C/C++ Bases

Introduction

Principales caractéristiques du langage C++

-compilé
-typage fort
-orienté objet
-permet le C
-pointeurs, références
-pas de garbage collector (gestion de la mémoire, memory leaks)

L'environnement de développement

-IDE Code::blocks
-Compilateur MinGW (Minimalist GNU for Windows)
-Gestionnaire de source Bitbucket
-Générateur de doc Doxygen

Critères de qualité dans un développement logiciel

-Lisibilité du code
-Commenter le code
-Optimisation du code

Vue d'ensemble des langages Orientés Objet

Il existe plusieurs langages OO : Java, Python, PHP, Javascript...

Programmation fonctionnelle : syntaxe

Données et expressions

int, float, bool, char, string, double
signed / unsigned
int a(val), b(val2)

Convention de nommage des variables: motMotMot (camelCase)

If elseif else
switch
while
for
do {} while ()

Variables et constantes, portée d'une variable (scope)

```
#define DIRECTORY_PATH "C:/..."
#define MACONSTANTE valeur

const type nomVar = val;
```

Fonctions et utilisation du code C dans un programme C++

```
type nom(var)
{
	return;
}
```

Print et Scan

```
printf(“machaine %d”,val)
printf(“Il est %d heures et %d minutes”,valHeure, valMinute)

scanf(“%d”,&maVal);
```

Lors d’un scanf il faut utiliser un passage par adresse et non par valeur car on souhaite mettre un résultat à l’adresse de la variable.

```
#define BIT(x) (1<<(x))
enum collisiontypes {
	COL_NOTHING = 0, //<Collide with nothing
	COL_RAY = BIT(0), //bit0 used for raytest
	COL_CHAR = BIT(1),
	COL_WALL = BIT(2),
	COL_AMMO = BIT(3),
	COL_ITEM = BIT(4),
	COL_TRIGGER = BIT(5)
```

};

Opérateurs et instructions de contrôle

= attribution de valeur
== égal (valeur)
! non
!= différent (valeur)
> supérieur
< inférieur
>= supérieur ou égal
<= inférieur ou égal
++ incrémente (ajoute 1)
-- décrémente (retire 1)
% modulo (on utilise par exemple %2 pour savoir si un nombre est pair ou impair)
+ addition
+= ajoute à la variable
- soustraction
-= soustrait de la variable
* multiplication
/ division
& adresse dans la RAM
&& ET logique
|| OU logique

Il vaut mieux faire une multiplication par 0,5 qu'une division par 2, car le nombre de cycles requis pour une division est 50 fois supérieur au nombre de cycles requis pour faire une multiplication.

Pointeurs et références

type * maVal pointeur
&maVal adresse dans la RAM de maVal

Exercices

1e exercice

Entrer un nombre

Tenter de trouver le nombre.
Le jeu ne s’arrête pas tant que le nombre n’est pas trouvé.
Indication si le nombre est supérieur ou inférieur

2e exercice

Randomizer le nombre à deviner, entre 0 et 100.

3e exercice

Ajouter un nombre de chances définies.

4e exercice

Intelligence artificielle qui devine le nombre, par dichotomie.

Jour 2

Programmation structurée

Structures

```
struct MaStructure {
	var maVal;
	var maDeuzVal;
}

//Sans pointeur
MaStructure varMaStruct= {maVal, maDeuzVal};

//Avec pointeur
MaStructure * varMaStruct= new MaStructure;
MaStructure->maVal = valeur;
MaStructure->maDeuzVal = valeur;
```

Gestion de la mémoire

Allocation et suppression
Attention aux pointeurs !

```
delete maVar;
```

La bibliothèque standard

#include <iostream>
using namespace std;

Exercices

1e exercice

Créer une structure pour le jeu qui contient les données du jeu.

2e exercice

Transformer la structure en pointeur vers la structure.
Ajouter les différentes fonctions qui permettent de gérer l'avancée du jeu:
-Test de victoire
-Mettre à jour le range
-Comparer les valeurs (faire un "pas")

3e exercice

Vous incarnez un héros. Ce héros se bat contre des monstres. Chaque combat se déroule en manches de pierre/papier/ciseau.
Jour 3

Programmation Orientée Objet

include <malib.h>
include "maclasse.h"

Création de classes et d'objets avec C++

```
//Fichier .h
class maClasse {
	public:
		maClasse(); // constructeur
		virtual ~maClasse(); // destructeur

		int maValPub;
	protected:
	private:
```

```
                int maValPriv;
}

//Fichier .cpp
#include "gameMapMesh.h"

maClasse::maClasse() {
        //ctor
}

maClasse::~maClasse() {
        /dtor
}
```

Membres et méthodes de classes (virtuelles / virtuelles pures)

Une méthode et un membre virtuelle pure (abstraite) sont des membres qui sont définis dans le header mais pas dans le .cpp
Elle sera redéfinie dans ses enfants.

