

# Appels de fonction

Alexis Nasr  
Franck Dary  
Pacôme Perrotin

Compilation – L3 Informatique  
Département Informatique et Interactions  
Aix Marseille Université

# Appels de fonction I

- Les fonctions (et procédures) en assembleur sont simplement des adresses dans le code.
- Un appel de fonction consiste à :
  - **sauter** à l'adresse de la première instruction de la fonction appelée,
  - **exécuter** les instructions de la fonction appelée,
  - **revenir** à la prochaine instruction de la fonction appelante (l'instruction qui suit l'appel).

# Arbre d'activation

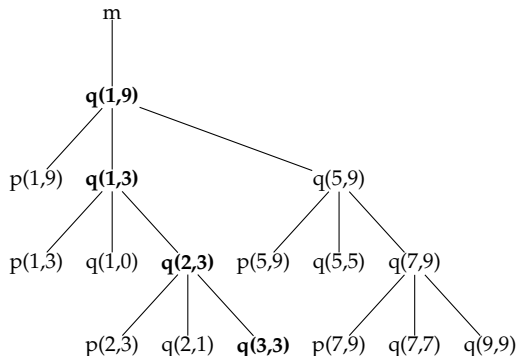
- Une fonction peut en appeler une autre, qui peut en appeler une autre ...
- A tout moment de l'exécution d'un programme, une seule fonction est **active**
- Les activations successives forment un **arbre d'activation**.
- Les activations se trouvant sur la branche de la fonction active sont en attente.

# Exemple : arbre d'activation du quicksort

```
int partition( int m, int n ) {  
    ...  
}
```

```
int quicksort(int m, int n) {  
    if(n > m) {  
        int i = partition(m, n);  
        quicksort(m, i-1);  
        quicksort(i+1,n);  
    }  
}
```

```
int main() {  
    quicksort(1,9);  
}
```



- Si la fonction active est  $q(3,3)$  alors  $\text{main}$ ,  $q(1,9)$ ,  $q(1,3)$  et  $q(2,3)$  sont en attente.
- Leurs paramètres doivent être sauvegardés en attendant la fin de  $q(3,3)$ .

# Communication I

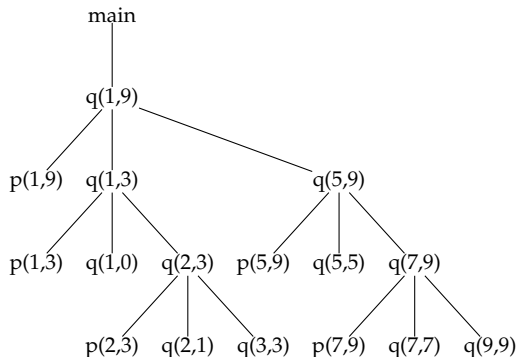
- La fonction appelée communique avec la fonction appelante à travers :
  - les paramètres d'appel,
  - la valeur de retour.
- De plus, la fonction appelée peut définir des variables locales.
- Où stocker les paramètres, la valeur de retour et les variables locales ?
- On ne peut pas les stocker dans des registres, ni dans la zone mémoire dédiée aux variables, car on ne peut savoir à l'avance combien il peut y avoir d'appels imbriqués.

# Trame de pile

- Bloc mémoire sur la pile, contenant toutes les informations sur une fonction n'ayant pas terminé son exécution
  - l'adresse à laquelle poursuivre l'exécution après l'appel
  - la valeur de retour de la fonction
  - ses paramètres
  - ses variables locales
- A chaque appel de fonction, une trame est créée et empilée.
- A la fin de l'appel, la trame est dépilée.

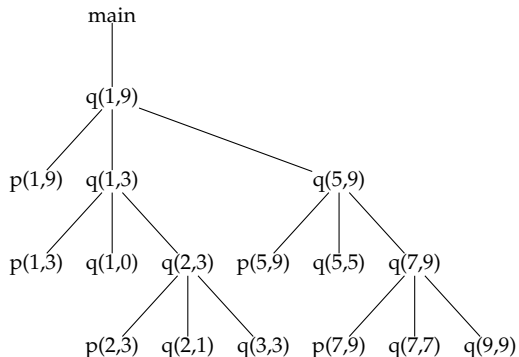
# Exemple : trames de piles lors de l'exécution de quicksort

main



# Exemple : trames de piles lors de l'exécution de quicksort

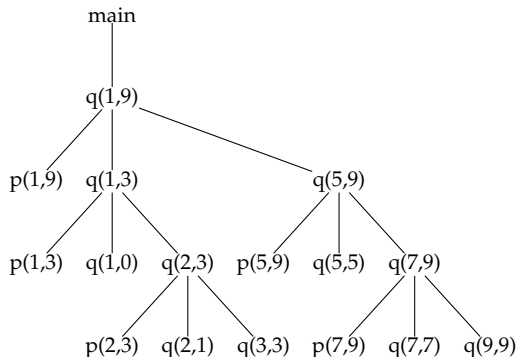
main
q(1,9)





