Piscine C++ - d07a

Clément “Kinder” CHAVANCE chavan_c@epitech.eu

Abstract: *Ce document est le sujet du d07a*

Table des matières

I REMARQUES GÉNÉRALES 2

II Exercice 0 4

III Exercice 1 8

IV Exercice 2 12

V Exercice 3 16

VI Exercice 4 20

Chapitre I

REMARQUES GÉNÉRALES

- REMARQUES GÉNÉRALES :
  - Si vous faites la moitié des exercices car vous avez du mal, c'est normal. Par contre, si vous faites la moitié des exercices par flemme et vous tirez à 14h, vous AUREZ des surprises. Ne tentez pas le diable.
  - Toute fonction implémentée dans un header ou header non protègé signifie 0 à l'exercice.
  - Toutes les classes doivent posséder un constructeur et un destructeur.
  - Toutes les sorties se font sur la sortie standard et sont terminées par un retour à la ligne sauf si le contraire est precisé explicitement.
  - Les noms de fichiers qui vous sont imposés doivent être respectés À LA LETTRE, de même que les noms de classes et de fonctions membres / méthodes.
  - Rappelez-vous que vous faites du C++ et non plus du C. Par conséquent, les fonctions suivantes sont INTERDITES, et leur utilisation sera sanctionnée par un -42 :
    - `*alloc`
    - `*printf`
    - `free`
  - De facon générale, les fichers associés à une classe seront toujours NOM_DE_LA_CLASSE.hh et NOM_DE_LA_CLASSE.cpp (s'il y a lieu).
  - Les repértoires de rendus sont ex00, ex01, ..., exN
  - Toute utilisation de `friend` se soldera par un -42, `no questions asked` .
  - Lisez attentivement les exemples, ils peuvent requérir des choses que le sujet ne dit pas...

- Ces exercices vous demandent de rendre beaucoup de classes, mais la plupart sont TRÈS courtes si vous faites ca intelligemment. Donc, halte à la flemme !
- Lisez ENTIÈREMENT le sujet d'un exercice avant de le commencer !
- REFLÉCHISSEZ. Par pitié.

- COMPILATION DES EXERCICES :
  - La moulinette compile votre code avec les flags : `-W -Wall -Werror`
  - Pour éviter les problèmes de compilation de la moulinette, incluez les fichiers nécéssaires dans vos fichiers include (*.hh).
  - Notez bien qu'aucun de vos fichiers ne doit contenir de fonction `main` . Nous utiliserons notre propre fonction `main` pour compiler et tester votre code.
  - Rappelez-vous, on fait du C++ maintenant, donc le compilateur est g++ !
  - Ce sujet peut être modifié jusqu'à 4h avant le rendu. Rafraichissez-le régulièrement !
  - Le repértoire de rendu est : `(DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/exN` (N étant bien sur le numéro de l'exercice).

Chapitre II

Exercice 0

	Exercice : 00	points : 3
Meeeeeeedic		
Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/ex00`		
Compilateur : `g++`	Flags de compilation: `-Wall -Wextra -Werror`	
Makefile : `Non`	Règles : `n/a`	
Fichiers a rendre : `Skat.h, Skat.cpp`		
Remarques : `n/a`		
Fonctions Interdites : `malloc, free, *printf, pointeurs`		

Soldats, bienvenue au SKAT (Special Kreog and Tactical). Si vous etes ici, c'est parce que vous avez envie de défendre notre beau pays, l' Australie contre cette engeance rampante que sont les dingos. Ces infâmes pourritures ne reculeront devant rien pour saper les bases de notre beau pays.

Je suis le sergent instructeur Hartog et je vais être chargé de faire de vous des foudres de guerre.

Mais en attendant d'aller leur montrer ce que vous valez, pour l'instant vous etes le niveau zéro de la vie sur terre.

Bien, commencons le briefing.
Vu qu'il est plus que probable que vous vous preniez une bastos sur le champ de bataille, nos scientifiques ont mis au point quelque chose qui vous sauvera les miches plus d'une fois : le stimpak. Il soude les os brisés, suture les plaies, etc... Du moment qu'il vous en reste un, vous aurez une chance de rentrer en vie à la maison.

