UNIVERSITÉ
CÔTE D'AZUR

Fonctions – Variables – Entrées/Sorties

Présentation: **Stéphane Lavirotte**
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Erick Gallesio

Mail: Stephane.Lavirotte@univ-cotedazur.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ
CÔTE D'AZUR

Fonctions

UNIVERSITÉ CÔTE D'AZUR

Définition de fonction K&R 1/2

- ✓ **En C traditionnel, les fonctions sont déclarées avec la forme suivante:**

```
type_résultat nom(paramètres)
   types des paramètres
{
   corps de la fonction
}
```

- ✓ **Exemple:**

```
int max(a, b)
   int a, b;
{
   return (a > b) ? a : b;
}
```

- ✓ **Si les paramètres sont omis dans la liste, ils sont considérés comme de type** `int`

UNIVERSITÉ CÔTE D'AZUR

Définition de fonction K&R 2/2

- ✓ *Lorsqu'une fonction est déclarée sous la forme K&R*
  - *Pas de vérification du type des paramètres !!!*
  - *Pas de vérification du nombre de paramètres !!!*

```
#include <stdio.h>
int max(a, b)
   int a, b;
{
   return (a > b) ? a : b;
}

int main() {
   printf("max(3,6) = %d\n", max(3., 6)); /* un "." qui traîne */
   printf("max(5,1) = %d\n", max(5.1)); /* un "." au lieu d'une "," */
   return 0;
}
==> max(3,6) = 1074266112
    max(5,1) = 1717986918
```

- ✓ *Pas d'erreur ni de warning* => **Ne jamais utiliser la forme K&R!!!**

UNIVERSITÉ CÔTE D'AZUR

Définition de fonction ANSI 1/2

- ✓ La norme ANSI redéfinit la façon de définir une fonction avec des *prototypes*:

```
type_résultat nom(liste typée de paramètres) {
    corps de la fonction
}
```

- ✓ Exemple:

```
int max(int a, int b) {
    return (a > b) ? a : b;
}
```

- ✓ Pas de type implicite

UNIVERSITÉ CÔTE D'AZUR

Définition de fonction ANSI 2/2

- ✓ Type du résultat
  - `void` (pas de type => procédure)
  - n'importe quel type scalaire (entier, réel, pointeur, enum)
  - structure et union
  - Attention: pas tableau
- ✓ Type des paramètres
  - n'importe quel type scalaire (entier, réel, pointeur, enum)
  - structure et union
  - tableau
    - paramètre formel et paramètre effectif doivent être compatibles (cf. pointeurs)
    - La taille peut être spécifiée (ou non)

```
int strlen(char str[MAX]):
int strlen(char str[]);
```

UNIVERSITÉ CÔTE D'AZUR

Appel de fonction: K&R vs ANSI

```
void f(a, b)                    /* Version K&R */
    int a; double b;
{
...
}

void g(int a, double b) {  /* Version ANSI */
...
}

f();                            /* pas d'erreur */
f(1, 2, 3, 4);                  /* pas d'erreur */
f("false", "even more");        /* pas d'erreur */
f(1, 2);                        /* problème potentiel */

g();                            /* erreur détectée */
g(1, 2, 3, 4);                  /* erreur détectée */
g("false", "even more");        /* erreur détectée */
g(1, 2);                        /* 2 converti en double */
```

UNIVERSITÉ
CÔTE D'AZUR

Résultat de fonction

- ✓ La valeur de la fonction est donnée par l'énoncé `return`
- ✓ Type de résultat: celui donné à la définition de la fonction
- ✓ Conversion éventuelle de la valeur du `return` vers le type de la fonction
- ✓ Si la fonction est de type `void`, pas de valeur après le `return`

UNIVERSITÉ CÔTE D'AZUR

Passage de paramètre

- ✓ Un seul mode: passage par valeur
- ✓ Pour les tableaux, on passe un pointeur sur le début du tableau (par valeur)

```
void f(int x) {
   x = x + 1;
   printf("In function f: %d\n", x);
}
void main(void) {
   int a = 1;
   f(a);
   printf("After call to f: %d\n", a);
}
==> In function f: 2
    After call to f: 1
```

- ✓ L'ordre d'évaluation n'est pas garanti

```
i = 5
f(i++, t[i]) /* t[5] ou t[6] ?????*/
```

