Les fonctions en C

Jean-Gabriel Luque

Département d'informatique de l'université de Rouen - Normandie

Qu'est-ce que c'est et à quoi ça sert?

Exemple:

```
int main(void) {
  int num;
  assert(scanf("%d", &num) == 1);
  printf("%d\n", num);

  return EXIT_SUCCESS;
}
```

Qu'est-ce que c'est et à quoi ça sert?

Exemple:

```
int main(void) {
  int num;
  assert(scanf("%d", &num) == 1);
  printf("%d\n", num);

  return EXIT_SUCCESS;
}
```

Fonctions appelées: `assert`, `scanf`, `printf`.
Fonction définie: `main`.

Qu'est-ce que c'est et à quoi ça sert?

Fichier `scanf.c`:

```
...
int __scanf (const char *format, ...)
{
  va_list arg;
  int done;
  va_start (arg, format);
  done = __vfscanf_internal (stdin, format, arg, 0);
  va_end (arg);
  return done;
}
ldbl_strong_alias (__scanf, scanf)
```

Fonctions appelées: `va_start`, `__vscanf_internal`, `va_end`
Fonction définie: `__scanf`
warning: `ldbl_strong_alias` est une macro.

Qu'est-ce que c'est et à quoi ça sert?

Fichier `vfscan_internal.c`:

```
...
int
__vfscanf_internal (FILE *s, const char *format,
      va_list argptr, unsigned int mode_flags) {
  va_list arg;
  const UCHAR_T *f = (const UCHAR_T *) format;
  UCHAR_T fc; /* Current character of the format.  */
  WINT_T done = 0; /* Assignments done.  */
  ....
}
```

+3000 lignes!

Qu'est-ce que c'est et à quoi ça sert?

"Bout" de code paramétrable qui peut

- agir sur des périphériques:
  lecture clavier, affichage, gestion de fichier etc.
  `scanf`: récupère des valeurs sur l'entrée standard (clavier).
- agir sur des composants internes:
  calcul mémoire, modification variables etc.
  `scanf`: Modifie les valeurs aux adresses passées en paramètre.
  `scanf("%d",&n)`
- renvoyer une valeur :
  résultat d'un calcul, d'une saisie, gestion des erreurs etc.
  Valeur de retour de `scanf` = nombre de valeurs correctement saisies.
  `assert(scanf("%d",&n) == 1);`

Qu'est-ce que c'est et à quoi ça sert?

Intérêt:

- Raccourcir le code
  `scanf` $= +3000$ lignes
- Améliorer la lisibilité
- Unifier un ensemble de fonctionnalités de même nature
  `scanf`: les formats permettent de saisir une multitude de valeur de type différent.

warning: ceci n'est pas décoratif.
Il est essentiel, pour un programme, d'être lisible et propre.

Comment utiliser une fonction?

Manuel: `man strtol`

```
STRTOL(3)                  Linux Programmer's Manual                  STRTOL(3)

NAME
       strtol, strtoll, strtoq - convert a string to a long integer

SYNOPSIS
       #include <stdlib.h>

       long strtol(const char *nptr, char **endptr, int base);

       long long strtoll(const char *nptr, char **endptr, int base);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       strtoll():
           _ISOC99_SOURCE
               || /* Glibc versions <= 2.19: */ _SVID_SOURCE || _BSD_SOURCE

DESCRIPTION
       The  strtol()  function converts the initial part of the string in nptr
       to a long integer value according to the given base, which must be  be-
       tween 2 and 36 inclusive, or be the special value 0.

       The string may begin with an arbitrary amount of white space (as deter-
       mined by isspace(3)) followed by a single optional '+' or '-' sign.  If
       base  is zero or 16, the string may then include a "0x" or "0X" prefix,
       and the number will be read in base 16; otherwise, a zero base is taken
       as  10  (decimal) unless the next character is '0', in which case it is
       taken as 8 (octal).

       The remainder of the string is converted to a long value in the obvious
       manner,  stopping  at the first character which is not a valid digit in
       the given base.  (In bases above 10, the letter 'A' in either uppercase
       or  lowercase  represents 10, 'B' represents 11, and so forth, with 'Z'
       representing 35.)
```

Comment utiliser une fonction?

Prototype: `type nom(liste paramètres)`

```
long strtol(const char* nptr, char **endprt, int base)
```

Prend en paramètre :

- un pointeur vers un `char`.
  `const` spécifie que la valeur à l'adresse ne sera pas modifiée.
- un pointeur vers un pointeur vers un `char`.
- un entier.

Renvoie une valeur de type `long`

Comment utiliser une fonction?

Mécanisme d'appel (exemple 1)

```
SYNOPSIS
       #include <stdlib.h>

       int abs(int j);
...
DESCRIPTION
       The  abs() function computes the absolute value of
        the integer argument j.
...
```

Le code

```
int a = -2;
int b = a + abs(a);
printf("%d\n", b);
```

Affiche 0.

Comment utiliser une fonction?

Mécanisme d'appel (exemple 1)

```
int a = -2;
int b = a + abs(a);
printf("%d\n", b);
```

Ligne 2:

```
b = a + abs(a)
b = -2 + abs(-2)
b = -2 + 2
b = 0
```

La valeur `0` est affectée à la variable `b`.

Comment utiliser une fonction?

Mécanisme d'appel (exemple 2):

```
#include <stdio.h>

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

La fonction scanf() analyse l'entrée selon un format qui contient des spécifications de conversion. Les résultats de ces conversions sont stockés dans les emplacements indiqués par les arguments pointeurs qui suivent le format.

Le code

```
int a ;
assert(scanf("%d",&a) == 1);
printf("%d",a);
```

Affiche la valeur saisie par l'utilisateur (sauf s'il y a eu une erreur de saisie).

Comment utiliser une fonction?

Mécanisme d'appel (exemple 2):

```
int a ;
assert(scanf("%d",&a) == 1);
printf("%d",a);
```

Valeur sur l'entrée standard 12.
L'appel à `scanf("%d",&a)`

- Place la valeur 12 à l'adresse de la variable a
- Renvoie la valeur 1.