Implémentez la classe suivante :

```
class Skat
{
    public:
        Skat(std::string const& name, int stimPaks);
        ~Skat();
```

```
    public:
        [...]        stimPaks();
        const std::string& name();

    public:
        void    shareStimPaks(int number, [...] stock);
        void    addStimPaks(unsigned int number);
        void    useStimPaks();

    public:
        void status();

    private:
        [...]
};
```

Explications :

- Un Skat possède un nom représente par une chaine de caractère et un nombre de stimpaks. Votre Skat portera le nom de bob et possédèra 15 stimpaks par défaut.
- La fonction membre stimPaks retourne le nombre de stimpacks que possède votre Skat.
  Cette fonction membre devra permettre de modifier le nombre de stimpaks de votre unité en dehors de la classe
- la fonction membre name retourne le nom de votre unité. Cette fonction membre ne modifie pas l'instance appellante.
- La fonction membre `shareStimPaks` permet de donner des stimpaks superflus a un coéquipier dans le besoin. Cette fonction membre incrémentera de `[number]` le `[stock]` de stimpaks.
  Vous devrez décrementer votre reserve de `[number]` stimpaks. Si le nombre de stimpaks demande est trop grand, afficher

```
1 Don't be greedy
```

  sur la sortie standard et ne rien faire. Sinon, afficher

```
1 Keep the change.
```

- la fonction membre `addStimPaks(unsigned int number)` ajoute `[number]` stimpaks à ceux que votre unité possède déjà. Si `[number]` vaut 0, afficher

```
1 Hey boya, did you forget something ?
```

sur la sortie standard.

- Votre unité peut utiliser un stimpaks via un appel à la fonction membre `useStimPaks()` . Elle affichera

```
1 Time to kick some ass and chew bubble gum.
```

sur la sortie standard si cela est possible. Sinon afficher

```
1 Mediiiiiic
```

- Une unité peut vous donner à tout moment son statut via un appel à la fonction membre `status` . Une unité s'exprimera de la manière suivante :

```
1 Soldier [NAME] reporting [NUMBER] stimpaks remaining sir !
```

où `[NAME]` représentera le nom de l'unité et `[NUMBER]` son nombre de stimpaks. Cette fonction membre devra pouvoir être appellée sur des objets Skat immuables.

Le code suivant devra compiler et afficher la sortie ci-après :

```
int main()
{
    Skat s(‘‘Junior’’, 5);

    std::cout << ‘‘Soldier ‘‘ << s.name() << std::endl;

    s.status();

    s.useStimPaks();

    return 0;
}
```

Sortie :

```
Kinder:ex00$ g++ -W -Wall -Werror *.cpp
Kinder:ex00$ ./a.out | cat -e
Soldier Junior$
Soldier Junior reporting 5 stimpaks remaining sir !$
Time to kick some ass and chew bubble gum.$
Kinder:ex_0$
```

Chapitre III

Exercice 1

	Exercice : 01	points : 2
KoalaBot, Assemble		
Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/ex01`		
Compilateur : `g++`	Flags de compilation: `-Wall -Wextra -Werror`	
Makefile : `Non`	Règles : `n/a`	
Fichiers a rendre : `KoalaBot.h, KoalaBot.cpp, Parts.h, Parts.cpp`		
Remarques : `n/a`		
Fonctions Interdites : `malloc, free, *printf`		

Messieurs, voici le KoalaBot Kreog mk5 (les 4 versions précédentes avaient tendance à être un peu trop explosives). Et c'est ce dernier qui sera envoyé lorsque ca sera trop chaud pour vous (Une mission de reconnaissance dans un champ de mine par exemple).

Ce dernier possède 3 parties amovibles : des bras. des jambes ainsi qu'une tête (le premier qui me dit mais monsieur il a 2 bras et 2 jambes je lui pète les genoux).

Ces différentes parties seront regroupées dans les fichiers `Parts.h,cpp` et possèderont les noms suivants : `Arms` , `Legs` , `Head` .

Elles seront toutes construites selon le modèle suivant:

- chaque classe possèdera un constructeur ayant la signature suivante:
  `Constructeur(std::string [...] serial, bool functionnal);`
- Elles devront posséder un serial representé par une chaine de caractères, ainsi qu' un booléen nous indiquant si cette dernière est fonctionnelle ou non.

  – La class Arms aura un serial par défaut de "A-01"

  – La class Legs aura un serial par défaut de "L-01"

- La class Head aura un serial par défaut de “H-01”

Ces dernieres seront toutes fonctionnelles par défaut.