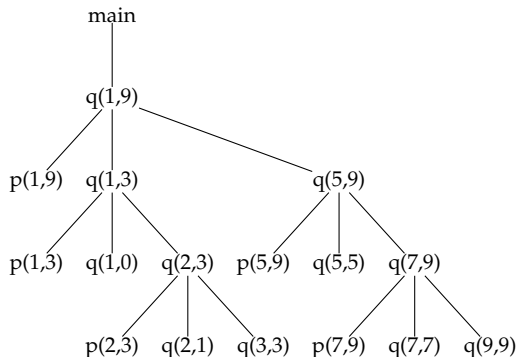
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
p(1,9)



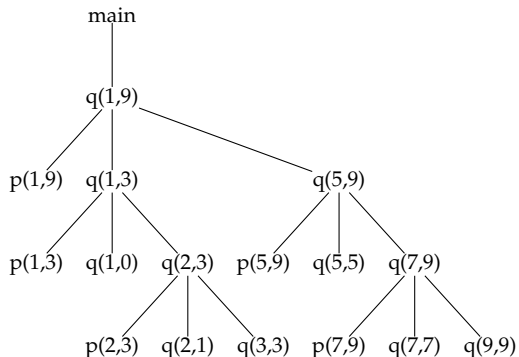
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)



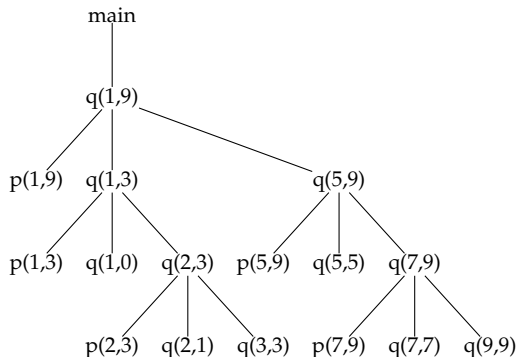
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)



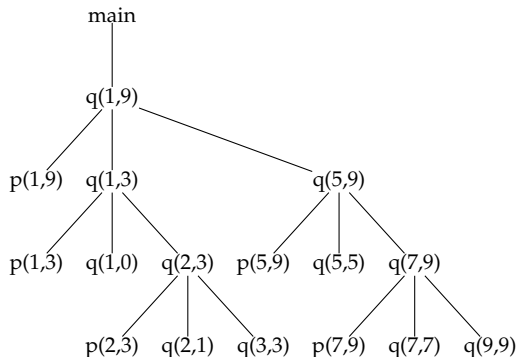
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
p(1,3)



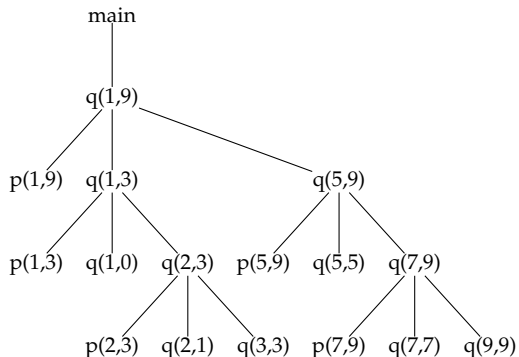
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)



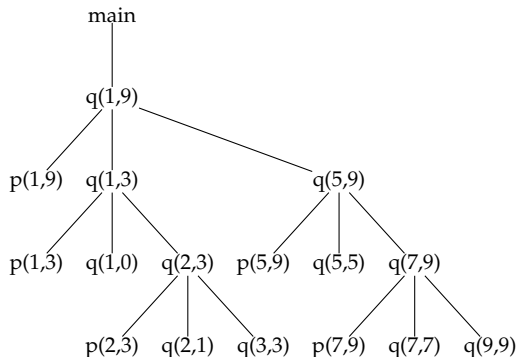
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(1,0)



