Programmation orientée objet

LIV 2024-2025

Travaux pratiques 3

Site du cours : `https://defelice.up8.site/poo.html`
Les exercices marqués de (@) sont optionnels.
Un fichier écrit en langage C++ se termine conventionnellement par .cpp.
Une commande de compilation est `g++ fichier_source1.cpp fichier_source2.cpp fichier_source3.cpp`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-Wextra` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-std=c++20` pour compiler selon la norme c++20
- `-g -fsanitize=address` pour que l'exécutable puisse detecter certaines erreurs de mémoire (executable plus lent, -g pour avoir plus de précision).
- `-g` pour ajouter des informations de débugage, utile avec debug (executable plus lent).

Exemple : `g++ -Wall -Wextra fichier1.cpp -o monprogramme`

Exercice 1. *Construction et destruction*

Écrire la classe `DestCons`. Lors de la construction ou destruction de chaque objet de la classe `DestCons`, un message doit être affiché avec l'adresse de l'objet.

Par exemple le code suivant :

```
int main(){
   DestCons a;
   return 0;
}
```

Pourrait produire l'exécution suivante :

```
.....$.\programme
Je suis construit à l'adresse 0x19c471fd83;
Je suis suis détruit à l'adresse 0x19c471fd83;
....$
```

Exercice 2. *@Tableau*

Implanter une classe `Tableau` qui implante des tableaux d'entiers de taille variable. Voici un exemple d'utilisation de la classe.

```
Tableau tab1{20};// créer un tableau de taille  20
                 // initialise chaque case avec la valeur 0
assert(tab1.taille()==20);  // renvoie la longueur du tableau
tab1=tab1; // rien ne doit se passer dans ce cas
for(int i=0;i<20;i++){
   assert(tab1.getCase(i)==0);}

// tab.setCase(20,4); Erreur à l'execution, message<il n'y a pas de case 20>
// tab.getCase(-1); Erreur à l'execution, message

tab1.setCase(4,2); // place 2 dans la cinquième case du tableau
assert(tab1.getCase(4)==2);
```

```
Tableau tab2{tab1}; // créer un nouveau tableau tab2, copie de tab1
assert(tab2.getCase(4)==tab1.getCase(4));
assert(tab2.getCase(4)==2);

assert(f(tab2)==2);// <int f(Tableau);> renvoie la somme des éléments du tableau
tab2.setCase(3,1);
assert(f(tab2)==3);

tab2.setCase(0,1);
assert(tab2.getCase(0)!=tab1.getCase(0))

Tableau tab3{6};
tab3.setCase(0,3);
tab2=tab3;
assert(tab2.getCase(0)==tab3.getCase(0))
tab3.setCase(0,2);
assert(tab2.getCase(0)!=tab3.getCase(0))

tab1.affiche(); // affiche le contenu du tableau sur la sortie
```

Comme pour n'importe quel exercice, il ne doit pas y avoir d'erreurs de mémoire (fuite,segfault,...). Vérifiez avec `-fsanitize=address`

Exercice 3. *Tableau extensible*

Implanter une classe `TableauExt` qui permet les mêmes choses (et passe les mêmes tests) que la classe `Tableau` mais qui permet en plus d'ajouter des cases au tableau.

Voici un exemple d'utilisation de la classe.

```
TableauExt tab2{10};// 10 cases
TableauExt tab1;
assert(tab2.taille()==10);
assert(tab1.taille()==0);

tab2.etend(4); // ajoute 4 cases (les cases 10,11,12,13) au tableau
assert(tab2.get(12)==0);
tab2.setCase(12,4);
tab2.suppr(2); // supprime les cases 13,12
// tab2.get(12); Erreur
tab2.etend(1);
assert(tab2.get(12)==0);
assert(tab2.taille()==13);
tab2++; // ajoute une case au tableau
```

Il ne doit pas y avoir d'erreurs ni fuites de mémoires.