- Elles devront posséder les fonctions membres publiques suivantes:
  - `bool isFunctionnal()` qui nous indiquera si la pièce est fonctionnelle ou non
  - `std::string serial()` qui nous renverra le serial de la pièce en question
  - `void informations()` qui affichera :

```
1     [Parts] [PARTSTYPE] [SERIAL] status : {OK | KO}
```

Précédé d’une tabulation, et suivi d’un retour à la ligne.

- `[PARTSTYPE]` devra être remplacé par le nom de la classe
- `[SERIAL]` sera bien sur le serial de la pièce en question
- Si la pièce est fonctionnelle, afficher “OK” sinon “KO”

`Toutes ces fonctions membres ne modifieront en rien l’instance appellante.`

Bien maintenant qu’on a les différentes parties, il ne nous reste plus qu’à les assembler.
Crèez la classe `KoalaBot`

- Cette dernière sera composée d’une instance de chacune des pièces créées précédemment (`Arms` , `Legs` et `Head` pour ceux qui ont la flemme de relire).
- Elle possédera un serial qui sera stocké sous la forme d’une chaîne de caractères, qui possédera la valeur “Bob-01” par défaut.
- Elle possédera une fonctiom membre `setParts` qui pendra en paramètre une référence sur une pièce constante.

On devra pouvoir l'appeller avec n'importe laquelle des classes presentes dans le fichier Parts.h

- Elle possèdera une fonction membre `swapParts` qui prendra en paramètre une référence sur une pièce. Cette dernière sera échangée avec la pièce du `KoalaBot`

On devra pouvoir l'appeller avec n'importe laquelle des classes présentes dans le fichier Parts.h

- Elle possèdera une fonction membre void `informations()` qui affichera

```
1    [KoalaBot] [SERIAL]
```

suivi d'un retour à la ligne, `[SERIAL]` sera bien sur le serial du `KoalaBot` , suivi des informations de chacunes des pièces qui le compose, dans l'ordre suivant:

  - Bras
  - Jambes
  - Tête

Cette fonction membre ne modifiera pas l'instance appellante

- Elle possèdera la fonction membre `bool status()` qui renverra `true` si toutes les pièces le composant sont fonctionnelles, sinon elle renverra `false`

Cette fonction membre ne modifiera pas l'instance appellante

Le code suivant devra compiler et afficher la sortie ci-après :

```
int main()
{
    KoalaBot kb;

    std::cout << std::boolalpha << kb.status() << std::endl;

    kb.informations();

    return 0;
}
```

Sortie :

```
Kinder:ex01$ g++ -W -Wall -Werror *.cpp
Kinder:ex01$ ./a.out | cat -e
true$
[KoalaBot] Bob-01$
       [Parts] Arms A-01 status : OK$
       [Parts] Legs L-01 status : OK$
       [Parts] Head H-01 status : OK$
Kinder:ex_1$
```

Chapitre IV

Exercice 2

	Exercice : 02	points : 5
Houston, on a un problème		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/ex02		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : KreogCom.h, KreogCom.cpp		
Remarques : n/a		
Fonctions Interdites : malloc, free, *printf		

Vous êtes un groupe, aussi soudé que les parisiens dans le métro aux heures de pointes. Aussi bien quand vous vous reposez à la caserne que quand vous serez éparpillés dans la jungle. Afin de pouvoir communiquer, et surtout de savoir où sont vos petits camarades, je vous présente le fleuron de notre technologie : le KreogCom . Ce bijou vous permettra de savoir en temps réel où sont vos petits camarades.

Le KreogCom est composé de la manière suivante :

```
class KreogCom
{
    public:
        KreogCom(int x, int y, int serial);
        ~KreogCom();

    public:
        void    addCom(int x, int y, int serial);
        KreogCom        *getCom();
        void    removeCom();

    public:
        void ping();
        void locateSquad();

    private:
```

```
        [...]

    private:
        [...]

    private:
        const int m_serial;
};
```

Explications:

- Un KreogCom possède : un numéro de série, représenté par un entier constant, ainsi qu'une position X et une position Y.

