POLYTECH SORBONNE

Automne 2020

Grilles de mot de passe

TP 1

Objectifs

- ✯ Se familiarisez avec la syntaxe du C++.
- ✯ Comprendre le compilateur

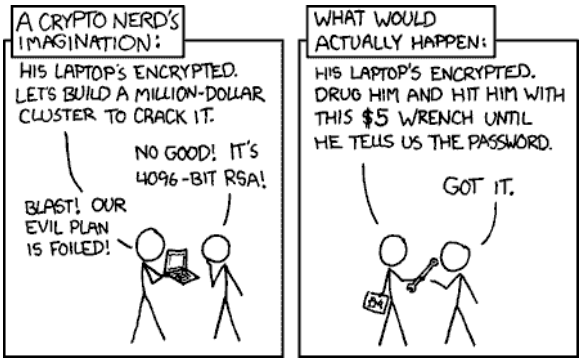
Contraintes

- Indentez vos fichier.
- La correction tiendra compte de la brièveté des méthodes que vous écrivez (évitez les fonctions de plus de 25 lignes) ; n'hésitez pas à découper une méthodes en plusieurs sous-méthodes (privées) plus courtes.
- Votre code ne doit pas donner d'erreurs avec Valgrind (ni fuite mémoire, ni autre erreurs).
- Vous ne devez pas utiliser de fonction C quand un équivalent C++ existe.
- Pour l'UML, vous pouvez utiliser UMLet ou Umbrello.
- Les noms de classe commencent par une majuscule.
- Les noms de méthodes et d'attributs commencent par une minuscule.
- La convention de nommage des accesseur est get_nom_attribut() et set_nom_attribut(...).
- Vous devez fournir un Makefile qui compile vos fichiers source et contient une règle clean ainsi qu'un programme de test.
- Le **code source et les diagrammes** doivent être *pusher* sur le git du TP
- En cas de problème avec le formulaire, envoyez votre archive par e-mail à `cecile.braunstein@lip6.fr`

Préparation du TP

- CLiquez sur le lien écrit dans moodle, votre dépôt git sera à l'adresse `https://github.com/Polytech-Sorbonne-Cpp-202` `tp\version-LOGIN` où *LOGIN* est votre login sur GitHub
- Cloner votre répertoire sur votre compte `git clone https://git` l'adresse qui vous a été attribué.
- Pendant le TP n'oubliez pas de comiter régulièrement
- N'oubliez pas de pusher l'ensemble
- Un ensemble de tests unitaires est fourni dans le répertoire `unitTest`, vous pouvez les compiler avec `make testcase`. Les tests utilise la bibliothèque `catch2`, pour en savoir plus `https://github.com/catchorg/Catch2`.

Concept

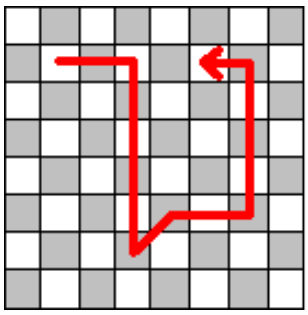


Comme l'illustre le *strip* de xkcd, il est souvent plus facile qu'on le croit d'obtenir un mot de passe. Pour augmenter la sécurité, on doit faire en sorte qu'il soit possible « d'oublier » le mot de passe et d'être incapable de le retrouver. Une technique simple pour atteindre ce but est d'utiliser une grille de mots de passe.

