

Du C# au C++

420-V31-SF – Programmation de Jeux Vidéo III

Le C#

Framework.

Tout est objet

Les langages de programmation

420-V31-SF – Programmation de Jeux Vidéo III

Le grand débat: Le meilleur paradigme

Paradigme :

Style auquel appartient la structure et la philosophie d'un langage de programmation.

Plusieurs langages, plusieurs paradigmes:

*Programmation structurée (COBOL, C) versus non-structurée (BASIC).
Principale différence: non-structurée permet les « GOTO »*

Programmation orientée-objet (Simula, Smalltalk, éventuellement C++, Java, C#, VB)

Programmation fonctionnelle (LISP, ML): tout est une fonction mathématique

Survol des langages de programmation

Génération	Description	Exemples
1	Valeurs numériques seulement (relatif au type de processeur)	Langage machine
2	Ajout de codes mnémotechniques aux valeurs numériques	Assembleur
3	Programmation structurée Langages procéduraux Programmation orientée objet	COBOL Pascal Fortran BASIC C, C++
4	Interface graphique Programmation événementielle Outils d'aide à la conception (rapport, requête, ...)	Java Visual Basic C# .NET
5	Intelligence artificielle Programmation fonctionnelle Robotique	LISP Prolog Smalltalk

Évolution du langage C et dérivés

Langage	Caractéristiques
C	Pour applications DOS, programmation système, développements de systèmes d'exploitation, applications scientifiques, applications de gestion... Très performant quant au code compilé
C++	Ajout de la programmation orientée objet Utilisation divers plug-in ou API (Windows API, MFC, QT, Open GL, SFML, etc). Assez complexe pour un programmeur débutant ou intermédiaire. Tout aussi performant que le C
C#	Très fortement lié avec avec le framework .NET Certaines simplifications par rapport au C++ Protection accrue face aux erreurs de programmation Tout est objet (même les types simples)!

420-V31-SF – Programmation de Jeux Vidéo III

Le C++: Variables et constantes

Déclaration de variable

Une variable doit être initialisée avant d'être utilisée (PAS D'INITIALISATION PAR DÉFAUT: votre variable sera utilisable mais contiendra du "junk")

Ex:

```
int x;
```

```
x+= 5;
```

Ce code est fonctionnel, compile et s'exécute, mais x prendra le contenu déjà présent de la mémoire où il a été utilisé. Son contenu est absolument imprévisible.

Déclaration de constante

Une constante en C++

```
const int MAX_ETUDIANT = 25;
```

Dans du code C++, quelque chose qui ressemble à ça! Il s'agit de valeurs de pré-compilation dans du code (const n'existe pas en C)

```
#define MAX_ETUDIANT 25
```

L'effet précis de cela: le compilateur va remplacer directement dans le code toutes les instances de MAX_ETUDIANT par 25, et ce, juste avant la compilation (Ça reste peu recommandé).

420-V31-SF – Programmation de Jeux Vidéo III

Conversion (Casting) en C++

Conversion

En C++, la conversion (Casting) entre les types de base est implicite.

Un plus petit sera converti automatiquement en un plus grand.

Un plus grand verra ses bits de trop enlevés (avec des résultats parfois imprévisibles)

Un intégral sera converti en nombre à la virgule flottante

Un flottant verra sa partie décimale tronquée

Conversion

On peut assigner un numérique à un booléen

0 donnera false et toute autre valeur donnera true

Un booléen attribué à un numérique donnera 1 si true et 0 si false

while (1) est d'une certaine manière équivalent à while(true) pour faire une boucle infinie dans un while. Par contre ça implique une opération de conversion à chaque boucle, donc évitez de le faire.

420-V31-SF – Programmation de Jeux Vidéo III

Chaînes de caractères

Pour la console

Pour pouvoir utiliser le jeu de caractère par défaut de la machine (donc normalement sur votre machine, les caractères canadien français) :

```
setlocale(LC_ALL, "");
```

Pour éviter la fermeture de la console en mode debug:

```
system("Pause");
```

string

Prérequis:

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

Déclaration

```
string s1 = "Allo";
string chemin =
"c:\\Daffy\\LibrairiesProgrammation";
```

En C++, une string est considéré comme un tableau de char. On peut accéder à chacun des caractères d'une chaîne en précisant son indice débutant à 0

```
string s1 = "Allo";
char c = s1[0]; //c aura la valeur A
```

string

Lecture

```
#include <iostream>
#include <string>

// Problème : arrête de lire au premier
// espace rencontré
cin >> s1;

// Lit au complet jusqu'au ENTER
getline (cin, s1);
```


string

fichage

```
#include <iostream>
#include <string>
#include <iomanip> // pour setw

cout << s1;

// Affiche sur 8 caractères
cout << setw (8) << s1;
cout << s1 << endl; // Change de ligne à
                    // chaque 8 caractères
```

string

Comparaison

```
if (s1 == s2) // Retourne un booléen
```

Note : Tous les autres opérateurs de comparaisons sont valides (==, !=, <, <=, >, >=)

