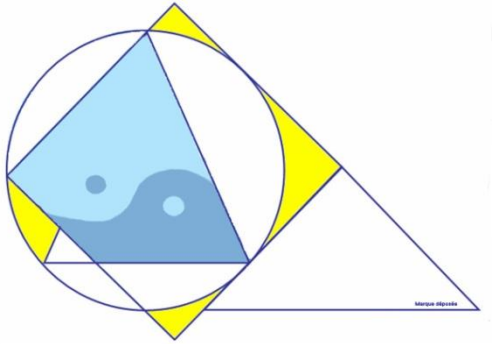


www.yantra-technologies.com

Le Langage C modulaire

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

Sommaire

1 - Définition

La programmation modulaire sert à décomposer une grosse application en modules, groupes de fonctions, de méthodes et de traitement, pour pouvoir les développer indépendamment, et les réutiliser dans d'autres applications.

Le principe est tout bête : plutôt que de placer tout le code de notre programme dans un seul fichier (main.c), nous le « séparons » en plusieurs petits fichiers.

http://fr.wikipedia.org/wiki/Programmation_modulaire
http://www.siteduzero.com/informatique/tutoriels/apprenez-a-programmer-en-c/la-programmation-modulaire

2 - Pourquoi ?

- La programmation est modulaire, donc plus compréhensible
- La séparation en plusieurs fichiers produit des listings plus lisibles
- La maintenance est plus facile car seule une partie du code est recompilée

=> La programmation modulaire simplifie l'ensemble d'un programme, facilite les futurs modifications, évolutions et réutilisations.

3- Le prototype

Le prototype d'une fonction (d'une procédure ou d'une méthode) désigne la syntaxe d'appel de cette fonction. Il spécifie ce que la fonction fait mais ne donne pas de détails sur la façon dont elle le fait.

```
// structure nombre complexe
typedef struct complexe {
    double a; // partie reel
    double b; // partie imaginaire
} Complexe;

Complexe* complexe_creer  (const double a ,const double b );
Complexe* complexe_copier (const Complexe* const Y);
Complexe* complexe_default();
void complexe_liberer     (Complexe** X);

void complexe_afficher(const Complexe* const X);

Complexe* complexe_multiplier (Complexe* X, const Complexe* const Y );
Complexe* complexe_diviser    (Complexe* X, const Complexe* const Y );
Complexe* complexe_additionner(Complexe* X, const Complexe* const Y );
Complexe* complexe_soustraire (Complexe* X, const Complexe* const Y );

int complexe_isegale    (const Complexe* const X, const Complexe* const Y );
int complexe_isdifferent(const Complexe* const X, const Complexe* const Y );
```

4 - Le header

Les fichiers header contiennent les prototypes de fonction.

complexe.h

```
#ifndef COMPLEXE_H
#define COMPLEXE_H

// structure nombre complexe
typedef struct complexe {
    double a; // partie reel
    double b; // partie imaginaire
} Complexe;

Complexe* complexe_creer  (const double a ,const double b );
Complexe* complexe_copier (const Complexe* const Y);
Complexe* complexe_default();
void complexe_liberer     (Complexe** X);

void complexe_afficher(const Complexe* const X);

Complexe* complexe_multiplier (Complexe* X, const Complexe* const Y );
Complexe* complexe_diviser    (Complexe* X, const Complexe* const Y );
Complexe* complexe_additionner(Complexe* X, const Complexe* const Y );
Complexe* complexe_soustraire (Complexe* X, const Complexe* const Y );

int complexe_isegale    (const Complexe* const X, const Complexe* const Y );
int complexe_isdifferent(const Complexe* const X, const Complexe* const Y );

#endif // COMPLEXE_H
```

5 - Le corps

Le corps de la fonction contient le code de la fonction

complexe.c

```
#include "complexe.h"
#include <malloc.h>

Complexe* complexe_creer  (const double a ,const double b ) {
    Complexe *res = (Complexe*) malloc(sizeof(Complexe));
    if ( res == NULL ) exit(10);
    res->a=a;
    res->b=b;
    return res;
}

Complexe* complexe_copier (const Complexe* Y) {
    if ( Y == NULL ) return NULL;
    return complexe_creer(Y->a,Y->b);
}

Complexe* complexe_default() {
    return complexe_creer(0,0);
}
```

6 - L'utilisation

On peut écrire dès la partie création des headers, le programme utilisateur et compiler, mais on ne peut faire un exécutable que si le corps des fonctions a été écrit.