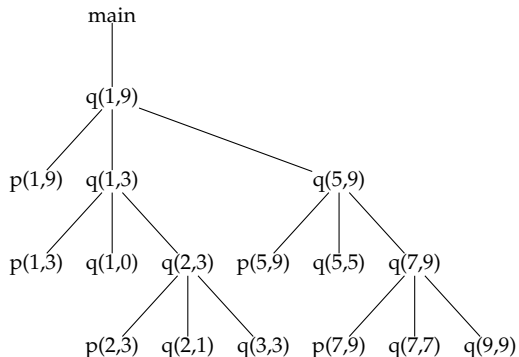
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)



# Exemple : trames de piles lors de l'exécution de quicksort

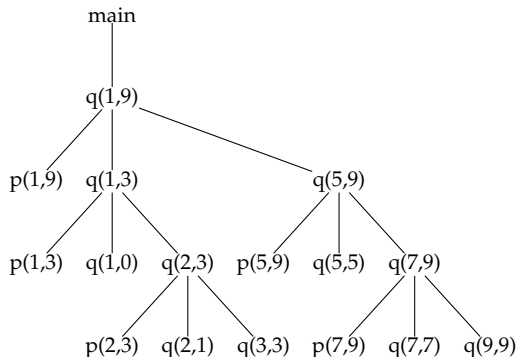
main
q(1,9)
q(1,3)
q(2,3)





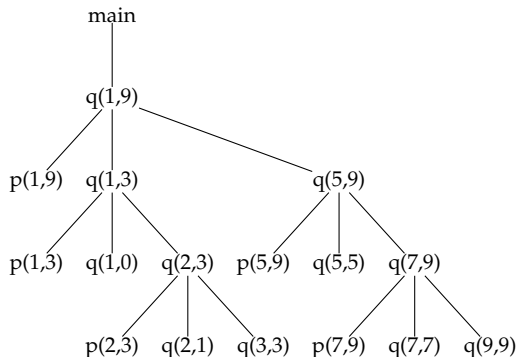
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(2,3)
p(2,3)



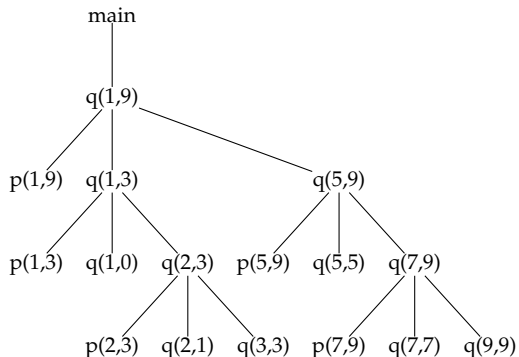
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(2,3)



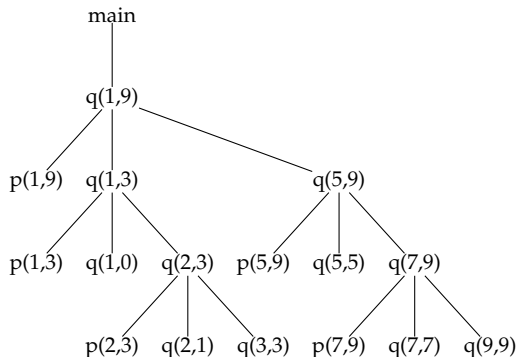
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(2,3)
q(2,1)



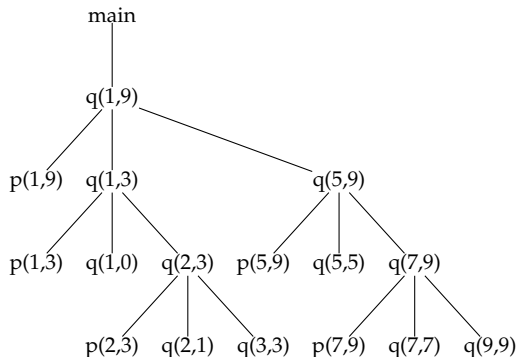
# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(2,3)



# Exemple : trames de piles lors de l'exécution de quicksort

main
q(1,9)
q(1,3)
q(2,3)
q(3,3)



# Organisation de la trame de pile

- Les éléments stockés dans la trame de pile doivent être rangés dans un ordre déterminé afin de pouvoir y accéder par un **calcul**.
- Un registre pointe sur un élément de la trame de pile active.
- En X86, ce registre s'appelle ebp (extended base pointer).
- L'accès aux éléments se fait par un calcul à partir de ebp.
- L'ordre dans lequel ils sont rangés est conventionnel, il est en général défini par le constructeur du processeur.
- Il est **primordial** de respecter cet ordre!

