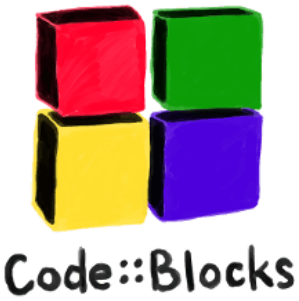
Travaux Pratiques

Classes – Objets - Constructeurs



Extrait du référentiel : BTS CIEL option A (Informatique et Réseaux)		Niveau(x)
C08 CODER	Langages de développement, de description, de création d'API et les IDE associés	4

Objectifs du TP :

- Formatage des entrées/sorties
- Création d'un objet sans attribut
- Utilisation d'un constructeur et d'un destructeur
- Exercice : Le cercle
- Surdéfinition – arguments par défauts :
  - surdéfinition de méthodes et initialisation d'objets
  - surdéfinition et arguments par défaut

Support d'activité :

- Logiciels : CodeBlocks et suite bureautique
- Fichiers : Objets-Classes-Constructeurs.PDF
- Ce document au format PDF

Vous rédigerez un compte-rendu numérique.
Pensez aux captures d'écran pour imager votre compte-rendu.
Sauvegardez votre travail régulièrement !
Des modifications peuvent exister selon la version du logiciel utilisée.

Question 1

Lisez attentivement le document lié au fichier « **Objets-Classes-Constructeurs.pdf** » se trouvant dans le dossier « **Support** » de l'activité.

FORMATAGE DES ENTRÉES/SORTIES

Soit le code source ci-dessous :

```
#include <stdio.h>

int main()
{
 int n ;
 float x ;

 printf("Donnez un entier et un flottant\n") ;
 scanf("%d %e", &n, &x) ;
 printf("Le produit de %d par %e\nest : %e", n, x, n * x) ;
 return(0) ;
}
```

Question 2

Saisissez le code C ci-dessus sous CodeBlocks puis compiler et exécuter le programme.

Réécrivez le code en ne faisant appel qu'aux nouvelles possibilités d'entrées-sorties du C++.

Compilez puis ***exécutez*** le nouveau code source en C++.

Obtenez-vous la même chose ?

OBJET SANS ATTRIBUTS

Soit le code source ci-dessous :

```
#include <iostream>

using namespace std ;

class bateau
{

} ;

int main()
{
 bateau MonBateau ;
 cout << "Objet MonBateau creer !" << endl ;
```

```
 return 0 ;
}
```

Question 3

Saisissez le petit programme ci-dessus (tout le code dans le même fichier source) sous CodeBlocks puis exécuter-le.

Que constatez-vous ?

UTILISATION D'UN CONSTRUCTEUR ET D'UN DESTRUCTEUR

Soit le code (donné de façon « **brute** ») ci-dessous :

```
#ifndef BATEAU_H
#define BATEAU_H

class bateau
{
 private :
 int longueur,largeur ;
 public :
 bateau(int,int) ;
 ~bateau() ;
 int calculersuperficie() ;
} ;
#endif

#include <iostream>
using namespace std ;
#include "Bateau.h"
bateau::bateau(int longue,int large)
{
 cout << "Je suis le constructeur" << endl ;
 longueur=longue ;
 largeur=large ;
}
bateau::~bateau()
{
 cout << "Je suis le destructeur" << endl ;
}
int bateau::calculersuperficie()
{
 return(longueur*largeur) ;
}

int main()
```

```
{
 int lon,lar ;
 bateau MonBateau ;

 cout<<"Longueur du bateau: " ;
 cin>>lon ;
 cout<<"Largeur du bateau: " ;
 cin>>lar ;
 bateau baigneur(lon,lar) ;
 cout<<"La superficie est de: "<<baigneur.calculersuperficie()<<"\n" ;
 return 0 ;
}
```

Question 4

Encadrez sur le programme ci-dessus :

En vert : la déclaration de la classe ;
En bleu : la définition du constructeur ;
En rouge : la définition du destructeur ;
En jaune la déclaration du constructeur ;
En orange la déclaration du destructeur ;
En noir la fonction principale.

Question 5

Créez un projet nommé « **Bateau** » sous CodeBlocks.

