Rappel de programmation

Prérequis

Un programme écrit en C/C++ requiert obligatoirement une fonction **int main()** pour q'un exécutable soit créé. Cette fonction contient la suite des instructions effectuées par le programme, qui sera lancée lors de l'ouverture de l'exécutable. On appelle aussi cette fonction **le point d'entrée** du programme.
Nous vous conseillons de créer un fichier **main.cpp** spécialement pour cette fonction.

Compilation Simple : 1 seule commande

g++ fichier.cpp -o NomExecutable
ou
g++ fichier1.cpp fichier2.cpp -o NomExecutable

-> Au moins un des fichiers .cpp doit contenir une fonction "int main()"

-> Tous les fichiers .cpp "utiles" (librairies locales et programme principale) doivent être compilés en même temps.

-> Compile et génère un exécutable directement depuis les fichiers .cpp

-> Si vous ne mettez pas l'option "-o LeNomExecutableQueVousAvezChoisi", le programme créé s'appellera par défaut "a.out"

Compilation séparée : 2 commandes

1 - Précompilation d'un fichier (uniquement les fichiers .cpp) :

g++ -c fichier.cpp
ou
g++ -c fichier1.cpp fichier2.cpp

-> Vérifie que votre fichier.cpp ou vos fichier1.cpp et fichier2.cpp sont corrects.

-> Génère un fichier.o pour chaque fichier.cpp

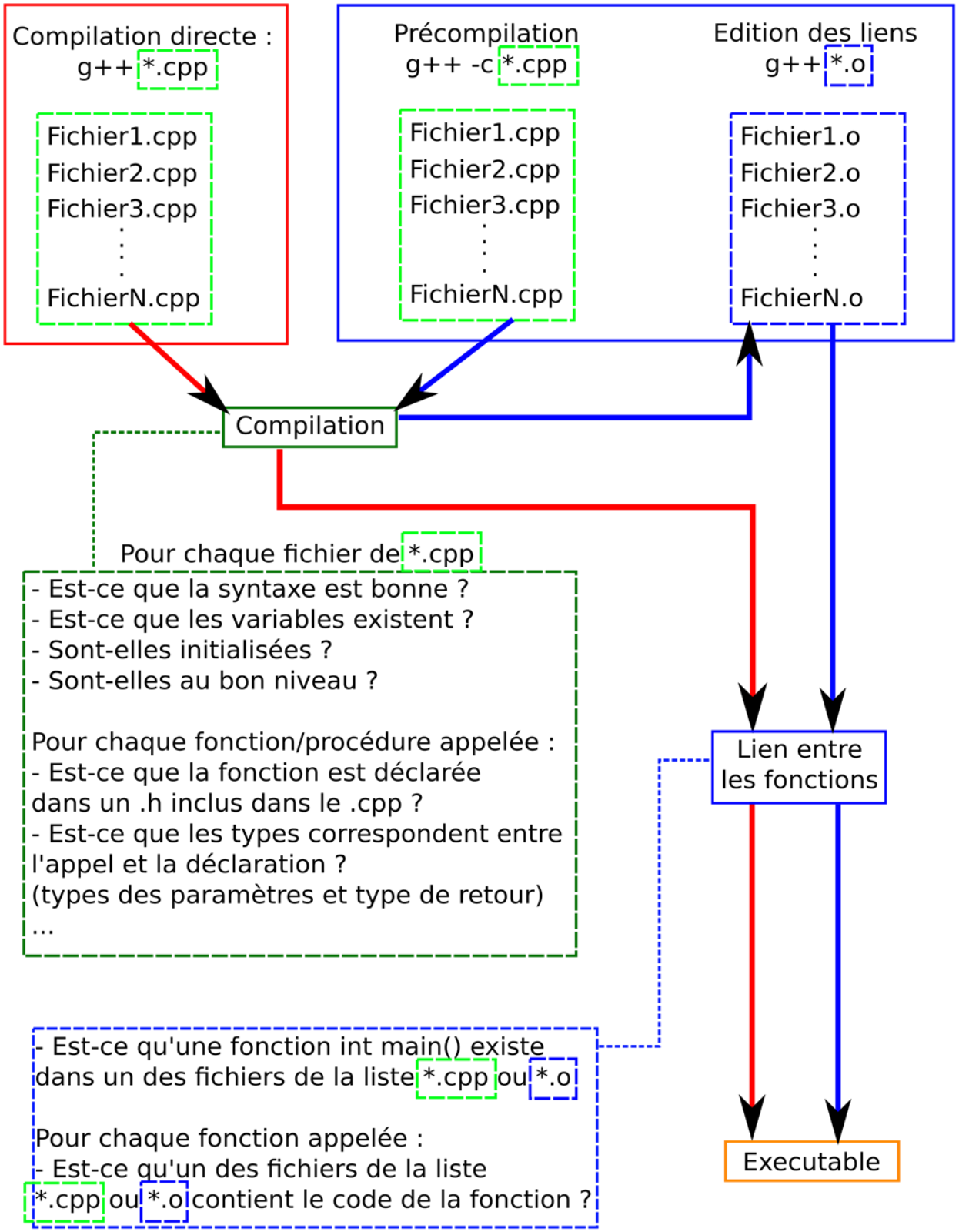
2 - Compilation de l'exécutable (Tous les fichiers .o) :

g++ fichier1.o fichier2.o -o NomExecutable

-> Au moins un des fichiers .o doit être issu d'un fichier .cpp contenant une fonction "int main()"

-> Fait le lien entre les différents fichiers .o pour créer l'exécutable

Comment G++ crée un exécutable ?



