Unités de compilation

Système et environnement de programmation

Université Grenoble Alpes

Plan

Le processus de compilation

Constitué de deux grandes étapes

- traduction : transforme le contenu d'un seul fichier
  - bloc de code en langage machine
  - bloc de données contenant les constantes
  - laisse des trous : variables et fonctions manquantes
- édition de liens : produit un programme complet
  - regroupe les fichiers traduits
  - retrouve les bibliothèques (standard ou autre)
  - s'assure que tout est défini et non dupliqué

La traduction est l'étape la plus lourde (optimisations)

Un exemple illustratif

monnaie.c

```
struct monnaie {
    int euros, centimes;
};
struct monnaie creer_monnaie(int e, int c) {
    struct monnaie m;
    m.euros = e; m.centimes = c; return m;
}
int eur(struct monnaie m) {
    return m.euros;
}
struct monnaie addition(struct monnaie a,
                        struct monnaie b) {
    a.centimes += b.centimes;
    a.euros += b.euros + a.centimes/100;
    a.centimes = a.centimes % 100; return a;
}
```

Avec un programme principal

main.c

```
#include "monnaie.c"

int main() {
        struct monnaie a, b, c;
        a = creer_monnaie(2, 63);
        b = creer_monnaie(40, 57);
        c = addition(a, b);
        printf("%d %d %d\n", eur(a),
                eur(b), eur(c));
        return 0;
}
```

On aimerait ne pas inclure `monnaie.c`

- oblige à tout retraduire même si uniquement `main.c` change
- mais contient `struct monnaie` et le typage des fonctions

Fichier d'entête

On appelle un fichier d'entête un fichier contenant

- des définitions de types
- des prototypes de fonctions
  - type de retour et des paramètres
  - pas de corps
  - noms des paramètres facultatifs

On le place dans un fichier de nom terminant en `.h`
Il se réfère à un fichier de même nom de base mais terminant en `.c`

Avec notre exemple sur la monnaie

monnaie.h

```
struct monnaie {
    int euros, centimes;
};

struct monnaie creer_monnaie(int e, int c);
int eur(struct monnaie m);
struct monnaie addition(struct monnaie a,
                        struct monnaie b);
```

main.c devient

main.c

```c
#include "monnaie.h"

int main() {
    struct monnaie a, b, c;
    a = creer_monnaie(2, 63);
    b = creer_monnaie(40, 57);
    c = addition(a, b);
    printf("%d %d %d\n", eur(a),
           eur(b), eur(c));
    return 0;
}
```

monnaie.c devient

monnaie.c

```c
#include "monnaie.h"

struct monnaie creer_monnaie(int e, int c) {
    struct monnaie m;
    m.euros = e; m.centimes = c; return m;
}
int eur(struct monnaie m) {
    return m.euros;
}
struct monnaie addition(struct monnaie a,
                        struct monnaie b) {
    a.centimes += b.centimes;
    a.euros += b.euros + a.centimes/100;
    a.centimes = a.centimes % 100; return a;
}
```

Avec nos modifications

Pas de gros changement apparent, mais :

- il manque le corps de `creer_monnaie`, `addition` et `eur` dans `main.c` il n'inclut plus tout le code du programme
- il manque un `main` dans `monnaie.c`
- aucun ne peut être complètement compilé en un programme

La compilation change :

```
clang monnaie.c main.c -o essai_monnaies
```

Plus tard on verra que :

- les `.c` peuvent être traduits indépendemment
- on regroupe tout lors de l'édition de liens

Comment éviter les inclusions multiples ?

monnaie.h

```
#ifndef __MONNAIE_H__
#define __MONNAIE_H__
struct monnaie {
    int euros, centimes;
};
struct monnaie creer_monnaie(int e, int c);
int eur(struct monnaie m);
struct monnaie addition(struct monnaie a,
                        struct monnaie b);
#endif
```

Car un `.h` peut en inclure un autre, qui en inclut un autre, qui...

Plan

Corrigé du partiel

Corrigé rédigé en ligne sur
`https://inf203.gricad-pages.univ-grenoble-alpes.fr`

But de l'exercice

Réaliser un exécutable `somme`, compilé à partir d'un code en langage C qui effectue une addition rédigée sous la forme d'une chaîne de caractères

Exemple : à partir de `"3216+128+72"` stockée dans le tableau `addition[]`, imprimer à l'écran :

```
Le résultat vaut 3416.
```

Lecture d'un chiffre

Donnez le code qui permet d'affecter à la variable `n` de type `int` la valeur du chiffre contenu dans le caractère `c` de type `char` (pris parmi `'0',...,'9'` qui se suivent dans la table ASCII).

Lecture d'un entier

Cas où `addition[]` ne contient que des chiffres, par exemple `addition[20]="2458"`.

Écrivez une fonction `sum()` qui renvoie un entier égal à la valeur entière du nombre décrit dans un tableau de caractères.
Exemple : `sum(addition)` renvoie l'entier 2458.

```
nombre = 0
Pour chaque chiffre x lu dans le tableau :
  nombre = nombre*10 + valeur du chiffre x
```

Lecture d'une somme

Cas où la chaîne `addition[]` peut contenir des +.
Réécrivez `sum()` pour effectuer la somme décrite dans `addition[]`.

Exemple : si `addition[]` contient `"3216+128+72"`, alors `sum(addition)` renvoie l'entier 3416.

Programme principal

Écrivez la fonction `main()` :

- définition de la chaîne de caractères `addition[]`
- appel à la fonction `sum()` sur cette chaîne
- impression à l'écran de :

`Le résultat vaut XXXX.`

où `XXXX` contient le résultat de l'addition.

Compilation

- Comment compilez-vous `somme.c` afin de produire l'exécutable `somme` ?
- Comment exécutez-vous `somme` dans un terminal bash ?