# Exemple d'organisation d'une trame de pile

	paramètre 1
	paramètre 2
	paramètre 3
	valeur de retour
	instruction à effectuer après l'appel
ebp	ancienne valeur de eip
	ancienne valeur de ebp
	variable locale 1
	variable locale 2
	variable locale 3
esp	

Rappel : la pile croît vers le bas.

# Calcul d'adresses

ebp + 20	paramètre 1
ebp + 16	paramètre 2
ebp + 12	paramètre 3
ebp + 8	valeur de retour
ebp + 4	instruction à effectuer après l'appel ancienne valeur de eip
ebp	ancienne valeur de ebp
ebp - 4	variable locale 1
ebp - 8	variable locale 2
ebp - 12	variable locale 3



# Calcul d'adresses

$ebp + 8 + 4 * 3 - 0$	$ebp + 20$
$ebp + 8 + 4 * 3 - 4$	$ebp + 16$
$ebp + 8 + 4 * 3 - 8$	$ebp + 12$
$ebp + 8$	$ebp + 8$
$ebp + 4$	$ebp + 4$
	$ebp$
$ebp - 4 - 0$	$ebp - 4$
$ebp - 4 - 4$	$ebp - 8$
$ebp - 4 - 8$	$ebp - 12$

paramètre 1
paramètre 2
paramètre 3
valeur de retour
instruction à effectuer après l'appel
ancienne valeur de eip
ancienne valeur de ebp
variable locale 1
variable locale 2
variable locale 3

# Calcul d'adresses

$ebp + 8 + 4 * 3 - 0$	$ebp + 20$
$ebp + 8 + 4 * 3 - 4$	$ebp + 16$
$ebp + 8 + 4 * 3 - 8$	$ebp + 12$
$ebp + 8$	$ebp + 8$
$ebp + 4$	$ebp + 4$
	$ebp$
$ebp - 4 - 0$	$ebp - 4$
$ebp - 4 - 4$	$ebp - 8$
$ebp - 4 - 8$	$ebp - 12$

paramètre 1
paramètre 2
paramètre 3
valeur de retour
instruction à effectuer après l'appel ancienne valeur de eip
ancienne valeur de ebp
variable locale 1
variable locale 2
variable locale 3

- Une variable locale a :  $ebp - a.adresse$
- Un argument a :  $ebp + 8 + 4 * nb\_args - a.adresse$
- La valeur de retour :  $ebp + 8$

# Conventions d'appel (*calling sequences*)

- Séquence d'opérations effectuées pour allouer et désallouer les trames de pile
- Certaines sont effectuées par la fonction *appelante* (*caller*) et d'autres par la fonction *appelée* (*callee*)
- Ordre des opérations au retour inversé : chacun désalloue ce qu'il a alloué.

# Appels — côté appelant

Lors d'un appel de fonction, il faut :

- 1 empiler les arguments (`param`),
- 2 réserver de la place pour le résultat,
- 3 empiler le pointeur de programme `eip` (fait automatiquement par `call`),
- 4 aller à l'adresse de la fonction (fait automatiquement par `call`),

A l'issue de l'appel (après le `call`), il faut :

- 1 dépiler le résultat,
- 2 désallouer la mémoire des arguments

# Appels — côté appelé

Quand on entre dans une fonction (`fbegin`), il faut :

- 1 empiler le frame pointer `ebp`,
- 2 initialiser la nouvelle valeur de `ebp`,
- 3 réserver de la place dans la pile pour les variables locales.

Quand on en sort (`fbend`), il faut :

- 1 stocker le résultat,
- 2 désallouer la mémoire des variables locales,
- 3 dépiler le frame pointer `ebp`,
- 4 dépiler le pointeur de programme `eip`  
(fait automatiquement par `ret`),
- 5 sauter vers l'adresse de retour (fait automatiquement par `ret`).

