Travail pratique # 4 : Tests Unitaires

École polytechnique de Montréal

Mazigh Ouanes, 1721035

Georges Louis, 1880098

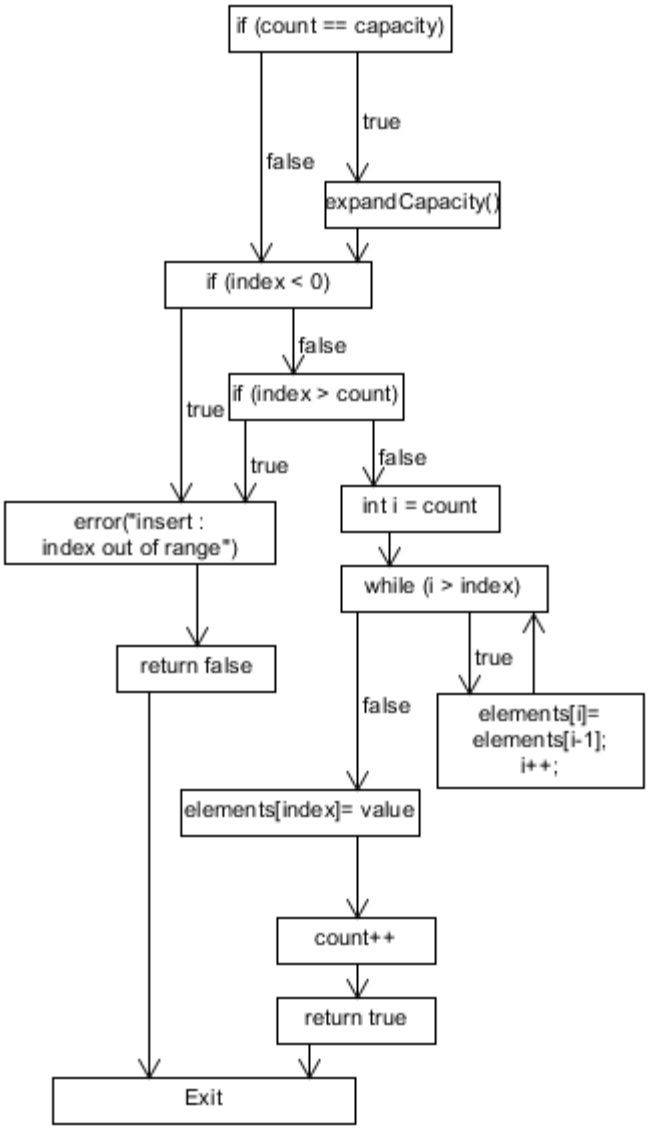
Équipe numéro : 20

Présenté à : Khalil Bennani

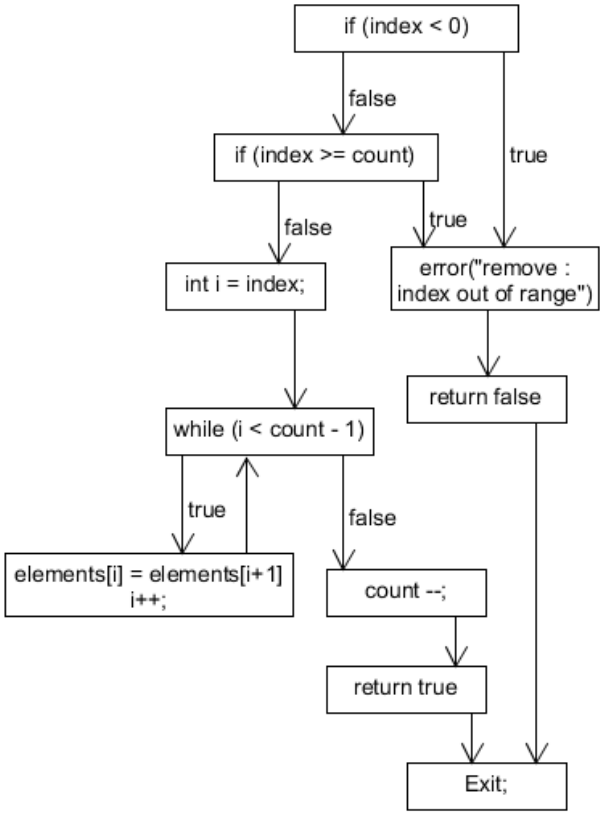
Date de remise (20-11-2017)

