SCIPER	Code decomposition and restrictions	violation list	Comment on decomposition and restriction about vector (no insert / replace on vector but OK on string)
216913	2.00		Impeccable ! La modularisation du code est bien maitrisée ! Seule petite remarque : ligne 107, le break n'est pas nécessaire dans le while, la condition mise dans le if ligne 108 aurait pu être mise dans la condition du while.
296190	2.00		Très bonne décomposition, commentaire généralement pertinant, et nom de fonction clair. Bonne idée pour accélerer la fonction factorial. Faites attention aux commentaires, trop nombreux et trop long ils perdent de leur intérêt, mais là c'est ok.
297806			
301678	2.00		Bon code bien organisé.
302064			
302461	1.50	rech_anagram	Bonne modularisation. Très bon choix des fonctions et de leur nom. Il y a cependant une fonction qui ne respecte pas la longueur imposée: la fonction rech_anagram. L'utilisation de l'instruction "continue" n'est pas toujours très lisible. Essayez de l'éviter. N'hésitez pas à mettre quelques commentaires au début des fonctions pour brièvement expliquer leur but. Il y a une boucle for sans rien à la ligne 408, ce qui est difficile à comprendre. Finalement, on peut remplacer if(... = false) par if(!...). Dans l'ensemble, bon travail.
311144	2.00		Très bon travail ! Le principe de décomposition a bien été compris
312271	2.00		Bonne décomposition. Pour alléger un peu le code, on peut ajouter des lignes blanches entre les fonctions, ce qui permet de mieux les distinguer.
314564	2.00		Attention à la longueur des fonctions (initializeDICO fait 42 lignes). Très bonne décomposition sinon.
314664	2.00		Très bonne décomposition, bravo. Vous auriez par contre pu séparer le typedef de la declaration de votre structure ProprMot, pour améliorer la lisibilité. Nom des fonctions bien trouvé. Quelques commentaires pour expliquer ce que font vos fonctions auraient cependant été utile.
315404	2.00		impeccable
315678	2.00		Le code est très bien décomposé !
315801			
315905			
315933	0.50	Manque de la recherche d'anagrames, Tri_alpha, Tri_alphabetique	La décomposition faite est tout à fait justifiable et même plutôt bonne, attention à bien respecter la taille max des fonctions
315980	2.00		Bonne décomposition. Tu n'as pas besoin de ta structure message si c'est la même chose que mots. Tu pourrais faire un operator< pour mots pour simplifier le tri du dico.
316105	2.00		Excellente décomposition, l'usage de commentaires explicatifs aurait grandement facilité la compréhension du coded.
316133			

316149	**1.50**	manque la recherche d'anagramme	Bonne décomposition, Nom de fonction bien choisi et pertinent. Les quelque commentaires sont généralement pertinent, mais quelques-un de plus ne seraient pas de trop. Dommage que tout le code pour la recherche d'anagramme soit en commentaire. Faites attention à bien regarder et corriger les warning que vous donne le compilateur, c'est généralement une bonne piste pour détecter en amont des erreurs potentielles. Soyez très attentif au moment du rendu à rendre la bonne version du code ! (un commentaire au début de votre code dit "Not Latest")
316473			
316708	**2.00**		Excellente décomposition. Afin d'augmenter la lisibilité du code, il faut supprimer les commentaires inutiles (commentaires de debug ou d'anciennes fonctions) ainsi que d'ajouter des commentaires explicatifs.
324414	**2.00**		Excellente décomposition. Bon usage de constructeurs.
324605	**2.00**		Belle décomposition, un peu plus de commentaires auraient été apréciés.
324718	**2.00**		Très bonne décomposition. L'utilisation d'une fonction pour la comparaison de deux mots dans le tri allège la synthaxe.
324806	**2.00**		Bonne décomposition, cependant, la création de deux structure contenant les même éléments était superflue, vous pouvez utiliser des alias ou des typedef pour avoir deux nom pour la même structure. Nom généralement parlant, bien ! Attention aux if/else imbriqué, essayez de ne pas dépasser 3 niveaux. Ne pas oublier quelques commentaire pour préciser le but de certaine fonctions !
324815	**2.00**		Très bonne décomposition!
325002	**2.00**		Excellente décomposition parfaitement soutenue par des commentaires très complets
325420			
325828	**1.00**	main, setDictionnary	La décomposition faite est bonne mais il aurait fallu décomposer les fonctions (beaucoup) trop longues en sous fonctions.
325856	**0.50**	fill_dictio_from_line, find_sorted_index, find_index_original	Dans l'ensemble, bonne modulrisation. Trois fonctions ne respectent cependant pas la limite de longueur fixée à 40 lignes et une est à la limite. Votre code étant assez long, il est très compliqué de s'y retrouver sans l'aide de commentaires. Lorsqu'il n'est pas clair du but d'une portion de code, n'hésitez pas à faire un bref commentaire l'expliquant. Les fonctions d'une ligne peuvent être écrites directement dans le code. Dans la mesure du possible, essayez de ne faire que trois niveaux de conditions if imbriquées, grand maximum. Bravo
325961	**2.00**		Bonne décomposition.
326003	**2.00**		Excellente décomposition.
326025	**1.50**		ok mais l'analyse/l'écriture du tri et de la recherche d'anagramme n'est pas finie
326030			
326251	**2.00**		Très bonne décomposition.
326346	**1.00**	setMots, setMess	La décompostition est bonne et permet un code facile à suivre, auquel s'ajoutent des commentaires très pertinents. Attention aux fonctions trop longues qui auraient pu être facilement raccourcies en enlevant quelques retours à la ligne dans le code. Il s'agit certes d'une bonne pratique, mais une seule ligne vide suffit amplement.
326349	**2.00**		Bonne modularisation. Le choix des fonctions ainsi que leur nom est pertinent. Il aurait été possible d'utiliser, pour la vérification des majuscules, directement les éléments 'A' à 'Z' de type char. La séparation des fonctions avec une ligne de commentaire est très agréable mais n'hésitez pas à ajouter quelques brefs commentaires pour expliquer le but de chaque fonction.
326369	**2.00**		Excellente décomposition, bien explicite et très facile à lire.

