TRAVAUX DIRIGÉS 2

Ce TD porte sur le chapitre « Les Bases du C++ », section « Utilisation avancée ».

Exercice 1:

Quelles sont les instructions suivantes qui ne compileront pas, on supposera que chaque ligne est exécutée indépendemment, exception faite de la déclaration en ligne 1, réutilisée si nécessaire. Justifier vos choix.

```
1   int   a   = 5;
2
3   int   b* = 5;
4   int  *b   = 5;
5   int* b    = 5;
6   int  * b  = 0;
7   int   b* = 0;
8
9  &int  c   = 5;
10  int& c    = 5;
11  int   c   = &5;
12  int   c   = &a;
13
14  int* c    = &a;
15  int  *c   = &a;
16  int   c*  = &a;
17  int* c    = a&;
18  int   c*  = a&;
```

Exercice 2:

(a) Donner le type, la dimension et les valeurs du contenu des variables définies par les expressions suivantes :

```
1   int      a[2]   = {1,2};
2   int*     b      = a;
3   char     s[2]   = {'o', 'n'};
4   char     t[]    = "Coucou";
5   float    y[]    = {};
6   float    y2[0]  = {};
```

(b) Les deux dernières instructions sont-elles valables ? Conseillées ?

Exercice 3:

Soit le code suivant :

```
1   double z[3] = {3.145, 2.58, 33.33, 42.00, 1e5};
```

(a) D'après la déclaration, indiquer la dimension et le type de la variable z.

(b) — L'instruction va-t-elle compiler ? Pourquoi ?

— Corriger en respectant la déclaration de z.
Si des valeurs doivent être ajoutées, elles le seront à la fin de la liste ; de même les valeurs supprimées seront d'abord les dernières.

(c) Quel est le type du contenu du tableau ?

(d) En partant du principe qu'un float est codé sur 32bits, donner l'adresse des 5 valeurs de la liste d'initialisation (en écriture décimale, pour simplifier la première adresse sera choisie à 0).

(e) Soit la boucle suivante exécutée à la suite de la ligne 1 corrigée précédemment en (b) :

```
1   for(int i=10; i < 20; ++i)
2     z[i] = 1234;
```

Cette boucle for est-elle autorisée (le compilateur va-t-il l'accepter sans erreur ou alerte) ?

— Si oui, expliquer ce qui est écrit en mémoire et où.

— Si non, donner la raison de l'erreur ou l'alerte générée par le compilateur.

Dans les deux cas, indiquer les valeurs contenues dans le tableau z.

Exercice 4:

Exprimer les types des expressions suivantes :

```
1  int i;
2  int *p;
3  int a[];
4  int f();
5  int **pp;
6  int (*pa)[];
7  int (*pf)();
8  int *ap[];
9  int aa[][];
10 int *fp();
```

Exercice 5:

Même question !

```
1  int ***ppp;
2  int (**ppa)[];
3  int (**ppf)();
4  int *(*pap)[];
5  int (*paa)[][];
6  int *(*pfp)();
7  int **app[];
8  int (*apa[])[];
9  int (*apf[])();
10 int *aap[][];
11 int aaa[][][];
12 int **fpp();
13 int (*fpa())[];
14 int (*fpf())();
```

Exercice 6:

Soit une fonction de prototype incomplet suivant

```
1   int traitement(...);
```

(a) Compléter la liste des arguments pour un passage :

— d'une variable "i" de type entier, par valeur

- d'une variable "a" de type entier, par référence
- d'une variable "a" de type entier, par référence et un entier "b" ayant 33 pour valeur par défaut
- d'un pointeur d'entier "pi" et d'un tableau de flottants simple précision "tf"
- d'un entier "b" par référence et d'un tableau de flottants double précision constants "tz"
- d'un entier "c" par valeur, une chaîne de caractères "chaine" et un flottant simple précision "z" valant 5 par défaut.
- d'une chaîne de caractères constants "chaine2", que peut-on aussi passer ?
- d'un pointeur constant vers un tableau d'entier "ti"

(b) Considérons une fonction principale qui ne contient que l'appel d'une des variantes de cette fonction ainsi que la déclaration des paramètres adéquates.
Tous ces prototypes peuvent-ils apparaître ensemble dans le fichier ? Si non, quels sont les prototypes incompatibles, pour quel appel, pourquoi ? Comment nomme-t-on ce concept, donner sa définition complète (utilité, limites, portée,...)

Exercice 7:

Proposer un code qui :

(a)
- Crée un tableau de 10 entiers
- Demande à l'utilisateur la saisie de chacune des valeurs
- Calcule et affiche les valeurs extrêmes

(b) On souhaite que les valeurs soient strictement positives.
- Proposer une boucle de saisie et modifier le type du tableau pour un type plus approprié.
- Que se passerait-il si on saisissait une valeur négative.

Exercice 8:

(a) Rappeler l'utilité de l'opérateur "sizeof()" : Que retourne-t-il, dans quelle unité, que prend-il en argument ?

(b) Soit le code suivant :

```
1  float z[5]{};
2  cout << sizeof(z) << endl;
```

i. Indiquer le type, la dimension, le type du contenu et les valeurs contenues par la déclaration en première ligne.

ii. Que retourne la seconde instruction ?

(c) À partir de ces observations, proposer une instruction retournant le nombre d'élément d'un tableau nommé "z".

(d) On place maintenant cette instruction dans une fonction.
- L'instruction a-t-elle le comportement attendu si le tableau est passé comme argument par valeur ? par référence ? par pointeur ? Justifier.
- Que retournerait cette fonction si on lui passait "z" par valeur. Expliquer.

(e) Quelles sont alors les limites du placement de cette instruction ?

Exercice 9:

(a) Considérons la déclaration suivante :

```
1  int affiche(int* argument[2]);
```

i. Quel est le type de retour ?

ii. Quel est le type de l'argument, peut-on le simplifier ? Justifier.

iii. Soit :

```
int i1 = 5;
int i2 = 6;
```

Définir en une ou deux lignes un argument correct à passer à cette fonction puis son appel.

iv. Écrire le corps de la fonction : Elle doit afficher les valeurs entières contenues dans le tableau. Si simplification, était-elle utile ?

(b) Considérons maintenant la déclaration suivante :

```
int affiche2(int (*argument)[2]);
```

i. Quel est le type de retour ?

ii. Quel est le type de l'argument, peut-on le simplifier ? Justifier.

iii. Soit :

```
int i1 = 5;
int i2 = 6;
```

Définir en une ou deux lignes un argument correct à passer à cette fonction puis son appel. Justifier le choix de l'appel.

iv. Écrire le corps de la fonction : Elle doit afficher les valeurs entières contenues dans le tableau.Si simplification, était-elle utile ?