Langage C

Modularité inter-fichiers

#include<stdio.h>

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech
LEnsE Laboratoire d'Enseignement Expérimental

Sylvie LEBRUN / Charles BOURASSIN / Julien VILLEMEJANE

Rappel : structure générale d'un programme

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES

Déclaration des fonctions ;

int main
{
appel aux fonctions;
return 0;
}

Définition des fonctions
```

Rappel : structure générale d'un programme

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES

Déclaration des fonctions ;

int main
{
appel aux fonctions;
return 0;
}

Définition des fonctions
```

Les fonctions ne sont utilisables que dans ce fichier

Si on veut les réutiliser ailleurs, on ne va pas faire des copier-coller ...

On va les répartir dans un ou plusieurs fichiers sources dits librairies → **Modularité interfichiers**

Avantages de la modularité interfichiers

- hiérarchisation des problèmes,
- lisibilité,
- réutilisation,
- modifications centralisées,
- compilation séparée.

Principe

1. **On crée un fichier source .c par thème et on l'inclut dans le projet Code Blocks :**

Fichier tableaux.c

Fichier fichiers.c

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES
/*Entête*/
void init_tab(int tab[],int dim) ;
/*Entête*/
void save_fichier(char *nom, int tab[],int
dim);

int main
{
appel aux fonctions;
return 0;
}

void init_tab(int tab[],int dim)
{code de la fonction}
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Principe

1. On crée un fichier source .c par thème et on l'inclut dans le projet Code Blocks :

On y copie les définitions

Fichier tableaux.c

```
void init_tab(int tab[],int
dim)
{code de la fonction}
```

Fichier fichiers.c

```
void save_fichier(char *nom,
int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES
/*Entête*/
void init_tab(int tab[],int dim) ;
/*Entête*/
void save_fichier(char *nom, int tab[],int
dim);

int main
{

return 0;
}
```

Principe

1. On crée un fichier source .c par thème et on l'inclut dans le projet Code Blocks :

On y copie les définitions

Fichier tableaux.c

```
void init_tab(int tab[],int
dim)
{code de la fonction}
```

Fichier fichiers.c

```
void save_fichier(char *nom,
int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES
/*Entête*/
void init_tab(int tab[],int dim) ;
/*Entête*/
void save_fichier(char *nom, int tab[],int
dim);

int main
{

return 0;
}
```

2. On crée un fichier d'en-tête .h par thème :

Fichier tableaux.h

Fichier fichiers.h

Principe

1. On crée un fichier source .c par thème et on l'inclut dans le projet Code Blocks :

On y copie les définitions

Fichier tableaux.c

```
void init_tab(int tab[],int dim)
{code de la fonction}
```

Fichier fichiers.c

```
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES


int main
{

return 0;
}
```

2. On crée un fichier d'en-tête .h par thème : on y copie les déclarations avec les en-têtes

Fichier tableaux.h

```
/*Entête*/
void init_tab(int tab[],int dim) ;
```

Fichier fichiers.h

```
/*Entête*/
void save_fichier(char *nom, int tab[],int dim);
```

Compilation séparée

3. On ajoute dans chaque fichier source les librairies nécessaires à sa compilation "autonome"

Fichier tableaux.c

```
#include <stdio.h>
void init_tab(int tab[],int dim)
{code de la fonction}
```

Fichier fichiers.c

```
#include <stdio.h>
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#define CONSTANTES SYMBOLIQUES


int main
{

return 0;
}
```

2. On crée un fichier d'en-tête .h par thème : on y copie les déclarations avec les en-têtes

Fichier tableaux.h

```
/*Entête*/
void init_tab(int tab[],int dim) ;
```

Fichier fichiers.h

```
/*Entête*/
void save_fichier(char *nom, int tab[],int dim);
```

Utilisation des nouvelles librairies

3. On ajoute dans chaque fichier source les librairies nécessaires à sa compilation "autonome"

Fichier tableaux.c

```
#include <stdio.h>
void init_tab(int tab[],int dim)
{code de la fonction}
```

Fichier fichiers.c

```
#include <stdio.h>
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#include "tableaux.h"
#define CONSTANTES SYMBOLIQUES


int main
{
appel aux fonctions de tableaux.c
return 0;
}
```

2. On crée un fichier d'en-tête .h par thème : on y copie les déclarations avec les en-têtes

Fichier tableaux.h

```
/*Entête*/
void init_tab(int tab[],int dim) ;
```

Fichier fichiers.h

```
/*Entête*/
void save_fichier(char *nom, int tab[],int dim);
```

Utilisation des nouvelles librairies

3. On ajoute dans chaque fichier source les librairies nécessaires à sa compilation "autonome"

Fichier tableaux.c

```
#include <stdio.h>
void init_tab(int tab[],int dim)
{code de la fonction}
```

Fichier fichiers.c

```
#include <stdio.h>
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#include "tableaux.h"
#include "fichiers.h"
#define CONSTANTES SYMBOLIQUES


int main
{
appel aux fonctions de tableaux.c
appel aux fonctions de fichiers.c
return 0;
}
```