# Exemple : fonction simple

Code-source *L* :

```
entier a;  
double( entier n ) {  retour n + n; }  
main() {  a = double( 3 ); }
```

Code trois adresses :

```
1          alloc 1 va      ; alloue variable globale  
2  fdouble: fbegin  
3          t0 = vn + vn   ; somme des arguments  
4          ret t0         ; sauvegarder la valeur de retour  
5          fend           ; terminer la fonction (instr. retour)  
6          fend           ; terminer la fonction (fin fonction)  
7  fmain : fbegin  
8          alloc 1        ; allouer place pour la valeur de retour  
9          param 3        ; premier paramètre  
10         t1 = call fdouble  
11         va = t1  
12         fend
```

# Exemple : fonction simple en x86

## Code-source L :

```
entier a;  
double( entier n ) { retour n + n; }  
main() { a = double( 3 ); }
```

## Code machine x86 :

fmain:

```
push ebp  
mov  ebp, esp  
push 3  
sub  esp, 4  
call fdouble  
pop  ebx  
add  esp, 4  
mov  dword[va], ebx  
pop  ebp  
ret
```

fdouble:

```
push ebp  
mov  ebp, esp  
mov  ebx, dword[ebp + 12]  
add  ebx, dword[ebp + 12]  
mov  [ebp + 8], ebx  
pop  ebp  
ret
```

# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```

120

116

112

108

104

100

096

092

va

ebx

eip

esp

ebp



# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```

120

116

112

108

104

100

096

092



va

ebx

eip

esp

ebp



# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```

120

116

112

108

104

100

096

092



va

ebx

eip

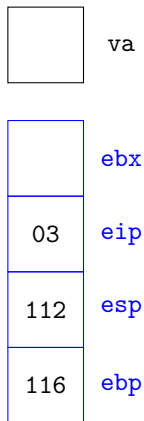
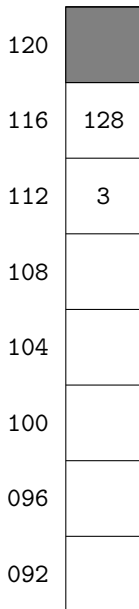
esp

ebp



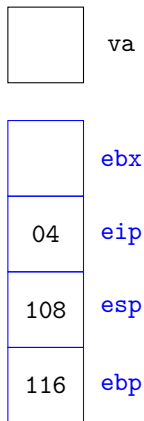
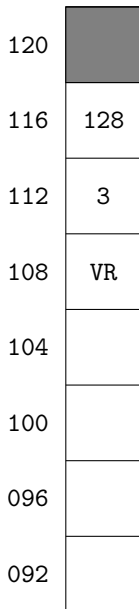
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



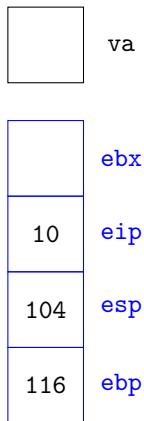
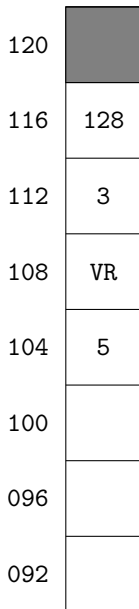
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



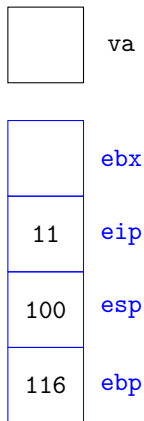
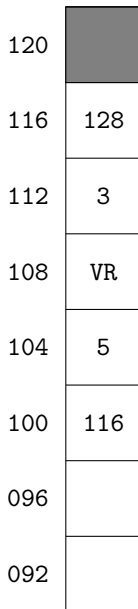
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



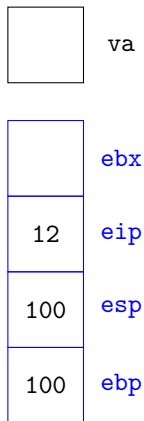
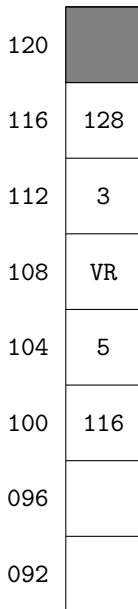
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