Autres aspects spécifiques à C++ : les surcharges

Utiliser la même fonction avec différentes variables.

```
void maClasse::maFonction() { }
void maClasse::maFonction(int p_maValeur) { }
void maClasse::maFonction(int p_maValeur, string p_maValeurDeux) { }
```

Public Protected Private

Public = accessible par tout le code
Protected = accessible par la classe ainsi que ses classes enfants
Private = accessible uniquement par la classe

Exercices

1e exercice

Créer une classe Perso avec des points de vie, d'attaque et un niveau

2e exercice

Créer une classe Héros et une classe Monstre, qui découlent de la classe Perso

Jour 4

Vocabulaire

Classe = Structure qui contient à la fois des variables (membres) et des fonctions (méthodes)
Objet = Instanciation de ma classe
Membre = Variable ou fonction qui appartient à ma classe
Méthode = Fonction membre de ma classe
Constructeur = Méthode membre de ma classe appelée lorsque mon objet est crée
Destructeur = Méthode membre de ma classe appelée lorsque mon objet est détruit
Surcharge = Redéfinir une fonction qui a déjà été définie, soit par mon parent si il s'agit de la même, soit avec des paramètres différents si il s'agit de ma classe.
Déclaration = Dire qu'un membre de ma classe existe, dans son fichier d'entête (header)
Définition = Coder la méthode de ma classe, dans son fichier .cpp

Tests et débogage

Les différents types d'erreurs

error : 'ma_classe' was not declared in this scope
- J'ai oublié d'include le header de ma classe (#include "maClasse.h")

undefined reference to 'ma_classe::mamethode()'
- J'utilise une méthode qui n'a pas été déclarée.

error: no 'void ma_classe::mamethode()' member function declared in class 'ma_classe'
- J'ai défini une méthode de ma classe qui n'a pas été déclarée.

- J'ai déclaré une méthode d'une classe mais je ne l'ai pas définie
- J'ai oublié de préciser que ma méthode fait parti de ma classe
  exemple: void mamethode() {} au lieu de void ma_classe::mamethode() {}
- J'ai mal déclaré une méthode, sa définition ne correspond à aucune déclaration
  exemple: void mamethode() {} en définition
  void ma_classe::mamethode(int param) {} en déclaration
- J'ai fait un héritage mais j'ai oublié d'include la classe parent
- J'ai oublié de faire un héritage en précisant que la classe parent est publique
- J'ai appelé un membre de ma classe parent alors que celui ci est privé

Techniques de débogage

Lire le message d'erreur

La réponse la plus évidente : lire le message d'erreur permet de donner des indications sur l'origine du problème.

L'affichage à l'écran

Rien de tel qu'un std::cout pour afficher une valeur à l'écran ou vérifier que le code passe bien à l'endroit souhaité, lorsqu'on est dans des structures de contrôle (if, switch…).

Le point d'arrêt

Pour mettre un point d'arrêt, dans code::blocks, il faut faire un clic droit sur la ligne concernée, et "add breakpoint".
Ensuite, il faut utiliser la flèche rouge afin de lancer le debug pour qu'il s'arrête aux points d'arrêt.

Monitorer une variable

Il suffit de faire un clic droit sur la variable et "watch maVariable"

Compiler un programme et lancer un executable

1) Dans la liste déroulante du menu de compilation, choisir "Release" au lieu de "Debug"
2) Dans le menu contextuel "Build", builder le projet (raccourci: F9)
3) Dans le menu contextuel "Build", compiler le fichier (raccourci: ctrl+shift+F9)
4) Votre exécutable se trouve dans ./mon_projet/bin/Release/mon_projet.exe

Exercices

Exercice 1

Un programme qui hérite la classe forme_geometrique et réalise le calcul d'aires de triangles, triangles rectangles, quadrilatère, rectangle, carré et losange

Références utiles

www.cplusplus.com
https://openclassrooms.com/courses/programmez-avec-le-langage-c