Libre à vous de lui rajouter des données membres, mais ces dernières devront être privées

- il possède un constructeur lui permettant de créer un `KreogCom` , aux positions (`[x],[y]`), et possédant le serial `[serial]`
- il possède une fonction membre `addCom` qui crée un nouveau `KreogCom` et le relie au `Com` actuel. Si le `KreogCom` courant n'est relié à aucun `KreogCom` , alors il sera relié au `KreogCom` nouvellement crée (cf ci-dessous).

```
(---> = relie a )
--------      ----------------------------
| this | ---> | new KreogCom x, y, serial |
--------      ----------------------------
```

Si le `KreogCom` courant était déjà relié à un autre `KreogCom` , alors le `KreogCom` nouvellement crée le remplacera (cf ci-dessous)

```
(---> = relie a )
--------      ----------------------------      ---------------------------------
| this | ---> | new KreogCom x, y, serial | ---> | KreogCom qui etait lie a this |
--------      ----------------------------      ---------------------------------
```

- Il possède une fonction membre `getCom` qui renverra un pointeur sur le `KreogCom` auquel notre `KreogCom` est relié. s'il n'est relié à rien, cette fonction membre renverra 0
- Il possèdera une fonction membre `removeCom` qui enlèvera le `KreogCom` auquel il est relié

Faites attention à ne pas briser la chaine de communication

- il possèdera une fonction membre `ping` , qui affichera les informations suivantes sur la sortie standard (suivi d'un retour à la ligne :

```
1     KreogCom [SERIAL] currently at [X] [Y]
```

[SERIAL], [X] et [Y] étant bien sur les données membres de votre `KreogCom`

Cette fonction membre ne modifiera en rien l'instance appellante

- il possèdera une fonction membre `locateSquad` qui affichera les informations de tous les `KreogCom` auquels il est relié, puis ses propres informations

```
1     [affiche infos KreogCom relies]
2     [affiche infos KreogCom courant]
```

Cette fonction membre ne modifiera en rien l'instance appellante

Lors de la destruction d'un KreogCom, la chaine de communication ne doit absolument pas être brisée

pour ceux qui lisent un sujet jusqu'au bout : Les kreogCom sont chainés entre eux :p

Le code suivant devra compiler et afficher la sortie ci-après :

```
int main()
{
    KreogCom k(42, 42, 101010);
    k.addCom(56, 25, 65);
    k.addCom(73, 34, 51);

    k.locateSquad();

    k.removeCom();
    k.removeCom();

    return 0;
}
```

Sortie :

```
Kinder:ex02$ g++ -W -Wall -Werror *.cpp
Kinder:ex02$ ./a.out | cat -e
KreogCom 101010 initialised$
KreogCom 65 initialised$
KreogCom 51 initialised$
KreogCom 51 currently at 73 34$
KreogCom 65 currently at 56 25$
KreogCom 101010 currently at 42 42$
KreogCom 51 shutting down$
KreogCom 65 shutting down$
KreogCom 101010 shutting down$
Kinder:ex_2$
```

Chapitre V

Exercice 3

	Exercice : 03	points : 5
Lock'n load baby		
Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/ex03`		
Compilateur : `g++`	Flags de compilation: `-Wall -Wextra -Werror`	
Makefile : `Non`	Règles : `n/a`	
Fichiers a rendre : `Phaser.h, Phaser.cpp, Sounds.h`		
Remarques : `n/a`		
Fonctions Interdites : `malloc, free, *printf`		

Maintenant que vous êtes à peu près formés, on va maintenant passer aux choses un peu plus marrantes :

Voici le Phaser Kreog'o Blaster mk2, votre meilleur ami pour le reste de votre vie. A partir de maintenant, vous allez vous entraîner, manger, dormir avec lui. Cette merveille possède 3 modes de tir différents :

- `REGULAR` , ca c'est pour quand vous voulez faire ca proprement.
- `PLASMA` , histoire de réchauffer un peu l'atmosphère.
- `ROCKET` , histoire de pouvoir frapper l'adversaire de manière chirurgicale

Par contre, pour des raisons budgetaires on nous a livré vos armes sous forme de pièces détachées, donc si vous voulez blaster du Dingo, va falloir vous retrousser les manches.

Pour cela vous allez implémenter la classe suivante: (À compléter)

```
class Phaser
{
    public:
        enum AmmoType
        { ... };

    public:
```

```
        Phaser(int maxAmmo, AmmoType type);
        ~Phaser();

    public:
        void fire();
        void ejectClip();
        void changeType(AmmoType newType);
        void reload();
        void addAmmo(AmmoType type);
    public:
        int getCurrentAmmos();

    private:
        static const int Empty;

    private:
        [...]
};
```

Vous devrez aussi implémenter la classe `Sounds` . Cette dernière possédera les variables de classe constantes suivantes:

- `std::string Regular` .
- `std::string Plasma` .
- `std::string Rocket` .