E1) Diagramme de flot de contrôle

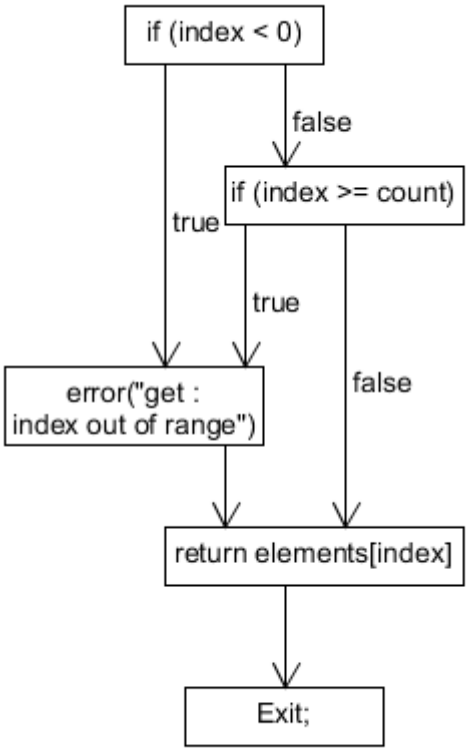
1) 1.1)Diagramme de flot de « «insert » :



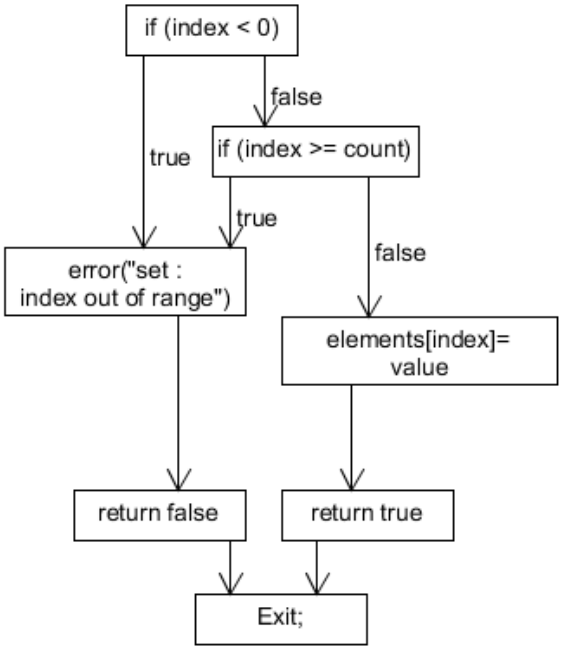
1.2)Diagramme de flot de « remove » :



1.3) Diagramme de flot de « get » :



1.4) Diagramme de flot de « set » :



2) La complexité cyclomatique de «insert» est de :
M= nb arcs – nb nœud +2= 16 - 13 + 2 = 5
M = nb conditions +1 = 4 + 1 = 5

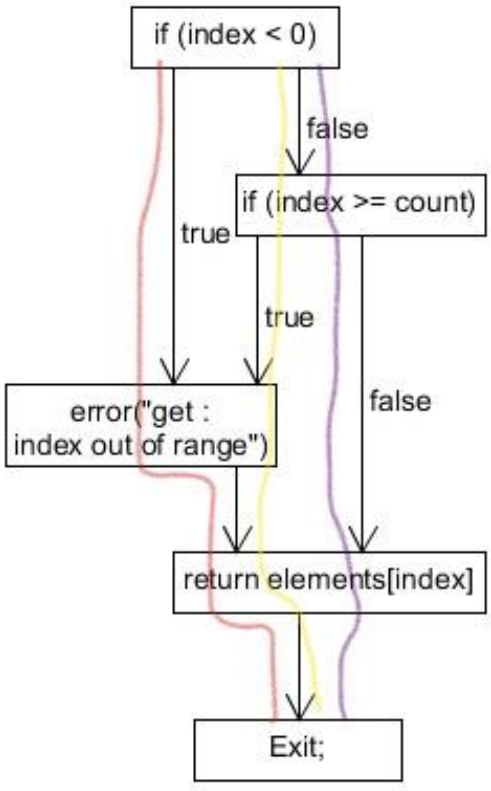
La complexité cyclomatique de «remove» est de :
M= nb arcs – nb nœud +2 = 12 – 10 + 2 = 4
M = nb conditions +1 = 3 + 1 = 4

