TME 2 -- Objets Simples

Contents

TME 2 -- Objet Simples

L'objectif de ce deuxième 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

Un développeur consacre un temps important à l'écriture (et la correction) de son code. Avoir une bonne maîtrise de son éditeur de texte est nécessaire. Aucun éditeur ne vous est imposé, en revanche, vous devez l'utiliser de façon efficace et il doit être capable de reconnaître du code C++.

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 défaut de 8. Enfin, n'insérez jamais le caractère `tabulation` dans votre code source, configurez votre éditeur favori pour que celui-ci substitue automatiquement des espaces blancs ordinaires à la place d'une tabulation.

Pour **vim**/**gvim**, ajouter dans votre **~/.gvimrc** 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))
```

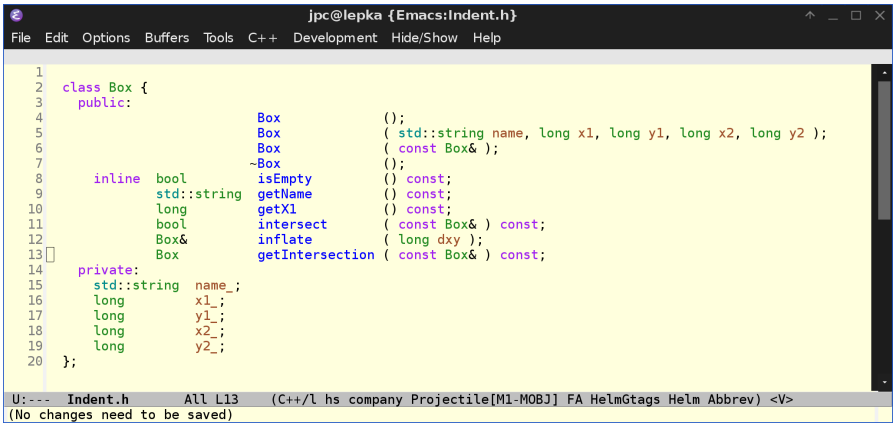

Figure 1: Figure 1 -- Exemple d'indentation du code C++

Organisation des Répertoires

On respectera l'arborescence décrite dans Organisation du Code & Compilation

Compilation

Ce deuxième TME ne contiendra que quatre fichiers:

1. `Box.h` : la *déclaration* de la classe `Box`.
2. `Box.cpp` : la *définition* de la classe `Box`.
3. `Main.cpp`: le programme de test.
4. `CMakeLists.txt`: configuration de `cmake`.

Configuration de `cmake`

Le contenu fichier `CMakeLists.txt` pour ce TME vous est fourni:

```
# -*- explicit-buffer-name: "CMakeLists.txt<M1-MOBJ/tme2>" -*-
#
# Pour voir les commandes lancées par cmake/make, utiliser:
#     > cmake -DCMAKE_VERBOSE_MAKEFILE:STRING=YES ../src

 cmake_minimum_required(VERSION 2.8.0)
 project(TME2)

 set( CMAKE_CXX_FLAGS            "-Wall -g" CACHE STRING "C++ Compiler Release op
 set( CMAKE_INSTALL_PREFIX       "../install" )

    include_directories( ${TME2_SOURCE_DIR} )

                    set( includes     Box.h )
                    set( cpps         Box.cpp
                                      Main.cpp
                       )
          add_executable( tme2        ${cpps} )