`Ces variables ne devront absolument pas être affectées dans les fichiers que vous devez rendre, nous le feront dans le main de test de la correction`

Explications :

- Un Phaser a un nombre de munition maximum, un nombre représentant le nombre de munitions actuelles présentes dans le chargeur, les sons correspondants aux différents modes de tir, un chargeur contenant ses munitions, ainsi que le type de munition par défaut de l'arme.
- Un Phaser contiendra par défaut un chargeur de 20 balles MAXIMUM, de type `REGULAR`
- Un `Phaser` sera entièrement chargé lors de sa création.
- la variable `Empty` représente un chargeur vide, il vous est très fortement conseillé de l'utiliser dans votre programme. Vous devrez l'initialiser avec la valeur 0.

- Lors d'un appel à la fonction membre `fire` , vous devrez afficher la chaine

```
1     Clip empty, please reload
```

suivi d'un retour à la ligne si le chargeur est vide. Sinon, vous afficherez le son de la munition contenu à la première case du chargeur. Après cela, la taille du chargeur est réduite de 1, éliminant ainsi la munition venant d'être tirée.

- La fonction membre `ejectClip` éjecte le chargeur présent dans l'arme, réduisant ainsi son nombre de munitions à 0.
- La fonction membre `changeType` affichera sur la sorie standard

```
1   Switching ammo to type : [TYPE]
```

suivi d'un retour á la ligne, où [TYPE] vaudra: la valeur du paramètre passée en minuscule (ex : `regular` pour `REGULAR`). Un appel à cette fonction membre change le type par défaut du Phaser

- La fonction membre `reload` affichera

```
1  Reloading ...
```

sur la sortie standard. Elle rechargera ensuite l'arme avec son type de munition par dèfaut.

- La fonction membre `addAmmo` ajoutera une munition du type `type` à la fin du chargeur. Si le nombre de munitions courantes est égal au nombre de munitions maximum de l'arme, afficher la chaîne

```
1     Clip full
```

sur la sortie standard et ne rien faire. Sinon, ajouter la munition au chargeur

- La fonction membre `getCurrentAmmos` renvoie le nombre de munitions présentes dans le chargeur de l'arme.

`La fonction membre` `getCurrentAmmos` devra pouvoir être appellée sur des objets Phaser immuables

Le code suivant devra compiler et afficher la sortie ci-après :

```
int main()
{
    Phaser p(5, Phaser::ROCKET);
    p.fire();
    p.reload();

    std::cout << ‘‘Firing all ammunitions’’ << std::endl;
    while (p.getCurrentAmmos() > 0)
        p.fire();

    return 0;
}
```

Et produire la sortie suivante :

```
Kinder:ex03$ g++ -W -Wall -Werror *.cpp
Kinder:ex03$ ./a.out | cat -e
Boooooooom$
Reloading ...$
Firing all ammunitions$
Boooooooom$
Boooooooom$
Boooooooom$
Boooooooom$
Boooooooom$
```

Dans ce cas, le son d'une roquette est "Boooooooom", à vous de trouver comment l'initialiser

Chapitre VI

Exercice 4

	Exercice : 04	points : 5
G-Squad		
Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d07a-promo-login_x)/ex04`		
Compilateur : `g++`	Flags de compilation: `-Wall -Wextra -Werror`	
Makefile : `Non`	Règles : `n/a`	
Fichiers a rendre : `Skat.h, Skat.cpp, Phaser.h, Phaser.cpp, Sounds.h, KreogCom.h, KreogCom.cpp, Squad.h, Squad.cp`		
Remarques : `n/a`		
Fonctions Interdites : `Aucune`		

Bon les petits gars, maintenant je n'ai plus rien à vous apprendre. Mais avant d'aller défourailler du Dingo, il vous reste à vous equiper. Parce que on est pas du genre à vous lacher en slip de bain sur la banquise.

Explications:

- Un Skat possédera maintenant un `Phaser` et un `KreogCom` .
- vous lui rajouterez un constructeur ayant la signature suivante:

```
Skat(std::string const& m_name, int stimPaks,
     int serial, int x, int y, Phaser::AmmoType type);
```

  Les paramètres `serial` , `x` , `y` et `type` correspondront bien sur aux informations nécessaires à la construction du `KreogCom` et du Phaser.

- Chaque Skat aura un Phaser contenant 20 cartouches.

`Cette classe ne devra plus posseder de constructeur par défaut`

- vous lui rajouterez les fonctions membres suivantes:
  - `void fire()` qui fera tirer votre Skat
  - `void locate()` qui donnera la position de votre Skat
  - `void reload()` qui fera recharger votre Skat
  - `KreogCom& com()` qui retournera le communicateur dont est équipé votre Skat.

