TP1
SLAM2
POO

Ce premier travail est la reprise du TD1 SLAM2 UML/MERISE.

Dans le langage que vous souhaitez (C++, C#, ObjC, Swift) vous allez écrire les classes issues du contexte suivant :

Une entreprise emploie des personnes en tant que salariés. Une personne à un seul employeur.

D'après la correction, vous devriez avoir au moins deux classes : Les classes doivent être déclarées dans des fichiers séparés en dehors du programme principale.

1. Rappel sur les classes à utiliser

```
Class entreprise
{
        // attributs
        // C#
        String raisonSociale ;
        String adresse ;
        String ape ;
        Personne[] lesSalariés ;

        //C++
        char * raisonSociale ;
        char * adresse ;
        char * ape ;
        personne * lesSalariés ;

        //Swift
        var raisonSociale : String
        var adresse : String
        var ape : String
        var lesSalarié : [personne]

        //méthodes constructeur
        //C#
        entreprise() ;
        entreprise(String raisonSociale = “”, …) ;
        //C++
        entreprise() ;
        entreprise(char * raisonSociale = “”, …) ;
        //Swift
        init()
        init(raisonSociale : String, adresse : String, …)
```

```
        //méthodes accesseurs
        // getter et setter sur chaque attribut

        //méthodes métiers : liste des salariés
        //C#, C++
        void liste() ; // printf
        //Swift
        func liste() // print

        //méthodes d'affichage des attributs de l'objet
        //C#, C++
        void affiche() ; //printf
        // Swift
        func affiche() // print

        //destructeurs
        // C++
        ~entreprise() ;
}
```

```
#ifndef entreprise_h
#define entreprise_h

#include "personne.h"
#include <string>
#include <vector>

class personne;

class entreprise
{
    private :

    std::string raisonSociale ;
    std::string adresse ;
    std::string ape ;
    std::vector<personne> lesSalaries; // lesSalariés de l'entreprise est un vecteur

    public :

    // Constructeurs
    entreprise();
    entreprise(std::string raisonSociale, std::string adresse, std::string ape, std::vector<personne>
        lesSalaries);
    // Méthodes
    void affiche();
    void liste();
    void embauche(personne * unSalarie);
    entreprise * afficheEntreprise();

};
```

```
Class personne
{
	// attributs
	// C#
	String nom ;
	String prenom ;
	Int age ;
	entreprise employeur;

	//C++
	char * nom ;
	char * prenom ;
	int age ;
	entreprise employeur ;

	//Swift
	var nom : String
	var prenom : String
	var age : Int
	var employeur : entreprise

	//méthodes constructeur
	//C#
	personne() ;
	personne(String nom = “”, …) ;
	//C++
	personne() ;
	personne(char * nom = “”, …) ;
	//Swift
	init()
	init(nom : String, prenom : String, …)

	//méthodes accesseurs
	// getter et setter sur chaque attribut

	//méthodes d’affichage des attributs de l’objet
	//C#, C++
	void affiche() ; //printf
	// Swift
	func affiche() // print

	//destructeurs
	// C++
	~personne() ; // print
}
```

```cpp
#ifndef personne_h
#define personne_h

#include "entreprise.h"

#include <string>
#include <vector>

class entreprise;

class personne
{
    private :

    std::string nom ;
    std::string prenom ;
    int age;
    entreprise * employeur; // employeur de la personne

    public :

    // Constructeurs
    personne();
    personne(std::string nom, std::string prenom, int age, entreprise * employeur);
    // Méthodes
    void affiche();
    std::string getNom();
    std::string getPrenom();
    int getAge();
    void employer(entreprise * employeur);

};

#endif /* personne_hpp */
```

2. Nature du projet

Vous réaliserez votre application dans un projet console (ligne de commande)

3. Fichiers d'interface et d'implémentation

C++ et ObjC autorisent une interface (.h) et une implémentation (.m ou .cpp) de vos classes. Ce n'est pas le cas en C# et Swift.

L'arborescence du projet est au minimum la suivante :

C# :

- theApp.cs
- entreprise.cs
- personne.cs

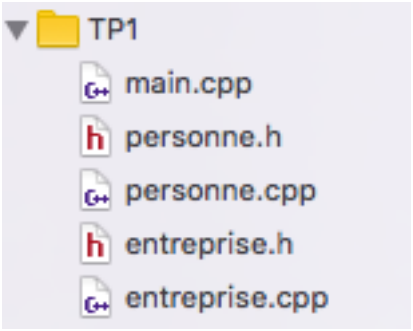
Swift :

- main.swift
- entreprise.swift

- personne.swift

C++ :

- theApp.cpp
- entreprise.h
- entreprise.cpp
- personne.h
- personne.cpp



ObjC :

- main.m
- entreprise.h
- entreprise.m
- personne.h
- personne.m

Attention à bien inclure vos fichiers de déclaration de vos classes si nécessaire suivant le langage est l'IDE.

4. Travail à faire

Il faudra créer des instances de personnes, et deux instances d'entreprises.

```
// Des personnes sans employeur
personne * unePersonne = new personne("Sergey","Brin",40,NULL);
personne * deuxPersonne = new personne("Larry","Pages",41,NULL);
personne * troisPersonne = new personne("Mark","Zuckerberg",37,NULL);
personne * quatrePersonne = new personne("Elon","Musk",44,NULL);
personne * cinqPersonne = new personne("Ray","Kursweil",60,NULL);

// Une liste de salarié
std::vector<personne> listeSalarie;
listeSalarie.clear();
listeSalarie.push_back(*unePersonne);
listeSalarie.push_back(*deuxPersonne);
listeSalarie.push_back(*troisPersonne);
listeSalarie.push_back(*quatrePersonne);

// Des entreprises avec une listes de salarié à la création
entreprise * uneEntreprise = new entreprise("IBM","Paris","32034",listeSalarie);
entreprise * deuxEntreprise = new entreprise("GOOGLE","San Francisco","88089",listeSalarie);
std::cout<<"Informations de l'entreprise :"<<std::endl;
uneEntreprise->affiche();
std::cout<<"Liste initiale des salariés suite au constructeur :"<<std::endl;
uneEntreprise->liste();

// Des personnes qui deviennent employées par des entreprises
unePersonne->employer(uneEntreprise);
unePersonne->affiche();
deuxPersonne->employer(deuxEntreprise);
deuxPersonne->affiche();
troisPersonne->employer(deuxEntreprise);
troisPersonne->affiche();
quatrePersonne->employer(uneEntreprise);
quatrePersonne->affiche();

// Une entreprise embauche une salarié supplémentaire
uneEntreprise->embauche(cinqPersonne);
// liste des employés des entreprises
uneEntreprise->liste();
deuxEntreprise->liste();
```