Copie

```
string s1 = "Allo";  
string s2 = s1;
```

Concaténation

```
string s1 = "Allo";  
s1 += " tout le monde!";  
string s2 = s1 + " Comment ça va?";
```

string

Longueur

```
string chaine = "allo";  
int longueur;  
longueur = chaine.length();
```

Insertion

```
string s1 = "Allo mes amis";  
// Donne « Allo à tous mes amis »  
s1 = s1.insert (5, "à tous ");
```

Suppression

```
string s1 = "Allo mes chers amis";  
// Donne « Allo mes amis »  
s1 = s1.erase (9, 6);
```

string

Remplacement d'une sous-chaîne

```
string chaine = "allo les amis";  
// Donne « allo mes amis »  
chaine = chaine.replace (5, 1, "m");
```

Permutation de 2 chaînes

```
string s1 = "bonjour monsieur";  
string s2 = "bonsoir madame";  
s1 = s1.swap (s2);
```

string

Recherche d'une sous-chaîne

```
string chaine = "allo les amis";  
string mot;  
mot = chaine.substr (9, 4); // amis
```

Position d'une sous-chaîne

```
string chaine = "allo les amis";  
int position;  
  
// Retourne 5  
// (recherche à partir de la position 0)  
position = chaine.find ("les", 0);  
  
// Retourne -1 car la sous-chaîne  
// n'est pas trouvée  
position = chaine.find ("salut", 0);
```

string

Il y a encore pas mal de possibilités sur String, mais bon, vous en avez fait pas mal en C# et pas mal tout ce que vous avez fait en C# est valable en C++. Il s'agit de fouiller un peu

420-V31-SF – Programmation de Jeux Vidéo III

Fonctions et paramètres en C++

Utilisation de paramètres

Envoie une COPIE de la variable originale. Il est donc impossible de modifier la variable originale.

```
void NomFonction(type variable)
```

```
void maFonction(int param1, int param2)  
{ ... }
```

```
int ageDuCapitaine  
(int nombreBidon, double autreNombreBidon,  
bool conditionQuelconque)  
  
{ return 0;}
```


Utilisation de paramètres

Attention, en C++, les instances de classe sont transmis par valeur aussi, donc copiées; attention aux copie lourdes.

Utilisation de paramètres

Paramètre par référence :

Envoie une référence de la variable originale. La fonction peut donc la modifier. Les tableaux sont toujours envoyés en référence.

```
void NomFonction(type & variable)  
void NomFonction(type tableau[ ] )
```

420-V31-SF – Programmation de Jeux Vidéo III

Programmation Orientée-Objet

struct vs. class

Parfois vous verrez le mot struct à la place de class, son utilisation est peu recommandé, mais ça existe.

Struct existait en C mais était bien moins élaboré

Par défaut, pour une structure(struct), tout est public et pour une classe(class), tout est privé.

C'est la seule différence.

Programmation Orientée-Objet (mode C++)

```
class Etudiant // Définition de la classe.  
{  
public:  
    bool inscrire(string cours);  
private: // Partie des déclarations privées.  
    int matricule;  
    int moyenne;  
    string cours[7];  
}; // Ne pas oublier le point-virgule ici.  
  
// Implantation des méthodes  
bool Etudiant::Inscrire(string cours)  
{  
    //...  
}
```

Pointeur interne

En POO, un objet peut se référencer lui-même, afin de retrouver ses propres données.

En java et en C#, on a la référence interne qui se nomme *this*.

Attention, en C++, *this* est un **pointeur** interne, avec tout ce qui vient avec.

Définition de méthode :

Il faut coder les méthodes en spécifiant à quelle classe elle se rapporte

```
void Point::initialise(int x, int y)
{
    this->x = x; //Notez la flèche au lieu du point
    this->y = y;
}
```

Note : Pour les variables membres (x et y dans ce cas-ci), on les précède du pointeur implicite `this` avec l'opérateur `->` pour éviter la confusion.

On utilise `->` au lieu du `.` Pour accéder aux méthodes quand il s'agit d'un pointeur.