La complexité cyclomatique de «get» est de :
M= nb arcs – nb nœud +2 = 6 – 5 + 2 = 3
M = nb conditions +1 = 2 + 1 = 3

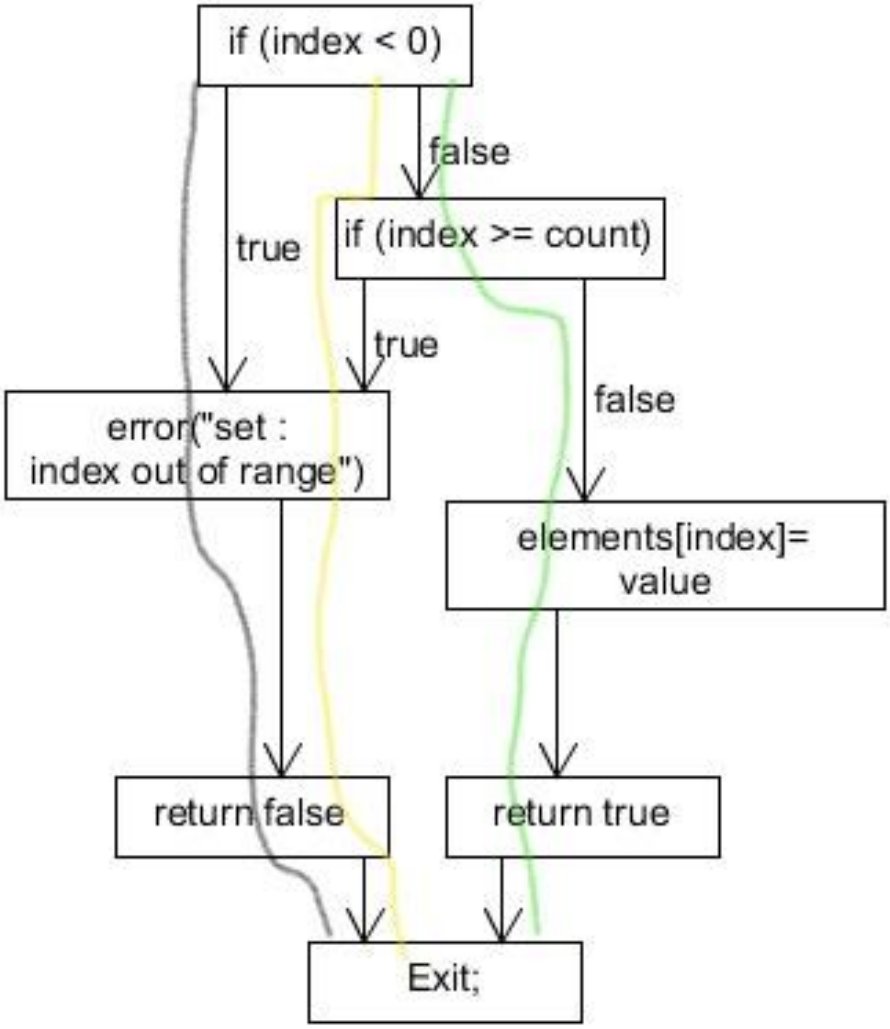
La complexité cyclomatique de «set» est de :
M= nb arcs – nb nœud +2= 8 - 7 + 2 = 3
M = nb conditions +1 = 2 + 1 = 3

3) chemins nécessaires à parcourir pour couvrir toutes les conditions.