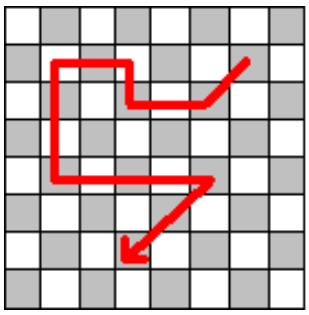
Le principe de ces grilles est de générer une grille de caractères aléatoires et de définir le mot de passe comme un chemin sur cette grille. Ainsi, l'utilisateur ne mémorise pas une suite de caractères mais un chemin. Deux ingrédients sont donc nécessaire pour accéder au système protégé : la grille et le chemin. La grille est inutile en tant que telle (si le chemin est suffisamment complexe) ; le chemin est aussi inutile sans la grille. Si un utilisateur se retrouve dans la situation décrite dans le strip précédent, il lui suffit de détruire la grille (e.g. de la brûler). Il sera alors dans l'incapacité physique de révéler son propre mot de passe !

l	c	8	^	k	4	,	G
(	j	a	3	$	b	@	l
m	&	m	e	l	O	n	5
k	%	@	,	F	w	A	K
b	*	X	q	:	=	h	w
P	L	!	'	+	h	U	s
I	7	.	?	[	\	~	x
(	2	c	k	A	U	c	W

(a) Un exemple de carte de mots de passe.



(b) Chemin donnant le mot de passe `6FVJ1vZ%nd^TpCeq`.



(c) Chemin donnant le mot de passe `eHYJVF6ixASvOcn%`.

1 Classe PassGrid

On souhaite implémenter une classe `PassGrid` qui représente une grille. Le constructeur de cette classe prend en argument la taille de la grille et initialise aléatoirement chaque case (on veut des caractères affichables, c'est-à-dire avec un code ascii entre 33 et 94). En plus du constructeur par copie, accesseurs en lecture et destructeur, la classe doit contenir :

- une méthode `reset()` qui recrée une grille aléatoire.
- une méthode `save(std::ostram) const` qui écrit la grille dans un flux passer en paramètre.
- une méthode `print()` qui affiche sur `std::cout` la grille ;
- méthode `load(std::istream)` qui charge la grille à partir d'un fichier.
- Un opérateur de lecture d'une case (i,j) de la grille : `char operator()(size_t i, size_t j) const`

Pour générer des nombres aléatoires, vous pouvez la fonction C `rand()` (man 3 rand) ou vous familiarisez avec les nouvelles bibliothèques `random` de C++11.

Les tests unitaires fournis vérifient :

- La taille de la grille
- La grille ne contient que des caractères imprimables

— Que deux générations successives de grilles sont différentes
— Que la méthodes reset change la grille (Ce test suppose que le constructeur par copie est implémenté).

2 Classe Path

On souhaite maintenant créer une classe `Path` qui représente un chemin dans la grille car on a notamment besoin de générer des mots de passe aléatoires pour les utilisateurs. Il y a 8 directions possibles : N, NE. E, SE, S, SW, W, NW que l'on peut représenter par des entiers. En plus des accesseurs en lecture et destructeur, la classe doit contenir :

— un constructeur `Path(int n, size_t hmax, size_t wmaw)` qui crée un chemin aléatoire de longueur `n` dans un rectangle de hauteur `hmax` et de largeur `wmax` ;
— une méthode `print()` qui affiche le chemin sous une forme lisible par l'utilisateur (par exemple `(2,4) N-S-SW-NE`), où `(2,4)` est le point de départ dans la grille.
— L'opérateur d'accès à une case du chemin

Les tests unitaires fournis vérifient :

— La case initial est dans les bornes
— Le chemin reste dans la grille.

3 Méthode generate()

Ajoutez à votre classe `PassGrid` une méthode `generate` qui prend un objet de type `Path` en argument et renvoie une chaîne de caractère correspondant au mot de passe.

4 Compléter les tests (BONUS)

Pour vous familiarisez avec les test unitaires ajouter les tests manquant qui démontrent que votre programme marche.

5 Écriture de la carte dans un fichier SVG (BONUS)

Ajoutez une option à votre programme qui prend en argument un nom de fichier et écrit dans ce fichier la grille au format SVG. Vous ne devez pas utiliser les fonctions C mais les fonctions C++ de manipulation de fichier (voir la classe `std::ofstream`). Pour le format SVG, vous pouvez regarder la documentation sur `http://www.svgbasics.com/simple_text.html`. Vous pouvez ouvrir les SVG avec votre navigateur web et les éditer avec inkscape.