Chapitre 7

Compléments

Chapitre 7 : Compléments

I Directives au pré-processeur

a) *Directive include*

- Elle permet d'incorporer dans le fichier **source** le texte figurant dans un autre fichier; ce dernier peut être un fichier d'entêtes de la librairie standard (stdio.h, math.h, stdlib.h, …) ou n'importe quel autre fichier.
- Elle possède deux syntaxes voisines :

#include <nom_fichier> : *le fichier est recherché dans un ensemble de répertoires systèmes*

#include "nom_fichier" **:** *le fichier est recherché dans le répertoire courant (à l'endroit où se trouve le fichier source).*

- La première syntaxe est généralement utilisée pour les fichiers d'entêtes de la librairie standard, alors que la seconde est destinée aux fichiers créés par l'utilisateur.

Chapitre 7 : Compléments

b) *Directive define*

Elle permet de définir :

- ✓ des constantes symboliques
- ✓ des macros avec paramètres

Constantes :

***Syntaxe* :**

#define *nom_constante* ***texte_remplacé***

***Exemples* :**

#define DEBUT **{**

#define FIN **}**

#define NBL **5**

#define NBC **8**

#define TAILLE **NBL*NBC**

Chapitre 7 : Compléments

Macros :

***Syntaxe* :** **#define** *nom_macro*(liste_de_paramètres) corps_de_la_macro

avec liste_de_paramètres : liste d'identificateurs séparés par des virgules.

***Exemples* :** **#define** MAX(a,b) (a > b ? a : b)

- ➢ Le préprocesseur remplacera dans le programme toute occurrence de la forme MAX(x,y) par :

  (x > y ? x : y)

- ➢ Une macro a une syntaxe similaire à celle d'une fonction mais à l'utilisation elle permet d'obtenir une meilleure performance en temps d'exécution.
- ➢ La différence entre la définition d'une ***constante symbolique*** et d'une ***macro avec paramètres*** se fait sur le caractère qui suit le nom de la macro :

  - si ce caractère est une **parenthèse ouvrante** ⇒ ***macro avec paramètres***
  - sinon c'est une constante symbolique.

- ➢ ***Il ne faut jamais mettre d'espace entre le nom de la macro et la parenthèse ouvrante.***

Chapitre 7 : Compléments

II Les fichiers : Généralités

- Le langage C donne la possibilité de lire et d'écrire des informations dans des fichiers stockés sur des supports externes.
- La taille d'un fichier (*quantité de données*) n'est pas fixe ni limitée (sauf par des contraintes matérielles liées à la capacité du support de stockage).
- Les opérations de manipulation de fichiers sont des opérations d'***entrée/sortie*** :
  - ✓ On **écrit** des données **dans** un fichier
  - ✓ On **lit** des données **depuis** un fichier

Chapitre 7 : Compléments

I Identification d'un fichier

- Toute opération d'E/S doit identifier le fichier sur lequel les traitements sont exécutés
- Un fichier a un **nom unique** et **permanent** appelé **nom externe** comme par exemple :

  /commun/BUT1/LgeC/enonces.pdf

- Le système d'exploitation dispose d'un dictionnaire qui associe à chaque fichier : sa localisation physique, ses droits d'accès, sa date de création ...

II Nom Interne d'un fichier

- Dans un programme, un fichier est identifié pas un ***nom interne*** dit **fichier logique** ou **variable fichier.**
- Le ***nom interne*** est déclaré comme une variable de type **File * .**
- Lors d'opérations d'E/S le fichier est manipulé à travers son ***nom interne.***
- Dans un programme c'est ce nom qui permet de l'associer au fichier physique existant sur le support externe.
- Le ***nom interne*** a une durée de vie qui correspond à celle du programme.
- Deux programmes différents peuvent utiliser le même fichier physique avec des ***noms internes différents.***

Chapitre 7 : Compléments

III Types de fichiers C

Il existe deux types de fichiers : fichiers textes et fichiers binaires.

➢ **Fichier texte :**

- ✓ formé de caractères ASCII, organisé en lignes, chacune ligne terminée par un caractère de contrôle de fin de ligne.
- ✓ un caractère de fin de fichier termine le fichier.
- ✓ toute tentative d'accès au delà de la fin du fichier déclenche une erreur.
- ✓ un fichier texte peut être édité avec des éditeurs de texte et affiché de manière lisible à l'écran.