326417	**2.00**		Très bonne décomposition!
326463	**2.00**		La décomposition est bien réalisée, les nom des fonctions sont généralement parlant, mais quelques commentaires précisant leur utilité aurait été appréciable. Vous avez choisi d'ordrer vos prototype selon le type de retour, hors il est plus utilie de les classer selon la fonctionnalité où ils sont utilisé. Une convention est par contre que peut importe l'ordre choisi pour les prototypes, celui pour les fonctions doit être le même.
326469			
326477	**2.00**		Très bonne modularisation. Très bon choix des fonctions et de leur nom. N'hésitez pas à mettre quelques commentaires au début des fonctions pour brièvement expliquer leur but.
326790	**2.00**		Bon travail dans l'ensemble ! Le principe de décompositon a bien été compris.
326822	**2.00**		Fonction main très succinte, bonne décomposition. Bonne utilisation des commentaires. Vous pouvez séparer la définition de main de la déclaration des prototypes pour plus de lisibilité. Warning : fonction read_messsage sur 44 lignes.
327096	**2.00**		décomposition exemplaire
327283	**2.00**		Excellente décomposition, code très lisible et clair. Petite remarque : l'usage de commentaires facilite la lecture du code, il ne faut pas hésiter à en mettre plus.
327501	**2.00**		Le problème est clairement décomposé. Rien à redire.
327538	**2.00**		Très bonne décomposition.
327637	**2.00**		Excellente décomposition du problème en sous-problèmes. Les fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code. Par contre, attention à certains noms de fonctions qui mériterait peut-être un peu plus de précision (par exemple ligne 52/54 "ajouter" et "afficher")
327683	**2.00**		Code très bien décomposé, la fonction separateur est peut être de trop cependant.
327694			
327708	**1.50**	lecture_messages	Super dans l'ensemble. La fonction lecture_messages est bien trop grande, il aurait fallu faire des fonctions annexes afin de l'alléger. Ligne 265 il n'y a pas besoin de mettre "==true" et "==false" puisque le if teste déjà si la condition est true, il suffit de l'inverser avec un ! au début de la condition à la place de mettre un "==false".
327732	**1.50**	saisie_dico	Bonne décomposition! Attention de respecter les 40 lignes par fonction. Aussi chaque fonction ne devrait faire qu'une seule tâche avec la fonction main au sommet qui contrôle le tout.
327821	**0.50**	entre tri	Le principe de décomposition n'a pas été assimilé, la fonction "tri" est illisible par le manque de sous fonctions et l'absence de commentaires. Le projet n'a clairement pas été fini à temps, il manque la partie recherche d'anagramme. Quelques pistes pour indiquer ce qui ne va pas dans la fonction "entre": - La déclaration des variables "char A='A'", "char Z='Z'" et "char point='.' est inutile, il faut traiter les caractères ASCII comme un tableau, pour tester si une lettre est en majuscule il suffit de tester 'A' < lettre < 'Z' - Une boucle infini "for(int i(1); i>0; i++)" à la ligne 33 peut simplement être réalisée avec "for(;;)" ou "while(true)". - Ligne 71-77, il suffit d'utiliser "op.size()" pour tester si un vecteur est vide - Le choix des noms de variables n'est judicieux du tout : "transition", "transfer", "op" pour le dictionnaire, "test", etc - Il y a des variables inutiles : "max"
327832	**2.00**		Bonne décomposition. Le nom des fonctions aident facilement à comprendre ce qui se passe.

328056	**2.00**		Super dans l'ensemble ! Mais l'utilisation des while(true) { ... if(...) { ... } else{ break; } } n'est pas une bonne pratique car ils rendent le code moins lisible et peuvent créer des boucles infinies si on ne fait pas assez attention. Ils peuvent être évité en modifiant légèrement le bloc et en mettant la condition du if soit dans la condition d'un do { ... } while (...) soit dans la condition de ton while à la place du true.
328073			
328162	**2.00**		Impeccable !
328194	**2.00**		Excellente décomposition. Bonne utilisation d'un enum pour gérer les différents cas d'entrée.
328239	**2.00**		Bonne décomposition dans l'ensemble. Quelques lignes vides et commentaires en plus pour organiser le code aurait pu être ajoutés.
328245	**2.00**		Très bon code : bonne décomposition, avec des commentaires complets.
328331	**2.00**		Très bon code : bonne décomposition. Bien compact, efficace. N'hésitez pas à faire des commentaires.
328354	**2.00**		Impeccable.
328390			
328433	**2.00**		Très bonne décomposition!
328434	**2.00**		
328436	**1.50**	entree_message	Bonne décomposition! Attention de respecter les 40 lignes par fonction.
328449	**1.00**	dico, cher_anagramme	Certaines fonctions auraient pu être encore plus décomposées afin d'éviter d'avoir de longues fonctions.
328515			
328686	**2.00**		Bon travail en général, le principe de décomposition a été compris
328696	**1.00**	Tri du dico pas complet et pas de recherche d'anagramme	Les quelques fonctions présentes sont bien nommée, et les commentaires sont judicieux. Dommage que le projet ne soit pas fini
328879	**2.00**		Bonne décomposition facile à lire. Peut-être qu'il y aurait eu moyen de faire plus simple la partie du tri dans le code et éviter autant de fonctions.
329093	**2.00**		Super code. Quelques commentaires sont toujours le bienvenue.
329098			
329110	**2.00**		Le code est bien décomposé. On remarque vite qu'est-ce qui fait quoi.
329113	**2.00**		Magnifique modularisation. Bon choix des fonctions et de leur nom. L'utilisation de "break" et "continue" rend parfois le code moins lisible. Dans la mesure du possible, essayez de les remplacer par, par exemple, des conditions dans des boucles while. Vous auriez pu ajouter quelques brefs commentaires en au début de chaque fonction pour expliquer leur but. Bon travail.
329169	**2.00**		Le code est bien décomposé, un peu plus de commentaires dans le code serait mieux tout de même
329286	**2.00**		Super dans l'ensemble ! Quelques fonctions auraient pu être rajoutées pour améliorer la modularisation du code. Notament le main est un peu plus lourd que ce qu'il devrait être.
329309	**1.50**	manque anagrames	Bonne décomposition.

329328	**2.00**		Excellente décomposition du code, de manière génerale, même si ce n'est pas noté ici, pensez à être consistant dans la notation des variables fonctions : par example : sortDicWord ou bien sort_dic_word
329464	**0.00**		fichier sans code
329505	**0.50**	creation_dico, tri_dico, creation_alpha_m	La décomposition est pertinente mais mériterait de créer plus de sous fonctions pour éviter de réutiliser le code et rendre le code plus lisible
329648	**2.00**		Très bonne modularisation. Très bon choix des fonctions et de leur nom. Les commentaires sont parfaits. Quelques remarques: la méthode main doit se contenter d'appeler les autres fonctions. La votre est un peu trop longue. L'utilisation de boucles while toujours vraie/fausse avec instruction break ou utilisation d'un booléan qu'on change à l'intérieur de la boucle sont à éviter. Elle sont peu lisibles et peuvent mener à des boucles infinies. Il aurait également été possible d'utiliser, pour la vérification des majuscules et à la place des nombres en code ASCII, directement les élément 'A' à 'Z' de type char. Bon travail.
329681	**1.50**	lecture_message,	bonne décomposition à part lecture message trop longue, gros doutes sur le tri multi-critère car refait entièrement sur un seul critère et débordement d'indice
329722	**1.50**	init_message	La fonction init_message est trop chargé, des fonctions annexes auraient pu être créées pour alléger. Les boucles for(...) { if (...) {...} else { break; } } ligne 204 et 315 peuvent être remplacées par une boucle while, ce qui rendra le code plus lisible. Mettre "==true" dans la condition d'un if après une expression qui est un boolean (ligne 213 et 275) est inutile, mettre seulement la condition sans le "==true" ferait la même chose. De même pour le "change == false" ligne 279 et "pareil == false" ligne 353 qui peut être remplacer par "!change" et "!pareil". La composition globale est tout de même pertinante !
329723	**2.00**		Super dans l'ensemble. Certaines fonctions sont un peu longues, des fonctions annexes auraient pu être créées pour les alléger. Les boucles for(...) { if (...) {...} else { break; } } peuvent être remplacées par une boucle while, ce qui rendra le code plus lisible.
329756	**2.00**		Bonne organisation du code. Quelques commentaires sont toujours le bienvenue.
329782	**2.00**		Bonne décomposition.
329793	**2.00**		Il y a une bonne décomposition du code. La fonction de tri aurait peut-être pu être mieux abstraite mais ce n'est pas faux.
329874			
329887	**2.00**		Très bon travail ! Le principe de décomposition est compris
329904	**2.00**		Bon travail ! Le principe de décomposition a bien été est compris
329910	**2.00**		Super dans l'ensemble, la modularisation est impéccable ! Mais l'utilisation des while(...) {if(...) { ... } else{ break; } } n'est pas une bonne pratique car ils rendent le code moins lisible. Ils peuvent être évité en modifiant légèrement le bloc et en mettant la condition de ton if après ta condition déjà présente dans ton while. Il n'est également pas nécessaire de caster un bool ligne 168, 171 et 174.
329920	**1.50**	calcul_nbt_nbd_alpha	Décomposition très pertinente, tache_1 et projet aurait pu être nommées de manière plus explicite
329942	**2.00**		Bonne décomposition, en géneral, par convention, les noms de fonctions et de variables ne commencent pas par une majuscule: VerifAnnagramme serait plutot verifAnnagramme
330043	**2.00**		Très bon travail ! Le principe de décomposition est compris