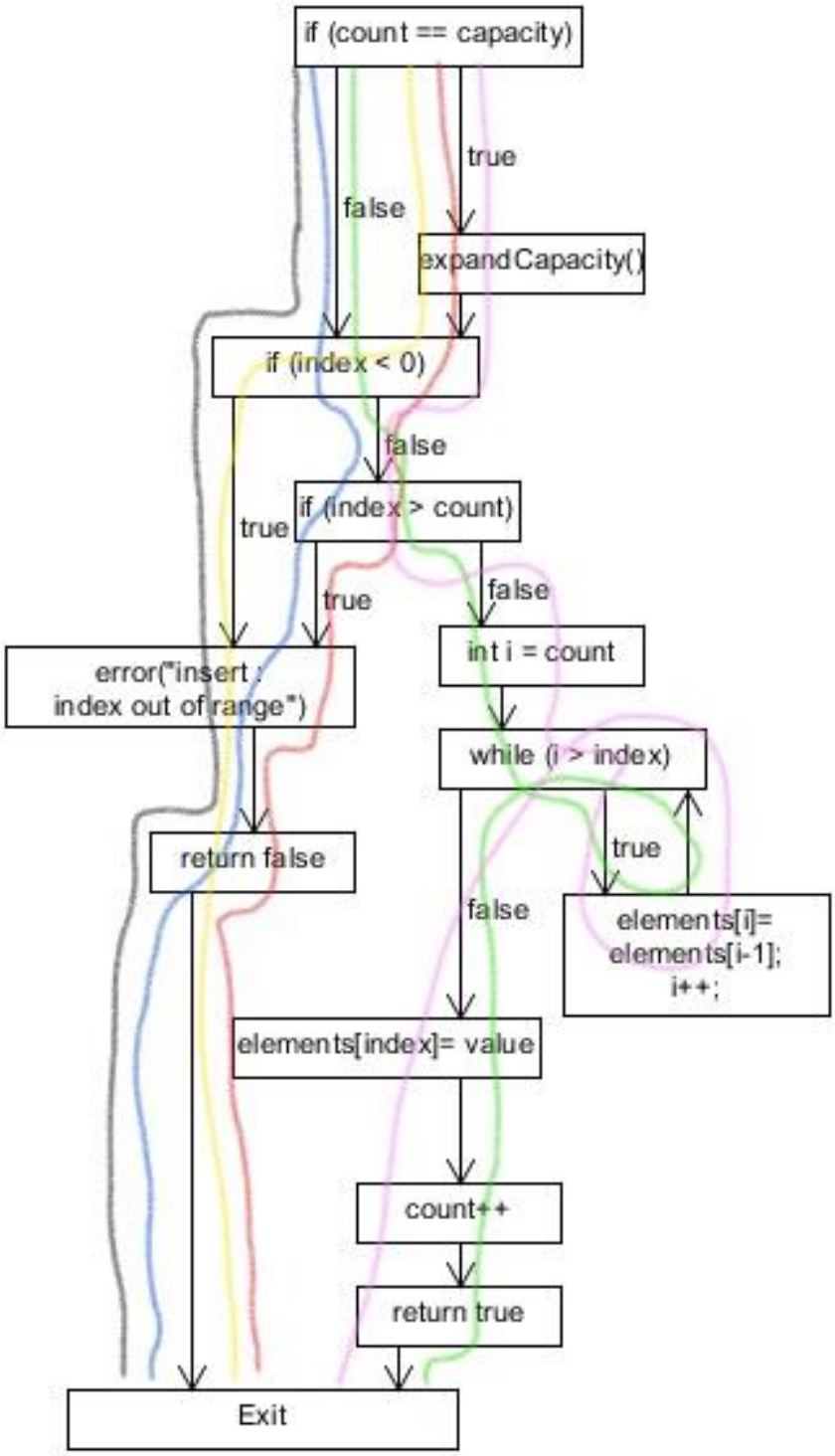
a) Get :