➢ **Fichier binaire :**

- ✓ contient des données non textuelles.
- ✓ il ne prend sens que s'il est traité par un programme adapté.
  - Exemples de fichiers binaires : code exécutable d'un programme, fichier son, video, etc...

Chapitre 7 : Compléments

IV Opérations essentielles sur les fichiers

a) Ouverture :

- L'association entre le **nom externe** (*fichier physique*) et le **nom interne** (*fichier logique*) se fait lors de l'ouverture du fichier, par l'instruction **fopen** qui :
  - recherche le fichier par son *nom externe* dans le catalogue du système
  - détermine la localisation physique du fichier : chaîne de caractères comprenant un nom de chemin et un nom physique de fichier
  - détermine les droits ou modes d'accès : lecture seule, écriture en fin de fichier, réécriture du fichier, …
  - indique s'il s'agit d'un fichier binaire ou d'un fichier texte
  - associe *l'identificateur interne* et le *nom externe* du fichier
- Cette opération d'initialisation, par l'usager, du *nom interne* est à réaliser **une seule fois** avant de commencer les traitements.

Chapitre 7 : Compléments

Modes d'accès :

- "r" : ouverture d'un fichier texte en lecture (ou "rt") ⇒ le fichier doit exister
- "w" : ouverture d'un fichier texte en écriture (ou "wt") ⇒ le fichier sera créé (s'il existe son contenu est écrasé)
- "a" : ouverture d'un fichier texte en écriture à la fin (*append*) ⇒ si le fichier n'existe pas, il est créé; s'il existe les nouvelles données seront ajoutées à la fin
- "rb" : ouverture d'un fichier binaire en lecture
- "wb" : ouverture d'un fichier binaire en écriture
- "ab" : ouverture d'un fichier binaire en écriture à la fin (append)
- "r+", "w+" : ouverture d'un fichier texte en lecture/écriture
- "a+" : ouverture d'un fichier texte en lecture/écriture à la fin

+ d'autres modes

Syntaxe **:** *nom_fichier_interne* = **fopen** (*chemin_accès_nom_fichier_externe*, *mode*)**;**

Chapitre 7 : Compléments

b) Fermeture :

La fonction **fclose** permet de casser le lien entre le *fichier logique* et le *fichier physique*.

Syntaxe **:** **fclose (***nom_fichier_interne***);**

c) Ouverture en lecture seule :

- On peut lire dans le fichier caractère par caractère avec la fonction **fgetc.**
- A l'ouverture du fichier, un "**marqueur** " est placé au début du fichier (il pointe le premier caractère).
- Chaque appel à la fonction permet de récupérer le caractère courant et déplace le marqueur sur le caractère suivant.
- La fin du fichier est atteinte si le caractère récupéré est la marque de fin qui est égale à -1 ou **EOF** (*end of file*).
- La fonction **feof** renvoie *vrai* si la fin de fichier est atteinte
- Dans un fichier texte, on peut utiliser les fonctions **fgets** pour lire une chaîne de caractères et **fscanf** pour lire des données formatées (des nombres par exemple).

Chapitre 7 : Compléments

d) Ouverture en écriture seule :

- En écriture seule, si le fichier existe, son contenu est effacé.
- Le pointeur est placé sur la fin de fichier (**EOF**) qui est ramené au début.
- La fonction **fputc** permet d'écrire un caractère à la fois dans le fichier
- Chaque appel à la fonction déplace le pointeur sur la marque de fin de fichier.

V Fichiers standards

Il existe 3 types de fichiers qui peuvent être utilisés sans qu'il ne soit nécessaire de les ouvrir ou de les fermer :

- ✓ *stdin* (standard input) : unité d'entrée (par défaut c'est le clavier)
- ✓ *stdout* (standard output) : unité de sortie (par défaut c'est l'écran)
- ✓ *stderr* (standard error) : unité d'affichage des messages d'erreur (par défaut c'est l'écran)