```
assert(scanf("%d",&a) == 1);
assert(1 == 1);
assert(true);
```

Comment utiliser une fonction?

Mécanisme d'appel (exemple 3):

```
#include <assert.h>

void assert(scalar expression);
```

Arrête le programme si le résultat de l'évaluation de "expression" est false (0).

La fonction ne renvoie rien et ne doit pas être utilisée dans une expression.

Comment écrire une fonction?

Schéma général d'un programme:

```
/* Nom du programme
  * Date de réalisation
  * Auteur
  * Descriptif
  */

/* Appel des bibliothèques */

#include <stdlib.h>
#include <stdio.h>

/* Déclarations des fonctions  et des macros*/

/* Fonction principale */
int main(void) {
  return EXIT_SUCCESS;
}

/* Définitions des fonctions */
```

Comment écrire une fonction?

Déclaration d'une fonction

```
/* Nom de la fonction
 * Description
 * Entrées: Liste des entrées avec leurs types
 * Sortie: type de la valeur renvoyée
 * AE: Assertion d'entrée
 * AS: Assertion de sortie
 */
 Prototype de la fonction
```

Assertion: Phrase qui a une valeur de vérité.
Le fonctionnement correct n'est garantie que si l'assertion d'entrée est vraie.
Une fois la fonction exécutée l'assertion de sortie est vraie.

Comment écrire une fonction?

Exemples de déclaration

```
/* iquo:
 * prend deux entiers a et b de type int en paramètre
 * ainsi qu'un pointeur *prem vers le type int
 * Lorsque b != 0: renvoie le quotient de a par b et
 * place le reste de la division de a par b à
 * l'adresse prem.
 * Entrées: a et b de type int, prem de type int*
 * Sortie: int
 * AE: b != 0
 * AS: iquo == a / b && *prem == a % b
 */
 int iquo(int a, int b, int *prem);

/* inc:
 * prend un pointeur vers un int et ajoute 1 à
 * la valeur se trouvant à cette adresse
 * Entrées: pa de type int*
 * Sortie: void
 * AE: Aucune
 * AS: *pa == old(*pa) + 1
 */
 void inc(int *pa);
```

Comment écrire une fonction?

Exemples de déclaration

ATTENTION: Il faut être plus précis!!!

```
/* iquo:
 * prend deux entiers a et b de type int en paramètre
 * ainsi qu'un pointeur *prem vers le type int
 * Lorsque b != 0: renvoie le quotient de a par b et
 * place le reste de la division de a par b à
 * l'adresse prem.
 * Entrées: a et b de type int, prem de type int*
 * Sortie: int
 * AE: b != 0 && prem pointe vers une adresse valide
 * AS: iquo == a / b && *prem == a % b
 */
 int iquo(int a, int b, int *prem);

/* inc:
 * prend un pointeur vers un int et ajoute 1 à
 * la valeur se trouvant à cette adresse
 * Entrées: pa de type int*
 * Sortie: void
 * AE: pa pointe vers une adresse valide.
 * la valeur contenue à cette adresse est modifiable [MIEUX]
 * AS: *pa == old(*pa) + 1
 */
 void inc(int *pa);
```

Comment écrire une fonction?

Règles de construction des assertions

- Utilisez le plus possible des expressions `C`.
- Opérateurs logiques : `&&` (et), `||` (ou), `^^` (ou exclusif), `!` (not).
- Lorsqu'il n'y a aucune condition sur les entrées, écrire `Aucune` ou `true`.
- Utilisez le nom de la fonction pour symboliser la valeur renvoyée.
- Utiliser `old` pour désigner la valeur d'une variable avant l'appel à la fonction.

Comment écrire une fonction?

Définition d'une fonction

```
prototype {
/* Optionnel pour sortir du programme lorsque
 * AE n'est pas satisfaite
 * assert (...);
 */

 /* corps de la fonction */

 /* Si le type de la fonction n'est pas void:
  * return valeur;
  */
}
```

Comment écrire une fonction?

Exemples de définition

```
int iquo(int a, int b, int *prem) {
   assert(b != 0);
 /* pas de moyen simple  de tester la validité d'une adresse
 * donc pas d'assert pour tester la validité
 */
   *prem =a % b;
   return a / b;
}
```

Attention: `return` arrête la fonction, rien de ce qui suit ne sera exécuté.

