PL1 - Travaux pratiques - Séance 5

Compilation séparée, *Makefile*

Avant de commencer cette séance :

1. créez un répertoire ***PL1/TP5*** et placez-vous dans le répertoire
2. récupérez les fichiers nécessaires à ce TP : `cp ~mounlaur/CCI_PL1/TP5.tar.gz .`
3. dé-compressez et dé-archiver ce fichier : `gunzip TP5.tar.gz ; tar -xvf TP5.tar`

Exercice 1 - Utilisation de `Makefile`

Q1. Examinez le contenu des fichiers **codage.c**, **code1.c** et **code2.c**, puis exécutez les trois commandes suivantes : `gcc -c code1.c` puis `gcc -c code2.c` puis `gcc -c codage.c`

Vérifiez que les fichiers **code1.o**, **code2.o** et **codage.o** ont été créés.
À partir de ces trois nouveaux fichiers créez le fichier exécutable **codage** par la commande suivante :
`gcc -o codage codage.o code1.o code2.o`
Exécutez-le, puis supprimez les fichiers **code1.o**, **code2.o**, **codage.o** et **codage**.

Q2. Lisez attentivement le contenu du fichier **Makefile**. **Notez bien** que les lignes de *commande* commencent par un caractère de tabulation, et **non pas par des espaces**. Exécutez la commande `make`. Quels fichiers ont été créés ?
Exécutez à nouveau la commande `make`. Que se passe-t-il ?
Supprimez le fichier **codage** et exécutez de nouveau la commande **make**. Que se passe-t-il ?
Éditez le fichier **code1.c** et modifiez-le en supprimant le premier caractère, puis en le remettant. Exécutez à nouveau la commande `make`. Que se passe-t-il ?

Exercice 2 - Ecriture de `Makefile`

Q1. Examinez les contenus des fichiers ***exo2.c***, ***lire.c*** et ***lire.h***. Ecrivez un ***Makefile*** permettant de produire l'exécutable ***exo2*** par compilation séparée à partir de ces fichiers. Compilez en utilisant ce ***Makefile***.

Q2. On souhaite maintenant remplacer dans ***exo2.c*** l'instruction `printf(...)` par l'appel d'une fonction `ecrire_entier` fournie dans un fichier ***ecrire.c***. Ecrire ce fichier et le fichier ***ecrire.h*** correspondant. Modifiez votre ***Makefile*** en conséquence. Recompiler et vérifiez que votre programme est correct.

Exercice 3 - Fractions

On souhaite écrire un programme permettant d'effectuer des opérations arithmétiques sur des fractions rationnelles, selon le lexique suivant :

Fraction : le type { num : un entier, den : un entier }

LireFraction : une action (le resulat f : une Fraction)
{lit au clavier un numérateur et dénominateur mémorise le résultat dans f }
EcrireFraction : une action (la donnée f : une Fraction)
{affiche la fraction f à l'écran}

AddFractions : une fonction (les données f1, f2 : des Fractions) → une Fraction
{AddFractions(f1,f2) renvoie f1 + f2 }
MultFractions : une fonction (les données f1, f2 : des Fractions) → une Fraction
{AddFractions(f1,f2) renvoie f1 * f2 }
EgalFractions : une fonction (les données f1, f2 : des Fractions) → un booléen
{EgalFractions(f1,f2) renvoie f1=f2 }

Ce programme sera construit selon l'architecture suivante :
- un fichier `type_fraction.h`, qui définit le type Fraction.
- un ficher `es_fraction.c` (et `es_fraction.h`) fournissant les implémentations des actions LireFraction et EcrireFraction.
- un ficher `op_fraction.c` (et `op_fraction.h`) fournissant les implémentations des fonctions AddFractions, MultFractions et EgalFractions.
- un fichier `fraction.c`, contenant le programme principal, et qui implémentera l'algorithme suivant :
  1. lecture au clavier de deux Fractions f1 et f2 ;
  2. calcul de f1+f2 et f1*f2 ; affichage des résultats ;
  3. affichage d'un message indiquant si f1 et f2 sont égales ou non.

Q1. En vous inspirant du répertoire ***Complexes*** (exemple vu en cours), écrivez ces 6 fichiers.

Q2. Ecrivez le fichier `Makefile` permettant de compiler l'ensemble de ces fichiers par compilation séparée en un excutable de nom ***fractions***. Compilez et testez votre programme.

Exercice 4 - Séquences

On souhaite maintenant écrire un programme permettant d'implémenter le lexique suivant :

SeqInt : le type Séquence d'entiers (de longueur inférieure à 100)

LireSeq : une action (le resulat s : une SeqInt)
{lit au clavier un entier l, puis l entiers et les mémorise dans s}
EcrireSeq : une action (la donnée s : une SeqInt)
{affiche la séquence s à l'écran}

AjouterFin : une action (la donnée-résultat s : une SeqInt, la donnée x : un entier)
{AjouterFin (s, x) ajoute x en fin de s}
SupFin : une action (la donnée-résultat s : une SeqInt)
{SupFin (s) supprime de s son dernier élément (s est inchangée si elle est vide)}
EstPresent : une fonction (la donnée s : une SeqInt, la donnée x : un entier) → booléen
{EstPresent(s,x) vaut vrai ssi x appartient à s}

Q1. En vous inspirant de l'exercice précédent, écrivez une implémentation C de ce lexique en séparant :
- la définition du type séquence (avec longueur explicite ou marque de fin) ;
- les primitives de lecture/écriture ;
- les primitives de recherche et modification.

Q2. Ecrivez un petit programme principal permettant de tester ces primitives.

Q3. Ecrivez un ***Makefile*** permettant de compiler ce programme, et exécutez-le.