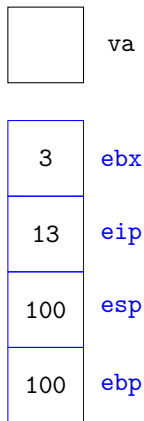
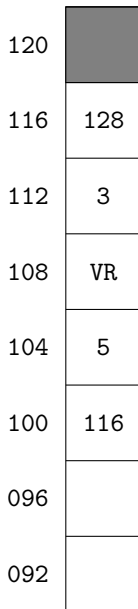
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



# Exemple d'appel de fonction

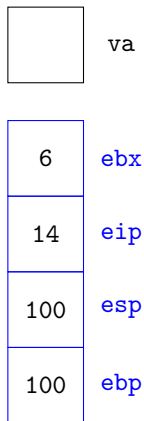
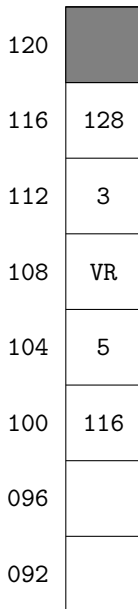
```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```





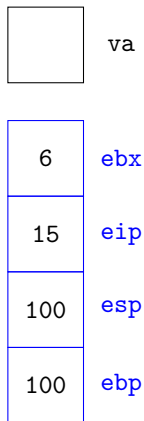
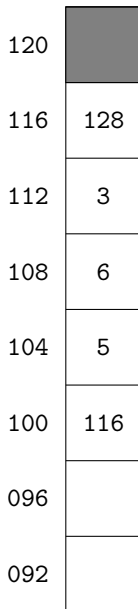
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



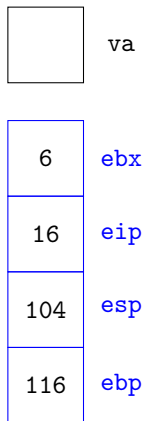
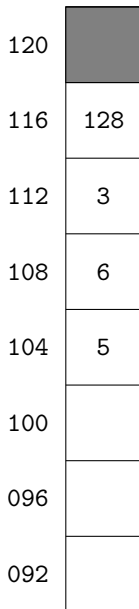
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



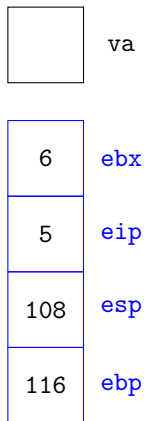
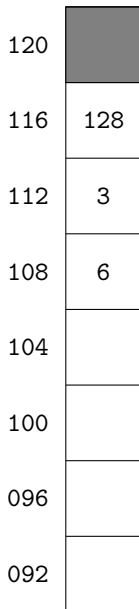
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



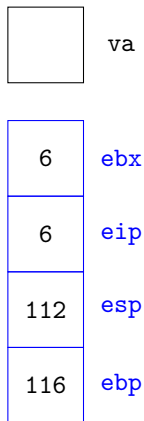
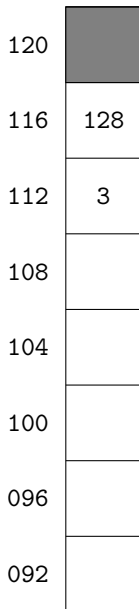
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



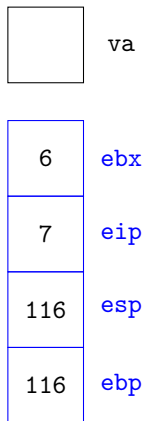
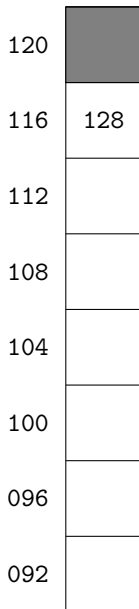
# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```



# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```

120

116

112

108

104

100

096

092



6

va

6

ebx

8

eip

116

esp

116

ebp

# Exemple d'appel de fonction

```
00 fmain: push ebp
01      mov  ebp, esp
02      push 3
03      sub  esp, 4
04      call fdouble
05      pop  ebx
06      add  esp, 4
07      mov  dword[va], ebx
08      pop  ebp
09      ret
10 fdouble: push ebp
11      mov  ebp, esp
12      mov  ebx, dword[ebp + 12]
13      add  ebx, dword[ebp + 12]
14      mov  [ebp + 8], ebx
15      pop  ebp
16      ret
```

120

116

112

108

104

100

096

092

6

va

6

ebx

9

eip

120

esp

128

ebp