Objet Simples

L'objectif de ce premier TME est de bien se familiariser avec les mécanismes base régissant le fonctionnement d'un objet C++. Plus particulièrement sur les constructeurs & destructeurs, et les contextes dans lesquels ils peuvent être appelés.

Présentation du Code

Comme il a été annoncé en cours un soin tout particulier devra être accordé à la présentation de votre code. A cet effet un exemple type d'indentation vous est fourni ci-après. Toujours pour une meilleure lisibilité, réduisez la taille des tabulations à 2 ou 4 caractères au lieu du defaut de 8. Enfin, n'insérez jamais le caractère *tabulation* dans votre code source, configurez votre éditeur favori pour que celui-çi substitue automatiquement des espaces blanc ordinaires à la place d'une tabulation.

Pour **`vim`**/**`gvim`**, ajouter dans votre `~/.vimrc` les lignes suivantes:

```
:set shiftwidth=2
:set expandtab

:autocmd FileType c,cpp set cindent
:autocmd FileType make set noexpandtab shiftwidth=8
```

Pour **`emacs`**, ajoutez dans votre `~/.emacs` les lignes suivantes:

```
;; CC-Mode configuration.
 (defun local-c-mode-hook ()
  (setq c-echo-syntactic-information-p t
                        c-basic-offset 2
                             tab-width 4
                       indent-tabs-mode nil)
  (c-set-offset 'topmost-intro          0)
  (c-set-offset 'inclass              '++)
  (c-set-offset 'access-label          '-)
  (c-set-offset 'comment-intro         '-)
  (c-set-offset 'arglist-cont-nonempty 'c-lineup-arglist-close-under-paren)
  (c-set-offset 'arglist-close         'c-lineup-arglist-close-under-paren)
  (c-set-offset 'defun-block-intro     '+)
  (c-set-offset 'statement-block-intro '+)
  (c-set-offset 'block-close            0)
  (c-set-offset 'case-label            '+)
  (c-set-offset 'statement-case-intro  '+))
 (add-hook 'c-mode-common-hook 'local-c-mode-hook)

 (setq auto-mode-alist
  (cons '("\\.\\(h\\|hh\\|hpp\\)\\'" . c++-mode)
   auto-mode-alist))
```

Compilation

Dans le cadre de ce premier TME, nous allons nous contenter de faire de la compilation monolithique. C'est à dire que tout le code de chaque exercice sera contenu dans un seul fichier source, par exemple `Box.cpp`. L'éxécutable sera créé par compilation de cet unique fichier.

Compilation monolithique d'un fichier source:

```
g++ -Wall -g -o box Box.cpp
```

Ici, l'éxécutable généré sera `box`. Attention l'argment `-o` doit immédiatement être suivi du nom de l'éxécutable (`box`).

Autres arguments donnés à `g++`:

- `-g` demande d'inclure le code nécessaire pour pouvoir utiliser un débogueur (`gdb` ou `ddd`).
- `-Wall` demande d'afficher tous les avertissements. Vous devrez toujours utiliser ce drapeau (`flag`) et à l'issue de la mise au point, votre programme ne devra générer aucun avertissement.

Mon Premier Objet: `Box`

Nous allons implémenter la classe `Box` présentée en cours.

Attention, dans le récapitulatif suivant, les prototypes des fonctions membre ne sont pas forcément donnés de façon exacte. Votre travail consiste à les achever conformément à ce qui a été vu en cours (et au bon sens).

La classe `Box` comporte les cinq attributs suivants:

- `_name` : le petit nom de l'objet courant (de type `std::string`).
- `_x1, _y1, _x2, y2` : les coordonnées des deux angles de la boite (de type `int`).

Les coordonnées `_x1` et `_y1` (respectivement `_y1` et `_y2` seront maintenues ordonnées telles que `_x1 <= _x2`. Une boite sera considérée comme vide si `_x1 > _x2` ou `_y1 > _y2`.

Constructeurs: (*CTOR*)

- `Box()`, constructeur par défaut. Doit construire une boîte vide.
- `Box(const std::string&, int x1, int y1, int x2, int y2)`, constructeur *ordinaire*.
- `Box(const Box&)`, constructeur par copie. Pour plus de clarté, lorsqu'une boite sera copiée on s'autorisera à en modifier le nom en y ajoutant le suffixe `"_c"`.

Destructeur: (*DTOR*)

- `~Box()`, destructeur (unique).

Accesseurs: (*accessors*)

- `getName()`
- `getX1()`
- `getY1()`
- `getX2()`
- `getY2()`

- `isEmpty()`
- `intersection(const Box&)`
- `print(std::ostream&)` : affiche l'état de la `Box` dans un flux, on choisi le format suivant: `<"NAME" x1 y1 x2 y2>`.

Modificateurs: (*mutators*)

- `makeEmpty()`
- `inflate(int dxy)`
- `inflate(int dx, int dy)`
- `inflate(int dx1, int dy1, int dx2, int dy2)`
- `getIntersection(const Box&)`

Afin de pouvoir bien suivre les appels aux constructeurs & destructeurs durant l'éxécution du programme, nous allons ajouter des affichages dans les corps de ces fonctions. Ils devront afficher: (huit espaces blancs en début de ligne)

```
Box::Box() <"NAME" x1 y1 x2 y2>
Box::Box(const std::string&, ...) <"NAME" x1 y1 x2 y2>
Box::Box(const Box&) <"NAME" x1 y1 x2 y2>
Box::~Box() <"NAME" x1 y1 x2 y2>
```

Question 1

Implanter la classe `Box` comme spécifié précedemment. Pour la valider, on utilisera le programme de test fourni çi-après. L'éxécution de ce programme va produire une trace d'éxecution indiquant à quel endroits les contructeurs et destructeurs sont appelés. Commentez la trace en mettant en correspondance les appels et les instructions du programme.

Programme de test à utiliser pour valider votre classe:

Question 2

Vérification des droits d'accès: tour à tour, déplacer chaque constructeur dans la partie `private` de la classe `Box`. Recompiler, que se passe-t-il et pourquoi.

Question 3

On désire ajouter une nouvelle méthode `getCenter()` à la classe, qui renvoie les coordonnées `(x,y)` du centre de la boite. Proposer une implantation de cette méthode ainsi qu'un petit programme de test.

Question 4

Exercice de style:

Pour illustrer les concepts d'`API` et d'encapsulation, nous allons changer la représentation interne de la `Box`. Les attributs deviennent:

- `_x` et `_y`: le centre de la boite.
- `_width` et `_height` : la taille de la boite.

Réimplémenter la classe en utilisant ces nouveaux attributs. Quelles conséquences cela a-t-il sur le programme de test?