UNIVERSITÉ CÔTE D'AZUR

Déclaration de fonction 1/2

- ✓ Déclarer une fonction = donner son type avec un header ou prototype
- ✓ C'est utile:
  - si la fonction est utilisée « en avant »
  - si la fonction est définie dans un autre fichier
- ✓ Définition K&R

```
double cos(); /* paramètres absents (inutiles) */
```

- ✓ Définition ANSI

```
double cos(double x); /* en-tête complet */
```

- ✓ Si utilisation de la fonction sans prototype
  - auto déclaration de la fonction par le compilateur
  - résultat de type entier
  - pas de contrôle du nombre et du type des paramètres

UNIVERSITÉ CÔTE D'AZUR

Déclaration de fonction 2/2

- ✓ **Même en ANSI C, la rétro compatibilité avec le C de K&R peut être source d'erreurs**

```
void f2(int a, int b); /* forward declaration */

void f1(int a, int b) {
    int x, y, z;
    x = f2(b, a); /* error detected */
    y = f3(b, a); /* auto declaration */
    z = f4(b, a); /* auto declaration */
}

void f2(float a, float b) /* error detected */
{ ... }

int f3(void) /* conform to autodeclaration !!!! */
{ ... }

double f4(int a, int b) /* error detected, even in K&R */
{ ... }
```

UNIVERSITÉ CÔTE D'AZUR

Fonctions à arité variable 1/2

- ✓ C'est une extension ANSI
- ✓ La liste de paramètres variable est dénotée par '. . . ' après le dernier paramètre fixe

```
#include <stdarg.h>
void ma_fonction(type1 arg1, type2 arg2, ...) {
}
```

- ✓ Il doit y avoir au moins un paramètre fixe
- ✓ Le fichier `<stdarg.h>` définit les macros suivantes:

```
void va_start(va_list ap, last_fixed_parameter)
type va_arg(va_list ap, type)
void va_end(va_list ap)
```

UNIVERSITÉ CÔTE D'AZUR

Fonctions à arité variable 2/2

✓ **Exemple**

```
#include <stdarg.h>

int max(int first, ...) { /* Liste terminée par un nombre < 0 */
   va_list ap;
   int M = 0;
   va_start(ap, first);
   while (first > 0) {
      if (first > M) M = first;
      first = va_arg(ap, int);
   }
   va_end(ap);
   return M;
}

void main(void) {
   int x = max(12, 18, 17, 20, 1, 34, 5, -1);
   ...
}
```

UNIVERSITÉ
CÔTE D'AZUR

Variables

UNIVERSITÉ
CÔTE D'AZUR

Variables 1/3

- ✓ *On ne considère ici que les programmes mono fichier*
- ✓ Variable globale
  - *Définition:* en dehors d'un bloc
  - *Durée de vie:* tout le programme
  - *Visibilité:* depuis son point de définition jusqu'à la fin de fichier (avec possibilité de masquage dans un bloc)
- ✓ Variable locale
  - *Définition:* dans un bloc
  - *Durée de vie:* la durée de vie du bloc
  - *Visibilité:* restreinte au bloc de définition

UNIVERSITÉ CÔTE D'AZUR

Variables 2/3

```
int counter = 0;

int f(void) {
    counter += 1; /* incrementing the global variable */
    x += 1; /* error: x unknown */
}

int x; /* now, x is known */

int g(void) {
   int counter = 0; /* mask global variable */
   counter += 1;    /* incrementing the local variable */
   x += 1;
}
```

UNIVERSITÉ CÔTE D'AZUR

Variables 3/3

- ✓ Toujours dans le cas de programmes mono-fichiers
- ✓ Variable statique
  - *Définition:* dans le bloc (doit être préfixée par `static`)
  - *Durée de vie:* tout le programme (comme une globale)
  - *Visibilité:* restreinte au bloc de définition (comme une locale)

```
void f1(void) {
    static int counter = 0;
    printf("f1 was called %d times\n", ++counter);
}
void f2(void) {
    static int counter = 0;
    printf("f2 was called %d times\n", ++counter);
}
void main(void) {
    f1(); f2(); f1();
}
==> f1 was called 1 times
    f2 was called 1 times
    f1 was called 2 times
```