```
int iquo(int a, int b, int *prem) {
   assert(b != 0);
   return a / b;
   /*Prg faux: la ligne n'est pas exécutée */
   *prem =a % b;
}
```

Comment écrire une fonction?

Exemples de définition type `void` $\Leftrightarrow$ pas de `return`

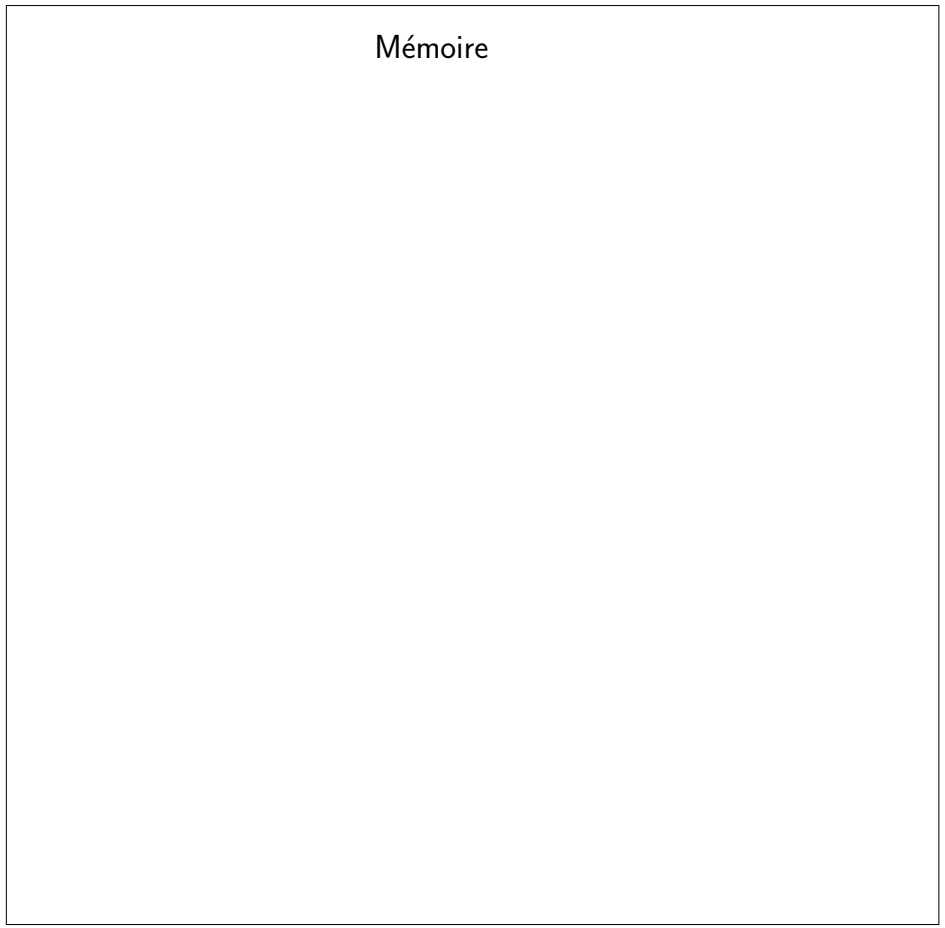
```
void inc(int *pa) {
/* pas de moyen simple  de tester la validité d'une adresse
 * donc pas d'assert
 */
   *pa += 1;
}
```

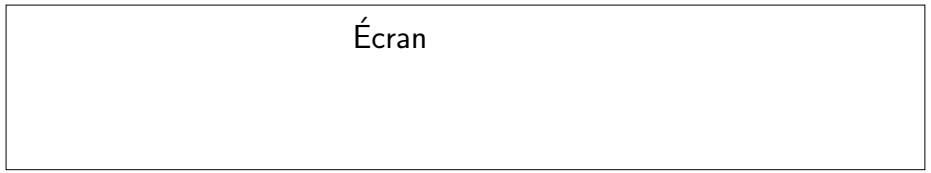
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



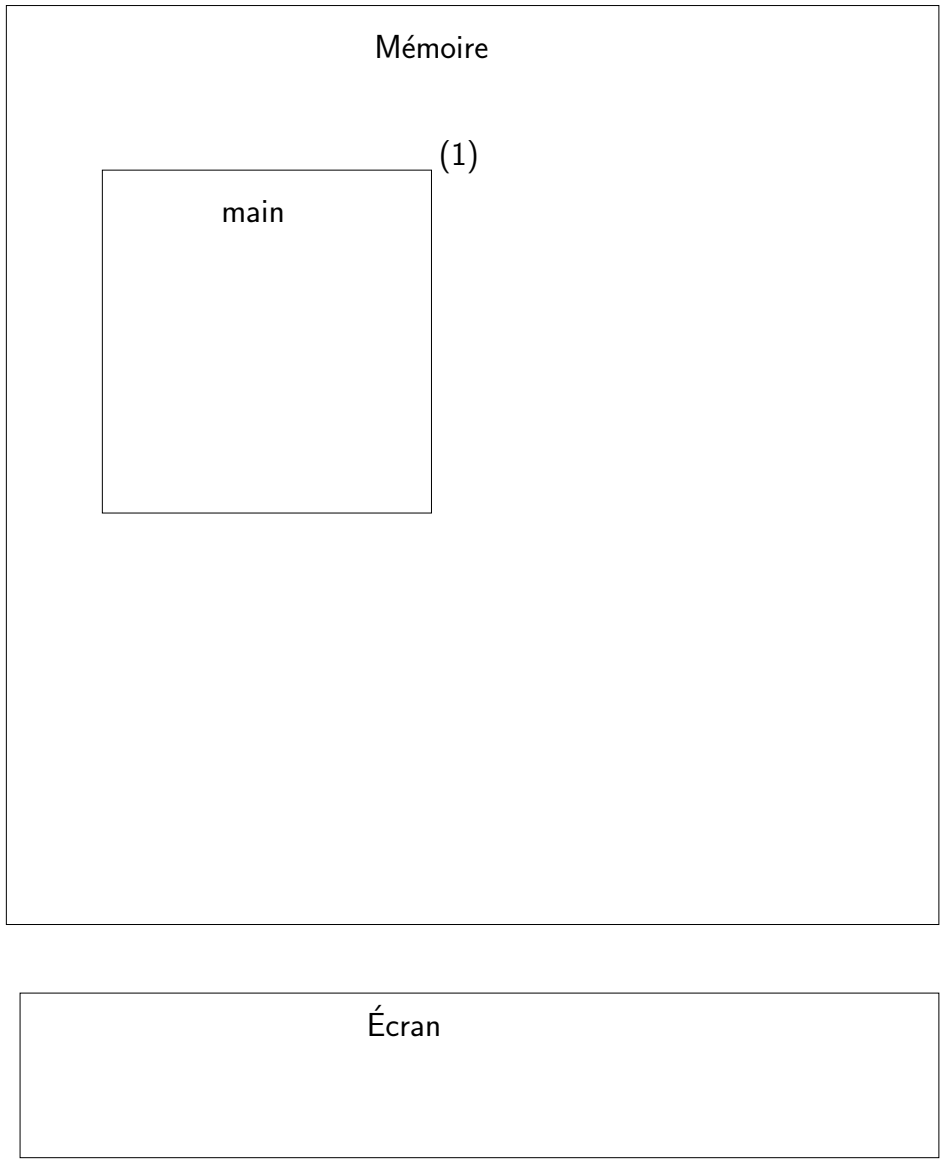


Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

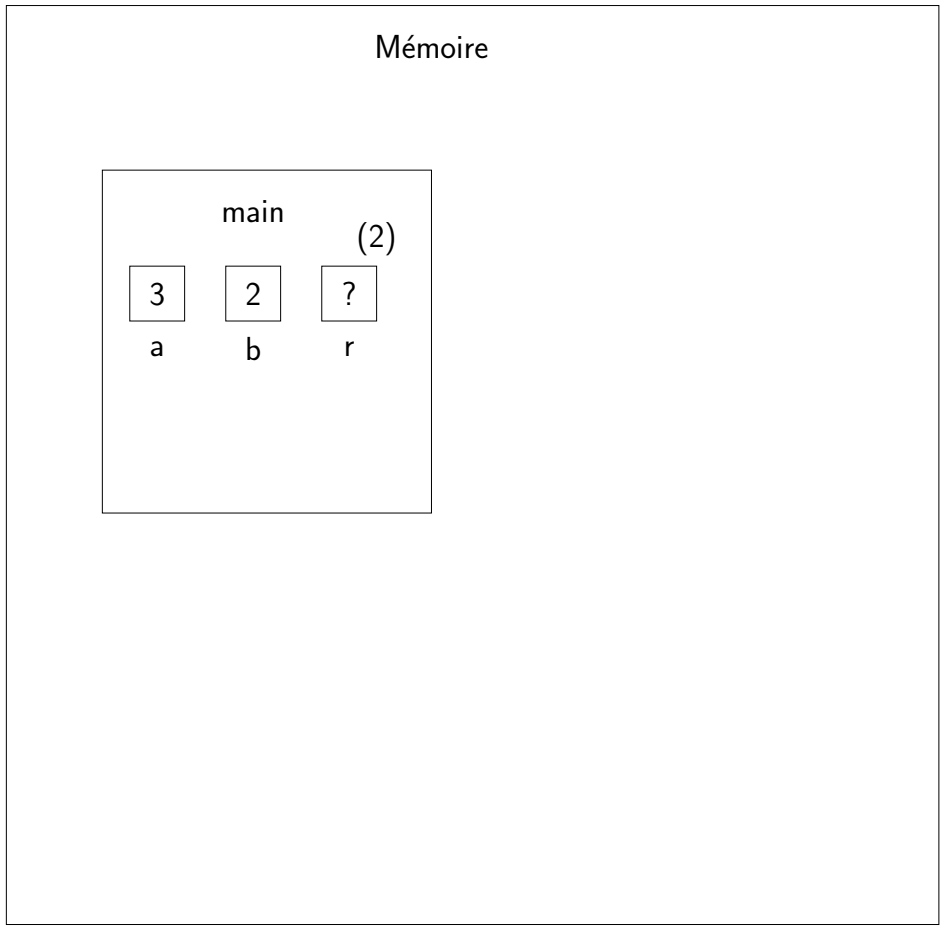


