

Programmation Procédurale – Structure des programmes

Polytech'Nice Sophia Antipolis

Erick Galesio

2015 – 2016

Rappels sur les fonctions

- Il existe deux formes de définition de fonctions
 - forme K&R (très peu de contrôles)
 - forme C-ANSI (paramètres contrôlés, pas de règle implicite)
- La forme ANSI est un sur-ensemble de K&R
 - les deux formes peuvent exister au sein d'un même programme
 - K&R peut être utilisé pour relâcher certains contrôles
 - **attention**: si on oublie le prototype d'une fonction, elle est *auto-déclarée*

Modularité

La modularité en C est basée sur les *fichiers*

Variables globales d'un fichier:

- variable **static**:
 - durée de vie: celle du programme (variable globale)
 - visibilité: de son point de définition à la fin du fichier
- variable **extern**:
 - l'allocation de la variable n'est pas faite par ce fichier
 - Il faut qu'un autre fichier déclare cette variable

Fonction d'un fichier:

- fonction **static** n'est visible que dans le fichier qui définit cette fonction
- un prototype de fonction peut être déclaré **extern**
 - la fonction est définie ailleurs
 - le mot-clé **extern** peut être omis

Programme multi-fichiers (1 / 3)

```
/* Fichier file1.c */
static int a;
int b;
extern int c;

void main(void){ .... }

double f1(char *s) {
    static int a, b;
    ..... f2(); ..... /* f2 est auto-déclarée */
}
```

```
/* Fichier file2.c */
static int a, b;
int c;

extern double f1(); /* prototype K&R */

void f2(void) { ... }

static int f3(...) { /* f3 utilisable que dans file2.c */
    f1("Test"); /* utilisation conforme au proto */
}
```

Programme correct

Programme multi-fichiers (2 / 3)

```
/* Fichier file1.c */
static int a;
int b;
extern char c;           /* déclaré comme int dans fichier2.c !!!! */

void main(void){ .... }

double f1(char *s) {
    static int a, b;
    ..... a = f2(100); .... /* utilisation résultat + paramètre !!!! */
}
```

```
/* Fichier file2.c */
int a, b;                /* définition multiple de b !!!! */
int c;
extern double f1();

void f2(void) { ... }

static int f3(...) {
    f1(42);              /* utilisation non conforme */
}
```

Pas d'erreur de compilation

Programme multi-fichiers (3 / 3)

Utilisation du préprocesseur

```
/* Fichier file1.h */  
extern int b;  
extern double f1(char *s);
```

```
/* Fichier file1.c */  
#include "file1.h"  
#include "file2.h"  
  
static int a;  
int b;  
  
void main(void){ .... }  
double f1(char *s) {  
    static int a, b;  
    ... f2();  
}
```

```
/* Fichier file2.h */  
extern int c;  
extern void f2(void);
```

```
/* Fichier file2.c */  
#include "file1.h"  
#include "file2.h"  
  
static int a, b;  
int c;  
void f2(void) { ... }  
static int f3(...) {  
    f1("Test");  
}
```

→ **Utiliser des fichiers .h**

Spécification dans un fichier .h

```
/* File stack.h    Specification of a stack of integers */

#ifndef _STACK_H_
#define _STACK_H_

void stack_init(int size);
void stack_push(int elem);
int  stack_pop(void);
int  stack_is_full(void);
int  stack_is_empty(void);

#endif
```

Implémentation dans un fichier .c

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

static int *the_stack;
static int stack_top = 0;
static int stack_len;

static void Error(const char *msg) {
    fprintf(stderr, "Stack: error %s\n", msg);
}

void stack_init(int size) {
    if (size < 0) Error("bad size");
    if (the_stack = (int *) malloc(size*sizeof(int)))
        stack_len=size;
    else
        Error("bad malloc");
}

void stack_push(int elem) {
    if (!stack_is_full())
        the_stack[stack_top++] = elem;
    else
        Error("stack overflow");
}
```


Programme modulaire (3 / 4)

```
int  stack_pop(void) {
    if (!stack_is_empty())
        return the_stack[--stack_top];
    else
        Error("empty stack");
}

int  stack_is_full(void) {
    return stack_top == stack_len;
}

int  stack_is_empty(void) {
    return stack_top == 0;
}
```

Utilisation de la pile d'entiers

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

void main(void)
{
    stack_init(10);

    while(!stack_is_full()) {
        int i;

        printf("Enter a number: "); scanf("%d", &i);
        stack_push(i);
    }

    while (!stack_is_empty()) printf("%5d ", stack_pop());
}
```