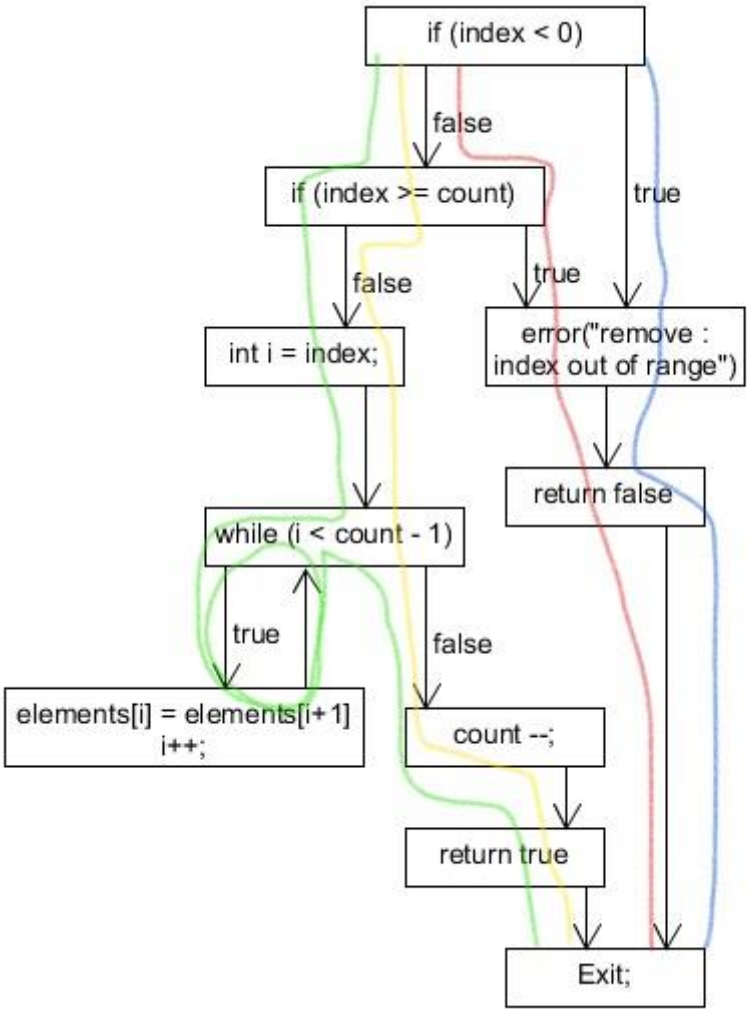
b) Set :


if (index < 0)
false
true
if (index >= count)
true
false
error("set :
index out of range")
elements[index]=
value
return false
return true
Exit;

c) Insert :


if (count == capacity)
true
false
expandCapacity()
if (index < 0)
false
if (index > count)
true
true
false
error("insert : index out of range")
int i = count
while (i > index)
return false
true
false
elements[i]= elements[i-1]; i++;
elements[index]= value
count++
return true
Exit

d) Remove :



E2) Cas de tests et de jeu de données

Tableau 1 : Résultat de la sortie de « Get » dépendamment des entrées

Possibilités	Entrées	Sortie
Possibilité 1 (Rouge)	Count =>1	error("get: index out of range"); return elements[-1];
	Index =>-1	
Possibilité 2 (Violet)	Count =>2	return elements[0];
	Index =>1	

Possibilité 3 (Jaune)	Count =>1 Index =>2	error("get: index out of range"); return elements[2];

Tableau 2 : Résultat de la sortie de « Set » dépendamment des entrées

Possibilités	Entrées	Sortie
Possibilité 1 (Noir)	int index =>-1 String value => “Test”	error("remove: index out of range"); return false;
Possibilité 2 (Jaune)	Int index =>2 String value => “Test”	error("remove: index out of range"); return false;
Possibilité 3 (Vert)	int index =>0 String value => “Test”	Return true;

Tableau 3 : Résultat de la sortie de « Insert » dépendamment des entrées

Possibilités	Entrées	Sortie
Possibilité 1 (Noir)	int index =>-1 String value => “Test”	error("remove: index out of range"); return false;
Possibilité 2 (Jaune)	int index =>-1 String value => “Test”	error("remove: index out of range"); return false;
Possibilité 3 (Rouge)	int index =>2 String value => “Test”	error("remove: index out of range"); return false;

Possibilité 4 (Bleu)	int index =>2 String value => “Test”	error("remove: index out of range"); return false;
Possibilité 5 (Vert)	int index =>0 String value => “Test”	Return true;
Possibilité 6 (Rose)	int index =>0 String value => “Test”	Return true;

Tableau 4 : Résultat de la sortie de « Remove » dépendamment des entrées

Possibilités	Entrées	Sortie
Possibilité 1 (Bleu)	Count => 2 index => -1	error("remove: index out of range"); return false;
Possibilité 2 (Rouge)	count =>2 Index =>3	error("remove: index out of range"); return false;
Possibilité 3 (Jaune)	count =>2 Index =>1	Return true;
Possibilité 4 (Vert)	count =>2	Return true;