330070	**2.00**		La fonction partie2 pourrait avoir un nom plus descriptif. Très bonne décomposition et code bien documenté en conséquence !
330101	**1.50**	lire_dico	Bonne décomposition! Attention de respecter les 40 lignes par fonction.
330128			
330146			
330186	**2.00**		Le main est un peu trop chargé, des fonctions annexes auraient pu être créé pour l'alléger. Il n'est pas nécessaire de mettre le proptotype de la fonction si la fonction arrive à la ligne juste en dessous (ligne 17, 48 et 78 par exemple). On ne gagne pas en lisibilité et le code compilerait sans puisque la fonction est avant le main. Il y a également des ; en trop ligne 163 et 218, il n'y a pas de ; après un while (seulement après un do while). Mettre "==true" dans la condition d'un if après une expression qui est un boolean (ligne 310) est inutile, mettre seulement la condition sans le "==true" ferait la même chose.
330541			
330583	**2.00**		Bonne structure de code.
330629			
330640	**2.00**		
330667	**2.00**		Très bon code : bonne décomposition, bonne utilisation des commentaires.
330686	**0.50**	incomplet	seulement 68 lignes et 3 fonctions en plus de main()
330762	**2.00**		Bonne décomposition, vous auriez pu aller encore plus loin en englobant la fin du main dans une fonction, mais ce n'est pas très important. Petit warning sur les noms des fonctionsqui ne sont pas toujours très parlants.
330833	**2.00**		Très bon travail en général, félicitation pour l'utilisation des templates !
331471	**2.00**		Impeccable. L'utilisation de enums est excellente et les noms des fonctions globalement bien choisis. Essayez quand même de mettre la fonction main en haut du programme pour plus de lisibilité.
332230	**2.00**		Très bonne décomposition!
339389	**2.00**		Bonne modularisation. Le choix des sous fonctions est pertinent. Ne pas hésiter à choisir des noms plus pertinents pour les fonctions. Il aurait été possible d'utiliser, pour la vérification des majuscules et à la place des nombres en code ASCII, directement les élément 'A' à 'Z' de type char. De manière générale, mettre des nombres "tombés du ciel" n'est pas une bonne pratique. Mettre des expressions globales au début du programme aurait aussi été une solution. Bravo.
339407	**2.00**		Décomposition pertinente, bien travaillé. Les noms de fonctions pourraient être un peu raccourcis, par exemple arranger_le_dico en arranger_dico. Pourquoi créer deux structure exactement les même, vous auriez pu utiliser un alias pour avoir deux nom différent. Attention à la tailles des fonction qui flirtent avec la limite imposée. (lire_mot et lire_message)
339421	**2.00**		Très bon travail ! Le principe de décomposition a bien été compris.
339442	**2.00**		Très bonne décomposition!

339450	**2.00**		Bonne décomposition dans l'ensemble ! Des fonctions comme "tri_nbd" et "tri_alpha" sont très similaires à "tri_nbd_m" et "tri_alpha_m" et le principe de réutilisation aurait donc du être appliqué. Le choix des noms de fonctions est parfois peu judicieux ("tri_nbt", "tri_nbd", "tri_alpha" qui ne font qu'assigner la bonne valeur nbt, nbd et alpha à chaque éléments du dictionnaire, aucune opération de tri ne prend place dans ces fonctions). Aussi, Préférer l'utilisation de "true" et "false" a "1" et "0". Sinon, le nombre de lignes maximales par fonctions est respecté et les restrictions sur les méthodes de vector sont respectées.
339452	**1.00**	TRIalpha ANNA	Bon travail dans l'ensemble, le principe de décomposition est compris mais pas assez appliqué (les fonctios TRIalpha et ANNA sont trop longues)
339458	**2.00**		Excellente décomposition des fonctions, controle aurait bénéficié d'un nom plus explicite
339621	**2.00**		Il y a beaucoup de fonctions dans cette décomposition mais pourtant on ne s'y perd pas, ce qui est bien.
339646	**2.00**		Bonne décomposition, même un peu trop décomposé et certaines fonctions n'auraient pas besoin d'exister (par exemple error_dico_maj où on peut simplement utiliser les constantes d'erreurs fournies).
339649	**2.00**		Très bien dans l'ensemble, le principe de décomposition a été compris. Le code a même été trop décomposé ce qui peut nuire à la clarté et facilité de maintenance du code.
339655	**2.00**		Bonne décomposition, les noms des fonctions ne sont pas toujours clairs pour un lecteur extérieur
339712	**1.50**	get_dictionnary	Très bonne décomposition malgré un manque de sous-fonctions pour la création du dictionnaire, get_word_word aurait mérité un nom plus explicite ou un commentaire sur son objectif
339713			
339719	**2.00**		Le main pourrai être un peu plus décomposé, autrement, la décomposition est bonne
339831	**1.00**	Annagrammes(55 l), main(212 l)	Il faut faire une fonction par tache et non tout faire dans le main. Faire une structure aurait pu aider à la clarté du code.
339870			
339943	**0.00**		Il n'y a qu'une seule fonction main => 0pt Utilisation des fonctions de la librairie algorithm interdite (e.g :sort, transform) Seulement une partie du tri a été implémenté et la recherche d'anagramme est absente,
339959	**2.00**		Ok, la décomposition est excellente. Il serait cependant préférable de plus commenter les fonctions afin de les rendre compréhensibles au premier coup d'oeil.
340062	**0.00**	Un seul main	Ca à l'air d'être un bon début de projet.
340374	**1.00**	main, LireLigne	Les fonctions main et LireLigne sont bien trop grandes, il aurait fallu faire des fonctions annexes afin de les alléger.
340423	**2.00**		Excellente décomposition. Il n'y a cependant strictement aucun commentaire dans le code, ce qui est dommage car il est difficile de comprendre tout ce qu'il s'y passe. Les noms de variables peu indicatifs n'aident pas à cet effet et sont à éviter.
340497	**2.00**		Très bon travail ! Le principe de décomposition est compris