                 install( TARGETS tme2        DESTINATION bin )
                 install( FILES   ${includes} DESTINATION include )
```

Mon Premier Objet: Box

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

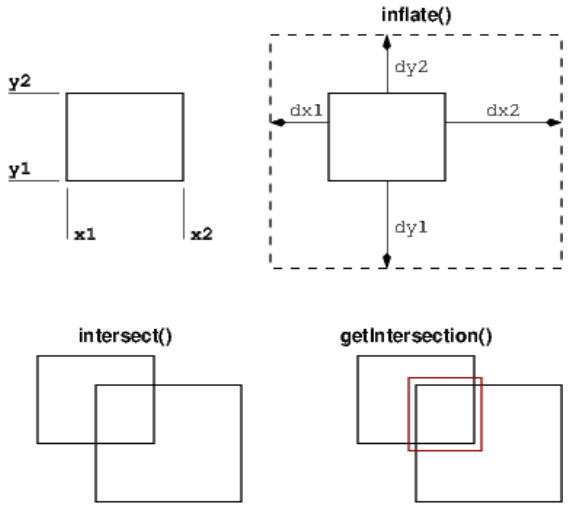

Figure 2: Figure 2 -- Spécification de la classe Box

Note

Dans le récapitulatif suivant, les prototypes des fonctions membre ne sont pas forcément donnés de façon exacte. Votre travail consiste à les compléter 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 boîte (de type **`long`**).

Les coordonnées `x1_` et `y1_` (respectivement `y1_` et `y2_` seront maintenues ordonnées telles que `x1_ <= x2_`. Une boîte 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, long x1, long y1, long x2, long y2)`, constructeur *ordinaire*.
- `Box(const Box&)`, constructeur par copie. Pour plus de clarté, lorsqu'une boîte 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()`
- `getWidth()`
- `getHeight()`
- `isEmpty()`
- `intersect(const Box&)`
- `print(std::ostream&)` : affiche l'état de la `Box` dans un flux, on choisit le format suivant: `<"NAME" x1 y1 x2 y2>`.

Modificateurs: (*mutators*)

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

Afin de pouvoir bien suivre les appels aux constructeurs & destructeurs durant l'exécution du programme, nous allons ajouter des affichages dans les corps de ces fonctions. Ils devront afficher:

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

Travail Demandé

Question 1

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

On mettra l'ensemble de la classe à l'intérieur d'un **`namespace`** `tme2`.

Programme de test à utiliser pour valider votre classe:

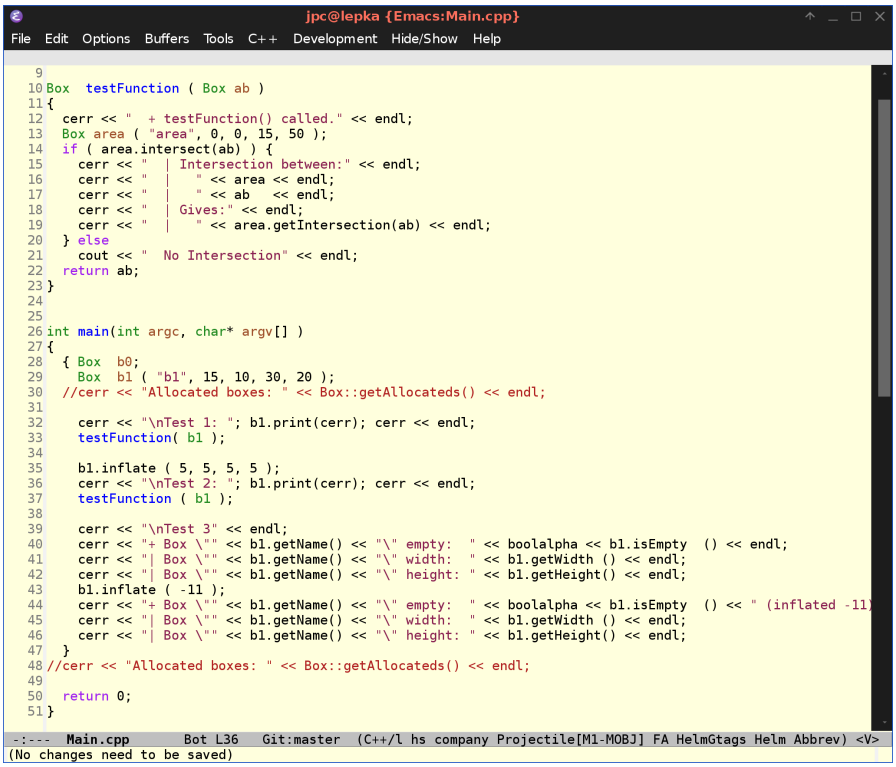

Figure 3: Figure 3 -- Test de la classe Box

On utilisera le test avec affichage complet dans un premier temps, pour suivre la création et la destruction des objets, puis dans un deuxième temps, en filtrant l'affichage des lignes de Debug, pour obtenir de façon plus claire le résultat du programme.

Note

Filtrage de l'affichage de Debug. Pour retrouver une trace d'exécution proche de celle fournie au TME 1, vous pouvez utiliser la commande suivante, qui filtre toutes les lignes contenant le terme *Debug* :

```
etudiant@pc:work> ../install/bin/tme2 2>&1 | grep -v Debug
```

Explication de cette ligne de commande un peu alambiquée :

- `2>&1` redirige le flot de sortie d'*erreur* (i.e. `cerr`) dans le flot de sortie *standart* (i.e. `cout`).
- `|` *pipe* la sortie standart du programme `tme2` sur l'entrée standard d'un autre, ici : `grep`
- `grep -v Debug` affiche les lignes de l'entrée standart *ne comportant pas* le terme *Debug*.

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

Mise en oeuvre des fonctions **`inline`** : mettre en **`inline`** toutes les fonctions de la classe `Box` ne comportant qu'une seule instruction.

Question 4

Nous allons maintenant utiliser les surcharges d'opérateurs:

1. Écrire la surchage d'opérateur pour l'affichage dans un flux, modifier le code en conséquence partout où cela est pertinent (dans et en dehors de la classe).
2. On désire ensuite utiliser l'opérateur `and` pour faire l'intersection entre deux `Box` (i.e. comme `getIntersection()`). Comment faire et où cela permet-il de changer le code ?

Question 5

On souhaite connaître à tout moment le nombre d'objets `Box` alloués. En vous référant au cours, proposer une solution (élégante).

Question Facultative

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 boîte.
- `width_` et `height_` : la taille de la boîte.

On mettra cette classe alternative dans un **`namespace`** `tme2Qf`. Que faudra-t-il changer dans `Main.cpp` pour utiliser cette seconde implantation?

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