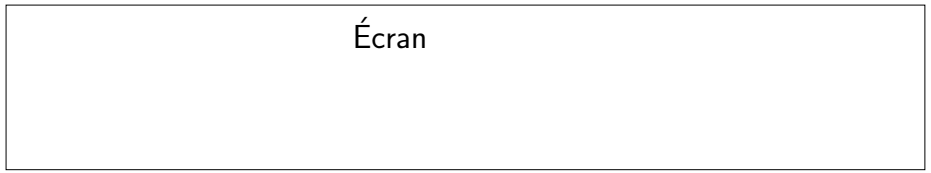
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



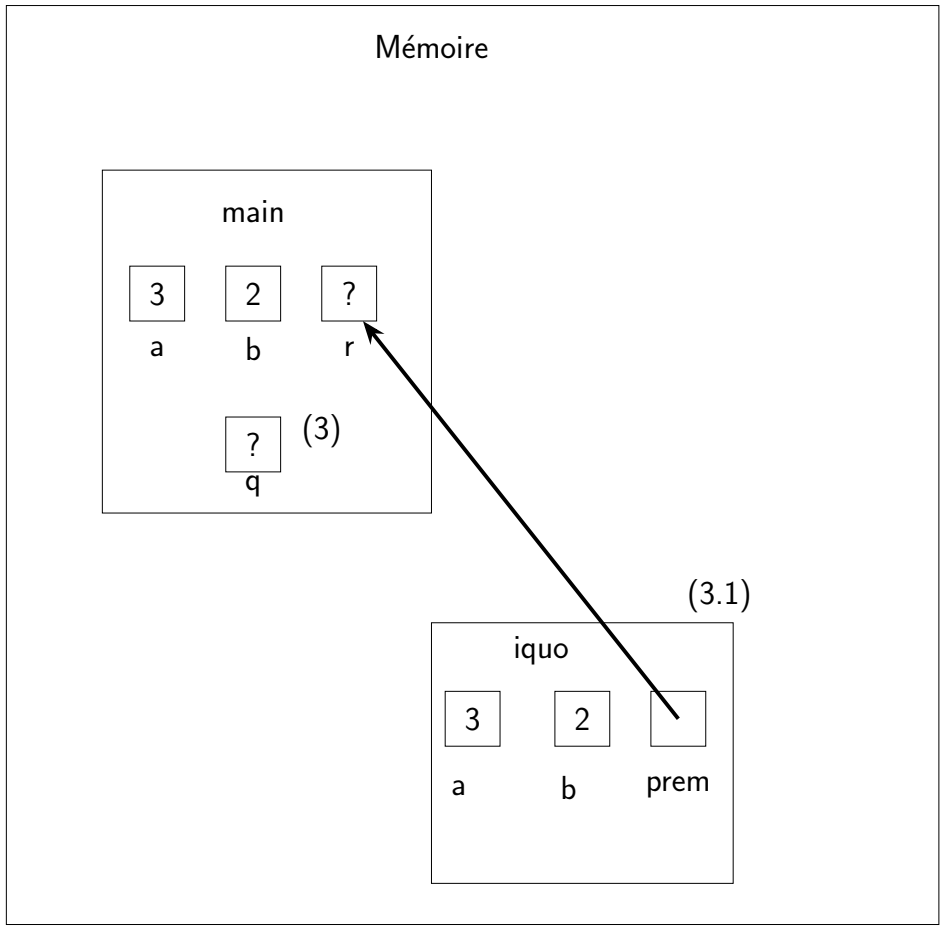


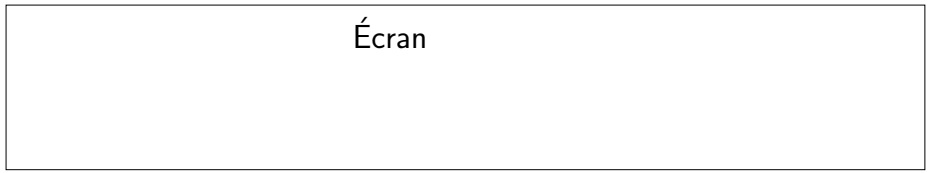
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



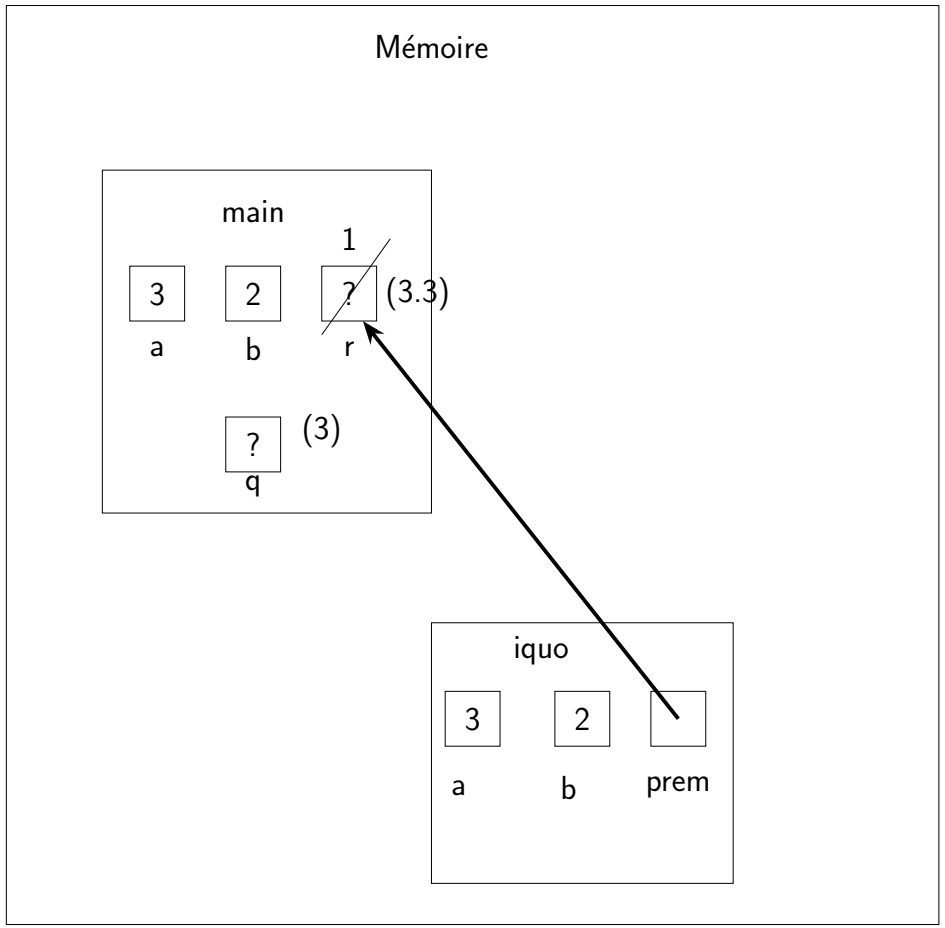


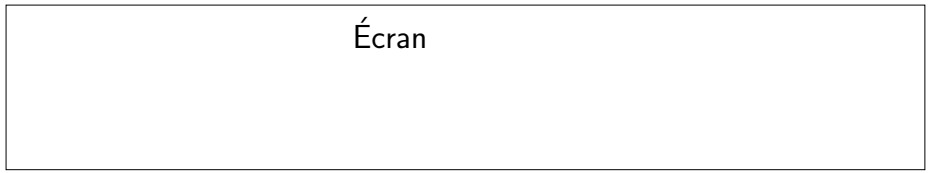
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



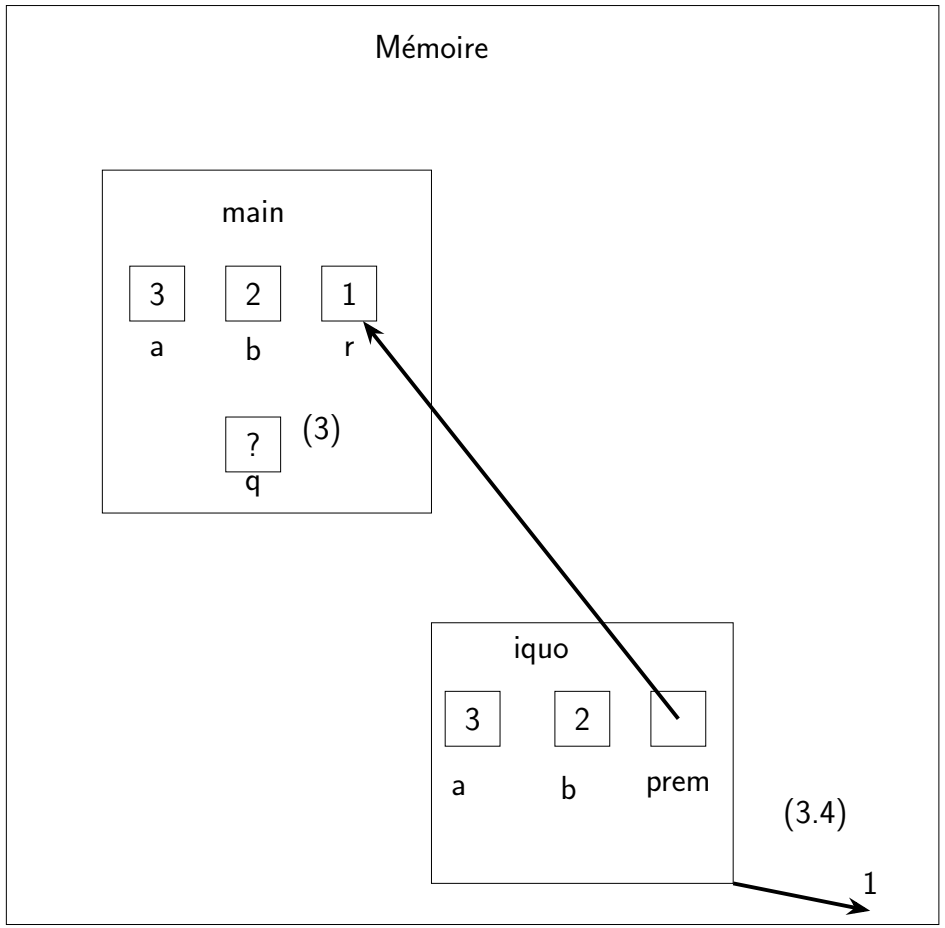


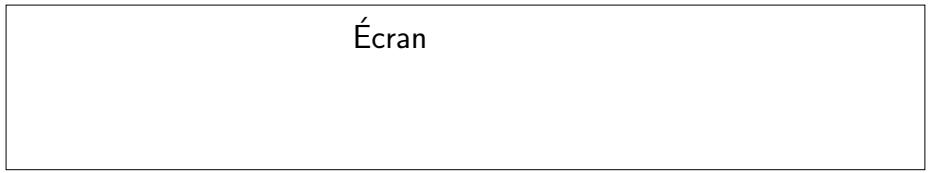
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



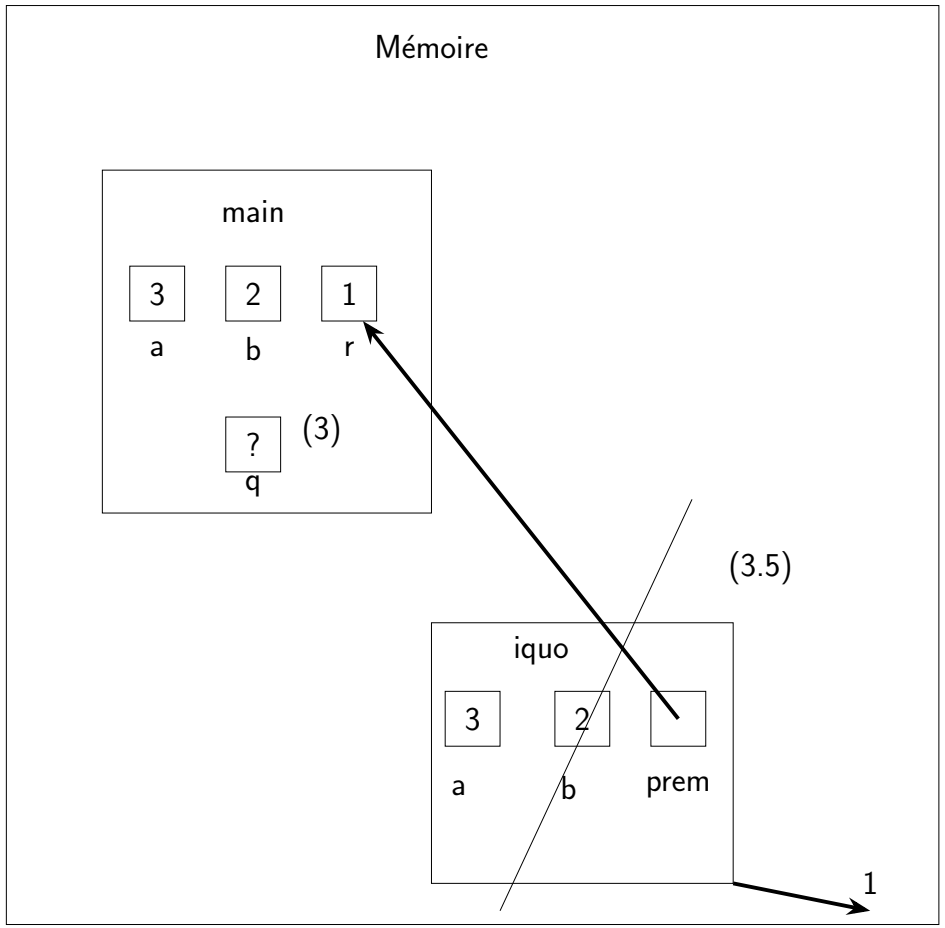


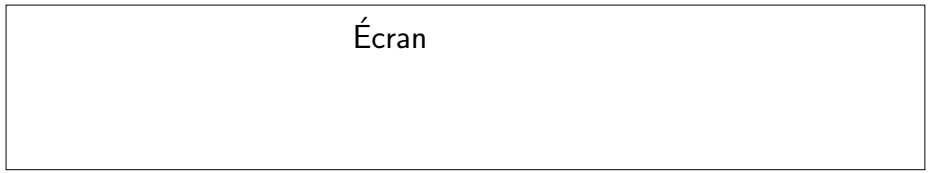