340645	**1.00**	init_dico_and_messages, main	Décomposition très pertinente, dommage qu'il n'y ait pas eu plus de sous-fontions pour alléger le main et l'initialisation du dictionnaire
340769	**2.00**		La décomposition est claire et simple, ce qui est très bien.
340803	**2.00**		Excellente décomposition, les fonctions faisant une ligne peuvent toutefois être évitées dans le cas où la ligne n'est pas particulièrement complexe
340814	**2.00**		Excellente décomposition, les commentatires sont très complets et bien expliqués.
340840	**2.00**		Code très bien documenté et décomposé !
340863	**2.00**		OK, le code est bien décomposé et commenté
340864	**2.00**		Excellente décomposition parfaitement soutenue par des commentaires très détaillés
340932	**2.00**		Excellent. Les noms sont parlants et accompagnés de commentaires très adéquats.
340933	**2.00**		Très bonne décomposition, il manquerait quelques commentaires pour ajouter de la lisibiliité.
340938			
340944	**2.00**		Bonne décomposition, évitez cependant les fonctions comme 'all' qui n'a pas de réel sens logique ici
340949	**2.00**		Bonne décomposition bravo, un peu plus de commentaires dans le code serait mieux tout de même
340952	**2.00**		Ok, la décomposition est excellente et très lisible. Concernant les commentaires, il est cependant préférable de faire un retour à la ligne complet proche de la limite de caractères pour rendre la lecture plus fluide. Les commentaires sont très clairs et expliquent bien le code. Petite remarque : au lieu de faire un répertoire contenant toutes les majuscules, on peut simplement utiliser le code ASCII associé à chaque caractère.
340991	**2.00**		Excellente décomposition, code très agréable à lire et lisible. Concernant les cout, il serait préférable de déclarer des variables constantes au début du code.
340992	**2.00**		Excellente décomposition sinon, code très lisible et clair. Il n'y a cependant strictement aucun commentaire explicatif, ce qui est dommage pour la compréhension de chaque fonction.
340997	**2.00**		Excellente décomposition. L'usage de commentaire aurait facilité grandement la compréhension des fonctions.
341008	**2.00**		Excellente décomposition. L'usage de commentaire aurait facilité grandement la compréhension des fonctions.
341015	**2.00**		OK
341021	**2.00**		La décomposition est pertinente. Les commentaires sont utiles et clairs.
341042	**2.00**		Très bonne décomposition. Bonne utilisation de struct et typedef. La fonction "get_ASCII" pourrait être remplacée par un cast en int, ou la comparaison pourrait être faite directement avec les caractères plutôt que leur code ASCII brut.
341045	**1.50**	main	Le main est beaucoup trop long, l'utilisation de sous-fonctions pour gérer les différentes parties aurait bien allégé le code. L'utilisation d'une struct ou d'un vector supplémentaire pour stocker la valeur nbD des mots du dictionnaire, plutôt que de la recalculer à chaque fois, aurait amélioré les performances.
341057			
341069	**2.00**		Bonne décomposition du problème.
341115	**2.00**		Très bonne décomposition.
341127	**2.00**		Décompostion claire et efficace.
341138	**2.00**		Le code a été bien décomposé entre fonctions. Certaines de ces fonctions sont assez confuses comme lecture_message mais rien de bien grave.

341148	**2.00**		Décomposition pertinente, mais les noms de fonctions ne sont pas toujours très bien choisis, vous pouvez penser à les rendre plus agréables pour un lecteur extérieur
341178	**2.00**		La décomposition du code est très bonne !
341181	**2.00**		Très bonne décomposition !
341189	**2.00**		Excellente décomposition du code !
341215	**2.00**		Excellente décomposition du code !
341237	**2.00**		Le code et clair et bien structuré. On comprend rapidement ce que fait chaque partie.
341270	**2.00**		La décomposition est correcte mais un peu difficile à lire en raison des noms de fonction peu explicite et la présence de nombreux retour à la ligne entre les fonctions.
341273	**2.00**		Bonne décomposition, il aurait été préférable de créer une fonction regroupant les dernières lignes du main() afin de rendre ce dernier plus aéré. Le nom des fonctions est clair à chaque fois et les commentaires sont pertinents et aident à la compréhension du programme. Attention toutefois à s'assurer qu'ils ne dépassent pas non plus la 87ème ligne.
341284	**1.50**	affichage_final	Dommage que la dernière fonction soit trop longue car le reste est vraiment très bon! Les noms sont parlants et la décomposition est excellente. Pour la suite, essayez peut-être juste de raccourcir légèrement le nom de vos fonctions qui sont parfois un peu lourds.
341354	**2.00**		Très bonne décomposition, les fonctions faisant une ligne peuvent toutefois être évitées dans le cas où la ligne n'est pas particulièrement complexe.
341363	**2.00**		Impeccable !
341372	**2.00**		Impeccable !
341400	**2.00**		Fonction main très succinte, bonne décomponsition. Warning : 3 fonctions sur 41 lignes. Warning 2 : privilégier le test de constantes char 'A' ou 'Z' plutôt que des valeurs brutes de code ASCII.
341439	**2.00**		Excellente décomposition. Le code est bien structuré, le choix des fonctions est approprié et clair. Pour alléger un peu la syntaxe, il est recommandé de remplacer un "else" contenant un unique "if" par l'instruction conditionnelle "else if" (p.ex. l. 331-332).
341450	**2.00**		Bonne composition de code ! Juste le tri du dico pourrait être simplifier si par exemple tu fais un operator < de Terme.
341451	**2.00**		Très bon code : bonne décomposition, fonction main succinte. Warning : fonction recherche_anagram sur 41 lignes.
341489	**2.00**		Très bon code : bonne décomposition. Il aurait pu y avoir une réutilisation des fonctions calculant nbT, etc. et création d'une autre fonction pour manipuler les opérations spécifiques au dictionnaire. Bonne utilisation des commentaires, très complets (mais pas besoin d'expliquer le "comment" des fonctions). Warning : fonction sort sur 42 lignes.
341551	**2.00**		Très bonne décomposition, ceci facilite grandement la lecture ansi que la compréhension du code.
341622	**2.00**		Très bon code : bonne décomposition. Bien compact, efficace. Warning: privilégier le test de constantes char 'A' ou 'Z' plutôt que des valeurs brutes de code ASCII.
341668	**2.00**		Super dans l'ensemble ! Mais l'utilisation des while(true) { ... if(...) { break; } } n'est pas une bonne pratique car ils rendent le code moins lisible et peuvent créer des boucles infinies si on ne fait pas assez attention. Ils peuvent être évité en modifiant légèrement le bloc et en mettant la condition inverse du if soit dans la condition d'un do { ... } while (...) soit dans la condition de ton while à la place du true. De même pour les boucles for(...) { if (...) { break; } } qui peuvent être remplacée par une boucle while. Cela rendra le code plus lisible et diminuera même sa taille ! Il y a également un ; inutile ligne 362.