Constructeur de classe:

On peut programmer un constructeur (appelée à la création de l'objet) et un destructeur (appelée à la destruction de l'objet)

Le constructeur a le même nom que la classe:

```
Point::Point()  
{  
    this->x = 0;  
    this->y = 0;  
}
```

Note: Il existe aussi un destructeur, qui a le même nom que la classe précédé du caractère ~. Il sert surtout à libérer les ressources utilisées par l'objet lorsque celui-ci cesse d'exister. (ex.: `Point::~~Point();`)

420-V31-SF – Programmation de Jeux Vidéo III

C++: .h versus .cpp

C++: .h versus .cpp

Lorsqu'on définit une classe, on place:

La définition de la classe et la déclaration de ses méthodes dans un **fichier .h**.

L'implantation des méthodes (le code) dans un **fichier .cpp** du même nom.

Pour utiliser la classe dans un autre fichier, on ajoute **#include "etudiant.h"** (guillemets doubles et le nom de la classe en question).

C++: .h versus .cpp

ATTENTION!!!

On pourrait en théorie définir les méthodes dans le .h et se passer complètement de tout .cpp, sauf celui du main.

Mais c'est totalement contraire à la philosophie du C++

Pourquoi donc?

C++: .h versus .cpp

1- La compilation

Le compilateur va compiler individuellement chaque fichier .cpp et va vérifier sa cohérence interne. Si un fichier .h et son .cpp lié ne changent pas, la compilation n'est pas refaite pour ceux-ci.

La compilation regarde au niveau des fonctions externes à la classe

Si elles sont déclarées

Si leur signature correspondent.

Si ça a du sens, le compilateur ne s'occupe pas du contenu des fonctions externe.

Pour des systèmes massifs, comme des jeux AAA, c'est un gain de temps énorme.

C++: .h versus .cpp

2- La programmation multi-système

Imaginez du code côté engin de jeu où on doit lancer du un objet graphique vers le module d'affichage de la console.

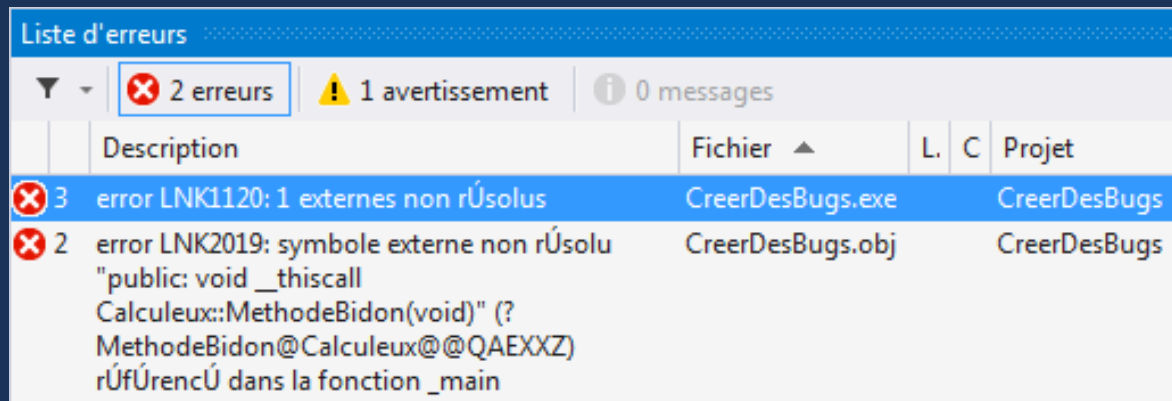
Peu importe la console, la définition de la classe et de ses méthodes sera identique (même .h),

Mais le code précis d'échange avec la machine sera différent (donc contenu du .cpp différent). Même principe si vous faites une application C++ pour PC , MAC, et Linux

Reste que dans les deux cas, c'est la même classe. Et peut-être même carrément le même fichier .h. Pour éviter des bugs et augmenter la compatibilité, on essaie de faire le moins de changement possible.

C++: .h versus .cpp

Voici une erreur terrorisante... Not!



	Description	Fichier ▲	L.	C	Projet
3	error LNK1120: 1 externes non résolus	CreerDesBugs.exe			CreerDesBugs
2	error LNK2019: symbole externe non résolu "public: void __thiscall Calculeux::MethodeBidon(void)" (? MethodeBidon@Calculeux@@QAEXXZ) référéncé dans la fonction _main	CreerDesBugs.obj			CreerDesBugs

C'est une erreur de linkage. Elle se produit presque tout le temps quand on essaie d'utiliser une fonction déclarée, mais non-définie.

La compilation interne de chaque cpp est bonne, mais quand le compilateur essaie de faire le lien entre la déclaration de la méthode et sa définition, il ne trouve pas la seconde.