

Programmation impérative

ENSIIE

Semestre 1 — 2020–21

Fonctions et procédures

Modularité

GCC : > 14,5 millions de lignes de code

Noyau Linux : > 19 millions de lignes de code

Windows Vista : 50 millions de lignes de code

- ▶ besoin de partage des tâches entre développeurs
- ▶ besoin de structurer le code
- ▶ besoin de maintenance
- ▶ besoin de réutilisabilité

Fonction

Grain le plus fin de modularité :

- ▶ fonction/procédure

Entités indépendantes

Liées entre elles par contrat :

- ▶ prototype
- ▶ requires/assigns/ensures

Routines

Réutilisation de code

- ▶ Éviter le copier/coller

Passage de paramètres

Procédure vs. fonction

Même opposition qu'entre instruction et expression

Fonction :

- ▶ retourne une valeur
- ▶ peut être utilisée dans des expressions

NB : en C, pas de vraie distinction

Contrat

Comment la procédure/fonction interagit avec le reste du programme

Aspects techniques :

- ▶ type des arguments
- ▶ type de retour

Aspects sémantiques :

- ▶ requires
quelle condition sur les paramètres ?
- ▶ assigns
quelle variables modifiées ?
- ▶ ensures
Quel état final ?

Squelette d'une procédure

```
/* @requires ...;
   @assigns ...;
   @ensures ...;          */
void nom_proc(type1 param1, ..., typen paramn) {
    /* declarations de variables locales */
    ...
    /* instructions */
    ...
}
```

Appel par valeur / Appel par référence

Deux possibilités pour passer des paramètres :

- ▶ Passage par valeur
 - les paramètres sont des expressions
 - leur valeur est calculée et copiée
 - on travaille sur la copie

- ▶ Passage par référence
 - les paramètres sont des variables
 - on travaille directement avec ces variables
 - modifications possibles

En C

Tous les paramètres sont passés par valeur

mais...

possibilité d'utiliser des pointeurs

- ▶ paramètre = adresse de la variable à modifier
- ▶ `&i` : adresse de la variable `i`

Exemple utile : `scanf`

Fonction

Identique à procédure, mais retourne une valeur

```
/* @requires ...;
   @assigns ...;
   @ensures ...;          */
type_retour nom_fonc(type1 par1, ..., typen parn) {
  /* declarations de variables locales */
  ...
  /* instructions */
  ...
  /* contient 1 ou plusieurs */ return expr;
}
```

Paramètre comme retour

```
int f(t1 p1, t2 p2) {
    ...
    return e;
}
```

```
:
```

```
x = ... + f(e1, e2);
```

```
void f(t1 p1, t2 p2, int *r)
    ...
    *r = e;
}
```

```
:
```

```
int tmp;
f(e1, e2, &tmp);
x = ... + tmp;
```

Utile si on a plus d'une valeur à retourner

Récursion

Une procédure/fonction peut appeler d'autres fonctions

Elle peut même s'appeler elle-même

- ▶ récursion

- ▶ attention problème terminaison
- ▶ attention problème dépassement de pile

Pile d'appel

⋮	
	trame de la fonction appelante
paramètre 1	trame de la fonction actuellement appelée
paramètre 2	
⋮	
paramètre n	
variables locales	