341670	**2.00**		Impeccable ! La modularisation du code est bien maitrisée !
341687	**2.00**		Bonne décomposition. Conseil : la fonction operateur_inf_mot contient des return false inutiles. Dès qu'un return est appelé, la fonction s'arrête.
341739	**2.00**		Décomposition parfaite, les lignes pour séparer les différentes parties rendent le code très agréable à lire
341743	**2.00**		Bonne décomposition, commentaires utiles, pensez à nettoyer votre code avant le rendu cependant
341899	**2.00**		Excellente décomposition du code !
341931	**2.00**		Très bon code : bonne décomposition, fonction main succinte, bonne utilisation des commentaires. Rien à redire.
341932			
341940	**2.00**		Très bonne décomposition du problème en sous-problèmes. La plupart des fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code. Pour alléger rech_anagramme et le rendre plus lisible, il pourrait cependant être davantage décomposé.
341951	**2.00**		Bonne modulariation. Pourquoi utiliser deux structures (Struct) contenant les mêmes informations? Des typeDef auraient pu les remplacer. Essayez d'éviter les for(){if(){break}}. Il est souvent possible de les remplacer par des boucles while. Dans print-dico, par exemple, vous auriez pu, à la place, itérer de 0 à (la taille du dictionnaire - 1) directement dans la boucle for. La subdivision des fonctions ainsi que leur nom est très claire. Quelques brefs commentaires auraient été les bienvenues pour expliquer le but des portions de codes. Très bon travail.
341953	**2.00**		Bonne structure. Evite de donner des noms de seulement un charactère pour tes variables (et sans majuscule).
341970	**2.00**		Parfait, rien à redire.
341975	**2.00**		Très bonne décomposition, le code est clair.
342052	**2.00**		Très bonne séparation des fonctionnalitées, manque simplement quelques commentaires pour expliquer ce que font les fonctions. Si vous choisissez la convention d'écrire f_ avant chaque fonction, respectez là tout au long du programme ! Attention aux while(true), ce sont des source de boucles infinie si mal conçue. Les fonctions sur une ligne ne sont pas justifiée si la ligne est simple.
342054	**2.00**		Bonne modularisation, très bon choix de fonctions et de noms de fonctions et bonne lisibilité. Attention avec les instructions break qui ne sont pas toujours très lisibles et ne pas hésiter à ajouter à chaque début de fonction un petit commentaire la décrivant. Bravo!
342060	**2.00**		Bonne décomposition, les noms des fonctions sont parlants mais vous auriez pu les raccourcir un peu. Par exemple nbr_caracteres_differents deviendrait nb_car_diff.
342063	**1.50**	anagramme_maker	Bonne décomposition, les fonction ont généralement un nom parlant. Malheureusement la fonction anagramme_maker est trop longue, pourquoi tant d'espace? Commentaire pas toujours pertinant ils doivent servir à expliquer ce que font des portions de codes, pas comment et pourquoi, mais bonne initiative d'en avoir mis !
342154	**2.00**		Bonne décomposition. Noms des fonctions pas très explicites (par exemple afficher -> afficher_anagrames). Fonctions de tri quasiment identiques -> réutilisation.
342200	**2.00**		Bonne modularisation. Code très lisible, bon choix de noms de fonctions, code très clair. Ne pas hésiter à mettre quelques brefs commentaires pour encore plus de lisibilité et pour expliquer le but des fonctions.

342204	**2.00**		Bonne décomposition dans l'ensemble. Pour alléger encore le main et le rendre encore plus lisible, la boucle while aurait pu être mise dans une fonction à part.
342215	**1.50**	Manque recherche anagrame	Code bien organisé et lisible.
342216	**2.00**		Bonne structure de code. Cela aurait été plus simple de faire une structure pour un mot, et ton dictionnaire serait un vector de mots.
342224	**2.00**		Excellente décomposition du code !
342283	**1.50**	main	Le code est bien décomposé, néanmoins la fonction main est trop longue ! Il pourrait être encore plus décomposé.
342391	**2.00**		Bonne organisation. Tu pourrais alléger le main en rajoutant quelques fonctions. Ta fonction erreurs, utilise plutot un bool au lieu de char a. Le nom de ta fonction booleen est très mal choisi, tu aurais pu aussi la mettre dans la struct sous operator<.
342575			
342745	**2.00**		Bonne modularisation. Code très lisible, bon choix de noms de fonctions, code très clair. Ne pas hésiter à mettre quelques brefs commentaires pour encore plus de lisibilité et pour expliquer le but des fonctions.
342800	**2.00**		Très bonne décomposition, vous démontez une certaine maîtrise du langage, attention cependant, un trop grand nombre de fonction lambda et de template finissent pas nuire à la lisibilité du programme, surtout quant ils ne sont pas réellement justifiable. Ne faites pas compliqué juste pour montrer vos compétences, un bon programmeur est un programmeur qui sait rester sobre et qui va au plus efficace, sans superflu. Attention aussi au fonction d'une ligne, souvent elles alourdissent le code sans vrai bénéfice.
343266	**1.50**	rech_anagramme 391-436	Très bonne décomposition de la tâche de lecture/tri du dictionnaire. Par contre, pour la recherche d'anagrame la tâche aurait dû être décomposée en plus de sous-fonctions.
343725	**2.00**		Très bonne décomposition du problème en sous-problèmes. La plupart des fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code. Pour alléger le main et le rendre plus lisible, il pourrait être davantage décomposé.
343736	**2.00**		Excellente décomposition du problème en sous-problèmes. Les fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code.
343876	**2.00**		Très bonne décomposition. L'usage de commentaires aurait grandement facilité la compréhension du code.
344139	**2.00**		Bon travail en général ! Le principe de décompositon a été compris Cependant certaines fonctions sont parfois inutiles (e.g "affiche_string", quelle est la différence avec un "cout << mot" ?) Les noms des fonctions est parfois trop verbeux : les fonctions "affiche_string", "affiche_dictionnaire" et "affiche_anagramme", qui prennent en arguments respectivement ces types auraient put être remplacées par une fonction "affiche" surchargée.
344193	**2.00**		Bonne décompositon ! Le main pourrait être alléger en faisant une fonction de lecture de message en plus par exemple.
344225			
344276	**1.50**		Il n'y a pas de recherche d'anagrammes. A part cela le code est bien décomposé !