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



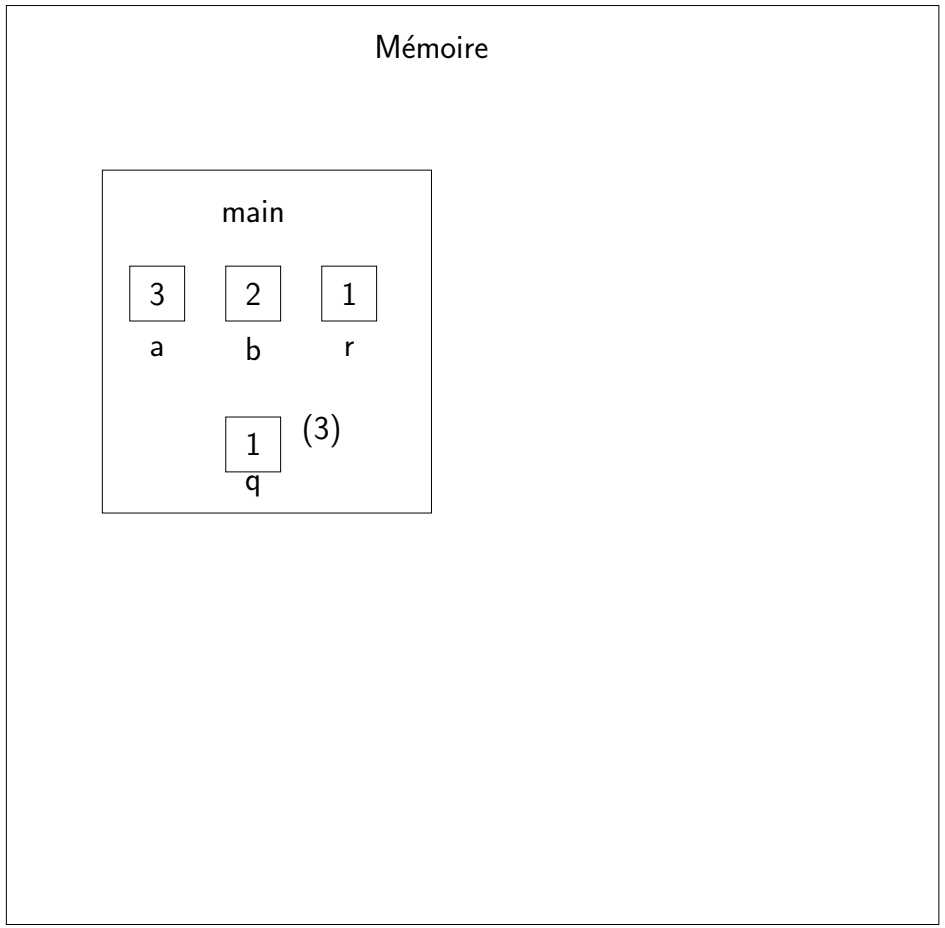


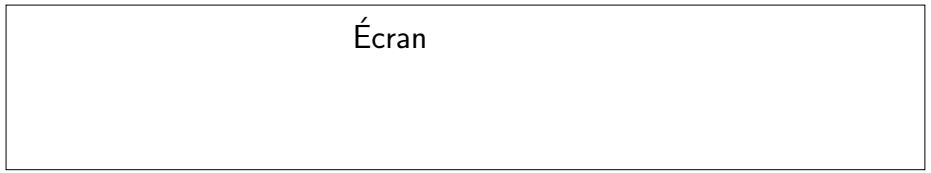
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



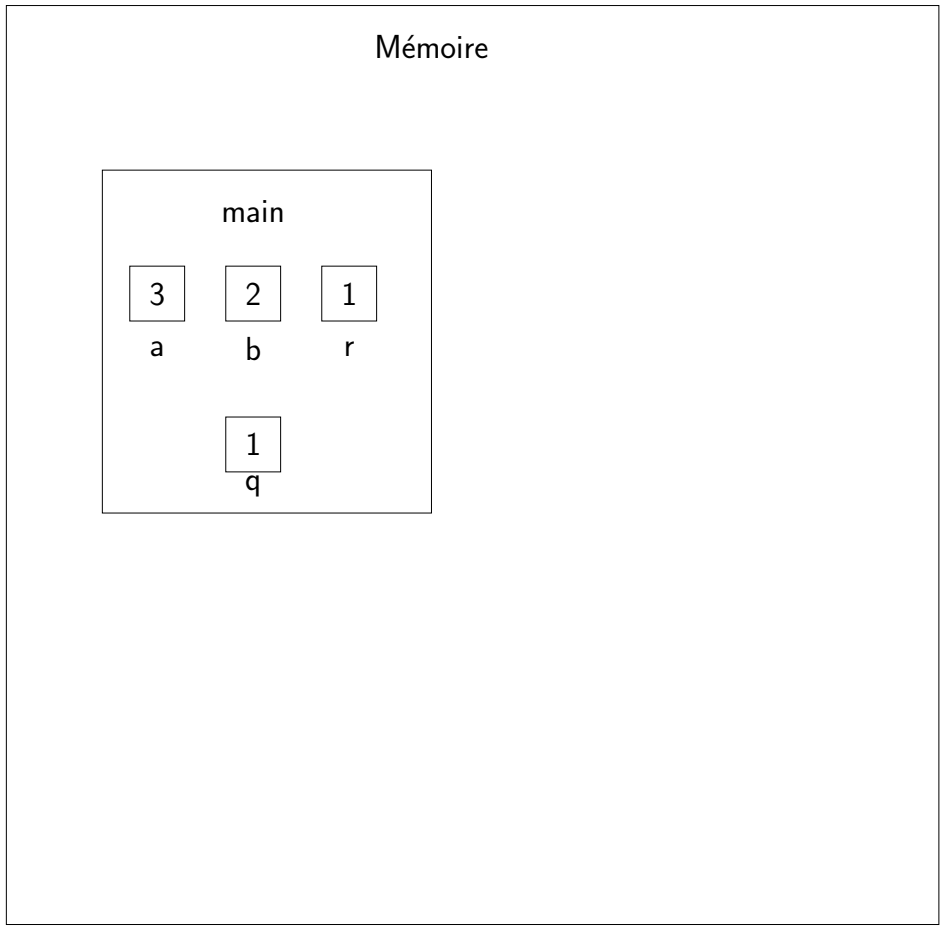


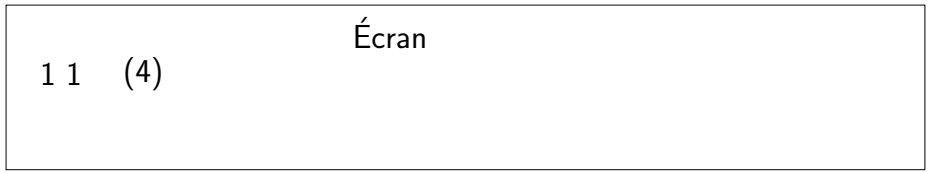
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



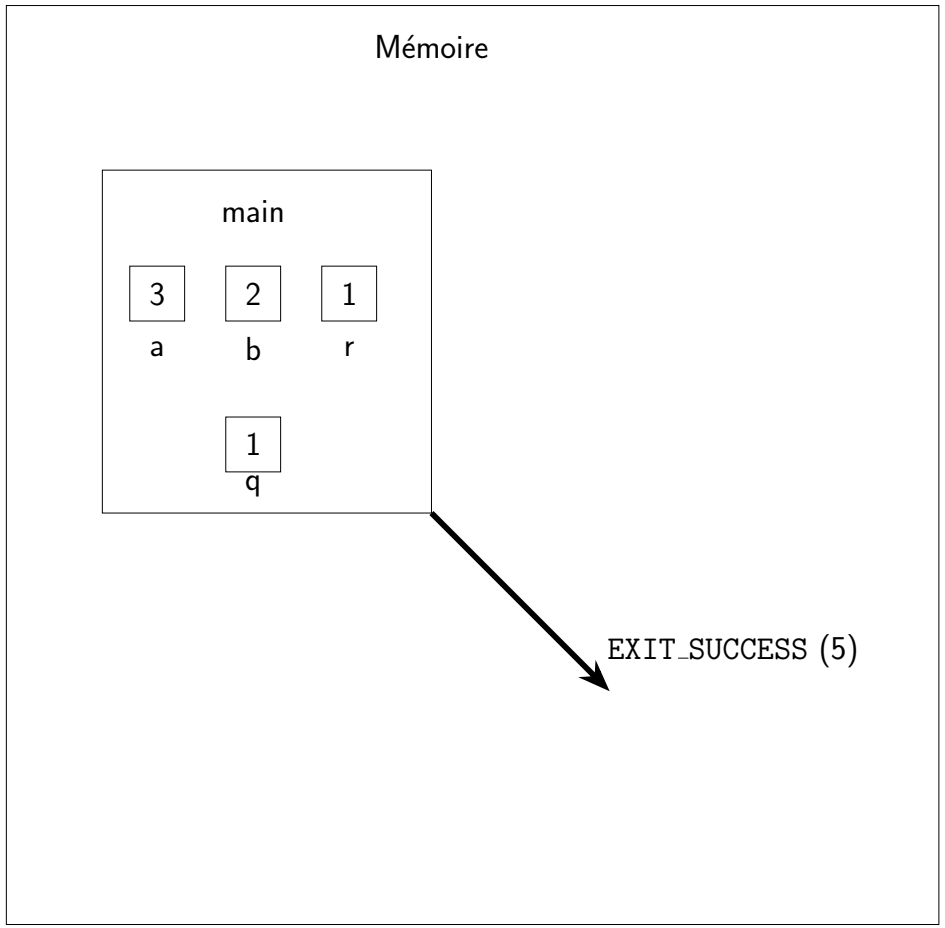


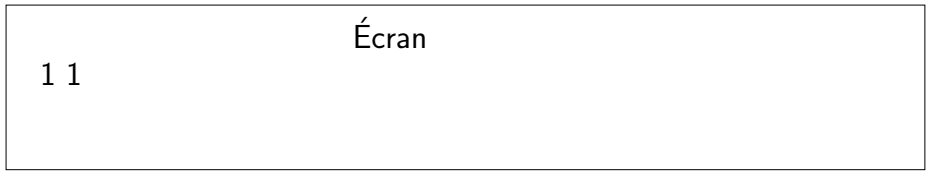
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



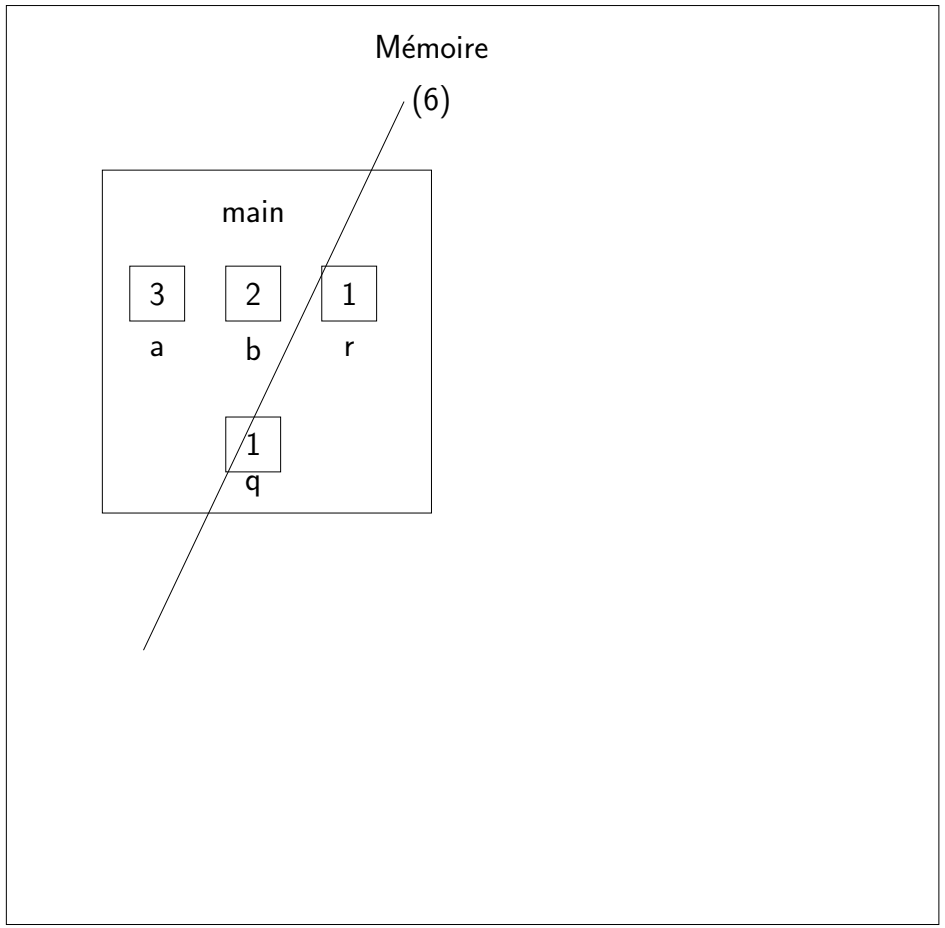


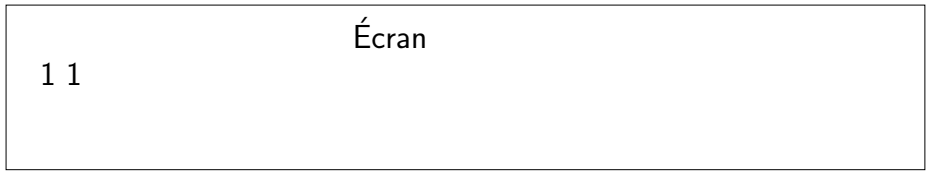
Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



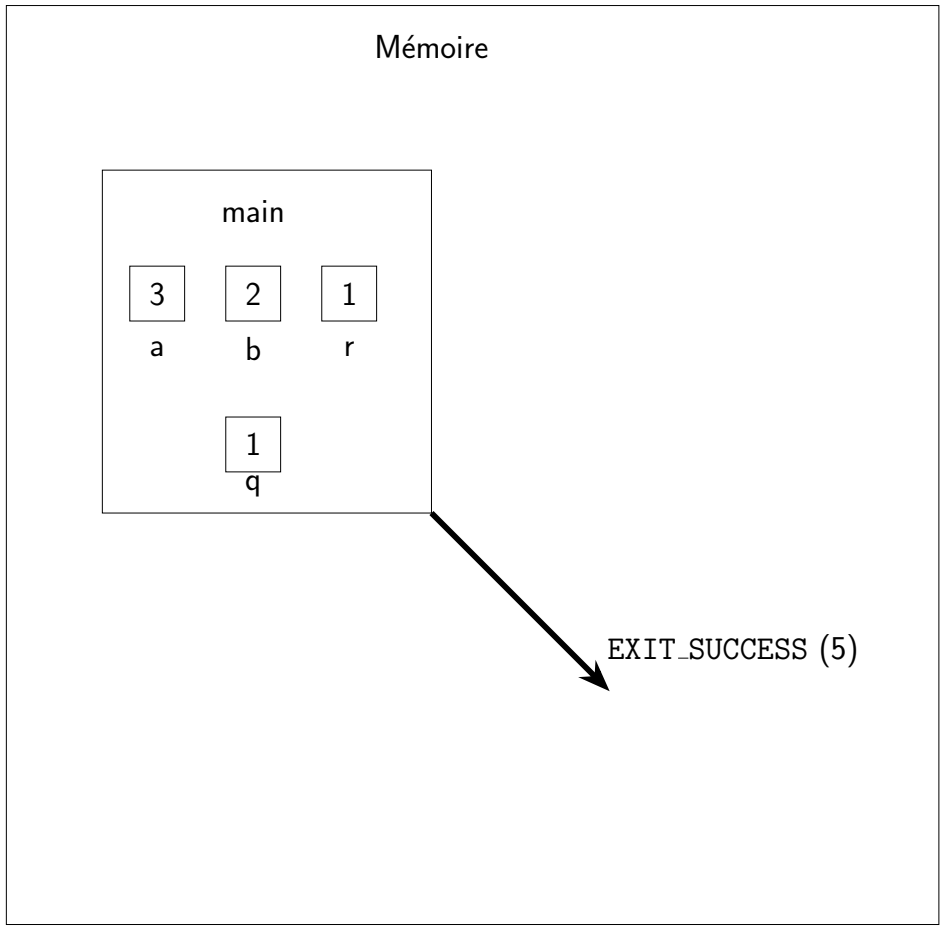


Comment s'exécute une fonction?

Sur un exemple

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```



Écran

1 1

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1) <-
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

$a^{\dagger}$	$b^{\dagger}$	r	q	$a^{\star}$	$b^{\star}$	$prem^{\star}$	$*prem^{\star}$	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)

- Il y a plusieurs variables qui ont le même nom.
  $^{\dagger} \Rightarrow$ variable de `main`, $^{\star} \Rightarrow$ variable de `iquo`
- - : variable non créée

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2) <-
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

$a^{\dagger}$	$b^{\dagger}$	r	q	$a^{\star}$	$b^{\star}$	$prem^{\star}$	$*prem^{\star}$	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)

? : pas de valeur affectée

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1) <-
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

a†	b†	r	q	a⋆	b⋆	prem⋆	*prem⋆	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)
|	|	(?)|	|	3	2	&r	?		(3.1)

- |: la variable est déclarée dans une autre fonction
- (val)|: Il y a un pointeur vers la variable dans la fonction (même couleur).

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5) <-
```

