UNIVERSITE IBN TOFAIL
Faculté des Sciences
Département de Mathématiques et d'Informatique
KENITRA

Année universitaire 2021-2022

POO en C++
Série 2

Exercice 1

On souhaite réaliser une classe Pile qui permet de manipuler les piles dont les éléments sont des pointeurs de type void* :

```
# define N 5
typedef void* T ;
class Pile
{
    T data[N];    // tableau contenant les éléments
    int sp;       // nombre d'éléments que contient la pile
public:
…
};
```

1. Donner un constructeur de Pile qui permet de créer une pile vide. Est-il nécessaire de définir pour cette classe un constructeur par recopie et un destructeur.
2. Définissez les fonctions membres suivantes:

   EstPleine() : qui permet de savoir si la pile est pleine
   EstVide() : qui permet de retourner true si la pile est vide
   push() : qui permet d'empiler (ajouter un élément à la pile)
   top() : qui permet de récupérer l'élément en haut de la pile
   pop() : qui permet de supprimer l'élément en haut de la pile

Exercice 2

1. Réaliser une classe `Ensemble` qui permet de manipuler les ensembles d'éléments d'un type T donnée, dont les membres données (privés) sont :
   - `data` : tableau dynamique contenant les éléments de l'ensemble.
   - `capacite` : taille du tableau data.
   - `cardinal` : nombre des éléments de l'ensemble.

   N.B. La taille du tableau aura une valeur par défaut CAPACITE_MAX, et sera augmentée de CAPACITE_PLUS en cas de besoin (ajout d'un élément lorsque le tableau est plein).
2. Définir pour cette classe les surcharges des opérateurs suivants :
   - Une surcharge de l'opérateur << qui permet d'ajouter un élément à l'ensemble, de telle sorte qu'on peut ajouter plusieurs éléments dans une même expression (par exemple : `e << n << m`; ajoute les entiers n et `m` à l'ensemble e).
   - Une surcharge de l'opérateur << qui permet d'afficher les éléments de l'ensemble.
   - Une surcharge de l'opérateur % qui permet de connaître si un entier donné appartient à l'ensemble :

     ```
     n % e        // vaut true si n appartient à e
     ```
   - Des surcharges des opérateurs == et != permettant tester l'égalité entre deux ensembles.
   - Une surcharge de l'opérateur < permettant de tester l'inclusion.

- Des surcharges des opérateurs +, *, et - donnant respectivement la réunion, l'intersection et la différence entre ensembles.

Exercice 3

Donner la sortie du programme suivant, et justifier votre réponse:

```
#include <iostream>
using namespace std;

class T{
   int i;
public:
   T(int n = 0){ cout << "+++ Constructeur \n"; i = n; }
   T( T & v){ cout << "*** Constructeur par recopie\n"; i = v.i;}
   ~T(){ cout << "--- Destructeur\n";}
};

void f1(T t) {}
void f2(T & t) {}

int main(){
   cout << "DEBUT\n";
   T u;
   T v = u;
   cout << "appel f1 : \n" ;f1(u);
   cout << "appel f2 : \n" ;f2(u);
   T * pt;
   pt = new T(2);
   delete pt;
   cout << "FIN\n";
   return 0;
}
```

Exercice 4

Complétez le programme suivant :

```
#include <iostream>
using namespace std;
class T{
   int n ;
   int *pn;
public:
   T(int);
   ...
};
T::T(int k){
   n = k;
   pn = new int[n];
   for(int i = 0; i < n; i++)
          pn[i] = 0;
}
int main()
{
   int tab[5] = {1,2,3,4,5};
   T a = tab;
   for(int i = 0; i < 5; i++)
          cout << a[i] << " ";
   cout << endl;
   T b = a;
   b[1] = 0;
   b[3] = 0;
   for(int i = 0; i < 5; i++)
          cout << b[i] << " ";
   cout << endl;
   return 0;
```

```
}
```

pour qu'il produit le résultat suivant :

```
1 2 3 4 5
1 0 3 0 5
```

Exercice 5

Complétez le programme suivant pour qu'il soit correct. Justifier tout membre (donnée ou fonction) ajouté :

```
...
class T{
   int x;
public:
   ...
};

int main()
{
   T a = 5;
   int i,j;
   i = a;
   j = 2 + a;
   cout << a << j << endl;
   return 0;
}
```

Exercice 6

On considère les déclarations et les définitions suivantes :

```
class A{
public:
   int x;
   A(int i ){ x = i; cout << " (A) ";}
};
int operator+(A a, int i){ cout << " (1) "; return (a.x + i);}
int operator+(A u, A v){ cout << " (2) "; return (u.x + v.x);}
A operator*(A u, A v){ cout << " (3) "; return (u.x * v.x);}
```

1. Supprimer les instructions erronées du programme suivant (justifier chaque suppression)

```
int main()
{
   A a = 1;
   int i = 2;
   cout << "\n********** DEBUT **********\n";
   cout << a + i << endl;
   cout << i + a << endl;
   cout << i + a + i << endl;
   cout << i + a + a << endl;
   cout << a * i << endl;
   cout << i + a * a << endl;
   cout << "\n********** FIN **********\n";
   return 0;
}
```

2. Que produit alors le programme après correction ? Justifier. (On donnera exactement la sortie du programme en respectant les sauts de lignes, et on justifiera ensuite le résultat de chaque instruction correcte)

Exercice 7

On considère les déclarations suivantes :

```
class A{
public:
   int x;
   A(int i ){ x = i;}
};

class B{
public:
   int x;
   B(A a){ x = a.x;}
   operator int (){ return x;}
};
//---------------------------------------
int operator+ (A a, int i)     { return (a.x + i);}
int operator+ (B b, A a){ return ( b.x + a.x);}
A operator* (A a, B b){
   A res = a;
   res.x = a.x * b.x;
   return res;
}
int f(A a){ return (a.x);}
//---------------------------------------
A a = 1;
B b = a;
int i = 2;
```

Les expressions suivantes sont-elles correctes. Si oui, quels sont les opérateurs, les conversions et les fonctions utilisés ?

```
(1) b + 10;          (2) 10 + b;          (3) a + 5;          (4) 5 + a;
(5) a + b + i;       (6) b + b;           (7) b + 10 *b;      (8) a * a + i;
(9) f(b);            (10) f(15);
```

Exercice 8

Complétez le programme suivant pour qu'il soit correct

```
…
class A{
   int x;
public:
   …
};
class B{
   int x;
public:
   …
};
…
int main()
{
   A a = 1;
   B b = 2;
   int i = a + b;
   int j = b + a;
   cout << i << j << endl;
   return 0;
}
```