E3) Implémentation et exécution des tests unitaires

[gelou@l4712-20 tests]$ '/usagers/gelou/LOG1000-20/TP4/tests/testsVector'

getTest::test1 : OK

setTest::test1set: index out of range : OK

setTest::test2set: index out of range : OK

setTest::test3 : OK

insertTest::test1insert: index out of range : OK

insertTest::test2insert: index out of range : OK

insertTest::test3 : OK

removeTest::test1 : OK

removeTest::test2remove: index out of range : OK

removeTest::test3remove: index out of range : OK

E4) Questions

1. Citez quatre utilités des tests unitaires en justifiant vos réponses?

a) Les tests unitaires nous aident durant le débogage. En cas d'échec d'un test, seulement les derniers changements portés au code doivent être déboguées.

b) Les tests unitaires permettent d'optimiser le code. Les révisions qu'apportent les tests unitaires peuvent amener des modifications considérables pour le projet. Ces modifications peuvent rendre le code plus performant parfois.

c) Les tests unitaires permettent de trouver des erreurs dans les codes plus tôt dans le code lorsqu'ils sont utilisés de manière stratégique tout au long du processus de développement, ainsi, ils permettent d'éviter des couts beaucoup plus dispendieux si des bogues se trouvent durant l'intégration ou le déploiement. Ainsi ils sont plus économiques.

d) Les tests unitaires contribuent à trouver des bogues plus rapidement, souvent avant l'étape de l'intégration. Lorsqu'on résout des bogues durant des étapes préliminaires et tout au long de la période de codage, il y'aura beaucoup moins d'impacts négatifs à la progression du développement du logiciel que s'ils sont résolus plus tard durant le processus.

2. Est ce qu'il sera meilleur de faire les tests unitaires à la fin de développement d'un projet? Pourquoi?

Non, il est plutôt préférable de faire les tests unitaires durant le développement d'un projet. Du d/but jusqu'à la fin. Ceci contribue au développement puisqu'on peut tester directement des fragments de code afin de savoir si ceux-ci retournent les résultats désires. Les tests unitaires aideront à trouver les bogues dans le code plus rapidement. Attendre de faire les tests unitaires jusqu'à la fin du développement du projet ne ferait qu'accumuler les erreurs qu'on aurait pu éviter. De plus, les bogues découverts plus tôt permettent de réduire le cout du projet car ceux-ci sont moins dispendieux à résoudre au début du développement du projet que vers la fin.

3. Quand considérons-nous que la couverture de test est optimale?

Nous ne sommes pas en mesure de faire la couverture des tests à 100%. La couverture optimale consiste à faire la couverture des conditions requises au complet. Sachant que dans une méthode d'une classe on peut retrouver parfois plusieurs conditions <if>, on veut donc décrire toute les toutes les conditions possibles qui peuvent être considérés par le <if-statement> ou qui seront rejetés.

4. Quel est la différence entre les tests de boîte blanche et les tests de boîte noire?

Le test de la boîte noire est utilisé pour tester le logiciel sans connaître la structure interne du code ou du programme. Le test de boîte blanche est la méthode de test de logiciel dans laquelle la structure interne est connue du testeur qui va tester le logiciel.

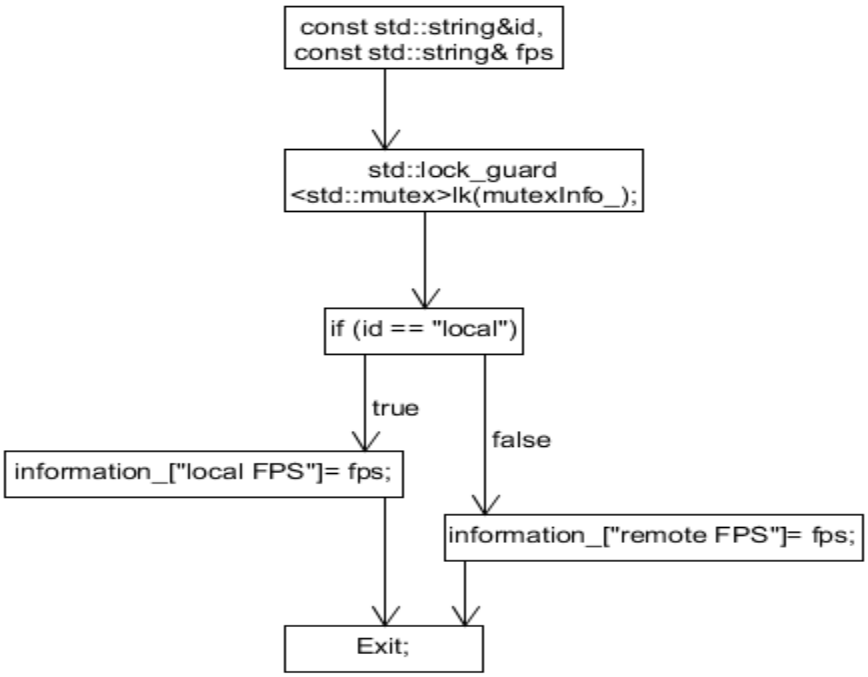
5. C'est quoi la différence entre la défaillance et erreur ?

