Unités de compilation
Corrigé du partiel

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

complexe.c

```
struct complexe {
    double r, i;
};
struct complexe creer_complexe(double r,
                               double i) {
    struct complexe c;
    c.r = r; c.i = i; return c;
}
double img(struct complexe c) {
    return c.i;
}
struct complexe addition(struct complexe a,
                         struct complexe b) {
    a.r += b.r; a.i += b.i; return a;
}
```

Avec un programme principal

main.c

```
#include "complexe.c"

int main() {
        struct complexe a, b, c;
        a = creer_complexe(2.0, 16.3);
        b = creer_complexe(40.0, 25.7);
        c = addition(a, b);
        printf("%lf %lf %lf\n", img(a),
                img(b), img(c));
        return 0;
}
```

On aimerait ne pas inclure `complexe.c`

- oblige à tout retraduire même si uniquement `main.c` change
- mais contient `struct complexe` 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 autour des complexes

complexe.h

```
struct complexe {
    double r, i;
};

struct complexe creer_complexe(double r,
                               double i);
double img(struct complexe c);
struct complexe addition(struct complexe a,
                         struct complexe b);
```

main.c devient

main.c

```c
#include "complexe.h"

int main() {
    struct complexe a, b, c;
    a = creer_complexe(2.0, 16.3);
    b = creer_complexe(40.0, 25.7);
    c = addition(a, b);
    printf("%lf %lf %lf\n", img(a),
           img(b), img(c));
    return 0;
}
```

complexe.c devient

complexe.c

```
#include "complexe.h"

struct complexe creer_complexe(double r,
                               double i) {
    struct complexe c;
    c.r = r; c.i = i; return c;
}
double img(struct complexe c) {
    return c.i;
}
struct complexe addition(struct complexe a,
                         struct complexe b) {
    a.r += b.r; a.i += b.i; return a;
}
```

Avec nos modifications

Pas de gros changement apparent, mais :

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

La compilation change :

```
clang complexe.c main.c -o essai_complexes
```

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 ?

complexe.h

```
#ifndef __COMPLEXE_H__
#define __COMPLEXE_H__
struct complexe {
    double r, i;
};

struct complexe creer_complexe(double r,
                               double i);
double img(struct complexe c);
struct complexe addition(struct complexe a,
                         struct complexe 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`