

Programmation C

Compilation Séparée, Bibliothèques

ING1

CY Tech

Fichier en-tête

Listing 1 – Problème des suites croisées

```
float suiteU (int int_n) {  
    .../...  
    flt_val = 2*suiteU(int_n-1) - suiteV(int_n-1);  
    .../...  
}  
  
float suiteV (int int_n) {  
    .../...  
    flt_val = 0.5*suiteV(int_n-1) - 3*suiteU(int_n-1);  
    .../...  
}
```

Déclaration de fonction

- Besoin de connaître le prototype pour l'appel
- Comment faire pour :
 - ▶ utiliser des fonctions réalisées par d'autres
 - ▶ utiliser des fonctions croisées (U_n , V_n)
 - ▶ fournir des bibliothèques sans fournir le code
- Besoin d'externaliser la déclaration et le code
- Réalisation d'un fichier contenant uniquement les prototypes

Déclaration de fonction "inline"

- Permet de répondre au problème des fonctions croisées
- Déclaration du prototype au début du fichier C

```
#include <stdio.h>

float suiteU (int int_n);
float suiteV (int int_n);

.../...

float suiteU (int int_n) {
    .../...
    flt_val = 2*suiteU(int_n-1) - suiteV(int_n-1);
    .../...
}
```

Création d'un fichier d'en-tête

- Permet de regrouper
 - ▶ les inclusions
 - ▶ les définitions de constantes
 - ▶ les définitions de structures
 - ▶ les déclarations de fonctions ainsi que les commentaires associés
 - ▶ ...
 - ▶ Ne comporte pas la déclaration de la fonction `main`

Fichier d'en-tête

- Avant : suite1.c
- Maintenant : suite2.c et suite2.h
- Possibilité de créer un *module*
 - ▶ Création des fichiers suite3.c, suite3.h
 - ▶ Création d'un fichier main.c, éventuellement main.h
 - ▶ Compilation séparée de suite3.o, main.o

```
gcc -c suite3.c -o suite3.o  
gcc -c main.c -o main.o
```

- ▶ Édition de liens avec suite3.o et main.o

```
gcc suite3.o main.o -o monProg
```

- Besoin de se protéger contre les inclusions multiples

Piège des modules

- Attention aux inclusions multiples
- Se produit lors d'inclusion de plusieurs fichiers
 - ▶ A inclu B
 - ▶ B inclu A
 - ▶ Boucle d'inclusion infinie
 - ▶ Compilation conditionnelle

Explication

- Dans A
 - ▶ Vérification de la présence de la variable A
 - ▶ Si non présente, on la définit et on inclut le fichier
 - ▶ Si présente, fin de l'inclusion
- Dans B
 - ▶ Idem, mais avec une variable différente
- Ce qui donne ...

Exemple

Listing 2 – A.h

```
#ifndef __A_H_  
#define __A_H_  
#include "B.h"  
/* Definition des fonctions */  
#endif
```

Listing 3 – B.h

```
#ifndef __B_H_  
#define __B_H_  
#include "A.h"  
/* Definition des fonctions */  
#endif
```



Cas particulier

- Attention : créer un fichier séparé pour les structures
- Sinon possibilité d'inclure des fonctions qui manipulent un type sans être défini

Exemple

Listing 4 – suite.h

```
#ifndef __SUITE_H_
#define __SUITE_H_
// inclusion des entetes de librairies
#include <stdio.h>
/* Definition des constantes symboliques d'erreur */
#define ERREUR_SAISIE -1

/*! Commentaires doxygen */
float suiteU(int int_n);
/*! Commentaires doxygen */
float suiteV(int int_n);

#endif
```



Exemple

Listing 5 – suite.c

```
#include "suite.h"

// Description de la fonction suiteU
float suiteU (int int_n) {
    .../...
}

// Description de la fonction suiteV
float suiteV (int int_n) {
    .../...
}
```

Bibliothèques

Bibliothèques et fichier en-tête

- Ne pas confondre fichier en-tête et bibliothèque
 - ▶ Fichier en-tête : prototype des fonctions
 - ▶ Bibliothèque : code des fonctions
- Utilité des bibliothèques
 - ▶ Regrouper un ensemble de fonctions traitant d'un même sujet (E/S, math, ...)
 - ▶ Statique : code des fonctions inclus dans l'exécutable
 - ★ Perte de place
 - ★ Gain de confidentialité
 - ▶ Dynamique : code des fonctions non inclus
 - ★ Gain de place
 - ★ Possible perte de confidentialité

- Code des fonctions inclus dans l'exécutable
- Possibilité d'exécuter le programme sur un système "quelconque"¹
- Produit un exécutable de plus grande taille
- Utile pour des fonctions très spécifiques ou propriétaires

Bibliothèque dynamique

- Code des fonctions non inclus dans l'exécutable
- Chargement de la bibliothèque en mémoire lors de l'exécution (sauf si déjà présente en mémoire)
- Occupe moins de place sur le disque
- Utile pour des fonctions très utilisées

Création de bibliothèques statiques

- Nécessité de créer un fichier objet
- Puis utilisation de la commande `ar`

```
ar r libnom.a fichiers_objets
```

- Ajout de nouvelle fonctionnalité par la même commande
- Obligation de créer un index avec la commande `ranlib` à chaque modification
- Utiliser toujours l'extension `.a`

Création de bibliothèques dynamiques

- Compilation des fichiers objets avec l'option `-fPIC`
- Création : utilisation de l'option `-shared`

```
gcc -shared libnom.so fichiers_objets
```

- Toujours utiliser l'extension `.so`

Utilisation des librairies

- Librairies statiques : utilisation de l'option `-L`

```
gcc -L ${HOME}/lib .... -lnom
```

- Librairies dynamiques : utilisation de l'option `-Wl`

```
gcc -Wl ...
```

- ▶ Documentez vous : librairies dynamiques s'inscrivent dans un cadre particulier, beaucoup de formalisme
- ▶ Utilisation de `ldconfig`, `soname`, ...