main,c

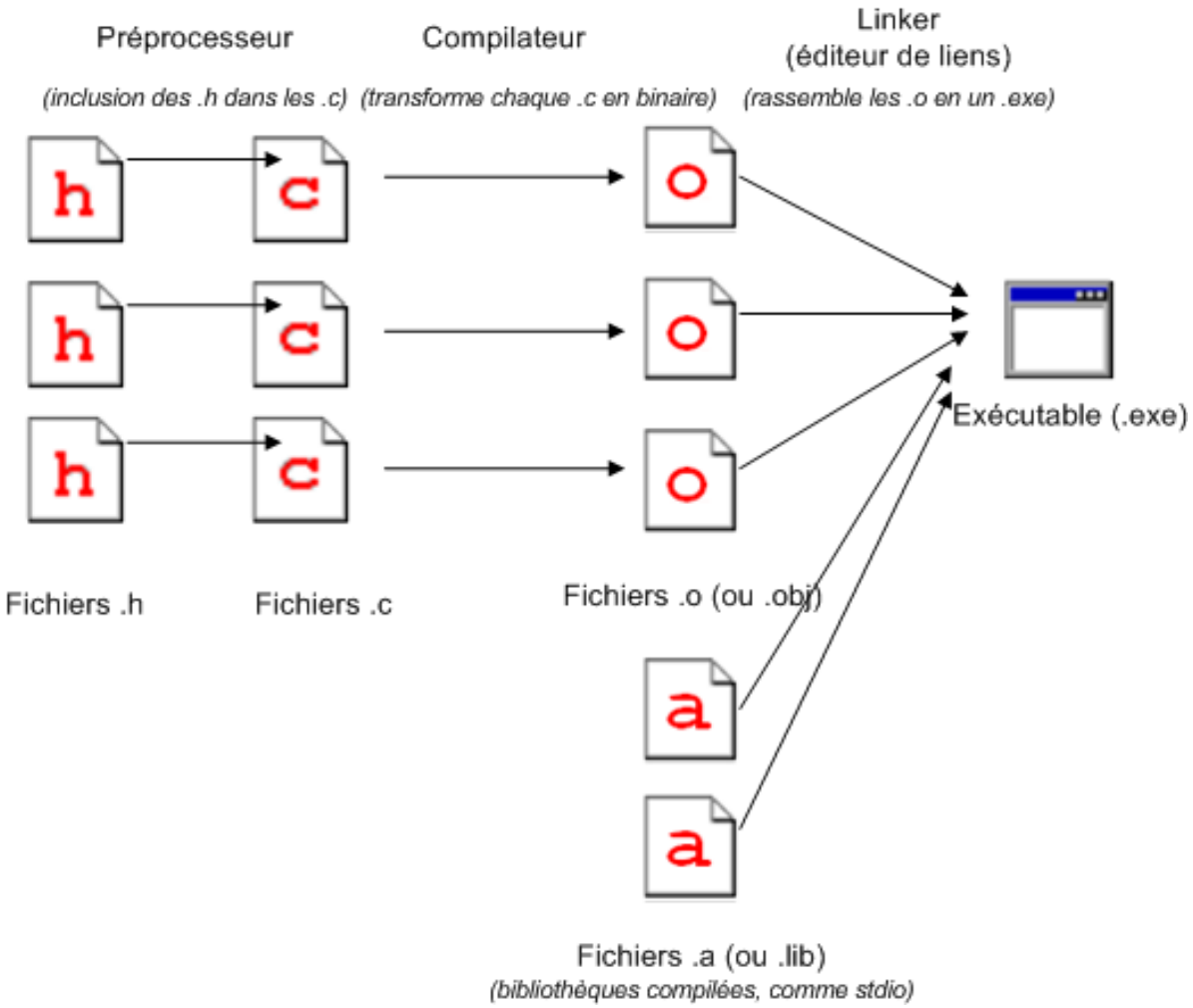
```c
#include <stdio.h>
#include <stdlib.h>
#include "complexe.h"

int main()
{
    Complexe* C1 = complexe_creer(1,1);
    Complexe* C2 = complexe_creer(2,2);
    printf("C1=");complexe_afficher(C1);
    printf("C2=");complexe_afficher(C2);
    complexe_additionner(C1,C2);
    printf("C1=");complexe_afficher(C1);
    printf("C2=");complexe_afficher(C2);
    return 0;
}
```

7 - La compilation séparée

La programmation modulaire entraîne la compilation séparée.

- Étape 1 : Le programmeur sépare ses fonctions dans des fichiers distincts regroupés par thème
- Exemple : complexe.h et complexe.c
- Étape 2 : création d'un librairie
- Étape 3 : Edition de liens (linker) qui a pour rôle de regrouper toutes les fonctions utilisées dans un même exécutable.

7 - La compilation séparée



7 - La compilation séparée

#création des fichiers binaires à mettre dans la libraire
- gcc –fpic –o complexe.o –Wall complexe.c

#création de la libraire statique
- ar –cr libcomplexe.a complexe.o
- ranlib libcomplexe.a

#création de la libraire dynamique
- gcc -o libcomplexe.so –shared complexe.o

#création du fichier contenant le programme principal
- gcc –fpic –o main.o –Wall main.c

#création exécutable dynamique
- gcc –o testcomplexe.dyn main.o –L. –lcomplexe

#création exécutable statique
- gcc -static –o testcomplexe.sta main.o –L. –lcomplexe

7 - La compilation séparée

- make
- automake
- cmake : http://www.cmake.org/
- qmake

8 - Répertoire projet

Projet :

- inc
- src
- obj
- lib
- exe
- test
- test_unitaire
- doc
- outils

8 - Répertoire projet

Projet :

- inc
- src
- release
  - obj
  - lib
  - exe
- debug
  - obj
  - lib
  - exe
- test
- test_unitaire
- doc
- outils