Chapitre 7 : Compléments

Exemples d'utilisation :

```
#include  <stdio.h>
#include  <string.h>

int main ()
{
     FILE *f_out;
     int i ;
     char * s = "salut" ,  * p;
     p = s;
     while (* p != '\0')     /* parcours de la chaîne */
     {
       printf("%c", *p);   /* affichage de chaque caractère */
       p++;
     }
     f _out= fopen ("titi.txt", "wt")
     i = 0 ;
     while  (i < strlen (s) )
     {
       fputc ( s[i], f_out);
       i++ ;
     }
     fclose (f_out) ;
}
```

```
#include <stdio.h>

int main ()
{
     FILE * f_in ;
     char c ;
     f_in= fopen ("titi.txt", "r");   /* ou "rt" ( read text) */
     while (! feof (f_in) )
     {
       c = fgetc (f_in) ;
       if (c != EOF) printf ("%c ", c );
     }
     fclose (f_in) ;
  }
```

Remarques :

- *strlen* : donne la longueur d'une chaîne de caractères
- *string.h* : librairie de fonctions sur les chaînes
- *FILE **: type fichier
- *f_in*, *f_out* : noms internes

Chapitre 7 : Compléments

IV La compilation séparée

- Un programme peut être fractionné en plusieurs fichiers que l'on compile séparément.
- On peut considérer la notion de **projet** comme étant une généralisation de la notion de *programme*.
- A cet effet, un projet peut être composé de plusieurs modules (ou unités).
- Un module ou unité est composé de deux parties :
  - ✓ *l'interface*
  - ✓ *l'implémentation*
- Dans le langage C, la partie *interface* est définie dans un fichier ayant l'extension **.h**, la partie *implémentation* dans un fichier d'extension **.c**.
- De manière générale, la partie *interface* (**.h**) contient la définition de constantes symboliques, de macros, de directives au préprocesseur, de types et d'entêtes de fonctions (*ainsi que les commentaires*) que le module désire rendre accessibles pour d'autres modules.
- La partie *implémentation* (**.c**) comprend toutes les notions que le module veut cacher aux autres modules : variables locales, définitions complètes des fonctions de l'interface (décrites dans le .h), d'éventuelles fonctions locales.
- Le fichier **.h** sera inclus dans le fichier **.c**

Chapitre 7 : Compléments

- Si un module **y** veut utiliser des ressources d'un module **x**, il doit inclure dans sa partie implémentation (**y.c**) la partie interface de **x** avec la directive *include* :

  #include "**x.h**"
- Pour éviter les doubles inclusions de fichiers d'entêtes, on définit une constante symbolique appelée, par convention, **X_H** dans le fichier **x.h** avec la directive conditionnelle *ifndef*.
- Chaque module peut être compilé séparément.
- Le résultat de la compilation de chaque module est un fichier objet ayant l'extension **.o.**
- Seule la partie implémentation est compilée et génère un fichier objet.
- A l'étape ***édition des liens***, tous les fichiers objets sont liés pour produire le fichier ***exécutable***.
- Un seul des modules composant le programme contient la fonction **main**.

Chapitre 7 : Compléments

Exemple :

On veut écrire un programme dans lequel on utilise des opérations sur des données de type fractions.

De plus, on veut rendre ces opérations utilisables dans différents autres programmes.

On va créer les fichiers suivants :

- Un fichier d'entêtes dans lequel on spécifie :
  - Les types dont on a besoin
  - Les entêtes des opérations de manipulation de ces types
- Un fichier d'implémentation dans lequel on définit complètement les opérations.
- Un fichier correspondant à un programme d'utilisation.

Chapitre 7 : Compléments

Fichier fraction.h :

```
#ifndef FRACTION_H
#define FRACTION_H
struct   T_fraction
{
  ….
};
typedef   T_fraction Fraction;
/*Saisie d'une fraction*/
Fraction Saisie(…);
/*Somme de deux fractions*/
Fraction Somme(……..);
….
#endif
```

Fichier fraction.c :

```
#include <stdio.h>
#include "fraction.h"

/*Saisie d'une fraction*/
Fraction Saisie(…)
{
…
}
/*Somme de deux fractions*/
Fraction Somme(……..);
{
…
}
…
```

Chapitre 7 : Compléments

Fichier test_fraction.c :

```
#include <stdio.h>
#include "fraction.c"
int main()
{
Fraction F;
…
F = Saisie(…);
…
}
```

Remarques :

- On peut créer plusieurs fichiers d'entêtes
- Un fichier d'entêtes peut ne contenir que des descriptions de types et/ou de fonctions
- Un fichier d'entêtes peut contenir des directives au pré-processeur
- Il en est de même pour le fichier d'implémentation