344310			
344312	**2.00**		Bonne décomposition, mais chaque fonction ne devrait faire qu'une seule tâche avec la fonction main au sommet qui contrôle le tout.
344350			
344385	**2.00**		Très bon code : bonne décomposition, très clair avec les commentaires. Warning: privilégier le test de constantes char 'A' ou 'Z' plutôt que des valeurs brutes de code ASCII.
344399	**2.00**		Très bon travail ! Le principe de décomposition est compris
344405	**2.00**		Bon travail en général ! Le principe de décompositon est compris. En général, on définit seulement les fonctions en début de fichier de fichier, leurs implémentations se situent après l'implémentation du main
344415	**2.00**		Belle décomposition, le main est très propre. Cependant les noms des fonctions pourraient être plus explicites
344463	**1.50**	read_message	Attention, la fonction read_message est bien trop longue, il faudrait la décomposer en plusieurs sous fonctions.
344514	**2.00**		Très bon code : bonne décomposition, bonnes utilisation des commentaires qui rendent la décomposition claire.
344516	**2.00**		Bonne décomposition. N'hésitez pas à faire des commentaires. Faire attention aux while (true) : peut conduire à une boucle infinie.
344526	**2.00**		Bonne décomposition dans l'ensemble. Cependant, le principe de ré-utilisation aurait pu être plus utilisé: il y a beaucoup de fonction qui se ressemblent (les fonctions de tri et de fusion ainsi que le test des majuscules) et la constante "NOT_IN_CAPITAL_LETTERS" qui est déclarée 2 fois localement dans 2 fonctions différentes.
344528	**1.50**	manque recherche anagrammes	Il manque la deuxième partie du projet consernant la recherche d'anagrammes. Pour la première partie, excellente modularisation. Le nom et le choix des fonctions sont claire. Les messages d'erreurs auraient pu être défini comme "const string" au début du progamme (c'est une bonne pratique de le faire) et quelques commentaires au début des fonctions auraient pu être présents pour expliquer leur tâche.
344538	**2.00**		Très bon code : bonne décomposition. Bien compact, efficace.
344544	**1.00**	Manque la recherche d'anagrames / main 128-174	Il manque la recherche d'anagrame. Les éléments qui sont dans le main doivent être décomposé en fonctions. Pour mieux organiser le code, les définitions des fonctions peuvent être placées après le main et garder seulement les prototypes des fonctions avant.
344700	**2.00**		La décomposition du code est claire et les commentaires aident beaucoup à sa lecture.
344708			
344732	**2.00**		Excellente décomposition du problème en sous-problèmes. Les fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code.
344736	**2.00**		Bonne décomposition. Essaye d'avoir le moins de chose dans ta fonction main.
344793	**2.00**		Très bonne décomposition !

344804	**2.00**		Très bon code : bonne décomposition. Vous avez utilisé deux structures qui ont exactement les mêmes objets, vous aurez pu réutiliser la même structure et les différentier avec le nom des instances. N'hésitez pas à mettre des commentaires pour clarifier les fonctions avec des noms pas très parlants.
344822	**2.00**		Excellente décomposition, le code est clair et lisible.
345007	**2.00**		La décomposition est très bonne.
345020	**2.00**		Excellente décomposition, les commentaires complètent très bien votre répartition de fonctions
345219	**2.00**		Beaucoup de fonctions sont très similaires à une autre, le principe de réutilisation n'est pas assez appliqué. Par exemple, la fonction "not_separateur" renvoie la négation du résultat de "separateur" si on l'appelle avec les mêmes valeurs en paramètre, mais est définie séparément. Les fonctions "suite_alpha" et "alpha" remplissent la même fonction, et sont quasiment parfaitement identiques. Similairement, plutôt que de définir deux struct avec exactement les mêmes champs, il serait possible d'utiliser un typedef. De plus, dans plusieurs fonctions (p.ex. rangement et tri_nbt), une variable est passée par référence, modifiée dans la fonction, puis sa valeur est renvoyée par la fonction et assignée à la même variable, ce qui est une perte de temps, surtout pour des variables de type vector. Finalement, une fonction comme "ajouter", qui ne contient qu'une instruction, pourrait être remplacée par cette instruction.
345259	**2.00**		Impeccable ! La modularisation du code est bien maitrisée !
345261	**2.00**		Décomposition OK Le bloc servant à déclarer les prototypes est difficilement lisible. essayez de rester concis en choisisssant vos nom de fonction, par exemple "trouver_les_proprietes_du_message" est trop long, utilisez des acronyme si c'est possible. Ne mélangez pas vos typedef avec les prototype, mettez-les tous ensemble dans leur coin.
345286	**2.00**		Excellente décomposition des fonctions, utilisation très pertinente des opérateurs surchargés pour la lisibilité des éléments de comparaison
345290	**1.00**	main, tri_dico	Il est très important dans un programme comme celui-ci d'avoir un main() le plus épuré possible, le but étant qu'il ne soit constitué que d'appels de fonctions. C'est compliqué de suivre son déroulement étant donné qu'il dépasse largement les 40 lignes, il aurait fallu décomposer chaque action (tri, recherche d'anagrammes) en plusieurs petites fonctions plus simples.
345298	**2.00**		Bonne décomposition bravo, il pourrait cependant y avoir un peu plus de commentaires dans le code
345308	**1.50**	main (92 lignes)	Décomposition quelques peu chaotique avec un main qui fait plus 92 lignes. Dans l'ensemble la règle des 40 lignes est respectée.
345310	**2.00**		Très bonne modularisation. Très bon choix des fonctions et de leur nom. N'hésitez pas à mettre quelques commentaires au début des fonctions pour brièvement expliquer leur but. L'utilisation de break est parfois difficile à comprendre. Dans la mesure du possible, essayez de les remplacer par des boucles while avec des conditions. Il aurait également été possible d'utiliser, pour la vérification des majuscules et à la place des nombres en code ASCII, directement les élément 'A' à 'Z' de type char. Bon travail.
345337	**2.00**		Bonne décomposition, mais chaque fonction ne devrait faire qu'une seule tâche avec la fonction main au sommet qui contrôle le tout.
345346	**0.50**	tri_dico, afficher_anagrammes, main	L'utilisation de struct aurait permi un allègement du code. Privilégier la décomposition des tâches en plusieurs sous-fonctions.
345352	**2.00**		Bonne décomposition, mais la fonction main aurait pu être séparée en plusieurs fonctions (taille limite de 40+-4 lignes).

