afficherComplexe(&complexes[i]);
    }
}
```