2. On crée un fichier d'en-tête .h par thème : on y copie les déclarations avec les en-têtes

Fichier tableaux.h

```
/*Entête*/
void init_tab(int tab[],int dim) ;
```

Fichier fichiers.h

```
/*Entête*/
void save_fichier(char *nom, int tab[],int dim);
```

Quelques remarques

1. **tableaux.c et fichiers.c peuvent être compilés séparément : intérêt pour les gros fichiers et la centralisation des modifications et du déboggage**

Fichier tableaux.c

```
#include <stdio.h>
void init_tab(int tab[],int dim)
{code de la fonction}
```

Fichier fichiers.c

```
#include <stdio.h>
void save_fichier(char *nom, int tab[],int dim)
{code de la fonction}
```

Fichier main.c

```
#include librairies standard
#include "tableaux.h"
#include "fichiers.h"
#define CONSTANTES SYMBOLIQUES


int main
{
appel aux fonctions de tableaux.c
appel aux fonctions de fichiers.c
return 0;
}
```

2. **Les .h (pour header) ne sont pas des fichiers source : ils ne sont pas compilables.**

Fichier tableaux.h

```
/*Entête*/
void init_tab(int tab[],int dim) ;
```

Fichier fichiers.h

```
/*Entête*/
void save_fichier(char *nom, int tab[],int dim);
```

Les #include

`#include<...>` ira chercher le .h dans les bibliothèques préinstallées.
exemple : `#include<stdio.h>`

Sinon :

- si le fichier **fichier.h** est dans le répertoire du projet courant on écrira :

```
#include"fichier.h"
```

- si le fichier **fichier .h** est dans un autre répertoire, on écrira le chemin d'accès complet au fichier, par exemple :

```
#include "C:\\transfert\\Sylvie\\Enseignement en C\\Enseignement en C 2013\\td8\\Fichiers pour les élèves\\fichier.h"
```

Compilation conditionnelle

Quand on ouvre un nouveau fichier .h, **tableaux.h** par exemple, CodeBlocks génère les lignes en vert ci-dessous :

```
#ifndef TABLEAUX_H_INCLUDED
#define TABLEAUX_H_INCLUDED
Mettre ici les prototypes des fonctions
#endif // TABLEAUX_H_INCLUDED
```

TABLEAUX_H_INCLUDED est un identificateur. S'il est rencontré pour la 1ère fois par le processeur, le texte situé entre #define et #endif est inclus. Sinon, il n'est pas pris en compte.

Application : protection des fichiers d'en-tête contre les inclusions multiples.

MODULARITÉ INTER-FICHIERS

POUR AMÉLIORER LA RÉUTILISATION

- **Regroupement de fonctions par catégorie dans des fichiers séparés / bibliothèques :**
  - un fichier **header .h** : comportant les **prototypes** de toutes les fonctions
  - un fichier **source .c** : comportant les **définitions** de toutes les fonctions répertoriées dans le .h

tableau1D.h	complexe.h
tableau1D.c	complexe.c

- **Recherche des fonctions facilitée**
- **Modification centralisée**
- **Compilation séparée**

MODULARITÉ INTER-FICHIERS

EXEMPLE - TABLEAU.H / TABLEAU.C

tableau1D.h

```
#ifndef TABLEAU1D_H_INCLUDED
#define TABLEAU1D_H_INCLUDED
#include <stdlib.h>
#include <stdio.h>

/*
 * afficheTab : Affiche un tableau 1D
 *         ENTREES :
 *           tab : tableau à afficher
 *           dim : dimension du tableau
 *         Auteur : Villou - 30/06/2017
 */
void afficheTab(int tab, int dim);

/* … */
void triTabCroissant(int tab, int dim);

[...]

#endif // TABLEAU1D_H_INCLUDED
```

tableau1D.c

```
#include "tableau1D.h"

/*
 * afficheTab : Affiche un tableau 1D
 *         ENTREES :
 *           tab : tableau à afficher
 *           dim : dimension du tableau
 *         Auteur : Villou - 30/06/2017
 */
void afficheTab(int tab, int dim){
        int i;
        for(i = 0; i < dim; i++)
        …
}
/* … */
void triTabCroissant(int tab, int dim){
        …
}
[...]
```

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "tableau1D.h"

int main(void){
        …
        afficheTab(montab, 10);
                                        …
}
```

EN-TÊTE DE FONCTIONS

POUR AMÉLIORER L'UTILISATION

```
/*
 * aireRectangle : Calcul de l'aire d'un rectangle
 *     ENTREES :
 *           a,b : Largeur et longueur (double)
 *     SORTIE :
 *           Aire du rectangle (double)
 *     Auteur : Villou – Création : 30/06/2017
 *           Modifié par Arthur, le 02/07/2017
 */
double aireRectangle(double a, double b)
```

- A inclure dans les fichiers **header .h** (voir diapo précédente)