`À vous de deviner lesquelles de ces fonctions membres sont des fonctions membres constantes`

- Vous modifierez aussi la classe `KreogCom` en lui rajoutant la fonction membre suivante : `void addCom(KreogCom* com)` , qui reliera le `KreogCom` courant avec celui passé en paramètre.

`Aucun appel à la fonction membre removeCom` ne sera fait dans les corrections pas la peine de la modifier

Bien, maintenant que vous ressemblez enfin à quelque chose, va maintenant falloir vous organiser un peu mieux que ca.

Pour cela, vous implémenterez la classe suivante :

```
1  class Squad
2  {
3      public:
4          Squad(int posXBegin, int posYBegin, Phaser::AmmoType ammoType,
5                int size);
6          ~Squad();
7
8      public:
9          void fire();
10         void localisation();
11
12     public:
13         [...] skats();
14         [...] at(int idx);
15         int size();
16
```

```
17    private:
18        [...]
19 };
```

Les sons de la classe `Sounds` devront être initialisés dans le fichier `Squad.cpp` , avec les valeurs suivantes :

- Regular : “Bang”
- Plasma : “Fwooosh”
- Rocket : “Boouuuuuum” !!!

Explications:

- Une escouade est composee de `[size]` Skats. Par défaut une escouade est composée de 5 `Skat` .
- Le dernier membre d’une escouade devra être 0.
- Les communicateurs de ces derniers possèderont comme serial l’emplacement du `Skat` dans l’escouade (`Skat` 0 -> `com serial` 0, ..., `Skat` n -> `com serial` n)
- Concernant leur position, le premier `Skat` sera situé à position `posXBegin` , `posYBegin` .
- La position en X sera incrémentée de 10 pour chaque Skat présent dans l’escouade.
- La position en Y sera incrémentée de 15 pour chaque Skat présent dans l’escouade.
  `ex : (Skat 0 -> X = 0, Y = 0; Skat 1 -> X = 10, Y = 15; ...;`
  `Skat n -> X = (n) * 10, Y = (n) * 15)`
- Les communicateurs de tous les Skats seront reliés entre eux, selon le schéma suivant :
  `( ---> = relie a )`

```
--------------      --------------      -------      --------------
| com Skat 0 | ---> | com Skat 1 | ---> | ... | ---> | com Skat n |
--------------      --------------      -------      --------------
```

- La fonction membre `fire` fera tirer tous les membres d’une escouade.
- La fonction membre `localisation` affichera la position de tous les membres d’une escouade, à partir du PREMIER membre.
- La fonction membre `skats` vous renverra tous les membres de l’escouade.
- la fonction membre `at` vous renverra le `Skat` situe à l’index `idx` . Si ce dernier est invalide, renvoyer 0.
- La fonction membre `size` vous renverra la taille de l’escouade.

Afin d'interagir plus facilement avec votre escouade, vous implémenterez la fonction suivante : `void foreach([...] beginIdx, [...] actionPtr)` dans les fichiers relatifs à la Squad. Cette fonction prendra en paramètre un index représentant le début d'un ensemble de Skat, et un pointeur sur fonction membre de la classe `Skat`, ne prenant aucun paramètre et retournant `void`.

Cette fonction devra pouvoir être appellè avec toutes les fonctions membres de la classe `Skat` ne prenant aucun paramètre et retournant `void`

Est ce qu'une fonction membre est différente d'une fonction membre const avec exactement la même signature int class::fct() ==? int class::fct() const

Le code suivant devra compiler et afficher la sortie ci-après :

```
int main()
{
    Squad s(0, 0, Phaser::REGULAR);

    s.fire();

    return 0;
}
```

Et produire la sortie suivante :

```
Kinder:ex04$ g++ -W -Wall -Werror *.cpp
Kinder:ex04$ ./a.out | cat -e
KreogCom 0 initialised$
KreogCom 1 initialised$
KreogCom 2 initialised$
KreogCom 3 initialised$
KreogCom 4 initialised$
Bang$
Bang$
Bang$
Bang$
Bang$
KreogCom 0 shutting down$
KreogCom 1 shutting down$
KreogCom 2 shutting down$
KreogCom 3 shutting down$
KreogCom 4 shutting down$
Kinder:ex04$
```

Ca sera tout. Rompez soldats, et bonne chance.