UNIVERSITÉ
CÔTE D'AZUR

Entrées / Sorties

Quelques fonctions de la bibliothèque standard

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

UNIVERSITÉ CÔTE D'AZUR

E/S avec format 1/3

✓ printf

```
int printf(const char *format, ...);
```

- le format spécifie le nombre, le type et les contraintes sur la représentation textuelle des expressions passées dans la liste variable
- Exemple:

```
printf("Name: %-15s average: %6.2f rank: %04d", name,
avg, rk);
```

✓ scanf

```
int scanf(const char *format, ...);
```

- le format spécifie le nombre, le type et les contraintes sur la représentation textuelle des objets ; les adresses sont passées dans la liste variable
- la valeur de retour est le nombre d'items effectivement lus
- Exemple:

```
scanf("%s %*s %f %d", name, &avg, &rk);
                          /* '%*s' = sauter une chaîne*/
```

UNIVERSITÉ CÔTE D'AZUR

E/S avec format 2/3

- ✓ `sprintf` **et** `sscanf`
  - ***Identiques à*** `printf` ***et*** `scanf` ***mais l'E/S se fait dans une chaîne au lieu d'un fichier***

- ✓ `snprintf`
  - ***La taille de la chaîne résultat est passée en paramètre (plus sûr)***
  - ***Exemple***

```
char str[MAXBUF];
int n;
snprintf(str, MAXBUF, "%3d error%s", n, n > 1 ? "s" : "");
```

UNIVERSITÉ CÔTE D'AZUR

E/S avec format 3/3

- ✓ `vprintf` **et** `vscanf`
  - *Entrée/Sortie avec une liste variable d'arguments*

```
int vprintf(const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format,
va_list ap);

int vscanf(const char *format, va_list ap);
int vsscanf(const char *str, const char *format, va_list ap);
```

- ✓ *Exemple*

```
void message(const char *format, ...) {
   static int count = 0;
   va_list ap;
   va_start(ap, format);
   printf("Message %d: ", ++count);
   vprintf(format, ap);
}
```

UNIVERSITÉ CÔTE D'AZUR

Le type FILE 1/3

- ✓ **Le type `FILE` est défini dans `<stdio.h>`**
- ✓ **Un objet de type `FILE` est un flux**
  - *supporte accès séquentiel et direct*
  - *accès par caractère*
  - *E/S bufferisées*

- ✓ **Ouverture d'un fichier:**

```
FILE *fopen(const char *name, const char *mode);
/* mode peut être "r", "w", "a", ... */
```

- ✓ **Fermeture d'un fichier**

```
int fclose(FILE *fp);
/* renvoie 0 si OK et EOF sinon */
```

UNIVERSITÉ CÔTE D'AZUR

Le type FILE 2/3

✓ Lecture fichier

```
int fscanf(filep, format, pointer_expression_list);
int getc(filep);
int fgetc(filep);
char *gets(string); /* allocation de la chaîne par
                       l'appelant */
char *fgets(string, size, filep);
int fread(buffer, size, nelem, filep);
```

✓ Écriture fichier

```
int fprintf(filep, format, pointer_expression_list);
int putc(character, filep);
int fputc(character, filep);
char *puts(string);
char *fputs(string, filep);
int fwrite(buffer, size, nelem, filep);
```

UNIVERSITÉ CÔTE D'AZUR

Le type FILE 3/3

✓ Accès direct

```
int fseek(FILE *stream, long offset, int whence);
long ftell(FILE *stream);
void rewind(FILE *stream);
```

- **Origine pour** `fseek`: `SEEK_SET`, `SEEK_CUR` **ou** `SEEK_END`

✓ Gestion des buffers

```
void setbuf(FILE *stream, char *buf);
int setvbuf(FILE *stream, char *buf, int mode,
size_t size);

mode: _IOFBF, _IOLBF ou _IONBF
```

UNIVERSITÉ CÔTE D'AZUR

Exemple: le programme `cat`

```
void copy(FILE *fp) {
   int c; /* et pas char! */
   while ((c = getc(fp)) != EOF)
      putc(c, stdout);
}

int main(int argc, char *argv[]) {
   FILE *fp;
   if (argc == 1) copy(stdin);
   else {
      while (--argc) {
         if ((fp = fopen(*++argv, "r")) == NULL) {
            fprintf(stderr, "cannot open %s\n", *argv);
            exit(1);
         }
         else {
            copy(fp);
            fclose(fp);
         }
      }
   }
   return 0;
}
```