345393	**1.50**	messagefonction	La fonction messagefonction est trop grande, quelques fonctions auraient pu être ajoutées afin de l'alléger.
345420	**1.50**	lecture_message	Certaines méthodes sont longues et pourraient être décomposées ou simplifiées. Néanmoins les autres fonctions sont bien décomposées.
345431	**1.50**	main	Code très clair et très bien documenté. Attention cependant à la fonction main qui est trop longue
345442	**2.00**		Bonne décomposition. Ta fonction main pourrait être allégée, par exemple tu pourrais faire une fonction de tri générale du dictionnaire qui appelle les 4 fonctions tri. Cela rend le code plus facile à comprendre.
345448	**2.00**		Excellente décomposition du problème en sous-problèmes. Les fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code.
345459	**2.00**		Bonne décomposition. Certaines fonctions pourraient avoir un nom plus parlant (p.ex "inverser" et "inverser2"). Les prototypes sont utilisés mais le main est quand-même placé tout à la fin, ce qui aurait pu être évité.
345473	**2.00**		Cette décomposition est assez confuse et très dure à lire. Il manque de l'abstraction du code. En revanche, le code est tout de même compréhensible.
345489	**2.00**		Bonne décomposition.
345491	**2.00**		Magnifique modularisation. Le nom des variables et des fonctions est claire, le choix de sous-fonctions est pertinent et les commentaires utiles à la compréhension du code. Des commentaires pour expliquer l'utilité de chaque fonction serait encore meilleur et l'utilisation de fonctions d'une ligne pas forcément nécessaire (fonction ajouter). La création de merge sort pour l'algorithm de tri semble très réussie.
345522	**2.00**		Très bonne décomposition. Pour plus de lisibilité, ne pas hésiter à faire de nouvelles fonctions plutôt que d'imbriquer des conditions les unes dans les autres. Également, ne pas hésiter à mettre plus de commentaires avant chaque fonction pour décrire ce qu'elles font.
345577	**2.00**		Super dans l'ensemble ! La boucle for(...) { if (...) { break; } } ligne 252 peut être remplacées par une boucle while, ce qui rendra le code plus lisible. La modularisation est impécable !
345618	**2.00**		Excellente décomposition du code ! Les commentaires pour séparer les différentes "sections" du code sont appréciées.
345671	**2.00**		Très bon code : bonne décomposition. Bien compact, efficace.
345710			
345712	**2.00**		Bonne décomposition, bravo ! les commentaires aident à la compréhension. code aéré, bien !
345716	**2.00**		Très bonne décomposition!
345765	**2.00**		Bonne décomposition et code très bref pourtant !
345770	**2.00**		Bonne décomposition. Ta fonction main pourrait être allégée, par exemple tu pourrais faire une fonction de saisie des anagrames qui te retournerait ton vector<Mot> messages.
345772	**2.00**		Bonne décomposition, bravo. Nom de fonction pertinent. Quelques commentaires n'auraient pas fait de mal. Code aéré et agréable
345773			

345785	**2.00**		Bonne modularisation. Code très lisible, bon choix de noms de fonctions, code très clair. Ne pas hésiter à mettre quelques brefs commentaires pour encore plus de lisibilité et pour expliquer le but des fonctions. Dans la mesure du possible, essayez d'ajouter des conditions dans les boucles while à la place d'utiliser des break.
345853	**1.50**		Pas de code pour la recherche d'anagramme.
345863	**1.50**	main	Bien dans l'ensemble, cependant, vous auriez pu créer quelques fonction supplémentaire pour alléger le main, par exemple pour la lecture. Attention aussi aux fonctions d'une seule ligne, ce n'est très pertinant si la ligne est simple. Nom de fonction parlant, bien !
345866	**2.00**		Magnifique modularisation. Le choix des fonctions ainsi que leur nom est pertinent. Le programme est très claire. Petite remarque: vous auriez directement pu utiliser les lettres de 'A' à 'Z' de type char pour la vérification des majuscules, au lieu du code ASCII. De manière générale, mettre des nombres "tombés du ciel" n'est pas une bonne pratique. Mettre des variables globales au début du programme aurait aussi été une solution. Quelques commentaires auraient pu aider à la lecture du code. Bon travail.
345936	**2.00**		Très bonne décomposition!
346039	**2.00**		Bonne décomposition!
346080	**2.00**		La décomposition est très bonne.
346151	**2.00**		Très bonne décomposition!
346183	**2.00**		bonne décomposition, le tri pourrait être plus efficace mais semble ok, attention: privilégier le test de constantes char 'A' ou 'Z' plutot que des valeurs brutes de code ASCII.
346193	**2.00**		Ok, la fonction tri_dico fait cependant 44 lignes. Il serait appréciable d'avoir des commentaires à chaque début de fonction pour une lexture plus claire. Dans l'ensemble, très bonne décomposition !
346210	**2.00**		Bonne décomposition, mais les noms de certaines fonctions ne sont pas hyper clairs. Essayez également de changer l'indentation globale de 2 à 4 espaces pour plus de lisibilité.
346220	**2.00**		Bonne décomposition des fonctions, faites attention au fonctions de même nom
346228	**2.00**		Excellente décomposition extrêmement claire.
346235	**2.00**		La décomposition est correcte. Certaines variables sont bizarrement nommées comme "structurevariablealpha" et parfois certaines fonctions ont des noms peut explicatifs (comme la fonction crea_dicotri)
346247	**2.00**		Excellente décomposition du problème en sous-problèmes. Les fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture très facile du code. Le code est également très bien organisé. Attention cependant à l'utilisation très fréquente de while(true) { ... if(...) { ... } else{ break; } }. Ce n'est pas une bonne pratique car ils rendent le code moins lisible et peuvent créer des boucles infinies si on ne fait pas assez attention.
346255	**1.50**	lecture_message	Bonne décomposition, mais la fonction lecture_message est trop longue. Elle serait allégée en séparant chaque cas par des sous-fonctions. Conseil : la fonction comparer_boite contient des return false inutiles. Dès qu'un return est appelé, la fonction s'arrête. Conseil 2 : définissez des constantes globales comme "tab", "space" au lieu d'utiliser les nombres du code ASCII pour les charactères spéciaux.

346301	**2.00**		Très bonne décomposition qui permet de suivre aisément le déroulement du programme. Essayez peut-être d'espacer un peu plus vos fonctions et de séparer la déclaration des constantes (messages d'erreur) du reste. La fonction enleve_g_nbT n'est pas utilisée, mais les noms des autres fonctions sont clairs, attention aussi à ne pas laisser des #includes inutilisés dans la suite du code (algorithm).
346370	**2.00**		Bonne décomposition, attention cependant à ne pas en faire trop (les fonctions de deux ligne ne sont pas toujours justifiée, par exemple ajouter). Certain nom de fonction un peu long et trop "bavard"(attribution_alpha). N'oubliez pas les commentaires !
346389	**1.50**	Manque la recherche d'anagrames	Bonne décomposition pour la partie lecture/tri du dictionnaire. Manque la rechercher d'anagrames.
346450	**2.00**		Bonne décomposition, Il serait cependant préférable de plus commenter les fonctions
346471	**2.00**		Magnifique modularisation. Attention: essayez, dans la mesure du possible, d'utiliser des boucles while à la place de for(){if(){break;}. Bon choix des noms de fonctions. Commentaires avant chaque fonction agréable pour comprendre les fonctions.
346514	**2.00**		Bonne décomposition, bravo ! nom de fonction parlant dans la plupart des cas. Quelques commentaire ne serait pas de trop. Une bonne pratique est de placer un entête contenant au moins votre nom et la date.
346515	**2.00**		Très bonne modularisation. Très bon choix des fonctions et de leur nom. Les commentaires permettent de comprendre le but des fonctions. Il aurait été possible d'utiliser, pour la vérification des majuscules et à la place des nombres en code ASCII, directement les élément 'A' à 'Z' de type char. L'utilisation de break est parfois difficile à comprendre. Dans la mesure du possible, essayez de les remplacer par des boucles while avec des conditions.
346518	**1.00**	main	Il n'y a pas de recherche d'anagrammes et le main est trop long. A part cela le code est bien décomposé !
346544	**2.00**		Excellente décomposition. La structure du code saute aux yeux.
346561	**1.50**	manque recherche anagrame	Très bonne modularisation. Bon choix des fonctions et de leur nom. L'utilisation de "break" rend parfois le code moins lisible. Dans la mesure du possible, essayez de les remplacer par, par exemple, des conditions dans des boucles while. N'hésitez pas à faire quelques brefs commentaires pour expliquer le but de vos fonctions. Attention: la longueur de la fonction main dépasse légérement la limite autorisée. Malheureusement, la partie de détection d'anagramme n'est pas terminée. Bon travail.
346564	**1.50**	tri_dico	Bonne décomposition! Attention de respecter les 40 lignes par fonction.
346565	**2.00**		Super code, bonne décomposition.
346576	**2.00**		La décomposition est bonne, les fonction on un nom parlant, et les quelques commentaires permettent bien de se retrouver dans le code, bon travail ! Faites cependant attention à ne pas faire de fonction de une ligne si elle ne sont pas justifiée, cela alourdi la lecture, voire peut amener à des pertes de performance à cause de copies inutiles.
346577			
346596	**2.00**		Attention, la fonction "Message_maj" renvoie un booléen qui est positif si le message n'est PAS en majuscule, ce qui est à éviter.