Saisissez le code source en créant **une modularité** du projet (un fichier d'en-tête « `Bateau.h` », un fichier « `Bateau.cpp` » et un fichier « `Main.cpp` »).

Déboguez le programme.

Compilez et ***exécutez*** le programme.

Vous n'arrivez toujours pas à obtenir l'exécutable ?
Ajoutez un constructeur par défaut (donc sans paramètre), sa déclaration dans le fichier d'en-tête `Bateau.h` et sa définition dans le fichier `Bateau.cpp`.
Compilez et exécutez le programme.

Compilez de nouveau puis ***exécutez*** le programme en mode débug (en pas à pas).

Ouvrez une fenêtre de suivi des variables locales.

Pour l'objet « `MonBateau` », quelles sont les valeurs de ses attributs longueur et largeur ?

Observez les différents appels du (des) constructeur(s) et du destructeur.

Indiquez dans quel ordre sont appelés le constructeur et le destructeur pour les deux objets présents dans la fonction `main( )`.

Il vous faut constater que les constructeurs sont appelés une fois et que le destructeur est appelé deux fois.

EXERCICE : LE CERCLE

On désire écrire une application qui calcule, puis affiche le périmètre et la surface d'un cercle.
Vous « construirez » une classe **`CCercle`**.
On demande d'utiliser un constructeur pour initialiser les paramètres du cercle.
Vous devrez déterminer au préalable les membres privés et les membres publics.

Désormais, vous ajouterez pour le nom des classes un **C** majuscule devant le nom de la classe.

Vous allez utiliser une valeur constante pour le nombre **`PI`** que vous déclarerez dans le fichier d'en-tête mais en dehors de la classe avec la ligne de code suivante :

```
const float pi = 3.14 ;
```

Question 6

Créez un projet « `Cercle` » sous CodeBlocks.

Saisissez le code source en créant **une modularité** du projet.

Compilez puis ***exécutez*** le programme.

SURDÉFINITION - ARGUMENTS PAR DÉFAUT

SURDÉFINITION DE MÉTHODES ET INITIALISATION D'OBJETS

Soit le code ci-dessous (sans modularité) :

```
class CSosie
{

private : // rien

public :

CSosie() ; // constructeur 1 par défaut
CSosie(int) ; // constructeur 2
CSosie(double) ; // constructeur 3
CSosie(int, float) ; // constructeur 4
CSosie(float, int) ; // constructeur 5

} ;

CSosie::CSosie()
```

```
{
}
CSosie::CSosie(int a)
{
cout << "sosie numero 1 a = " << a << "\n" ;
}
CSosie::CSosie(double a)
{
cout << "sosie numero 2 a = " << a << "\n" ;
}
CSosie::CSosie(int a, float b)
{
cout << "sosie numero 3 a = " << a << "\tb = " << b << "\n" ;
}
CSosie::CSosie(float a, int b)
{
cout << " premier argument : " << a << "\n" ;
cout << " second argument : " << b << "\n" ;
}

int main()

{

int n = 5 ;
float x = 2.5, y = 12.6 ;
unsigned char c = 7 ;
int m = 10, p = 20 ;
double z = 1258.9 ;

CSosie objet1(n) ;
CSosie objet2 = x ;
CSosie objet3(m, y) ;
CSosie objet4 = CSosie(x, p) ;
CSosie objet5(c) ;
CSosie objet6(m, p) ;
CSosie objet7(n, c) ;
CSosie objet8(p, z) ;
CSosie objet9(z, z) ;
CSosie objet10 ;
CSosie objet11 = objet3 ;
CSosie objet12(2, 8.35) ;
objet12 = objet3 ; // Affectation d'un objet sur un objet déjà créé
return 0 ;

}
```

Question 7

Sans exécuter le code source ci-dessus, ***déterminez*** quelle est la méthode (ou les méthodes) qui sera (ou seront) appelée(s) (pour chacun des appels réalisés dans la fonction principale) et indiquer le type des arguments envoyés.

Complétez le tableau ci-dessous selon l'exemple donné pour l'objet 1.