Une erreur est une mauvaise interprétation de la demande pour l'implémentation du code. Malgré l'erreur celui-ci fonctionnera bien, mais ne fera pas ce qui était demander. Par contre, une défaillance provient d'une bonne interprétation du code, mais il y a une inhabilité d'exécution pour retourner la spécification qui est demandée.

E5) Contribution au projet Ring

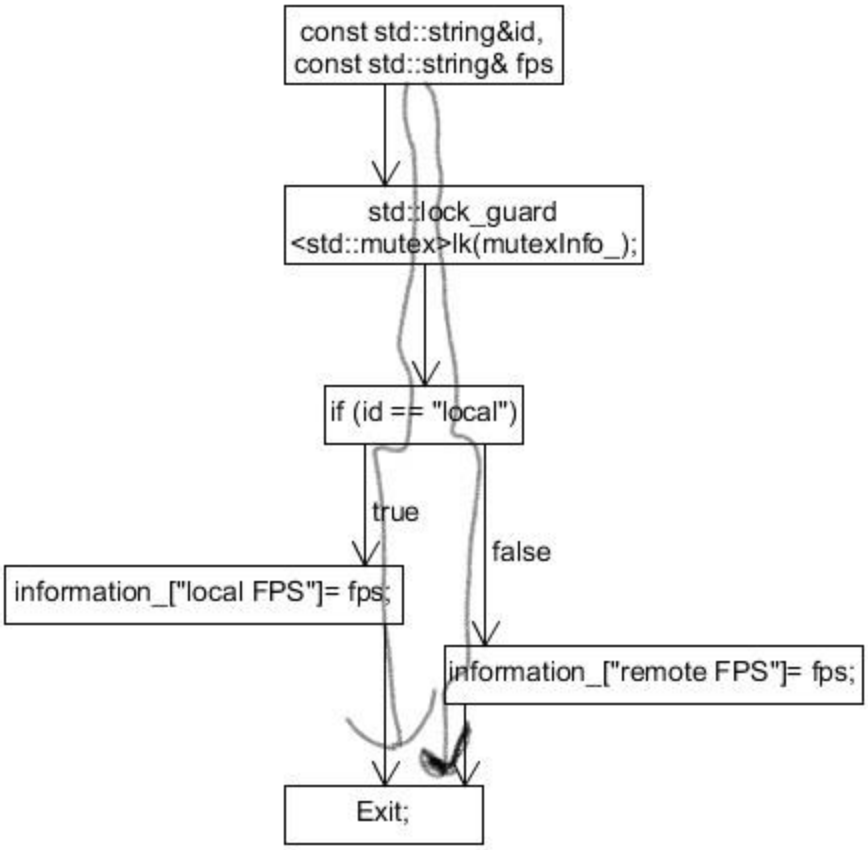
Smartools.cpp setFrameRate(const std ::string& id, const std ::string& fps)

1) Diagramme de flot de controle pour la function setFrameRate(const std ::string& id, const std ::string& fps) :



2) La complexité cyclomatique de la méthode setFrameRate(const std ::string& id, const std ::string& fps) est égale à :
M= nb arcs – nb nœud +2 = 6 – 6 + 2 = 2
M = nb conditions +1 = 1 + 1 = 2

3) les chemins nécessaires à parcourir pour couvrir toutes les conditions sont les suivants :



4) Oui, ils couvrent tous les chemins de la méthode setFrameRate(const std ::string& id, const std ::string& fps). Le paramètre id est celui qui donnent toutes les possibilités dans cette méthode. Dans notre cas, nous avons couvert tous les chemins possibles.