a†	b†	r	q	a⋆	b⋆	prem⋆	*prem⋆	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)
|	|	(?)|	|	3	2	&r	?		(3.1)
|	|	(1)|	|	3	2	&r	1		(3.2)-(3.5)

- On peut sauter des lignes faciles à expliquer.
- La valeur de `r` (fct `main`) a été modifiée par la ligne (3.3)

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3) <-
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

a†	b†	r	q	a⋆	b⋆	prem⋆	*prem⋆	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)
|	|	(?)|	|	3	2	&r	?		(3.1)
|	|	(1)|	|	3	2	&r	1		(3.2)-(3.5)
3	2	1	1	-	-	-	-		(3)

On revient dans `main`: les variables de `iquo` ne sont plus définies.

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4) <-
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5)
```

$a^{\dagger}$	$b^{\dagger}$	r	q	$a^{\star}$	$b^{\star}$	$prem^{\star}$	$*prem^{\star}$	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)
|	|	(?)|	|	3	2	&r	?		(3.1)
|	|	(1)|	|	3	2	&r	1		(3.2)-(3.5)
3	2	1	1	-	-	-	-		(3)
3	2	1	1	-	-	-	-	1 1	(4)

Prévoir une colonne affichage

Comment s'exécute une fonction?

Trace d'exécution

```
...
/* iquo:
 * ...
 */
 int iquo(int a, int b, int *prem);

 ...
 int main(void) { (1)
    int a = 3, b = 2, r; (2)
    int q = iquo (a, b, &r); (3)
    printf("%d %d\n", q, r); (4)
    return EXIT_SUCCESS; (5)
  } (6)
  ...
int iquo(int a, int b, int *prem) { (3.1)
    assert(b != 0); (3.2)
    *prem =a % b; (3.3)
    return a / b; (3.4)
 } (3.5) <-
```

a†	b†	r	q	a⋆	b⋆	prem⋆	*prem⋆	Affichage	Nb ligne
-	-	-	-	-	-	-	-		(1)
3	2	?	-	-	-	-	-		(2)
|	|	(?)|	|	3	2	&r	?		(3.1)
|	|	(1)|	|	3	2	&r	1		(3.2)-(3.5)
3	2	1	1	-	-	-	-		(3)
3	2	1	1	-	-	-	-	1 1	(4)
3	2	1	1	-	-	-	-		(5)-(6) Fin

Indiquer la fin du programme