346718	**2.00**		Très bonne décomposition. Attention, certaines fonctions renvoient une valeur qui n'est ensuite pas utilisée, elles pourraient donc être de type void. De plus, une fonction comme "add", qui ne contient qu'une opération, pourrait être remplacée par cette instruction.
346944	**1.50**	main	La fonction main est trop grande, quelques fonctions auraient pu être ajoutées afin de l'alléger. En général le main ne continent casiment que des appels d'autres fonctions.
346973	**1.00**	Manque la recherche d'anagrames, calcul 269-325	Bonne décomposition pour la partie lecture/tri du dictionnaire. Manque la rechercher d'anagrames. Analyse du message dans "calcul" devrait être décomposée en plusieurs sous-fonctions. Le noms des fonctions n'est pas très explicite, mais commentaires expliquent bien.
347035	**2.00**		Bonne décomposition des fonctions, right_place aurait pu être sous divisé ou présenté plus clairement
347036	**2.00**		Très bonne décomposition , beaucoup de sous-fonctions très pertinentes, n'hésitez pas à séparer vos noms de fonction par des _ pour les rendre plus lisibles
347038	**2.00**		Très bonne décomposition, il manquerait quelques commentaires et il serait peut être préferable de sauter un peu moins de lignes de manière génerale
347056	**2.00**		Excellente décomposition, clarté du code et lisibilité impeccable.
347057	**2.00**		
347087	**2.00**		Il y a une bonne décomposition. On se retrouve bien dans le code.
347105	**2.00**		Bon travail ! Le principe de décomposition a été compris, il y a même trop de fonctions, certaines n'étant pas justifiable. Des fonctions comme "ResetFreqAlpha", "Print" ne sont utilisées qu'une seule fois et ne sont composées que d'une seule instructions Dans la fonction "Permuter", il est possible dans ce cas de juste écrire "mot1 = tmp" pour copier les valeurs des attributs
347115	**2.00**		Excellente décomposition. Le code est clair et lisible.
347213	**1.50**	Manque la recherche d'anagrames	Bonne décomposition pour la partie lecture/tri du dictionnaire. Manque la rechercher d'anagrames. Pour mieux organiser le code la structure aurait pu être mise au dessus des prototypes plutôt qu'au milieu.
347262	**1.00**	dico_tri, rech_anag	Décomposition globale pertinente mais il manque des sous fonctions pour alléger le code
347263	**2.00**		très bonne décomposition, nom de fonction parlant, ne pas oublier de mettre quelques commentaires. Essayez de corriger les warning envoyé par le compilateur, c'est généralement de bonnes remarques.
347288	**1.00**	Tri_alpha, message_contient_mot, rech_anagramme, main	La décomposition est très pertinente, dommage que vous ayez espacé vos lignes partout, cela rend le code beaucoup plus long et moins lisible
347301			
347317			
347321	**2.00**		Bonne structure. Des fonctions sont inutiles comme ajouter(l.34), en plus elle crée des copies pour rien.
347328	**2.00**		Excellente décomposition. Il n'y a cependant que trop peu de commentaires dans le code, ce qui est dommage car il est difficile de comprendre tout ce qu'il s'y passe. Les noms de variables peu indicatifs (tels que ok, ou un nom de fonction ne comprenant qu'une lettre) n'aident pas à cet effet et sont à éviter.

347341	**1.00**	read_dico_input 73-118, read_message_input 174-221	À part les 2 fonctions de lecture, la décomposition est plutôt bonne. Ces 2 fonctions doivent être décomposées en plus de sous-fonctions. Commentaires pour chaque fonction très biens. Code très bien organisé, par contre pour une meilleur lisibilité les définitions des fonctions peuvent être placées après le main et garder seulement les prototypes des fonctions avant.
347346	**2.00**		Très bon travail en général ! Les noms de variables et de fonctions sont courts et explicites, attention à la fonction main qui est de type int et qui doit donc retourner quelquechose.
347369	**1.50**	Recherche d'anagrame incomplète	Pour lecture/tri du dictionnaire plutôt bonne décomposition. Recherche d'anagrame incomplète. Noms des fonctions pas très explicites (affichage1, affichage2). Fonctions de tri quasiment identiques -> réutilisation.
347381	**2.00**		Impeccable ! La modularisation du code est bien maitrisée !
347394	**2.00**		Très bon code : bonne décomposition, bonne séparation des fonctions avec les lignes.
347428	**2.00**		Bonne décomposition. Pour plus de concision, le principe de réutilisation aurait pu être appliqué pour les fonctions qui calculent le nbT, nbD et alpha du message, qui sont quasiment identiques à celles utilisées pour le dictionnaire. De plus, une fonction comme "ajouter", qui contient une seule opération, pourrait être remplacée par cette instruction.
347445	**2.00**		Très bonne décomposition, bravo. Le mode debug à l'aide d'une variable est une bonne pratique, bien pensé. Nom de fonction parlant. Attention au if/else imbriqué, limitez vous a 3 niveaux maximum, fonction supplémentaire ici ?
347471			
347499	**2.00**		Excellent usage des commentaires qui complémentent à merveille une bonne décomposition. Essayez de favoriser la fonction main comme première du fichier, permettant ainsi plus de clareté.
347510	**2.00**		Bonne décomposition, mais chaque fonction ne devrait faire qu'une seule tâche avec la fonction main au sommet qui contrôle le tout.
347529			
347530	**2.00**		Bonne décomposition du problème en sous-problèmes. La plupart des fonctions sont concises et utilisent d'autres sous-fonctions ce qui permet une lecture facile du code. Pour alléger le main et le rendre plus lisible, il pourrait être davantage décomposé.