Appel de	Appel possible ? (**OUI**/**NON**)	Méthode(s) appelée(s) - Type(s) des arguments	Ambiguïté ? (**OUI**/**NON**)
Objet 1	**OUI**	Constructeur 2 - int	**NON**
Objet 2			
Objet 3			
Objet 4			
Objet 5			
Objet 6			
Objet 7			
Objet 8			
Objet 9			
Objet 10			
Objet 11			
Objet 12			

Question 8

Saisissez le code source ci-dessus en créant une modularité du projet dans un projet nommé « `Surdef` ».

Modifiez le code source afin de lever toute ambiguïté en mettant en commentaire les lignes faisant défaut !

Compilez puis ***exécutez*** le programme en pas à pas. ***Vérifiez*** le numéro du constructeur appelé pour chaque objet.

Comparez « la pratique ou le réel » avec votre théorie « le tableau ».

Pourquoi la définition du constructeur 1 est-elle vide ? Il n'y a rien dans le bloc.

L'objet 10 est-il initialisé ? Si oui, pour quelles valeurs ?

Que se passe-t-il pour l'objet 11 ? Est-il correctement initialisé ? À votre avis comment et par quel constructeur ? Vérifiez avec la fenêtre de suivi des variables.

L'objet 11 a bien été initialisé. Mais par quel contructeur ? Vous le verrez un peu plus tard…

SURDÉFINITION ET ARGUMENTS PAR DÉFAUT

Soit le code ci-dessous (sans modularité) :

```
#include <iostream>
using namespace std ;

class CSosie
{

private : // rien

public :
CSosie() ; // constructeur 1 par défaut
CSosie(int) ; // constructeur 2
CSosie(double) ; // constructeur 3
CSosie(int, int = 15) ; // constructeur 4 avec argument par défaut

void fct1(int, int = 12 ) ; // méthode fct1 avec argument par défaut
void fct1(int, float = 13.7, int = 15) ; // // méthode fct2 avec 2 arguments
par défaut
} ;
CSosie::CSosie()
{
}
CSosie::CSosie(int a)
{
cout << "sosie numero 1 a = " << a << "\n" ;
}
CSosie::CSosie(double a)
{
cout << "sosie numero 2 a = " << a << "\n" ;
}
CSosie::CSosie(int a, int b)
{
cout << "sosie numero 3 a = " << a << "\tb = " << b << "\n" ;
}
void fct1(int a, int b)
{
cout << " premier argument : " << a << "\n" ;
cout << " second argument : " << b << "\n" ;
}
void fct1(int a, float x, int b)
{
cout << " premier argument : " << a << "\n" ;
cout << " second argument : " << x << "\n" ;
cout << " troisieme argument : " << b << "\n" ;
}

int main()
```

```
{
int n = 5 ;
float x = 2.5 ;
int m = 10, p = 20 ;
CSosie objet1 ; // Appel 1
CSosie objet2(n) ; // Appel 2
CSosie objet3(x) ; // Appel 3
CSosie objet4(m, n) ; // Appel 3
objet1.fct1(m , p) ; // Appel 4
objet1.fct1(m) ; // Appel 5
objet1.fct1() ; // Appel 6
objet1.fct1(n, x) ; // Appel 7
objet1.fct1(p, x, m) ; // Appel 8
return 0 ;

}
```

Question 9

Sans exécuter le code source ci-dessus, ***déterminez*** quelle est la méthode (ou les méthodes) qui sera (ou seront) appelée(s) (pour chacun des appels réalisés dans la fonction principale) et indiquer le type des arguments envoyés.

Complétez le tableau ci-dessous selon l'exemple donné pour l'appel 1.

Appel de	Appel possible ? (**OUI**/**NON**)	Méthode(s) appelée(s) - Valeur(s) des arguments envoyés	Ambiguïté ? (**OUI**/**NON**)
Appel 1	**OUI**	Constructeur 1 - aucune	**NON**
Appel 2			
Appel 3			
Appel 4			
Appel 5			
Appel 6			
Appel 7			
Appel 8			

Question 10

Saisissez le code source ci-dessus en créant une modularité du projet dans un projet nommé « ArguDefaut ».

Modifiez le code source afin de lever toute ambiguïté en mettant en commentaire les lignes faisant défaut !

Compilez puis exécuter le programme en pas à pas. ***Vérifiez*** le numéro du constructeur appelé et les valeurs affectées à chaque objet.

Comparez « la pratique » avec votre théorie « le tableau ».