Structurer ses programmes : Créer et utiliser une librairie

Séparation en deux types de fichiers : les fichiers d'entête (.h) et les fichiers de code (.cpp).

Rappel : L'inclusion de fichier

Si vous désirez ré-utiliser des structures et des fonctions définies dans un autre ensemble de fichiers .h/.cpp, il vous suffit d'inclure le .h au début de votre fichier. Il existe également un grand nombre de librairies dites "systèmes", déjà utilisables sur vos ordinateurs.

Pour inclure une librairie:
-> Librairie système : **#include <librairie>**
Exemple : #include <iostream>
-> Librairie locale : **#include "fichier.h"**

Dans
Listes et exemples des librairies systèmes (en anglais):

http://www.cplusplus.com/reference/

Contenu des fichiers

Fichiers d'entête .h (ou "headers"):

-> Déclaration des structures :

```
struct maStructure
{
        int t1;
        int t2,
};
```

-> Déclaration des fonctions et procédures (Prototypage)

```
void maProcedure(int arg1, float arg2) ;
int maFonction(int arg1, int arg2);
```

Fichiers de code .cpp :

-> Codage des fonctions et procédures

```
void maProcedure(int arg1, float arg2)
{
        // Instructions
}

int maFonction(int arg1, int arg2)
{
        int monResultat;
        // instructions
        return monResultat;
}
```

Programmer proprement

Afin de pouvoir plus rapidement lire votre code et vos algorithmes, nous vous recommandons de respecter au minimum quelques règles de présentation de votre code et de nommage de vos fonctions et variables. Voici pour le moment deux règles liées à la programmation par blocs et à l'imbrication de blocs d'instructions.

Sachez également que de nombreuses entreprises demandent de respecter des **Chartes de programmation**, afin d'uniformiser l'apparence et la notation des variables et des fonctions.

Programmation par bloc

En programmation, il n'est pas rare de rencontrer ce que l'on apelle des **blocs d'instructions.** En C++, un bloc d'instructions est un ensemble d'instructions situées entre deux accolades.

Exemple :

```
{
	int var1;
	var1 = 128;
}
```

Lorsque vous codez une fonction, vous créer ainsi un bloc d'instructions qui sera exécuté lors de l'appel de la fonction. Il en va de même pour les instructions telles que if/else, les boucles etc ...

Les accolades ouvrantes et fermantes d'un bloc d'instructions doivent être seules sur leurs lignes respectives.

Cela permet d'identifier plus facilement le début et la fin du bloc.

Les niveaux d'imbrication

Il est extrèmement fréquent de trouver des blocs d'instructions imbriqués (un bloc dans un bloc dans un bloc). C'est le cas par exemple d'une fonction contenant une instruction *if* exécutant un bloc d'instructions :

```
int maFonction(int a1)
{
	int var1 = 0;
	if ( a1 > 0 )
	{
		var1 = a1 + 128;
	}
	return var1;
}
```

Les instructions se trouvent alors à différents niveaux d'imbrication : l'entête de la fonction se trouve au niveau 0, les instructions de la fonction se trouvent au niveau 1 et les instructions du if se trouvent au niveau 2.

Afin de mieux différencier ces niveaux, il est nécessaire de respecter **l'indentation** de chaque instruction, c'est à dire l'alignement de l'instruction selon son niveau.

Chaque instruction doit être précédée d'un nombre de tabulations égal à son niveau.

Voici un petit exemple de fichier .cpp contenant plusieurs niveaux d'imbrication.

```
// Niveau 0 : 0 tabulations
#include <iostream>
using namespace std;
int main()
{// -> Début du bloc d'instructions : +1 tabulation
	// niveau 1
	bool monBooleen = true;

	if( monBooleen == false )
	{// -> Changement de niveau : +1 tabulation
		// Niveau 2
		cout << "!??" << endl;
		// -> Fin du bloc d'instructions : -1 tabulation
	}
	else
	{// -> Changement de niveau : +1 tabulation
		//Niveau 2
		cout << "Hello World" << endl;
		// -> Fin du bloc d'instructions : -1 tabulation
	}
	// -> Fin du bloc d'instructions : -1 tabulation
}
```